UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"Confronto tra protocolli di comunicazione tastiera-computer"

Relatore: **Prof. Roberto Corvaja**

Laureando: **Enrico Disarò**

ANNO ACCADEMICO 2023 – 2024

Data di laurea: 15 luglio 2024

“Roccioso.”

Indice

Indice v

Indice delle figure vii

Introduzione ix

Il progetto della tastiera 11

1.1 Scopo del progetto 11

1.2 Lato hardware 12

1.2.1 Il microcontrollore 12

1.2.2 I tasti 13

1.2.3 Il circuito 13

1.3 Lato software 15

Il protocollo seriale 17

2.1 Storia delle prime tastiere 17

2.2 Il protocollo PS/2 19

2.2.1 Interfaccia elettrica 19

2.2.2 Principi della comunicazione 20

2.2.3 Comunicazione tastiera-computer 21

2.2.4 Comunicazione computer-tastiera 22

2.3 Implementazione del protocollo 24

2.3.1 Interfaccia elettrica 24

2.3.2 Codice 25

Il protocollo HID 35

3.1 Descrittori HID 36

3.1.1 HID descriptor 36

3.1.2 Report descriptor 37

3.1.3 Physical descriptor 40

3.2 HID per le tastiere 40

3.2.1 Struttura dei report 40

3.2.2 Principi della comunicazione 41

3.2.3 Comunicazione tastiera-computer 42

3.2.4 Limitazioni 43

3.2.4 Comunicazione computer-tastiera 44

3.3 Implementazione del protocollo 45

3.3.1 Oggetti 46

3.3.2 Creazione del report descriptor 48

3.3.3 File di configurazione 50

3.3.4 Codice 51

Conclusioni 57

Bibliografia 61

Repositories GitHub 61

Indice delle figure

Figura 1: Piedinatura del Raspberry Pi Pico [1] 12
Figura 2: Funzionamento di un tasto meccanico 13
Figura 3: Diagramma schematico del circuito 14
Figura 4: Schema del PCB 14
Figura 5: connettore DIN 5-pin a 180° 18
Figura 6: Connettore PS/2 per tastiera 18
Figura 7: piedinatura del connettore PS/2 [2] 19
Figura 8: Circuito con open collectors [2] 20
Figura 9: valori assunti da Clock e Data nella trasmissione di un frame [2] 21
Figura 10: trasmissione di un frame da computer a tastiera [2] durante (a) viene inibita la trasmissione e inviato il request-to-send durante (b) avviene l'effettiva trasmissione del frame 23
Figura 11: schema del circuito per il collegamento del connettore PS/2 24
Figura 12: struttura di un HID descriptor [3] 36
Figura 13: Struttura di un oggetto [3] 37
Figura 14: usage table per funzioni generiche di desktop [4] 38
Figura 15: esempio di report descriptor per una tastiera [3] 39
Figura 16: esempio di report 41
Figura 17: confronto schemi di comunicazione attraverso polling e interrupt 43
Figura 18: esempio di report in stato di errore 44
Figura 19: richiesta GET_REPORT secondo lo standard USB 45
Figura 20: richiesta GET_REPORT secondo lo standard Bluetooth 45
Figura 21: Main Items table 46
Figura 22: Global Items table 47
Figura 23: Local Items table 47
Figura 24: nomi utilizzati per indicare le componenti del report 48

Introduzione

Il computer è ormai uno strumento incredibilmente diffuso e utilizzato nei più diversi contesti, il quale è stato protagonista di una incessante evoluzione, che continua ancora oggi. Al suo fianco sono sempre rimaste le periferiche, dispositivi che hanno permesso all'uomo di interfacciarsi con la macchina nel corso del tempo, e che hanno avuto necessità di reinventarsi nel corso del tempo per restare al passo con le necessità e le capacità degli elaboratori in continuo cambiamento. Questa tesi si propone di studiare i principali tipi di protocolli (passati e presenti) utilizzati per la comunicazione tra computer e tastiera, probabilmente la periferica più indispensabile. La necessità di mettere mano nel firmware dei dispositivi (e una moderata influenza da parte della mia passione per le tastiere meccaniche) mi ha spinto a progettare e costruire un macropad su cui implementare i sistemi di comunicazione trattati.

La struttura della tesi è brevemente riportata qui sotto:

Capitolo 1: presentazione del progetto del tastierino per macro.

Capitolo 2: storia delle prime tastiere e analisi del protocollo PS/2, il primo standard per la comunicazione tastiera-computer, con proposta di implementazione in linguaggio C.

Capitolo 3: presentazione del protocollo HID, analisi delle strutture dati utilizzate per la comunicazione e proposta di implementazione in linguaggio CircuitPython.

Capitolo 4: confronto tra i due protocolli e conclusioni.

Capitolo 1

Il progetto della tastiera

1.1 Scopo del progetto

Ci troviamo in un'epoca in cui ottenere strumenti elettronici e informatici risulta incredibilmente semplice, e questo vale ovviamente anche per le tastiere. Il mercato è inondato di modelli di tutti i tipi, per ogni esigenza e pronti all'uso. Questa tesi si propone di implementare anche una parte applicativa riguardo ai protocolli trattati, provando ad utilizzarli per far comunicare una tastiera con un computer. Tuttavia, nessuna delle periferiche acquistabili permette di mettere mano sul suo funzionamento a basso livello, perché lo scopo di un venditore è ovviamente quello di fornire un dispositivo pronto all'uso per l'utente finale, il quale non si interessa di software o hardware, ma si preoccupa solo di avere una tastiera ben funzionante.

Questo è il motivo che mi ha portato a decidere di applicare le conoscenze informatiche ed elettroniche acquisite negli ultimi anni per provare a realizzare personalmente una tastiera, ideandola dall'inizio alla fine: in questo modo avrei avuto il pieno controllo di ogni area del suo funzionamento. Così, dopo alcune settimane di ricerca e progettazione sono riuscito a creare un dispositivo adatto ai miei obiettivi.

Il prodotto finale è un tastierino di 12 pulsanti, programmato per supportare tasti macro e funzioni personalizzate, pensato come un utile strumento per la semplificazione di azioni ripetitive, che possa accompagnare nello sviluppo di progetti. L'interfaccia con il computer e il funzionamento sono gli stessi di una normale tastiera per la scrittura grazie alla possibilità di implementare i protocolli di comunicazione convenzionali supportati da ogni macchina, ma il formato ridotto rende la gestione leggermente più semplice e aggiunge un po' di originalità al progetto.

1.2 Lato hardware

I componenti fondamentali necessari per la realizzazione di una tastiera sono un microcontrollore, alcuni interruttori meccanici e un circuito stampato. Di seguito presento quanto utilizzato nel mio progetto.

1.2.1 Il microcontrollore

Il microcontrollore che ho scelto per il progetto è una scheda Raspberry Pi Pico con un microprocessore RP2040. Esso si presenta come un controllore ad alte prestazioni e basso costo, open-source e supportato da una grande comunità distribuita in tutto il globo. Le principali caratteristiche che lo hanno reso un'ottima scelta per la realizzazione della tastiera sono:

- 2MB di memoria flash e 264KB di SRAM
- Processore RP2040 con clock fino a 133MHz
- 26 GPIO (general purpose input-output) pins
- Possibilità di creare porte seriali virtuali
- Supporto di linguaggio C/C++ e CircuitPython
- Supporto programmazione drag-and-drop

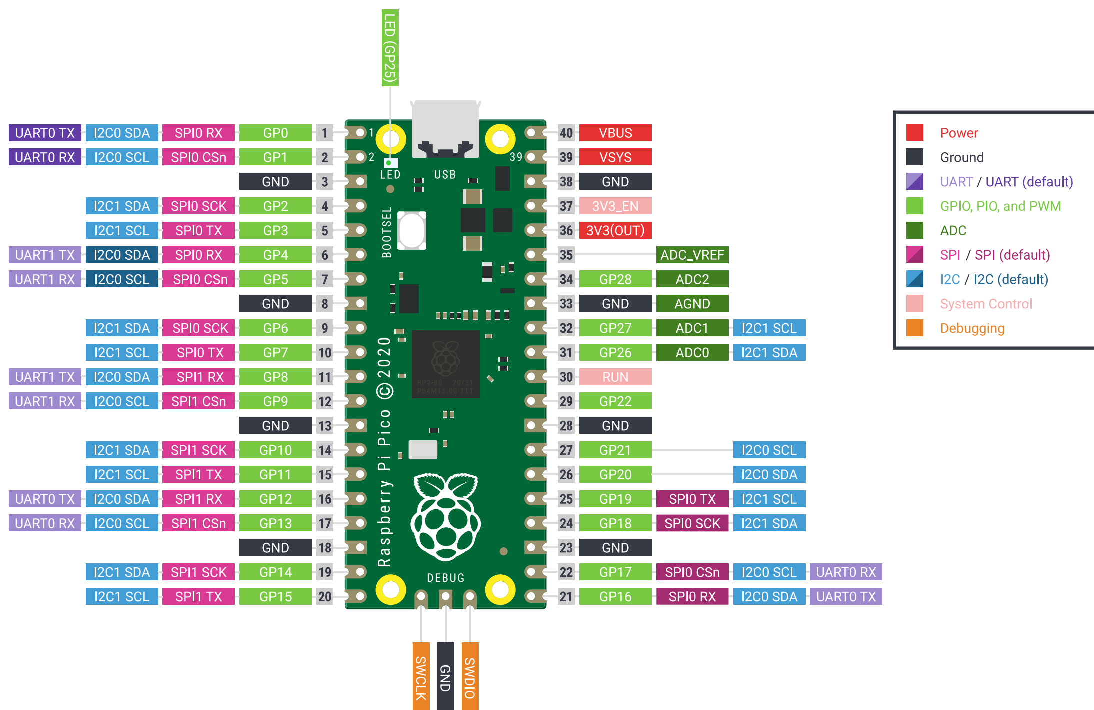

Figura 1: Piedinatura del Raspberry Pi Pico [1]

1.2.2 I tasti

Il mondo dei tasti meccanici è immenso e variegato, ne esistono di ogni tipo, ognuno con le proprie caratteristiche peculiari. Per gli scopi di questa tesi, è sufficiente sapere che un tasto meccanico funziona a tutti gli effetti come un interruttore: è dotato di due piedini metallici che vengono messi a contatto quando il tasto viene premuto, e questo permette di chiudere il circuito a cui sono collegati.

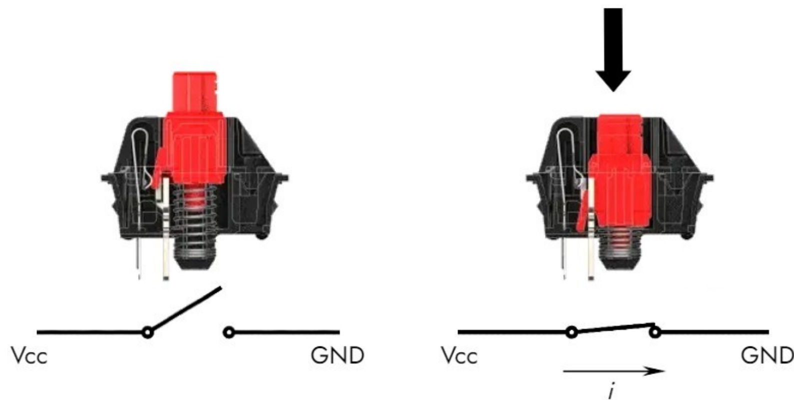

Figura 2: Funzionamento di un tasto meccanico

1.2.3 Il circuito

Esistono due approcci principali alla progettazione di un circuito per una tastiera: il cablaggio diretto e il cablaggio a matrice. Il primo è più banale e intuitivo, con lo svantaggio di essere poco scalabile e adatto solo alla realizzazione di periferiche con un numero contenuto di tasti. Siccome il progetto ne prevede solo 12, ho ritenuto questo approccio il più adatto per le mie necessità.

