

Integrazione di Sistemi Embedded

Lab 07 Report

Matteo Perotti (251453)
Giuseppe Puletto (251437)
Luca Romani (255244)
Giuseppe Sarda (255648)

January 29, 2019

Abstract

E' stato creato un sistema che si compone di due parti: un programma Python che viene eseguito da PC e un programma C caricato e fatto girare su scheda NUCLEO-F401RE. La comunicazione tra i due dispositivi avviene mediante un cavo USB che emula una seriale. Il debug è stato eseguito grazie alla modalità di SEMIHOSTING che ha permesso il reindirizzamento dello std-output dal processore ARM Cortex M4 passando per il debugger/loader ST-LINK fino al PC.

La seguente relazione spiega i passi operativi in ordine temporale che hanno portato al progetto così come consegnato. La versione finale verrà descritta solo al capitolo 4

1 Descrizione dell'algoritmo

Le specifiche sono disponibili nel documento "lab07.pdf". Di seguito verranno chiarite le specifiche non definite:

1. Qualora venga inviato un comando di accensione LED mentre esso è già acceso, il comando viene considerato non valido e semplicemente ignorato. Stessa cosa se il comando chiede di spegnere un LED già spento.
2. E' necessario che dal PC i comandi arrivino nel formato corretto. Non devono essere inviati comandi più lunghi di due caratteri.
3. Per ottenere un risultato corretto è indispensabile non attendere per più di 65 secondi dopo l'accensione/spegnimento del LED prima di premere il pulsante.

2 Programma Python

Il programma Python, interfacciandosi tramite linea seriale col microcontrollore, permette di inviare o ricevere informazioni da quest'ultimo, informazioni quali i comandi per spegnere e accendere il led del micro o il tempo impiegato dall'utente per finire una simulazione. Può essere utilizzato sia tramite terminale che tramite interfaccia grafica. Il progetto Python+ 3 consta di quattro file .py: *measurer.py*, *lib.py*, *constants.py* e *gui.py*.

2.1 Avvio - *measurer.py*

Il programma Python si avvia eseguendo il file *measurer.py* e passandogli un parametro, "-t" o "-g". "-t" permette l'esecuzione via terminale, mentre "-g" quella via interfaccia grafica. Il file si serve oltre che delle librerie *os* e *sys*, del file *lib.py*.

Se *measurer.py* viene eseguito passandogli un numero non corretto di parametri o un parametro diverso da "-t" o "-g", il programma segnala l'errore via terminale e termina con un exit status uguale a 2. Se avviato correttamente con l'opzione "-g", *measurer.py* esegue il file *gui.py* e l'interfaccia grafica si apre.

Se avviato correttamente con l'opzione "-t", *measurer.py* interagisce con l'utente via terminale tramite un menù. L'utente può chiudere il programma o inviare il comando "L0" (spegni il LED) o "L1" (accendi il LED) digitando sul terminale rispettivamente q, y o n. Se si digita q, *measurer.py* si chiude ritornando come exit state 0; se si digita y (n), *measurer.py* esegue la funzione *turn_on_LED_terminal()* (*turn_off_LED_terminal()*) importata dal file *lib.py*. Il valore di ritorno di *turn_on_LED_terminal()* (*turn_off_LED_terminal()*) viene utilizzato come exit state al momento della chiusura del programma. Se si digita uno o più caratteri diversi dai tre prima specificati, il programma segnala l'errore via terminale, chiedendo nuovamente all'utente cosa voglia fare.

2.2 Funzioni via terminale - *lib.py*

Il file *lib.py* contiene la definizione delle funzioni *delay_cmd()*, *send_ser_cmd(command_to_send)*, *turn_on_LED_terminal()* e *turn_off_LED_terminal()*, *t_score_byte_object_conversion_to_seconds(t_score_byte_object)*.