L'idea di base è piuttosto semplice: ogni tasto è gestito da un GPIO pin diverso del controllore, ed è sufficiente collegare un piedino del tasto a GND e l'altro al pin associato. La pressione dell'interruttore provoca la chiusura del circuito che collega il pin a terra, permettendo il passaggio di corrente tra i due punti e consentendo così al controllore di rilevare il cambiamento di stato. La lettura degli input è immediata per ogni pin, e il circuito non richiede elementi aggiuntivi. Di seguito vengono riportati i file per la progettazione del circuito stampato utilizzando il software CAD KiCad 7.0

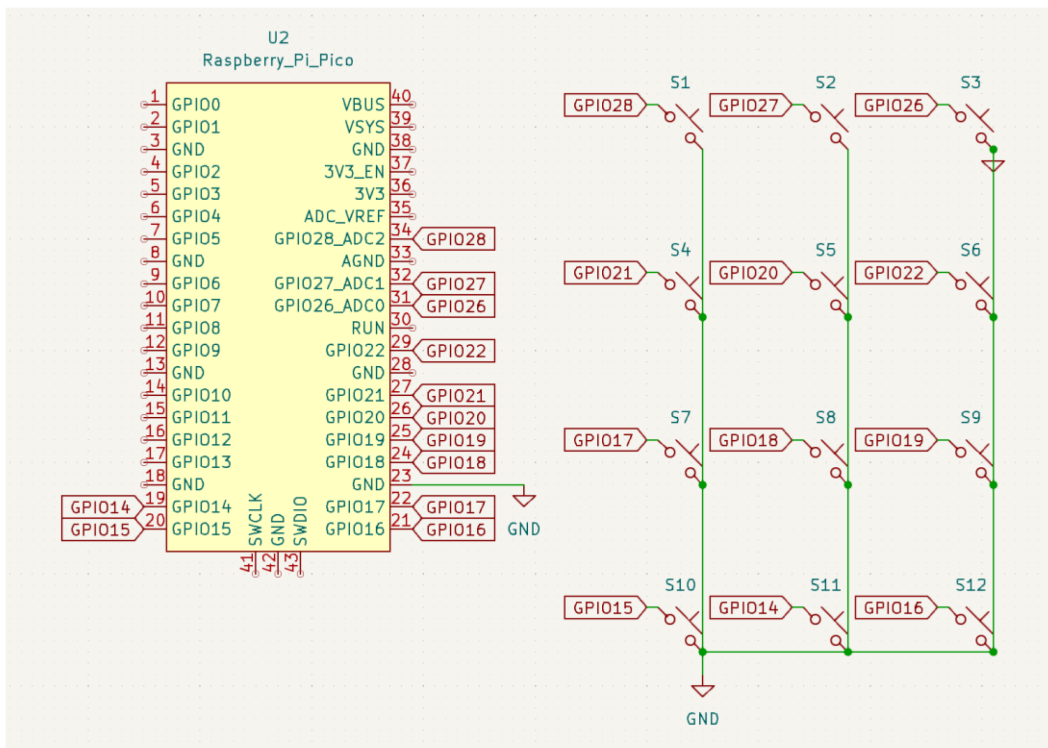

Figura 3: Diagramma schematico del circuito

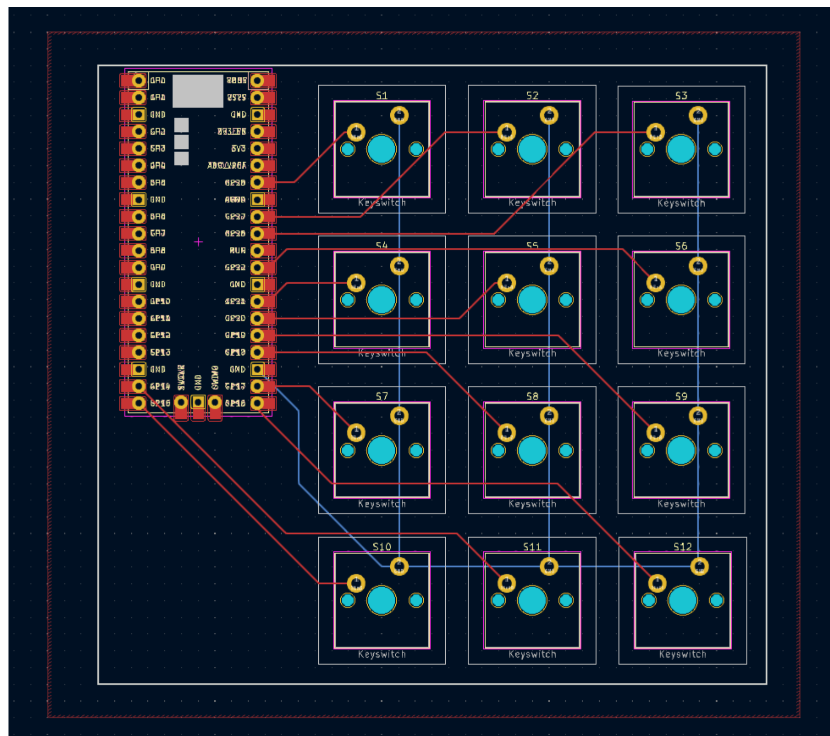

Figura 4: Schema del PCB

1.3 Lato software

Per realizzare il firmware della tastiera ho utilizzato i linguaggi CircuitPython (versione di Python ottimizzata per i microcontrollori) e C, in base al protocollo da implementare. Le funzionalità che ho pensato di associare ai tasti macro comprendono:

- Comandi multimedia
- Apertura rapida di programmi
- Scorciatoie da tastiera per comandi frequenti
- Scrittura di caratteri non disponibili per il layout di tastiera italiano

Dal punto di vista dell'effettivo funzionamento, gli approcci possibili per la programmazione di una tastiera macro sono due: fare eseguire i comandi direttamente al dispositivo (come, ad esempio, scrivere una serie di caratteri predefinita), oppure comunicare al computer semplicemente la pressione di un tasto e lasciare a lui la gestione delle operazioni ad esso associate. Il metodo che ho deciso di adottare è il secondo, in quanto si basa su un funzionamento completamente uguale a quello di una normale tastiera, e permette quindi di fare una analisi fedele ai protocolli reali.

Per farlo, ho deciso di sfruttare l'esistenza di codici che comunicano la pressione dei tasti funzione da F13 a F24, non presenti sulle normali tastiere e che non hanno alcuna funzionalità associata nei moderni sistemi operativi. Sarà tuttavia necessario eseguire nel computer un programma che risponda con l'operazione corretta quando rileva l'uso di questi tasti.

Il codice sarà presentato e analizzato più in dettaglio nei capitoli successivi.

Capitolo 2

Il protocollo seriale PS/2

La comunicazione seriale è un processo per la trasmissione di dati che prevede l'invio sequenziale di bit attraverso un mezzo di comunicazione, oggetto di molte applicazioni nell'ambito della trasmissione dati e delle telecomunicazioni, tra cui anche il collegamento di periferiche ad un computer. L'idea di fondo è quella di trasmettere i bit in serie, ovvero in sequenza uno dopo l'altro, così da leggerli al ricevitore nello stesso ordine. Prima di entrare nei dettagli del funzionamento, forniamo alcuni cenni storici sull'evoluzione delle prime tastiere.

2.1 Storia delle prime tastiere

Ancora prima dell'arrivo dei computer e delle loro necessità, dispositivi per la scrittura di caratteri alfanumerici erano già protagonisti di una grande diffusione. Dapprima con la creazione delle semplici macchine da scrivere meccaniche, poi i terminali per le telescriventi fino ad arrivare alle primitive tastiere che accompagnavano i primi elaboratori, ancora basati su schede forate. Sono questi gli strumenti che hanno gettato le basi per la progettazione delle moderne tastiere, le quali ereditano il layout fisico e le funzionalità presenti.

Le prime tastiere pensate per computer, non più basate su funzionamento meccanico ma elettriche, fanno la loro comparsa agli inizi degli anni 1980. La comunicazione con il computer avveniva tramite connettore DIN, capace di fornire alimentazione e di supportare la comunicazione seriale. Questi connettori sono stati impiegati per svariati scopi nel corso del XX secolo, e trovavano già prima applicazioni nella trasmissione analogica di audio, video e molto altro.

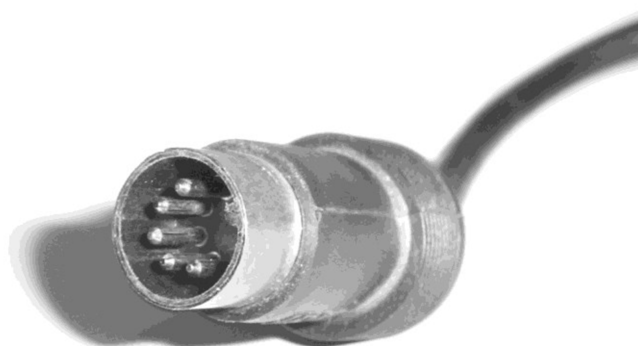

Figura 5: connettore DIN 5-pin a 180°

La prima grande svolta nel collegamento delle tastiere avviene nel 1987, quando IBM rilascia la sua seconda generazione di personal computers (IMB PS/2), con annessa una grande novità. Tali dispositivi presentavano infatti un nuovo ingresso per mouse e tastiera, dato da un connettore mini-DIN a 6 pin, progettato *ad hoc* per la connessione delle periferiche. Questa idea ebbe un grande un successo, e cominciò ad acquisire sempre più popolarità, fino a diventare di fatto lo standard utilizzato a livello globale per l'interfacciamento di tastiere ai pc.

Figura 6: Connettore PS/2 per tastiera

Il passo successivo dell'evoluzione arriva all'inizio del XXI secolo, quando Intel e Microsoft rilasciano la specifica *PC System Design Guide 2001*, che prevede il supporto di collegamento di periferiche tramite USB da parte di ogni computer, tramite protocollo HID. Le porte PS/2 sono ora ufficialmente considerate interfacce legacy, ma hanno continuato ad essere saltuariamente supportate nelle schede madri per molti anni.

2.2 Il protocollo PS/2

Il protocollo utilizzato dalle tastiere con connettore DIN a 5 pin e da quelle con il mini-DIN a 6 pin presenta differenze minime, e dal punto di vista elettrico i due connettori risultano identici. Per comodità, in questo capitolo verrà analizzato il protocollo PS/2 per via della sua maggiore diffusione.

2.2.1 Interfaccia elettrica

Ogni connettore implementa un canale seriale bidirezionale e sincrono, quindi gli unici collegamenti necessari sono uno per il passaggio dei dati e uno per stabilire il clock della comunicazione, oltre ai due per l'alimentazione del dispositivo (Vcc e GND). Per questo motivo, nonostante il connettore presenti 6 pin disponibili, solamente 4 di questi sono effettivamente utilizzati.

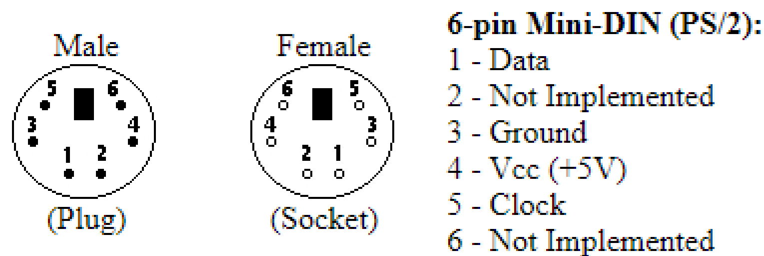

Figura 7: piedinatura del connettore PS/2 [2]

Il microcontrollore interno della tastiera ha 2 pin dedicati per la trasmissione di dati (C - clock, D - data) e 2 per la ricezione (B - clock, A - data). I pin C e D sono collegati alla base di transistor npn in configurazione ad *open collector*, i quali hanno il collettore connesso alle linee per la trasmissione di Clock e Data, e poi collegato a Vcc con una resistenza di pullup (1-10 kΩ); l'emettitore è invece connesso a GND. L'attivazione di uno dei due pin accende il transistor e permette il passaggio di corrente, collegando la linea direttamente a GND e facendo scaricare tutta la tensione sulla resistenza di pullup; quando invece i pin sono disattivati, il transistor resta spento e la linea risulta in corto a Vcc. Questa configurazione permette quindi ai pin di controllare il canale a loro associato, secondo la relazione:

$$Clock = \neg C$$

$$Data = \neg D$$

Di conseguenza, nel caso in cui una linea non venga utilizzata, il suo valore sarà "alto" (5V), mentre quando il pin alla base del transistor è attivo la linea ha valore "basso" (0V).

Per quanto riguarda i pin A e B, questi possono leggere il valore delle linee di Clock e Data in quanto sono direttamente collegati ad esse tramite un amplificatore.

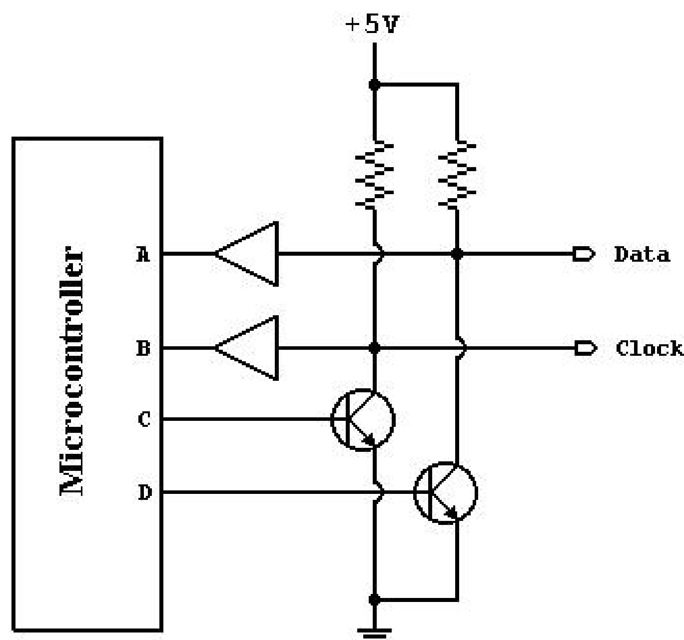

Figura 8: Circuito con open collectors [2]

Una configurazione simile può essere presente all'altro capo delle linee, dal lato computer.

Trattandosi di un circuito di elettronica digitale, da questo momento lo stato "alto" o "attivo" verrà indicato anche con '1', mentre "basso" o "spento" con '0'.

2.2.2 Principi della comunicazione

Come già accennato, la comunicazione è seriale, sincrona e bidirezionale, ed è quindi necessario permettere ad entrambe le parti di comunicare. Il protocollo prevede che il computer abbia la precedenza nelle trasmissioni, ma di default è la tastiera ad inviare i dati. Essa può farlo durante l'*idle state*, ovvero quando entrambe le linee di Data e Clock sono a livello "alto" perché non sono utilizzate da nessuno dei dispositivi (pin C e D = '0' da entrambe le parti). Il computer può prendere il controllo della comunicazione in ogni momento abbassando il livello del bus di Clock a '0' (operazione che inibisce ogni eventuale trasmissione della tastiera), e successivamente impostando Clock a '1' e Data a '0'. Questa operazione comunica alla tastiera l'arrivo di dati da parte dell'host. È importante notare che il clock è sempre e comunque generato dalla tastiera, e il computer può solo alzare o abbassare il livello del canale di Clock come un interrupt. La frequenza di clock per la sincronizzazione rientra tra i 10-16.7 kHz.

2.2.3 Comunicazione tastiera-computer

La trasmissione di dati dalla tastiera avviene sempre per iniziativa di quest'ultima e non è mai sollecitata da parte del computer (tranne in caso di errori in ricezione). Quando la tastiera intende iniziare la trasmissione si assicura il via libera controllando che la linea di clock sia a '1' per almeno 50µs; se così non è, significa che la comunicazione è stata inibita dal computer.

I dati sono inviati un byte alla volta, racchiusi in frames da 11 bits, secondo il seguente schema:

- 1 bit iniziale (*start bit*), sempre pari a 0
- 8 bit di dati (*data bits*), inviati dal meno al più significativo
- 1 bit di parità dispari (*parity bit*), impostato a 1 se il numero di '1' nei dati è pari, e impostato a 0 se il numero di '1' è dispari
- 1 bit finale (*stop bit*), sempre pari a 1

L'invio dei singoli bit viene scandito dal clock, e avviene ovviamente sulla linea dei dati; quando la linea di clock è a '1' la tastiera scrive il bit, e questo viene letto dal computer quando il clock è a '0'. Considerando che la frequenza di clock è tra i 10 e i 16.7 kHz, un periodo di clock dura tra i 60 e i 100µs; la transizione di stato della linea di Data può avvenire almeno 5µs dopo l'inizio del clock alto, e almeno 5µs prima della sua fine. Se la comunicazione viene inibita dal computer mentre i bit vengono inviati, la trasmissione è interrotta e la tastiera si prepara alla ri-trasmissione. Eventuali dati generati in questo periodo vengono salvati in un buffer temporaneo.

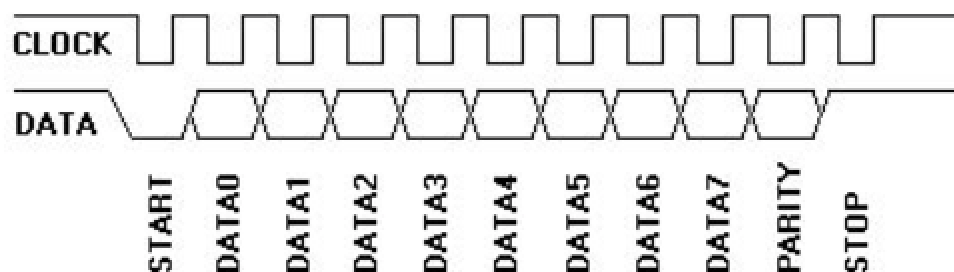

Figura 9: valori assunti da Clock e Data nella trasmissione di un frame [2]

I messaggi che vengono inviati si chiamano *scan codes*, e possono essere di due tipi: *make codes*, utilizzati quando un tasto viene premuto o mantenuto premuto, e *break codes*, che ne

indicano invece il rilascio. Ogni tasto è associato a codici make e break unici, in modo tale che l'invio di uno scan code generico permetta di stabilire con certezza che cosa è successo a quale tasto. È opportuno sottolineare come gli scan codes siano semplicemente associati ai tasti, e non indichino il carattere da stampare; non esiste una correlazione esplicita tra un codice scan e un codice ASCII, quindi è dovere del computer gestire l'informazione e decidere che carattere scrivere o quale azione eseguire. Tipicamente i make codes hanno lunghezza di un solo byte, ma per tasti meno utilizzati possono raggiungere anche la lunghezza di due o quattro bytes. Per quanto riguarda i break codes invece, questi sono uguali al make code del tasto a cui sono associati, con la differenza che possiedono un byte aggiuntivo all'inizio, impostato a 11110000 (0xF0).

Quando un tasto viene premuto e non immediatamente rilasciato, si dice che questo entri nello stato di *typematic* (automatic typing), il che significa che la tastiera continuerà ad inviare il make code di questo fino al rilascio del tasto o alla pressione di un altro. È consentito essere in stato di typematic ad un solo tasto alla volta. Ci sono due importanti parametri che definiscono lo stato di scrittura automatica:

- la latenza (*typematic delay*), ovvero l'intervallo di tempo che intercorre tra la prima pressione del tasto e il passaggio in stato di typematic, compresa tra gli 0.25 e gli 1.00 secondi e impostata a 0.5 s di default;
- la frequenza (*typematic rate*), ovvero la velocità con cui vengono inviati i make codes del tasto, compresa tra i 2.0 e i 30.0 caratteri per secondo e valore di base 10.9 cps.

Oltre agli scan codes associati ai tasti, ne esistono anche altri per la configurazione, la modifica di impostazioni, o la comunicazione di errori. Tra i più rilevanti si trovano:

- ACK (0xFA), utilizzato come risposta positiva nella comunicazione tra i dispositivi;
- ERROR (0xFC), utilizzato per comunicare l'evento di un errore;
- BAT SUCCESSFUL (0xAA), utilizzato per segnalare che la configurazione iniziale della tastiera (Basic Assurance Test) ha avuto successo.

2.2.4 Comunicazione computer-tastiera

Come già spiegato in precedenza, è la tastiera il protagonista principale della trasmissione di dati nel protocollo PS/2, il che è comprensibile visto che è un dispositivo progettato letteralmente per potersi interfacciare con un computer inviando dei simboli. Tuttavia, il

computer può assumere il controllo del canale di comunicazione in ogni momento, attraverso la seguente procedura:

- inibire la trasmissione da parte della tastiera impostando il clock a ‘0’;
- comunicare l’imminente invio di dati rilasciando la linea di clock e impostando la linea di dati a ‘0’ (questo passo è chiamato *request-to-send*);
- aspettare l’inizio del clock da parte della tastiera e procedere all’invio

La trasmissione di dati da parte del computer consiste in pacchetti da un singolo byte, incapsulati in frames con struttura del tutto uguale a quelli già visti. La differenza principale consiste nel fatto che in questo caso la linea Data viene letta dalla tastiera quando il clock è a ‘1’, e il computer ha la possibilità di cambiarne lo stato quando il clock è a ‘0’ (il contrario rispetto a quanto accadeva nella trasmissione nell’altro senso). Inoltre, al termine dell’invio del frame, la tastiera scandisce due cicli di clock aggiuntivi, nel secondo dei quali può abbassare la linea Data per comunicare la corretta ricezione dei dati (*ACK*).

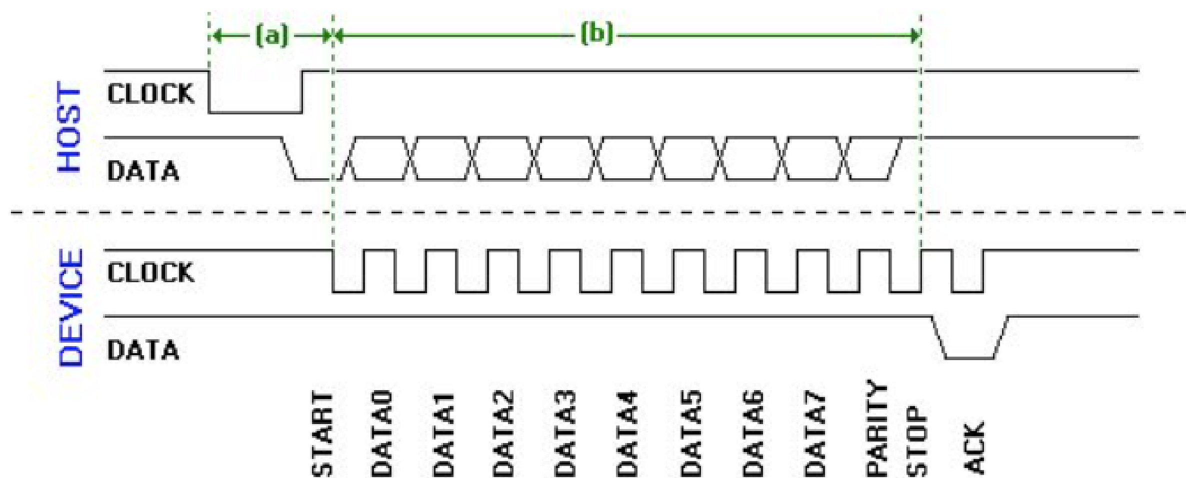

Figura 10: trasmissione di un frame da computer a tastiera [2]
durante (a) viene inibita la trasmissione e inviato il request-to-send
durante (b) avviene l’effettiva trasmissione del frame

Il computer può decidere di comunicare con la tastiera per svariati motivi, come la necessità di modificarne il comportamento temporaneamente, comunicare errori o cambiare impostazioni. Tra i comandi più significativi ci sono:

- RESET (0xFF): imposta la tastiera a tutti i valori di default ed effettua il BAT;
- RESEND (0xFE): indica un errore in ricezione e chiede il re-invio dell’ultimo byte;

- DISABLE (0xF5): disabilita gli input da tastiera fino a nuovo ordine;
- ENABLE (0xF4): riabilita gli input da tastiera;
- SET/RESET LEDS (0xED): modifica lo stato degli indicatori led della tastiera.

2.3 Implementazione del protocollo

2.3.1 Interfaccia elettrica

Il primo passo fondamentale è stato quello di fornire al controllore un modo per connettersi al computer tramite la porta PS/2 dedicata. Per fare questo mi sono procurato un cavo con connettore mini-DIN a 6 pin, e lo ho collegato al controllore rispettando il circuito presentato in figura, ispirato a quello già visto nella *figura 8*. I pin etichettati come *KEY* gestiscono i tasti e sono collegati come presentato nel circuito della *figura 3*.

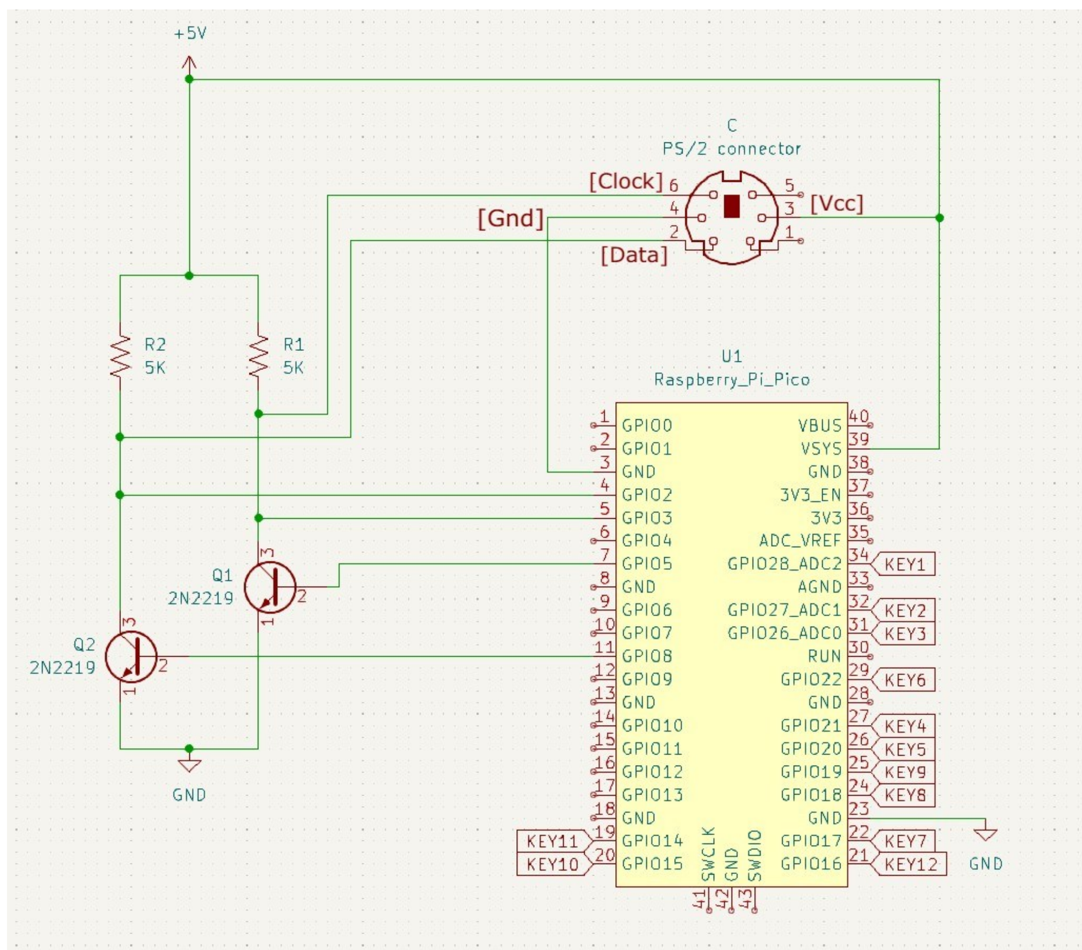

Figura 11: schema del circuito per il collegamento del connettore PS/2

2.3.2 Codice

Per la realizzazione del firmware ho deciso di optare per il linguaggio di programmazione C, perché la SDK (*software development kit*, l'insieme delle librerie, strumenti e documentazioni disponibili per un linguaggio) di C per Raspberry Pi Pico è completa e molto flessibile; C è inoltre molto veloce e offre la possibilità di eseguire operazioni a livello molto basso, come la manipolazione di singoli bit. Questo lo rende molto adatto allo scopo, in quanto garantisce la sincronizzazione tra i dispositivi e rende possibile la trasmissione e ricezione dei dati un bit alla volta.