La funzione *turn_on_LED_terminal()* (*turn_off_LED_terminal()*) invia alla porta seriale specificata nel file *constants.py* il comando per accendere (spegnere) il led. Non riceve alcun parametro e ritorna un intero. Esso vale 3 se il LED è già acceso (spento), altrimenti ritorna 0.

La funzione `send_ser_cmd(command_to_send)` riceve una stringa come parametro d'ingresso che contiene il comando che l'utente desidera inviare alla seriale; ritorna una stringa con il tempo impiegato dall'utente nel premere il pulsante o un errore; utilizza il pacchetto PySerial per gestire la comunicazione via seriale e le altre funzioni di `lib.py` `delay_cmd()` e `t_score_byte_object_conversion_to_seconds(t_score_byte_object)`.

Prima di tutto, `send_ser_cmd()` apre la porta seriale e imposta baudrate, parity bit e byte size della comunicazione, nonché un timeout per leggere la seriale. Il timeout è impostato a 100 s ed impedisce al programma Python di bloccarsi qualora dalla seriale non ricevesse i 5 byte che si aspetta. Poi, se il comando è valido, cioè il led è spento (acceso) e l'utente desidera accenderlo (spegnerlo), dopo un ritardo casuale di 10-20 secondi, viene inviato il comando sulla seriale, non prima però di averlo convertito da stringa a una variabile di tipo byte. Dopo aver letto 5 byte dalla seriale, converte quanto ricevuto in stringa e la passa come parametro di ritorno. La funzione si preoccupa anche di aggiornare `LED_STATE`, la variabile globale che permette al programma Python di ricordare lo stato del LED del micro. Qualora il comando non sia valido, ritorna come stringa un errore "E0001" ("E0000") qualora si voglia accendere (spegnere) il led del micro che è già acceso (spento). La funzione `delay_cmd()` né riceve, né ritorna alcun parametro. Usa la funzione `randint` della libreria Python `random` e `sleep` della libreria Python `time`. Se eseguita, genera un ritardo casuale tra 10 e 20 secondi. La funzione `t_score_byte_object_conversion_to_seconds(t_score_byte_object)` riceve una variabile byte come parametro d'ingresso e ritorna una stringa come parametro di ritorno. La variabile byte dovrebbe contenere il valore di un intervallo di tempo nella forma "Thhhh", dove "hhhh" è il valore dell'intervallo di tempo in ms spesso in esadecimale. Ritorna una stringa contenente il tempo convertito in millisecondi e la relativa unità di tempo.

2.3 File delle costanti - *constants.py*

Il file *constants.py* contiene il valore delle costanti usate dai file *.py*.

2.4 Graphical user interface GUI

L'interfaccia grafica è stata creata usando uno script in python e adoperando il modulo "tkinter". La GUI consente all'utente di mandare i comandi "L0" e "L1" mediante l'utilizzo di due tasti presenti a schermo. Per quel che concerne l'utilizzo di funzioni e l'inizializzazione della porta seriale, l'interfaccia grafica si appoggia allo script `lib.py`.

2.4.1 Widgets

L'interfaccia grafica ha una finestra principale chiamata "root" che viene creata mediante l'inizializzazione dell'interprete "Tk()" del modulo di tkinter e al suo interno contiene gli oggetti:

- `main_Frame`
- `led_on`
- `led_off`
- `quit_button`
- `text_score_lab`
- `t_score_lab`
- `error_lab`

`mainFrame` è un frame che funziona da contenitore per i widget dell'interfaccia grafica. È stato istanziato per gestire in maniera separata il comportamento della finestra principale e dei widget. Gli altri oggetti sopra elencati sono i widget dell'interfaccia grafica e in particolare:

led_on è un bottone verde che se premuto richiama la funzione `turn_on_LED` per inviare il comando "L1" tramite la seriale alla scheda Nucleo. All'interno di questa funzione viene richiamata la funzione `send_ser_command()` di `lib.py` il cui valore di ritorno è salvato all'interno della variabile locale `tmp`. In base al valore di `tmp` la funzione `turn_on_LED` aggiorna `t_score` con il nuovo tempo dell'utente oppure aggiorna il valore di `error_flag` dell'etichetta `error_lab` senza modificare `t_score`.