Nel codice non sarà presente la logica completa necessaria per la gestione della ripetizione dei caratteri (stato typematic), perché per questa precisa applicazione non è particolarmente utile ripetere all'infinito una funzionalità macro. Sono comunque aggiunte alcune variabili utili per lo scopo, ed è sufficiente aggiungere la logica necessaria nel ciclo principale.

Nella prima parte del programma vengono importate le librerie necessarie e definiti i codici, i pin utilizzati e alcuni parametri della tastiera. Come spiegato in precedenza, i codici hanno una lunghezza di 8 bit e sono incapsulati in frames con un bit di start, uno di stop e uno di parità dispari. Per comodità i codici sono salvati con il bit di stop e la parità già calcolata, in modo da ridurre al minimo l'overhead al momento della trasmissione ed evitare di comporre il frame ogni volta. La frequenza di clock impostata nel codice è di circa 12.5 kHz, con periodo 80µs. I valori numerici per gli intervalli di tempo sono diminuiti di 2µs per tenere conto del tempo necessario all'esecuzione della funzione che permette di mettere in pausa il programma.

```c
#include <stdint.h>
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/gpio.h"

//codici di risposta
#define EXT 0x02E0  // extension
#define REL 0x03F0  // release
#define ACK 0x03FA  // acknowledge
#define ERR 0x03FC  // error
//costanti di tempo
#define BREAK 336   // periodo tra le ritrasmissioni
#define CLOCK_CYCLE_US 38 //metà del periodo di clock in us
#define HALF_CLOCK 18  //metà di CLOCK_CYCLE_US
#define SHORT_SLEEP 48
```

```
//scan codes dei tasti F12-F24
#define F13 0x0208
#define F14 0x0310
#define F15 0x0318
#define F16 0x0220
#define F17 0x0328
#define F18 0x0330
#define F19 0x0238
#define F20 0x0240
#define F21 0x0348
#define F22 0x0350
#define F23 0x0257
#define F24 0x035F

//definizione dei pin
uint CLOCK_PIN = 6;   //pin che controlla la linea di clock
uint DATA_PIN = 8;    //pin che controlla la linea dei dati
uint CLOCK_READER = 3; //pin che legge la linea di clock
uint DATA_READER = 2; //pin che legge la linea dei dati
uint LED = 25;        //pin per il led integrato (per caps lock)

//array dei pin associati ai tasti
uint gpios[12] = {
    28, 27, 26,
    21, 20, 22,
    17, 18, 19,
    15, 14, 16
};

//array degli scan codes associati ad ogni tasto
const uint32_t KEY_SCAN_CODES[12] = {
    F13, F14, F15,
    F16, F17, F18,
    F19, F20, F21,
    F22, F23, F24
};

//definizione delle variabili che gestiscono il comportamento della tastiera
static unsigned char LAST_BYTE = 0x00;   // salva l'ultimo byte per la
                                            ritrasmissione
static unsigned char ENABLE = 1;         // abilita/disabilita l'invio di
                                            scan codes
static unsigned char REPEAT_RATE = 50;   // rate di trasmissione in
                                            typematic mode
static unsigned char REPEAT_DELAY = 100; // ritardo prima di entrare in
                                            modalità typematic
static unsigned char ELAPSED_TIME = 0;   // tempo passato da quando un tasto
                                            è stato premuto
```

FUNZIONE TRANSMIT( )

Transmit() è una funzione fondamentale, in quanto è quella che si occupa fisicamente dell'invio dei dati comandando le linee di clock e data attraverso i pin designati, seguendo lo schema di comunicazione presentato nella parte teorica. I bit sono estratti dal keycode uno alla volta grazie ad operazioni di shift e AND sui bit (*bitwise AND*), e vengono inviati in formato little endian, ovvero dal meno significativo al più significativo. Uno shift iniziale a sinistra aggiunge uno 0 come bit meno significativo, assicurando quindi la presenza dello start bit.

```
//funzione che gestisce l'invio di messaggi
void transmit(unsigned int keycode){

    LAST_BYTE = keycode;
    // prepara il bit di start nel messaggio da inviare (zero)
    // il left shift aggiunge uno 0 nella posizione meno significativa
    keycode <<= 1;

    unsigned char index;
    //invia il frame un bit alla volta in formato little endian (dal meno
      al più significativo)
    for(index = 0; index < 11; index++) {
        //una iterazione dura un intero periodo di clock
        gpio_put(DATA_PIN, !((keycode>>index) & 0x0001)); //invia il bit
                                                             meno significativo
        sleep_us(HALF_CLOCK);     //aspetta metà del periodo di clock
        gpio_put(CLOCK_PIN, 1);          //abbassa la linea di clock
        sleep_us(CLOCK_CYCLE_US);       //attendi metà ciclo di clock

        gpio_put(CLOCK_PIN, 0);         //alza la linea di clock
        sleep_us(HALF_CLOCK);     //aspetta metà del periodo di clock

    }

    //i pin di data e clock dovrebbero già essere in questo stato
    //questa operazione assicura la corretta gestione di eventuali errori
    gpio_put(CLOCK_PIN, 0);
    gpio_put(DATA_PIN, 0);

}
```

FUNZIONE RECEIVE( )

Receive() è la funzione che gestisce le linee di clock e data in fase di ricezione dei dati, e permette di leggere correttamente un bit alla volta grazie ad operazioni di *bitwise OR* e shift. Prima di iniziare a ricevere si assicura che il clock sia alto e la linea dati bassa, e al termine dell'operazione invia il bit di ACK per comunicarne l'esito positivo.

```
//funzione per ricevere dei comandi dal computer
int receive(void){

    int buffer = 0;
    //aspetta lo stato di clock alto e data basso
    //la tastiera legge i dati quando il clock è alto
    //quindi questo ciclo fa aspettare la ricezione dello start bit
    while(!gpio_get(CLOCK_READER) || gpio_get(DATA_READER));

    //cicla per ricevere il frame dall'host bit per bit (8 data bits e 1
       parity bit + stop bit) in formato little endian
    unsigned int index;
    for(index = 0; index < 10; index++) {
        gpio_put(CLOCK_PIN, 1);     //abbassa il clock
        sleep_us(CLOCK_CYCLE_US);   //attendi mezzo periodo

        gpio_put(CLOCK_PIN, 0);    //alza il clock
        sleep_us(HALF_CLOCK);      //attendi prima di leggere la linea data
        buffer |= ((int) gpio_get(DATA_READER) << index);   //legge il bit
        sleep_us(HALF_CLOCK);       //completa il mezzo periodo di clock
    }
    //aspetta un intero clock basso
    gpio_put(CLOCK_PIN, 1);
    sleep_us(CLOCK_CYCLE_US);

    //invia un bit di ACK (0) all'host
    gpio_put(CLOCK_PIN, 0);     //alza il clock
    sleep_us(HALF_CLOCK);
    gpio_put(DATA_PIN, 1);      //trasmette il bit di ACK
    sleep_us(HALF_CLOCK);

    gpio_put(CLOCK_PIN, 1);     //abbassa il clock
    sleep_us(CLOCK_CYCLE_US);
    gpio_put(CLOCK_PIN, 0);     //rilascia il clock
    sleep_us(5);
    gpio_put(DATA_PIN, 0);      //rilascia la linea dei dati

    return buffer;
}
```

FUNZIONI SENDCODE( ) e FOLLOWCOMMAND( )

Sono due funzioni ausiliare per l'invio e la lettura dei dati. In particolare, *sendCode()* gestisce la trasmissione dei codici, invocando *transmit()* su frame singoli e aggiungendo l'eventuale codice che indica il rilascio del tasto; *followCommand()*, invece, ha lo scopo di interpretare i codici ricevuti dal computer con *receive()* e agire di conseguenza.

```
//funzione per preparare il codice corretto da inviare
//in base a pressione (keyState = 1) o rilascio (keyState = 0)
void sendCode(uint32_t keycode, char keyState){

    //keyState è 1 ==> tasto premuto
    if(keyState){
        // controlla se il codice è un codice esteso
        if((keycode & 0xff0000) == 0xe00000){
            //se è esteso, invia prima l'estensione, poi il codice
            transmit(EXT);
            sleep_us(BREAK); //pausa tra le due trasmissioni
            transmit(keycode);
            sleep_us(BREAK);
        }else{
            //se il codice non è esteso invialo subito
            transmit(keycode);
            sleep_us(BREAK);
        }
    }
    else{ //keyState è 0 ==> tasto rilasciato
        // controlla se il codice è un codice esteso
        if( (keycode & 0xff0000) == 0xe00000 ){
            // trasmette estensione, codice di rilascio e infine codice del
tasto
            transmit(EXT);
            sleep_us(BREAK);
            transmit(REL);
            sleep_us(BREAK);
            transmit(keycode);
            sleep_us(BREAK);
        }else{
            // trasmette codice di rilascio e poi codice del tasto
            transmit(REL);
            sleep_us(BREAK);
            transmit(keycode);
            sleep_us(BREAK);
        }
    }//end_if_else_keyState
}
```

```
//funzione per interpretare un comando ricevuto ed eventualmente inviare
  una risposta
void followCommand(unsigned int command){
    command &= 0xff;
    unsigned int arg;   //variabile per eventuali dati aggiuntivi
    switch(command){
        case 0xed: // set LEDs
            transmit(ACK);
            arg = receive();
            transmit(ACK);
            //legge eventuali indicazioni riguardo il led caps lock
            if( (arg & 0x04) ){
                gpio_put(LED, 1); // accendi CapsLock LED
            }else{
                gpio_put(LED, 0); // spegni CapsLock LED
            }
            break;
        case 0xee: // echo
            transmit(0x03ee);  // risponde all'host con lo stesso messaggio
            break;
        case 0xf0: // scan code set
            transmit(ACK);
            arg = receive();    // leggi lo scan set da impostare
            transmit(ACK);
            // se arg è 0 rispondi con lo scan set corrente (0x02)
            if( !(arg & 0xff) )
                transmit(0x0341);
            break;
        case 0xf2: // read ID
           //rispondi con un deviceID pari a 0xab83 in due frames separati
            transmit(ACK);      // acknowledge
            transmit(0x02ab);
            sleep_us(BREAK);    //breve ritardo per assicurare la ricezione
            transmit(0x0283);
            break;
        case 0xf3:  //imposta typematic delay e frequenza di ripetizione
            transmit(ACK);
            arg = receive();
            transmit(ACK);
           //**modifica le impostazioni di REPEAT_RATE e REPEAT_DELAY in
           //base al contenuto di arg; codice omesso perché molto lungo e
           //non aggiunge informazioni utili**
            break;
        case 0xf4: // enable (abilita l'invio di tasti)
            transmit(ACK);
            ENABLE = 1;
            break;
```

```
        case 0xf5: // disable (disabilita l'invio di tasti)
            transmit(ACK);
            ENABLE = 0;
            break;
        case 0xfe: // resend last byte
            transmit(ACK);
            transmit(LAST_BYTE);
            break;
        case 0xff: // reset
            transmit(ACK);
            sleep_us(BREAK);    // piccolo delay per simulare un reset
            transmit(0x03aa);   // invia BAT successful
            break;

    }//end_switch
}
```

FUNZIONE MAIN( )

La prima parte della funzione principale del programma si occupa di inizializzare i pin del controllore definiti in precedenza attraverso la chiamata di appositi metodi.

```
// main routine
void main(void){

    //inizializza i pin digitali per lettura/scrittura di data e clock
    gpio_init(CLOCK_PIN);
    gpio_set_dir(CLOCK_PIN, GPIO_OUT);
    gpio_init(DATA_PIN);
    gpio_set_dir(DATA_PIN, GPIO_OUT);
    gpio_init(CLOCK_READER);
    gpio_set_dir(CLOCK_READER, GPIO_IN);
    gpio_pull_down(CLOCK_READER);
    gpio_init(DATA_READER);
    gpio_set_dir(DATA_READER, GPIO_IN);
    gpio_pull_down(DATA_READER);

    //inizializza i pin digitali per i tasti
    for(int i = 0; i < 12; i++) {
        gpio_init(gpios[i]);
        gpio_set_dir(gpios[i], GPIO_IN);
        gpio_pull_up(gpios[i]);
    }
```

Una volta impostati correttamente tutti i pin e definite alcune variabili, il programma può entrare nel ciclo principale, il quale viene eseguito per tutto il tempo in cui la tastiera è accesa. Ad ogni iterazione del ciclo, il programma controlla lo stato delle linee di clock e data e agisce di conseguenza:

- Se il CLOCK è tenuto basso dal computer significa che la trasmissione è inibita e l'unica cosa da fare è quindi aspettare.
- Se il CLOCK è alto ma la linea di DATA è bassa allora il computer è pronto a trasmettere, quindi la tastiera chiama la funzione *receive()* per ricevere il messaggio.
- Se entrambe le linee sono alte allora è possibile controllare se ogni tasto è premuto/rilasciato in quel momento ed eventualmente trasmettere il suo scan code.

Il ciclo principale risulta piuttosto breve ed intuitivo grazie alla definizione delle funzioni *sendCode()* e *followCommand()* che si fanno carico dell'elaborazione dei dati in ricezione e trasmissione, riducendo il lavoro del main ad una semplice chiamata a funzione

```
//array che tiene traccia dei tasti che sono premuti
  (tasto premuto --> valore corrispondente array = 1)
//al posto di 1 si può assegnare la variabile ELAPSED per tenere conto del
  delay per la modalità typematic
    unsigned char keyStamps[12];
    int i = 0, buffer = 0;

    //ciclo principale
    while(1){
    start:
        //controlla se l'host sta inibendo la comunicazione (clock basso)
        if(!gpio_get(CLOCK_READER)){
            sleep_us(SHORT_SLEEP);  //se è così, aspetta

        //controlla se l'host sta chiedendo di trasmettere (clock alto e
data basso)
        }else if(gpio_get(CLOCK_READER) && !gpio_get(DATA_READER)){

            buffer = receive();     //ricevi il messaggio in arrivo
            followCommand(buffer);  //elabora il messaggio
```

```
        //se l'host consente la trasmissione e l'invio di scan codes è
          abilitato, controlla se sono premuti dei tasti
        }else if(ENABLE){

            //cicla su tutti i pin dei tasti per vedere se sono premuti
            for(i = 0; i < 12; i++){
                //prima di ogni scan controlla se l'host ha abbassato il
                  clock per inibire la trasmissione
                if(!gpio_get(CLOCK_READER))
                    goto start;

                //se il pulsante è premuto (il circuito che connette il pin
                  a terra è chiuso quindi il valore del pin è 0)
                //determina il ritardo e poi trasmette il codice
                if(!gpio_get(gpios[i])){

                //se il tasto è appena stato premuto invia subito il codice
                    if(!keyStamps[i]){
                        sendCode(KEY_SCAN_CODES[i], 1);
                        //keyStamps[i] = ELAPSED_TIME;
                        keyStamps[i] = 1;
                    }
               //se il tasto era già premuto determina se il
                 REPEAT_DELAY è stato raggiunto ed entra in typematic mode
               /* qui ci va la eventuale logica necessaria per la gestione
                  dello stato Typematic */
               //
                }
                //se il tasto era premuto in precedenza e ora non lo è più
                  invia il codice di rilascio
                else if(keyStamps[i]){
                    sendCode(KEY_SCAN_CODES[i], 0);
                    keyStamps[i] = 0;
                }
            //}//end keys-for

            sleep_us(SHORT_SLEEP);   //piccola pausa tra un ciclo e l'altro
        }//end if else


    }//end_while
}
```

Capitolo 3

Il protocollo HID

I dispositivi di interfaccia umana (HID - human interface devices) sono una classe di dispositivi pensata per unificare sotto il supporto di drivers generici un grande numero di periferiche, da un lato rimpiazzando i vari connettori personalizzati con una interfaccia comune, e dall'altro fornendo un protocollo di comunicazione standardizzato per tutti.

Nonostante il protocollo HID abbia visto una prima diffusione con supporto su USB, è stato progettato per essere bus-agnostic, ovvero non ha necessità di sapere il mezzo attraverso cui avviene la comunicazione; HID lavora ad "alto livello" e non si occupa della trasmissione fisica dei dati, ma si affida ad un protocollo di supporto che lo faccia al posto suo. HID è stato pensato per dispositivi a bassa latenza e bassa banda, ma con la possibilità di specificare la velocità di comunicazione attraverso il mezzo, qualunque esso sia. I trasporti più utilizzati sono USB per le connessioni cablate, e Bluetooth per quelle wireless.

La documentazione ufficiale della definizione di classi HID v1.11 *[3]* presenta come obiettivi principali del protocollo:

- Essere il più compatto possibile per risparmiare spazio sul dispositivo
- Essere estensibile e robusto
- Essere auto-descrittivo e consentire applicazioni con software generico

La versione 1.11 risale al 2001 ed è l'ultimo aggiornamento rilasciato del protocollo; considerando che è tutt'ora implementato in qualsiasi computer, tablet o telefono, appare piuttosto chiaro come l'approccio sia incredibilmente versatile e ben strutturato.

3.1 Descrittori HID

Come già accennato, il protocollo HID è pensato per fornire una interfaccia comune a dispositivi di tipo diverso. Affinché la comunicazione avvenga senza problemi, e il computer possa interpretare correttamente i dati, è necessario che ogni dispositivo identifichi con precisione le sue funzionalità al momento della connessione. Per poter fare ciò vengono utilizzati dei *descrittori*, ovvero delle strutture dati con un formato definito e un significato ben preciso (approccio già utilizzato, tra l'altro, nel protocollo USB), che esplicitano fin da subito cosa rappresentano i dati inviati e come sono organizzati. Dal punto di vista pratico, un descrittore è semplicemente una serie di valori numerici. I descrittori utilizzati sono tre: *HID descriptor*, *Report descriptor* e *Physical descriptor*

3.1.1 HID descriptor

L'HID descriptor è il primo ad essere inviato, e ha lo scopo di definire il tipo e la lunghezza dei descrittori successivi, oltre alla versione della documentazione che questi rispettano. In aggiunta è presente anche il campo *bCountryCode*, che indica per quale stato è localizzato l'hardware; per la maggior parte dei dispositivi non viene utilizzato (valore impostato a 0x00), ma nel caso delle tastiere è particolarmente utile perché il layout dei tasti cambia in base alla lingua per cui sono progettate. Per esempio, la localizzazione per l'Italia è associata a 0x0E, quindi una tastiera con layout italiano avrà tale valore.

Part	Offset/Size (Bytes)	Description	Sample Value
bLength	0/1	Size of this descriptor in bytes.	0x09
bDescriptorType	1/1	HID descriptor type (assigned by USB).	0x21
bcdHID	2/2	HID Class Specification release number in binary-coded decimal—for example, 2.10 is 0x210).	0x101
bCountryCode	4/1	Hardware target country.	0x00
bNumDescriptors	5/1	Number of HID class descriptors to follow.	0x01
bDescriptorType	6/1	Report descriptor type.	0x22
wDescriptorLength	7/2	Total length of Report descriptor.	0x3F

Figura 12: struttura di un HID descriptor [3]

3.1.2 Report descriptor

Alla base della comunicazione HID ci sono i *report*, ovvero dei pacchetti di dati organizzati che forniscono informazioni sullo stato del dispositivo. I report descriptors sono assolutamente fondamentali, in quanto specificano il formato dei report, e rendono quindi possibile al computer decifrare i dati che riceve. Un report descriptor definisce la struttura sia dei dati che il dispositivo invia (*input report*), sia di quelli che si aspetta di ricevere (*output report*).

Un altro elemento fondamentale per il protocollo sono gli *oggetti* (*items*), delle strutture dati di dimensione ridotta utilizzate per rappresentare una singola informazione riguardante il dispositivo. In particolare, vengono utilizzati per comporre i report descriptors. Sono formati da 0-4 bytes di dati (opzionali, solitamente 1 byte) e un header di 1 byte che definisce completamente l'oggetto, attraverso i 3 campi che lo costituiscono: uno per la lunghezza dei dati (*bSize*), uno per il tipo dell'oggetto (*bType*), e uno per il suo tag, ovvero un valore numerico che indica la funzione dell'oggetto (*bTag*).

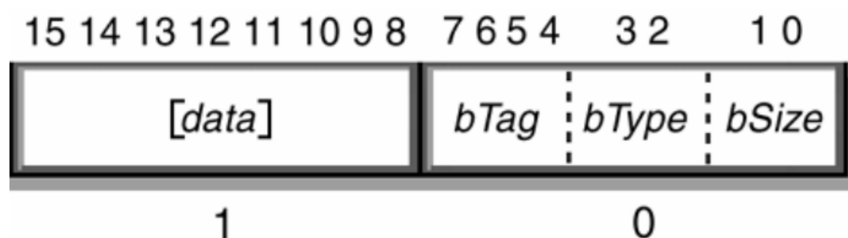

Figura 13: Struttura di un oggetto [3]

I tipi di oggetto sono i seguenti:

- Oggetti principali (*main items*): utilizzati per definire e raggruppare campi di dati in un report descriptor; sono rilevanti perché sono quelli che effettivamente riservano i bit per ogni dato nei report utilizzati per la comunicazione. Hanno valore di bType pari a '00', e in base a bTag possono indicare presenza di dati di input, output o opzioni. I main items permettono anche di raggruppare più oggetti in una collezione.
- Oggetti globali (*global items*): permettono di stabilire dei valori validi per ogni oggetto principale successivo, come ad esempio l'intervallo dei valori che possono assumere.
- Oggetti locali (*local items*): permettono di stabilire dei parametri validi solamente per l'oggetto o per la collezione successiva.

I report descriptor specificano la struttura dei report, ma se il loro lavoro si fermasse a questo il computer saprebbe solo come leggere i dati, e non come interpretarli. Ad esempio, se il report descriptor di un mouse comunica la presenza di un campo con valore tra i -127 e i +127, è opportuno comunicare al computer anche che tale numero indica la coordinata X del puntatore. Un compito del genere è piuttosto semplice a parole, ma è invece molto difficile da realizzare in pochi byte. Per far fronte a tale necessità, sono state stilate delle tabelle che raggruppano tutti i possibili utilizzi dei vari input di ogni tipo di dispositivo, chiamate *usage tables [4]*; i report descriptor contengono riferimenti a queste per esplicitare in modo semplice e poco dispendioso il significato di ogni oggetto di input. Riprendendo l'esempio del mouse, il significato di "coordinata X del mouse" viene comunicato indicando la entry 0x30 (X-value) della tabella 0x01 (Generic Desktop Page).

Generic Desktop Page (0x01)

Usage ID	Usage Name	Usage Types	Section
00	*Undefined*		
01	Pointer	CP	4.1
02	Mouse	CA	4.1
03-03	*Reserved*		
04	Joystick	CA	4.1
05	Gamepad	CA	4.1
06	Keyboard	CA	4.1
07	Keypad	CA	4.1
08	Multi-axis Controller	CA	4.1
09	Tablet PC System Controls	CA	4.1
0A	Water Cooling Device [6]	CA	4.1
0B	Computer Chassis Device [6]	CA	4.1
0C	Wireless Radio Controls [13]	CA	4.1
0D	Portable Device Control [23]	CA	4.1
0E	System Multi-Axis Controller [33]	CA	4.1
0F	Spatial Controller [39]	CA	4.1
10	Assistive Control [49]	CA	4.1
11	Device Dock [57]	CA	4.15
12	Dockable Device [57]	CA	4.15
13	Call State Management Control [73]	CA	4.16
14-2F	*Reserved*		
30	X	DV	4.2
31	Y	DV	4.2
32	Z	DV	4.2

Figura 14: usage table per funzioni generiche di desktop [4]

La struttura di un report descriptor risulta quindi piuttosto semplice: è una serie di oggetti lunghi 2 byte, dei quali il primo è l'header e definisce l'oggetto, mentre il secondo è il valore numerico assegnato. La maggior parte di questi oggetti sono utilizzati per strutturare e descrivere i dati da inviare, ai quali viene poi riservato fisicamente dello spazio nel descrittore con i main items. Talvolta vengono aggiunti anche dei bit di padding con lo scopo di allineare la dimensione del report ad un multiplo intero di byte. I report descriptor saranno analizzati in maniera più approfondita nel capitolo implementativo.

LEGENDA

main item

global item

local item

Item		**Value (Hex)**
Usage Page (Generic Desktop),		05 01
Usage (Keyboard),		09 06
Collection (Application),		A1 01
Usage Page (Key Codes);		05 07
Usage Minimum (224),		19 E0
Usage Maximum (231),		29 E7
Logical Minimum (0),		15 00
Logical Maximum (1),		25 01
Report Size (1),		75 01
Report Count (8),		95 08
Input (Data, Variable, Absolute),	;Modifier byte	81 02
Report Count (1),		95 01
Report Size (8),		75 08
Input (Constant),	;Reserved byte	81 01
Report Count (5),		95 05
Report Size (1),		75 01
Usage Page (Page# for LEDs),		05 08
Usage Minimum (1),		19 01
Usage Maximum (5),		29 05
Output (Data, Variable, Absolute),	;LED report	91 02
Report Count (1),		95 01
Report Size (3),		75 03
Output (Constant),	;LED report padding	91 01
Report Count (6),		95 06
Report Size (8),		75 08
Logical Minimum (0),		15 00
Logical Maximum(101),		25 65
Usage Page (Key Codes),		05 07
Usage Minimum (0),		19 00
Usage Maximum (101),		29 65
Input (Data, Array),	;Key arrays (6 bytes)	81 00
End Collection		C0

Figura 15: esempio di report descriptor per una tastiera [3]

3.1.3 Physical descriptor

I descrittori fisici sono utilizzati per fornire informazioni riguardo la specifica parte del corpo umano che viene utilizzata per attivare i vari controlli di un dispositivo. Tuttavia, questi sono completamente opzionali e vengono usati solo in pochi casi, poiché aggiungono complessità al protocollo ma restituiscono pochi vantaggi per la maggior parte dei dispositivi. Ritengo corretto menzionarne l'esistenza per completezza di informazione, ma non è utile approfondirne il funzionamento in quanto non vedono alcun utilizzo per quanto riguarda le tastiere.