led_off è un bottone grigio che consente all'utente, come il widget **led_on**, di trasmettere il comando "L0" tramite seriale per spegnere il LED della Nucleo richiamando la funzione **turn_off_LED** che lavora in modo speculare rispetto a quella dell'altro bottone.

quit_button è un bottone rosso che consente all'utente di uscire dal programma. Se premuto invoca la funzione "quit()" dell'oggetto root che fa uscire lo script dal **mainloop()** dell'interfaccia grafica.

text_score_lab è un'etichetta che contiene del testo (Your time score is).

t_score_lab è un'etichetta che mostra il valore della variabile **t_score** aggiornato in tempo reale.

error_lab è un'etichetta sensibile alla variabile **error_flag**.

2.4.2 Struttura

Per quello che riguarda il geometry manager si è deciso di adoperare "pack()" per gestire **mainFrame** e "grid()" per i vari widget. Dato che root ha un solo frame è stato usato **pack()**, in questo modo è stato più facile da configurare l'opzione di riempimento totale di **mainFrame** nella root e il suo ridimensionamento sempre in funzione della finestra padre. Al fine di ottenere questo risultato è stato necessario impostare in **pack()** i flag di "expand" e "fill" rispettivamente a "BOTH" e "1". Inoltre bisogna anche specificare quali righe e colonne del frame dovranno ridimensionarsi, per questo motivo sono stati inseriti due cicli for annidati che per ogni riga e ogni colonna vanno a mettere la variabile "weight" delle funzioni "rowconfigure()" e "columnconfigure()" a 1. Avendo in totale cinque widget, si è scelto **grid()** per avere un'organizzazione di tipo matriciale della GUI, in seguito viene riportata la matrice dell'interfaccia:

	1	2	3
1	led_on		led_off
2	text_score_lab	t_score_lab	error_lab
3		quit_button	

Anche per il manager **grid** è necessario cambiare delle variabili per ottenere un ridimensionamento automatico di ogni widget in funzione della finestra padre, di conseguenza è stata richiamata la funzione "grid_rowconfigure()" e "grid_columnconfigure" per andare a specificare quale riga e colonna dovessero essere ridimensionate dinamicamente. Per capire il fattore di scalamento del widget bisogna assegnare un valore alla variabile "weight" delle due funzioni sopra citate, in questo caso per tutti i widget, tranne il quit button, è stato scelto il valore 1.

3 Programma C

Il programma C deve permettere al microcontrollore di ricevere dati da seriale, interpretarli, accendere e spegnere un led, contare intervalli di tempo, formattarli e poi inviarli nuovamente tramite seriale.

3.1 CubeMX

Grazie al software **CubeMX** è stato possibile ottenere fin dall'inizio una struttura consistente del programma in modo molto agevole.

UART1 Dagli schematici è possibile vedere che il Cortex-M4 e l'ST-LINK sono collegati tramite seriale UART2. Per il programma è stata tuttavia usata UART1 per non passare dall'ST-LINK. Quest'ultima è collegata ai pins PA9 e PA10 del controllore principale e ai connettori D8 e D10 della scheda. In CubeMX è stato quindi selezionata l'USART1 e le impostazioni sono state regolate da specifiche; in aggiunta è stata selezionata anche l'inizializzazione degli interrupts relativi. Questo serve per poter gestire in modo non-blocking la ricezione e la trasmissione.

Connessione fisica La connessione per la trasmissione e ricezione dei comandi è stata effettuata tramite convertitore TTL-232R-3v3. Sono stati utilizzati tre connettori per assicurare un riferimento comune e connettere RX della scheda (D2 - P10) a TX dell'adattatore (cavo arancione) e TX della scheda (D8 - P9) ad RX dell'adattatore (cavo giallo).