3.2 HID per le tastiere

Come già accennato, il protocollo HID si basa sui report, ovvero dei pacchetti di dati ben definiti che vengono inviati tra i dispositivi per comunicare. Prima di tutto analizziamo quindi come sono strutturati i report di cui fanno uso le tastiere.

3.2.1 Struttura dei report

I report di input di una tastiera hanno una lunghezza totale di 8 byte; questa scelta è direttamente influenzata dai primissimi sviluppi del protocollo HID attraverso Universal Serial Bus, poiché tale dimensione è la massima supportata da USB per un singolo trasferimento in blocco. La loro struttura è la seguente:

- Byte 0: è un bitfield dove ogni bit è associato ad uno dei tasti modificatori (in ordine: Ctrl sinistro, Shift sinistro, Alt sinistro, GUI/Win sinistro, Ctrl destro, Shift destro, Alt destro, GUI/Win destro). Un valore pari a 1 indica che il relativo tasto è premuto, 0 altrimenti.
- Byte 1: riservato per uso futuro, impostato a 0x00.
- Byte 2-7: ogni byte indica un eventuale tasto premuto, riportando il codice ad esso associato nella HID usage table. Siccome sono 6 byte, è possibile comunicare la pressione di fino a 6 tasti (non modificatori) nello stesso momento.

Nella *figura 16* viene riportato come esempio il report che indica la pressione dei tasti Ctrl sinistro (bit 0 del primo byte), Alt sinistro (bit 2 del primo byte), F, S, e barra spaziatrice (con associati rispettivamente i codici 0x07, 0x16, 0x2C dalla HID usage table per le tastiere).

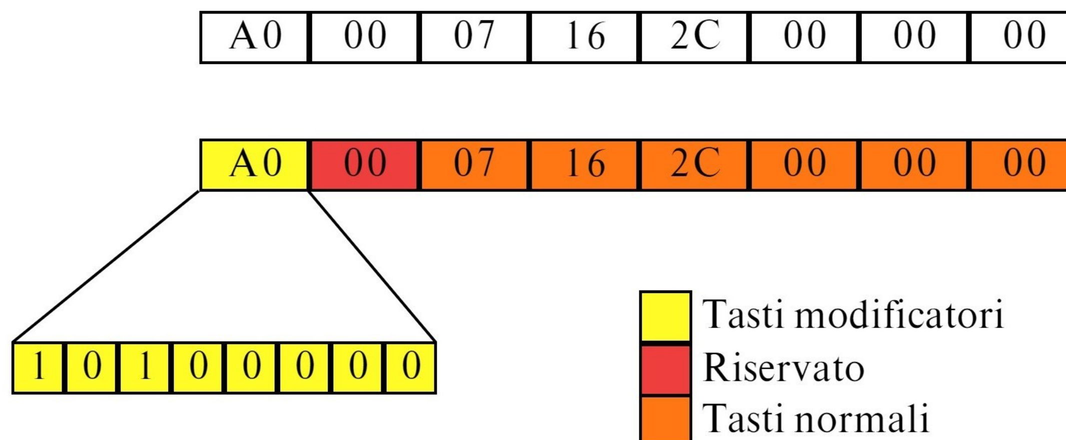

Figura 16: esempio di report

I report di output consistono invece in un singolo byte, in quanto le informazioni che la tastiera può ricevere dal computer sono molto limitate. La struttura è la seguente:

- Bit 0: led NUM LOCK (0 = spento, 1 = acceso; così per tutti i led)
- Bit 1: led CAPS LOCK
- Bit 2: led SCROLL LOCK
- Bit 3: comando '*compose*', una funzionalità che permette ad alcune tastiere di unire alcuni caratteri per formarne uno speciale altrimenti non disponibile, come lettere accentate (ad esempio: ` + e = è).
- Bit 4: comando '*kana*', una funzionalità utilizzata per rendere possibile l'input di caratteri kanji per la lingua giapponese.
- Bit 5-7: riservati per uso futuro, impostati a 0.

Un esempio di come è impostato un report descriptor è dato dalla *figura 12*, e il significato dei vari oggetti verrà spiegato nel capitolo riguardante l'implementazione del protocollo.

3.2.2 Principi della comunicazione

Durante la fase di inizializzazione gli unici dati che la tastiera può inviare sono i descrittori, mentre per il resto del tempo vengono inviati i report. Il computer ha invece la possibilità di interrogare la tastiera e impartire ordini tramite una serie di messaggi, tra cui:

- **GET_REPORT**: è una richiesta che permette al computer di ricevere un report del dispositivo. La definizione delle classi HID v.1.11 riporta quanto segue: "*Questa richiesta è utile a tempo di inizializzazione per gli oggetti assoluti e per determinare lo stato degli oggetti di opzione. Questa richiesta non è destinata ad essere utilizzata per il polling del dispositivo su base regolare*".
- **SET_REPORT:** è un messaggio che permette al computer di inviare un output report alla tastiera.
- **SET_IDLE** e **GET_IDLE**: sono delle richieste che permettono rispettivamente di interrompere la lettura degli input report dal dispositivo e di richiedere da quanto tempo tale direttiva è attiva. Sono solitamente utilizzate per opzioni di risparmio energetico.
- **SET_PROTOCOL** e **GET_PROTOCOL**: per le tastiere esistono due varianti del protocollo HID: il report protocol, utilizzato dai sistemi operativi, e il boot protocol, attivato solo per le comunicazioni con il BIOS. L'unica differenza tra i due risiede nel fatto che il report protocol prevede la possibilità di personalizzare i report descriptor, mentre il boot protocol stabilisce uno standard di base da seguire. Le due richieste permettono rispettivamente di comunicare alla tastiera quale protocollo utilizzare e chiede quale dei due è attualmente in uso.

3.2.3 Comunicazione tastiera-computer

Esistono due modalità principali per effettuare la comunicazione tra i dispositivi, fondamentalmente diverse tra loro. La prima usa della tecnica del *polling*, quindi richiede che il computer mandi periodicamente delle richieste alla tastiera (a intervalli solitamente tra gli 1 e i 50ms) per ricevere aggiornamenti sullo stato di questa, sollecitando l'invio dei report. Siccome la specifica HID proibisce l'utilizzo di richieste GET_REPORT per questi scopi, le interrogazioni avvengono attraverso richieste definite ad un livello inferiore; se per esempio la comunicazione avviene tramite USB, sarà possibile utilizzare una richiesta USB-IN. Il secondo approccio fa invece uso di *interrupt*, e vede quindi la tastiera inviare di propria iniziativa i report quando ha qualcosa da comunicare.

Il metodo del polling è generalmente utilizzato per collegare tastiere che non hanno limitazioni dal punto di vista energetico, perché genera un traffico molto maggiore rispetto a quello realmente necessario; questo permette tuttavia una gestione più intuitiva della comunicazione, perché si sviluppa una semplice serie di richieste e risposte tra i dispositivi. Il

polling è molto usato nelle tastiere collegate tramite USB. L'approccio tramite interrupt è invece preferibile per applicazioni dove è importante ridurre al minimo i consumi, come nel caso di tastiere Bluetooth, spesso e volentieri alimentate da una batteria limitata. Nonostante questo metodo generi un traffico molto ridotto, è necessario per il computer gestire correttamente i report in ingresso (che possono arrivare in qualsiasi momento), e sono necessari alcuni accorgimenti per mantenere la sincronizzazione tra i dispositivi ed evitare la collisione di eventuali dati in direzioni diverse.

Dal punto di vista puramente del protocollo HID, la tastiera prepara un report per l'invio solo quando ha qualcosa da comunicare, e sarà poi lavoro del protocollo di appoggio per la trasmissione dei dati inviarlo al momento opportuno, in base all'approccio utilizzato. Esistono inoltre una serie di ottimizzazioni che è possibile mettere in pratica ai livelli inferiori, esternamente al protocollo HID; per esempio, nel caso in cui venga utilizzato il polling e tra una richiesta e l'altra non è stato registrato alcun cambiamento nella tastiera, il protocollo USB può decidere di rispondere con un messaggio NAK-USB invece che con un intero report HID.

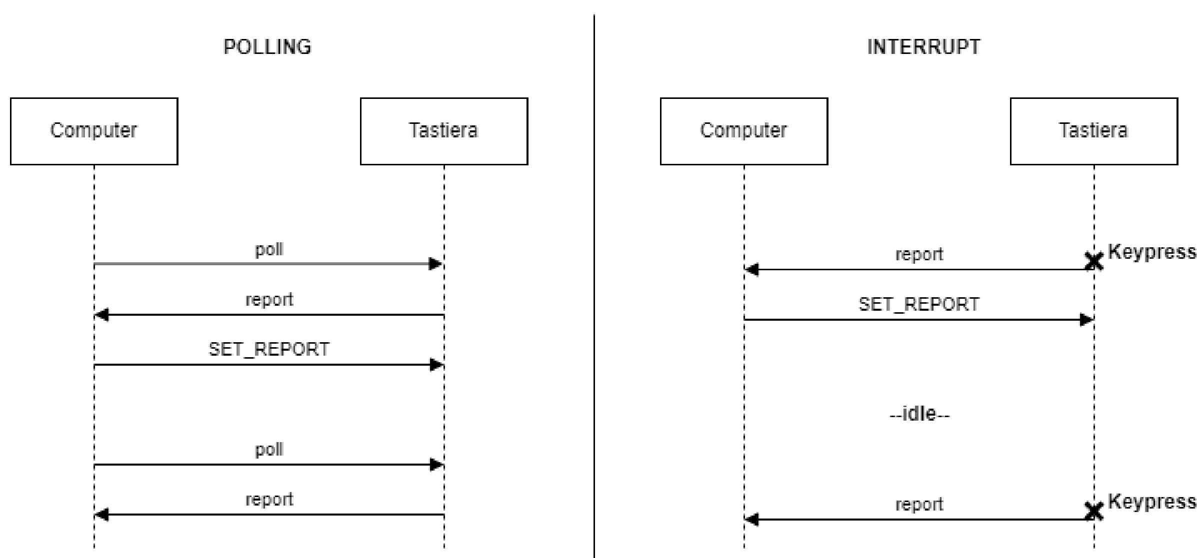

Figura 17: confronto schemi di comunicazione attraverso polling e interrupt

3.2.4 Limitazioni

Come spiegato in precedenza, un report contiene 6 bytes per segnalare i tasti premuti in un dato momento: il numero massimo di tasti che la tastiera può comunicare in contemporanea è quindi sei. L'abilità di una tastiera di gestire correttamente un certo numero di tasti *n* in uno stesso momento è chiamata *n-key rollover*.

Nonostante il limite imposto dai report, è comunque possibile aumentare il numero *n* facendo figurare la singola tastiera come molteplici dispositivi differenti: in questo modo può inviare più report sotto diversi alias (differenziandoli con l'aggiunta di un campo '*Report ID*'), e comunicare più di sei tasti senza problemi. Questa tecnica è molto utilizzata e, infatti, se si controllano i dispositivi collegati al proprio computer è probabile notare la presenza di (solitamente) 4 tastiere HID, nonostante ne sia collegata una sola.

Nel caso in cui i tasti premuti siano più di quelli consentiti, il report assumerà un formato di errore in cui tutti i bytes destinati ai tasti sono impostati a '0x01', valore riservato per indicare proprio lo stato non valido della tastiera. In questa situazione, il primo byte del report continuerà comunque a riportare in modo corretto lo stato dei tasti modificatori, in quanto ognuno ha un bit dedicato.

Figura 18: esempio di report in stato di errore

Il rilascio di un tasto è comunicato semplicemente dalla scomparsa del suo valore corrispondente nel report. Per questo motivo, non è possibile per un tasto entrare nello stato di *typematic*, almeno dal punto di vista hardware. Per fare in modo che la pressione ininterrotta di un tasto provochi la scrittura continua del carattere a schermo, è necessario intervenire a livello software, dal lato del computer.

3.2.4 Comunicazione computer-tastiera

Il computer può inviare in ogni momento dei messaggi tramite le richieste presentate in precedenza, con lo scopo di richiedere aggiornamenti, inviare dati, o modificare delle impostazioni. Il formato che seguono queste richieste può variare in base al mezzo di comunicazione usato, ma contengono in ogni caso almeno la direzione del flusso dei dati (computer-tastiera o tastiera-computer; ad esempio, GET_REPORT indica un flusso dei dati tastiera-computer) e il nome della richiesta. Le richieste specifiche possiedono poi ulteriori tratti comuni.

Part	Description
bmRequestType	10100001 (flusso tastiera-computer)
bRequest	GET_REPORT (0x01)
wValue	Report Type and Report ID
wIndex	Interface
wLength	Report Length
Data	Report

Figura 19: richiesta GET_REPORT secondo lo standard USB

Part	Description
MessageType	GET_REPORT (0x4)
Size	spazio riservato nel buffer
Reserved Bit	0
ReportType	1=input, 2=output, 3=feature
ReportID	report ID (se presente)
BufferSize	dimensione del buffer (se specificato da size)

Figura 20: richiesta GET_REPORT secondo lo standard Bluetooth

Le *figure 19* e *20* riportano il formato della richiesta GET_REPORT per USB e Bluetooth. È possibile notare, nonostante le differenze, la presenza in entrambi del tipo di richiesta, il tipo del report, il Report ID e un parametro riguardo la dimensione dei dati richiesti.

3.3 Implementazione del protocollo

Come spiegato in precedenza, il protocollo HID lavora ad un livello di astrazione che non prevede il controllo diretto del mezzo di trasmissione, e ha bisogno di essere supportato da un protocollo a sé stante per la comunicazione fisica tra i due dispositivi. Per questa implementazione ho deciso di collegare la tastiera al computer tramite USB, in quanto rappresenta l'approccio più utilizzato nelle applicazioni reali per le tastiere cablate, e il collegamento è reso molto semplice dalla possibilità del Raspberry Pi Pico di connettersi ad un host tramite la porta micro-usb integrata.

Per la realizzazione del firmware ho deciso di utilizzare il linguaggio *CircuitPython*, sviluppato da Adafruit [6]. Siccome l'oggetto di questo capitolo è il protocollo HID, ho ritenuto opportuno cercare di concentrare solo su di esso lo sviluppo del firmware, tralasciando quanto riguarda le impostazioni e le operazioni del protocollo USB. Per fare ciò ho deciso di servirvi della libreria *usb_hid* per CircuitPython [7], che si occupa interamente dell'interfacciamento con il protocollo USB, e al tempo stesso permette di configurare un dispositivo HID in maniera personalizzata e gestire liberamente la lettura e l'invio dei report, a costo di alcune limitazioni. Tra queste, in particolare, c'è il fatto che l'HID descriptor è generato in automatico e la gestione delle richieste HID in ingresso (GET_REPORT, SET_IDLE, ecc…) è a carico della libreria.

Fino ad ora è stata particolarmente sottolineata l'importanza dei report nell'ambito della comunicazione. Il primo passo per l'implementazione è quindi definire il report descriptor che verrà inviato al computer in fase di setup del collegamento iniziale.

3.3.1 Oggetti

Come già spiegato, un report descriptor è una collezione di oggetti, che si dividono in *main, global* e *local items*. Il significato degli oggetti è dato dal loro tag, e la specifica HID 1.11 definisce tutti i tipi di oggetto che è possibile utilizzare, insieme al valore corrispondente. Le tabelle che presentano questi dati sono molto approfondite e piuttosto lunghe; vengono quindi qui riportate delle versioni semplificate che contengono solo gli oggetti effettivamente utili per questa applicazione. Il valore *nn* indica la lunghezza in byte della sezione di dati dell'oggetto; a meno di casi eccezionali, questo valore è 01.

	1-byte prefix			
Item name	**bTag**	**bType**	**bLength**	**value**
Input	1000	00	*nn*	bit 0 = Data(0) | Costant(1) bit 1 = Array(0) | Variable(1) bit 2 = Absolute(0) | Relative(1)
Output	1001	00	*nn*	bit 0 = Data(0) | Costant(1) bit 1 = Array(0) | Variable(1) bit 2 = Absolute(0) | Relative(1)
Collection	1010	00	*nn*	(0x01) = Application
End Collection	1100	00	*nn*	No value

Figura 21: Main Items table

Item name	1-byte prefix bTag	bType	bLength	value
Usage Page	0000	01	*nn*	Value of HID usage page
Logical Minimum	0001	01	*nn*	Minimum value that the variable/array will report
Logical Maximum	0010	01	*nn*	Maximum value that the variable/array will report
Report Count	1001	01	*nn*	Number of data fields for the item
Report Size	0111	01	*nn*	Size of the report in bits

Figura 22: Global Items table

Item name	1-byte prefix bTag	bType	bLength	value
Usage	0000	10	*nn*	Usage from HID usage tables
Usage Minimum	0001	10	*nn*	Mininum value of usageID from HID usage tables
Usage Maximum	0010	10	*nn*	Maximum value of usageID from HID usage tables

Figura 23: Local Items table

Esempio di utilizzo: per creare un oggetto *Logical Maximum* con valore di 0x04, si procede nel seguente modo:

- I primi 4 bit sono dati dal valore bTag della tabella (*figura 22*): in questo caso 0010;
- I 2 bit successivi sono dati da bType: siccome è un oggetto globale, sono pari a 01;
- I 2 bit successivi sono dati da bLength, che indica la lunghezza dei dati associati all'oggetto in bytes: in questo caso 0x04 occupa solo un byte, quindi il valore è 01;
- Il valore associato è 0x04.

Il prefisso è quindi la sequenza di bit 0010 01 01 (= 0x25); è possibile aggiungere l'oggetto desiderato al report descriptor attraverso i valori [0x25, 0x04].

3.3.2 Creazione del report descriptor

La prima informazione da fornire nel report descriptor è il tipo di dispositivo HID che si sta configurando. Le tastiere appartengono alla sezione *Generic Desktop Page (0x01)*, e lo usageID corrispondente è *0x06* (*figura 13*). Il secondo passo è quello di raggruppare i comandi in una collezione di tipo Applicazione; è buona pratica raggruppare in una collezione oggetti logicamente legati tra loro (come quelli che insieme formano un report), e il tipo Applicazione indica che la loro struttura segue un formato standard facilmente interpretabile dalle applicazioni.

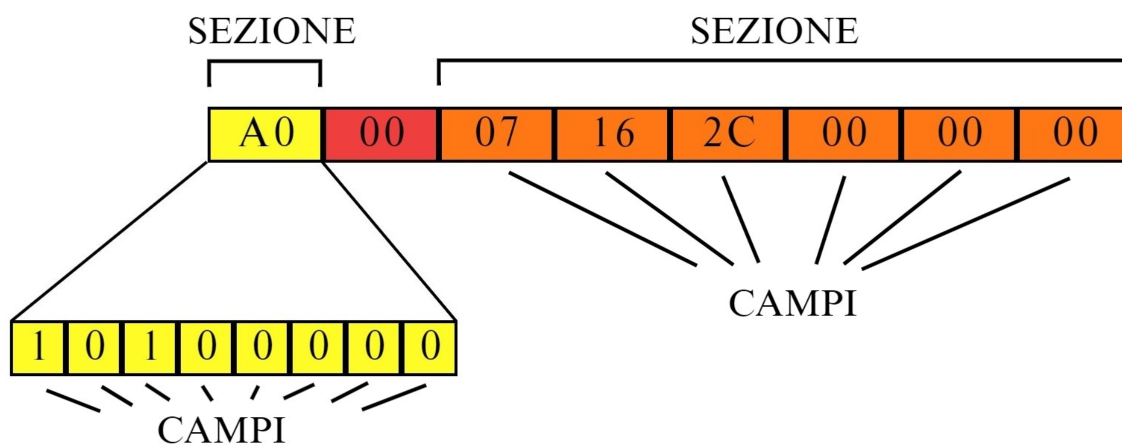

Figura 24: nomi utilizzati per indicare le componenti del report

Il capitolo 3.2 ha introdotto che la struttura di un report si divide in sezioni indipendenti tra loro, con un significato e una struttura propri (ad esempio l'input report è formato dal bitfield dei tasti modificatori, il byte riservato e i bytes per i tasti normali). Per definire correttamente l'impostazione di un report, è necessario fornire per ognuna delle sue componenti le seguenti informazioni:

- dimensione e struttura della sezione, attraverso gli oggetti REPORT COUNT e REPORT SIZE. *Esempio:* la sezione dei tasti modificatori è divisa in 8 campi (REPORT COUNT = 8), ognuno della dimensione di un bit (REPORT SIZE = 1).
- range di valori che può assumere ogni campo, attraverso gli oggetti LOGICAL MINIMUM E LOGICAL MAXIMUM. *Esempio:* un campo della sezione con le informazioni sui led assume il valore 0 per indicare che un led è spento (LOGICAL MINIMUM = 0) o 1 per indicare che è acceso (LOGICAL MAXIMUM = 1).

- range degli *usageID* a cui può fare riferimento il campo, attraverso gli oggetti USAGE MINIMUM e USAGE MAXIMUM. *Esempio:* la sezione che riporta i tasti premuti può comunicare la pressione dei tasti da F13 a F24, a cui corrispondono gli usageID da 0x68 (USAGE MINIMUM = 0x68) a 0x73 (USAGE MAXIMUM = 0x73).

- usage page a cui fanno riferimento i valori di utilizzo, attraverso l'oggetto USAGE PAGE. *Esempio:* la sezione che comunica i tasti normali premuti riferisce i valori alla tabella per tastiere (USAGE PAGE = 0x07).

- funzione della sezione (se forma un report di input o output) e alcuni parametri, attraverso gli oggetti INPUT e OUTPUT. Il valore assegnato è un bitfield dove ogni bit indica una caratteristica indipendente del report; in particolare:

  - bit 0: indica se ogni campo della sezione porta dei dati significativi (0 = Data) oppure è una costante utilizzata per il padding (1 = Constant).

  - bit 1: indica se ogni campo della sezione rappresenta una variabile singola (0 = Variable) o è strutturato come un array (1 = Array).

  - bit 2-8: hanno altri utilizzi, non particolarmente rilevanti per questa applicazione; saranno sempre impostati a 0.

  - bit 9-31: riservati

L'implementazione del report descriptor secondo tale schema è la seguente:

```
MACROPAD_REPORT_DESCRIPTOR = bytes((
    0x05, 0x01,             # Usage Page (Generic Desktop)
    0x09, 0x06,             # Usage (Keyboard)
    0xA1, 0x01,             # Collection (Application)
    #inizia la collezione di oggetti

        #DEFINIZIONE DELL'INPUT REPORT
        #inzio sezione dei tasti modificatori
        0x05, 0x07,             # Usage Page (Key Codes)
        0x19, 0xE0,             # Usage Minimum (224)
        0x29, 0xE7,             # Usage Maximum (231)
        0x15, 0x00,             # Logical Minimum (0)
        0x25, 0x01,             # Logical Maximum (1)
        0x75, 0x01,             # Report Size (1) [in bits]
        0x95, 0x08,             # Report Count (8)
        0x81, 0x02,             # Input (Data, Variable)
        #fine sezione dei tasti modificatori
```

```
        #inizio sezione del byte riservato
        0x75, 0x08,             # Report Size (8)
        0x95, 0x01,             # Report Count (1)
        0x81, 0x01,             # Input (Constant)
        #fine sezione del byte riservato

        #inzio sezione dei tasti normali
        0x05, 0x07,             # Usage Page (Key Codes)
        0x19, 0x68,             # Usage Minimum (0)
        0x29, 0x73,             # Usage Maximum (101)
        0x15, 0x68,             # Logical Minimum (0)
        0x25, 0x73,             # Logical Maximum (101)
        0x75, 0x08,             # Report Size (8)
        0x95, 0x06,             # Report Count (6)
        0x81, 0x00,             # Input (Data, Array)
        #fine sezione dei tasti normali

        #DEFINIZIONE DELL'OUTPUT REPORT
        #inizio sezione dei led
        0x05, 0x08,             # Usage Page (Leds)
        0x19, 0x01,             # Usage Minimum (1)
        0x29, 0x05,             # Usage Maximum (5)
        0x75, 0x01,             # Report Size (1)
        0x95, 0x05,             # Report Count (5)
        0x91, 0x02,             # Output (Data, Variable)
        #fine sezione dei led

        #inizio sezione padding dei led (per completare un byte)
        0x75, 0x03,             # Report Size (3)
        0x95, 0x01,             # Report Count (1)
        0x91, 0x01,             # Output (Constant)
        #fine sezione padding dei led

    #termina la collezione di oggetti
    0xC0            #End Collection
))
```

3.3.3 File di configurazione

Tra le caratteristiche principali del linguaggio CircuitPython torna particolarmente utile la capacità di stabilire una connessione USB tra il controllore e il computer con estrema facilità. Viene infatti offerta la possibilità di creare un file chiamato *boot.py*, all'interno del quale configurare il tipo di connessione USB che si desidera avviare nel momento in cui il dispositivo

viene collegato ad un host. In questo caso, il file conterrà il report descriptor e alcune righe di codice per inizializzare il Raspberry Pi Pico come un dispositivo HID.

```
import usb_hid
#
# qui ci va la DEFINIZIONE DEL REPORT DESCRIPTOR PRESENTATO
#
macropad = usb_hid.Device(       # Crea il dispositivo per la libreria
    report_descriptor = MACROPAD_REPORT_DESCRIPTOR,
    usage_page=0x01,             # Generic Desktop Control
    usage=0x06,                  # Keyboard
    report_ids=(1,),             # I report hanno reportID 1
    in_report_lengths=(8,),      # Invia report di 8 byte
    out_report_lengths=(1,),     # Ricevi report di 1 byte
)

usb_hid.enable((macropad,))    #configura il Pico come un dispositivo HID
                                con il report descriptor fornito
```

3.3.4 Codice

La prima parte del codice si occupa semplicemente di importare le librerie necessarie e definire alcune variabili utili, accompagnate da commenti esplicativi.

```
import time
import board
import digitalio
import usb_hid
# riferimento alla tastiera per funzioni di libreria
this_keyboard = usb_hid.devices[0]

# variabile che contiene il report
# byte 0 = tasti modificatori
# byte 1 = riservato
# byte 2-7 = tasti normali
report = bytearray(8)

# array che tiene conto di quali tasti sono premuti
# keyStamps[i] = -1 ==> il tasto i non è premuto
# altrimenti il valore indica la posizione dell’usageID di i nel report
keyStamps = [-1 for i in range (12)]

# variabile che tiene conta quanti tasti sono premuti (massimo 6)
keys_pressed = 0
```

Vengono poi definiti gli *usageID* associati ai tasti del macropad (F13-F24) e inizializzati i pin del controllore utilizzati per rilevarne lo stato e controllare il led.

```
# array degli UsageID associati ai tasti
usageIDs = [
    0x68, 0x69, 0x6A,   #F13, F14, F15,
    0x6B, 0x6C, 0x6D,   #F16, F17, F18,
    0x6E, 0x6F, 0x70,   #F19, F20, F21,
    0x71, 0x72, 0x73    #F22, F23, F24
]

# definisci i pin collegati ai tasti
buttons = [
    board.GP28, board.GP27, board.GP26,
    board.GP21, board.GP20, board.GP22,
    board.GP17, board.GP18, board.GP19,
    board.GP15, board.GP14, board.GP16
]

# inizializza i pin
key = [digitalio.DigitalInOut(pin_name) for pin_name in buttons]
for k in key:
    k.direction = digitalio.Direction.INPUT
    k.pull = digitalio.Pull.UP

led = digitalio.DigitalInOut(board.GP25)
led.direction = digitalio.Direction.OUTPUT
```

FUNZIONE FOLLOWCOMMAND()

FollowCommand() è una funzione con lo scopo di elaborare un output report; siccome l'unico led presente sul controllore è associato al tasto Caps Lock, è l'unico parametro che viene controllato.

```
# funzione che elabora i report in ingresso
def followCommand(command):
    #il secondo bit del report indica il led per il CAPS LOCK
    if (command[0] & 0x02):
        led.value = 1
    else:
        led.value = 0
```

FUNZIONE PRESSKEY()

PressKey() è una funzione che, invocata alla pressione di un tasto, prepara correttamente il report e lo invia. Controlla se il numero massimo di tasti premuti in contemporanea è stato raggiunto e, in caso affermativo, invia un report che comunica lo stato di errore.

```
# funzione che prepara il report per segnalare la pressione di un tasto
def pressKey(key_index):
    global keys_pressed

    if keys_pressed == 6:       #comunica lo stato di errore
        sendError()

    else:                   #registra la pressione del tasto e invia il report
        keys_pressed += 1
        keyStamps[key_index] = keys_pressed - 1
        report[(keys_pressed - 1) + 2] = usageIDs[key_index]

        sendReport()
```

FUNZIONE RELEASEKEY()

ReleaseKey() è una funzione che, invocata al rilascio di un tasto, modifica il report e lo invia. Fa in modo che tutti i campi dei tasti normali diversi da 0x00 occupino i byte più a sinistra possibile. Dopo di questo aggiorna anche gli indici di posizione dell’array keyStamps.

```
# funzione che prepara il report per segnalare il rilascio di un tasto
def releaseKey(key_index):
    global keys_pressed

    keys_pressed -= 1
    byte_index = keyStamps[key_index]       # salva la posizione corrente
                                 in cui è registrato il codice nel report
    keyStamps[key_index] = -1

    for i in range(byte_index + 2, keys_pressed + 2):
        report[i] = report[i+1]                 # copia il byte successivo
                                                  in quello corrente
    report[keys_pressed + 2] = 0x00             # cancella l'ultimo byte

    for ks in keyStamps:                        # aggiorna gli stamps
        if ks > byte_index:
            ks -= 1

    sendReport()
```

ALTRE FUNZIONI

Sono definite alcune funzioni ausiliarie che rendono più leggibili le operazioni fatte dagli altri metodi: *sendReport(), modifierKeys(), sendError()*.

```
# funzione che invia il report
def sendReport():
    this_keyboard.send_report(report)


# funzione che restituisce un byte con lo stato dei tasti modificatori
def modifierKeys():
    return 0x00   #visto che questa tastiera non ha tali tasti ritorna 0x00


# funzione che invia un report con lo stato di errore, ma il primo byte
corretto
def sendError():
    first_byte = modifierKeys()
    this_keyboard.send_report(bytearray([first_byte, 0x00, 0x01, 0x01,
0x01, 0x01, 0x01, 0x01]))
```

CICLO PRINCIPALE

È infine presente il ciclo principale del programma, il quale possiede una struttura piuttosto semplice. Per prima cosa controlla se è stato ricevuto un OUTPUT report dalla precedente iterazione del ciclo; in caso affermativo viene chiamata la funzione *followCommand()* per elaborare il messaggio in ingresso. In seguito, il codice itera su tutti i pin utilizzati e controlla se i relativi tasti sono premuti o meno. Se un tasto è premuto e non è ancora elencato tra quelli attualmente premuti viene chiamata la funzione *pressKey()* per aggiornare il report; se invece un tasto non risulta premuto, ma possiede un valore valido in keyStamps, significa che è appena stato rilasciato, e viene quindi invocata la funzione *releaseKey()*.

```
while True:

    # controlla se ci sono report in ingresso con ID = 1
    command = this_keyboard.get_last_received_report(1)
    if command is not None:
        followCommand(command)
```

```
    # cicla su tutti i pin e controlla se i tasti sono premuti
    for i in range(12):
        # se il tasto è premuto, e prima non lo era
        if (not key[i].value) and (keyStamps[i] < 0):
            pressKey(i)         # invia il report che indica che il tasto è
                                  stato premuto

        # se il tasto non è premuto, e prima lo era
        if (key[i].value) and (keyStamps[i] >= 0):
            releaseKey(i)       # invia il report che indica che il tasto è
                                  stato rilasciato

        # piccola pausa tra le iterazioni sui tasti
        time.sleep(0.0005)

    # piccola pausa tra le iterazioni del ciclo principale
    time.sleep(0.05)
```

Capitolo 4

Conclusioni

Sono stati presentati nella loro completezza i due principali protocolli di comunicazione utilizzati dalle tastiere; ma è possibile stabilire oggettivamente quale dei due sia il migliore? Proviamo a vedere cosa si può concludere operando una serie di confronti.

STRUTTURA

Risulta evidente che i due protocolli siano stati sviluppati con un approccio differente. Il PS/2 definisce in maniera completa il modo in cui deve avvenire la comunicazione, sia dal punto di vista software, attraverso l'introduzione di codici associati a tasti e comandi, sia dal punto di vista hardware, specificando nel dettaglio la modalità con cui devono essere fisicamente trasmessi i dati. Al contrario, il protocollo HID è stato esplicitamente pensato per non occuparsi del lato hardware, e deve affidarsi ad un altro protocollo per l'effettiva trasmissione dei dati (USB, Bluetooth, I2C…).

FUNZIONALITÀ

Il protocollo PS/2 presenta molte funzionalità pensate per le tastiere, come lo stato *typematic* e il *n-key rollover*. Il protocollo HID, invece, non ha la possibilità di integrare direttamente queste funzioni, e deve ricorrere a soluzioni alternative più complesse. Questa carenza, tuttavia, è perfettamente comprensibile considerando che HID è un protocollo progettato per interfacciare un grande numero di dispositivi diversi, mentre PS/2 lavora esclusivamente con le tastiere.

COMPLESSITÀ E DISTRIBUZIONE DEL LAVORO

Osservando le implementazioni dei due protocolli è possibile notare che i codici hanno lunghezza molto diversa; uno dei motivi principali per questo è sicuramente il fatto che PS/2 deve occuparsi anche della trasmissione fisica dei bit, mentre HID passa semplicemente i dati al protocollo di livello inferiore, ma è opportuno prendere in considerazione anche altri fattori. Il protocollo HID è *multi purpose*, e può funzionare con più dispositivi: è quindi necessario definire correttamente al momento del collegamento le strutture che utilizza (i descrittori). Il codice è invece piuttosto semplice, e si occupa solo di comunicare quali tasti sono premuti. Lo stesso non si può invece dire per il protocollo PS/2, in quanto deve gestire autonomamente molti parametri come lo stato typematic, i ritardi nella digitazione, e le richieste del computer.

In breve, il protocollo HID vede tutta la sua complessità nelle operazioni in fase di inizializzazione e invia poi solo dati molto semplici, lasciando al computer il lavoro di interpretarli correttamente; il protocollo PS/2 ha invece una fase di inizializzazione molto leggera, ma per tutto il resto del tempo è soggetto ad un carico di lavoro significativo, in quanto elabora i dati e li invia al computer già pronti per essere usati.

HOT-PLUGGING

Il termine *hot-plugging* indica l’azione di collegare o scollegare un dispositivo mentre il computer è acceso. Il protocollo PS/2 non supporta questa pratica: dal punto di vista software, un computer può riconoscere dispositivi che utilizzano PS/2 solo se questi sono collegati fin dal momento della sua accensione; non è garantito che una tastiera connessa ad un computer già acceso venga rilevata. I problemi sono presenti anche dal punto di vista hardware, in quanto il connettore non è progettato per evitare cortocircuiti in fase di inserimento, quindi l’hot-plugging non è elettricamente sicuro.

Il protocollo HID, invece, non ha particolari restrizioni software e permette di collegare dispositivi in qualsiasi momento. L’hot-plugging è generalmente reso sicuro anche dal punto di vista elettrico grazie ad alcune precauzioni; ad esempio, i connettori USB (i più diffusi per il collegamento di tastiere HID) riducono enormemente il rischio di cortocircuiti grazie alla loro struttura che permette ai pin di alimentazione di collegarsi prima dei pin dei dati.

CONCLUSIONI

I due protocolli presentano ovviamente i propri pro e contro, e ci sono, nel complesso, troppi parametri secondo cui stabilire quale sia il migliore. Tuttavia, entrambi rispecchiano perfettamente le scelte che sono state seguite nelle fasi del loro progetto: PS/2 è un protocollo ottimizzato per le tastiere, con l'obiettivo di essere molto veloce e responsivo; HID è pensato per connettere molti dispositivi ad un host in modi diversi, rendendo il suo punto forte la portabilità.

Ad ogni modo, se il protocollo HID è stato reso lo standard universale per la comunicazione di molti dispositivi negli ultimi venti anni, delle motivazioni ci sono. L'approccio che è stato seguito è molto più moderno rispetto a PS/2, in quanto segue la tendenza di "stratificare" il protocollo, assegnando ad entità diverse il controllo di mansioni indipendenti. In questo caso, il lavoro è diviso tra il protocollo HID che si occupa di generare correttamente i dati, e un protocollo di sostegno come USB o Bluetooth che invece li trasmette. Oltre ad alleggerire il codice per l'implementazione, questo permette di lavorare allo stesso modo indipendentemente dal metodo di connessione, e ha anche aperto la strada allo sviluppo delle tastiere wireless. Nonostante HID sia un protocollo multi purpose, è stato possibile adattarlo perfettamente alle tastiere, riuscendo a colmare le sue carenze (*stato typematic, n-key rollover*) con aggiunte a livello software. Riesce a pareggiare il protocollo PS/2 anche in quanto a responsività, perché nonostante PS/2 sia stato progettato per essere "veloce", rispecchia gli standard di 40 anni fa; di conseguenza, anche se HID è meno efficiente e invia più dati, non è affatto più lento. Giusto per fare un confronto, PS/2 lavora al massimo a 16,7kHz, e per inviare un frame di 11 bit impiega quindi circa 660µs. Un collegamento USB 1.0 a bassa velocità lavora a 1.5Mbit/s, e per trasmettere un report di 64 bit impiega circa 43µs.

Dal punto di vista dell'utente medio di una tastiera, è molto difficile avere una esperienza diversa con i due protocolli, perché comunque il loro funzionamento 'a scatola chiusa' è lo stesso: comunicare al computer i tasti premuti. La differenza che forse può influenzare maggiormente l'esperienza è la capacità o meno si supportare l'*hot-plugging*.

PS/2 merita in ogni caso del credito perché rappresenta il primo protocollo ben progettato per la comunicazione tastiera-computer, ma ha i suoi limiti ed è ormai decisamente superato. HID si fonda su un modello molto più scalabile ed affidabile, e si è decisamente guadagnato la sua grande diffusione.

Bibliografia

[1] Raspberry Pi Foundation, "Raspberry Pi Pico Datasheet – An RP2040-based microcontroller board".

[2] Adam Chapweske, "The PS/2 Mouse/Keyboard Protocol", www.Computer-Engineering.org, 2003

[3] USB Implementers' Forum, "Device Class Definition for Human Interface Devices (HID) – version 1.11", www.usb.org, 2001

[4] USB Implementers' Forum, "HID Usage Tables FOR Universal Serial Bus (USB) – version 1.5", www.usb.org, 2022

Repositories GitHub

[5] James Huffman, "PS/2 Keyboard From Scratch", github.com/HuffmanCS/PS2-Keyboard/, 2022.

[6] Adafruit Industries, "cicruitpython", github.com/adafruit/circuitpython/.

[7] Adafruit Industries, "usb_hid", github.com/adafruit/circuitpython/tree/main/shared-bindings/usb_hid.