TIM1 E' necessario contare con una risoluzione del millisecondo, ed il valore massimo esprimibile in tale unità di misura è 0xFFFF, ovvero 65,535 secondi. Dal reference manual del microcontrollore è possibile vedere che il timer TIM1 è collegato ad APB2. Da CubeMX si può impostare il clock relativo affinché sia di 42 MHz, e il timer affinché il prescaler lo faccia scendere ancora fino ad 1 kHz. In questo modo viene evitata la creazione di una funzione di conversione apposita per ottenere il tempo in millisecondi a partire dal contenuto del registro del timer: con queste impostazioni essi si equivalgono.

GPIO Vengono usati il LED2 a disposizione dell'utente e il blue PushButton. Di quest'ultimo è importante attivare l'interrupt corrispondente allo stato di output asserito, ovvero al falling edge (si può ricavare dallo schematico che il pulsante cortocircuita a gnd il pin del microcontrollore).

Clock settings Come già spiegato nel precedente paragrafo è stato modificato il clock che arriva ad APB2. Il prescaler relativo è stato messo a 4. Il main clock è 84 MHz, viene quindi diviso per 4 per poi essere moltiplicato per 2 ed arrivare ad APB2.

3.2 Scrittura del programma

Il programma utilizza funzioni messe a disposizione dall'HAL della ST. Gli interrupt di ricezione e trasmissione UART, insieme con quello del push-button, sono stati astratti con tre funzioni callback. Per prima cosa sono definiti il buffer di trasmissione e ricezione (5 e 2 bytes rispettivamente).

Prima del ciclo infinito viene chiamata la funzione *HAL_UART_Receive_IT*, che abilita la ricezione in modalità non blocking con interrupt (è disponibile anche in modalità blocking, oppure con DMA). Tutto il resto del programma è implementato nelle funzioni di callback: non appena il buffer di ricezione è pieno viene eseguita quella di ricezione, che si occupa di fare il parsing del comando ricevuto a condizione che non sia già in corso una misurazione. Esso viene ignorato se non valido, mentre nel caso contrario viene eseguita l'azione corrispondente sul LED e viene avviato il timer. Un flag viene asserito per indicare che è in corso una misurazione, e un'altra variabile indica se il LED è correntemente acceso o spento. In seguito viene ri-abilitata la ricezione dei dati da seriale.

Quando il pulsante è premuto viene immediatamente salvato il valore attuale del counter in una variabile globale ed il timer viene fermato e resettato. Poi viene eseguito un controllo sul flag di misurazione: se non è asserito, allora l'azione viene ignorata. Altrimenti il flag di misurazione viene deasserito e il valore della variabile di supporto è convertito in esadecimale e formattato tramite la funzione *sprintf* su un'altra variabile temporanea. Con una funzione che implementa banalmente un ciclo for, questa "stringa" è copiata sul buffer di trasmissione senza il null character finale, per essere poi inviata tramite la funzione non-blocking *HAL_UART_Transmit_IT*. Il controllo sul flag è eseguito successivamente al salvataggio del valore del counter per non introdurre un errore nel conteggio dovuto alla latenza delle operazioni di controllo.

La funzione callback di fine trasmissione può non fare nulla ed è implementata con una *__NOP()*.

3.3 Test e debug

Ogni periferico è stato testato singolarmente prima che venisse assemblato il programma. Il debug è stato eseguito mediante l'utilizzo della *printf* grazie alla modalità di semihosting; essa è stata implementata in una macro che riceve in ingresso una stringa che viene passata ad una *printf* ma

con l'aggiunta di un carattere ' n' finale. Come riportato sul sito della ST questa è una condizione necessaria per il corretto funzionamento del semihosting stesso.

I comandi sono stati inviati e ricevuti da PC per mezzo del programma minicom.

4 Extra: la Leaderboard

Una volta aver soddisfatto le specifiche e testato in modo soddisfacente il sistema si è deciso di aggiungere una *feature* extra alla parte "front end" del progetto e permettere all'utente di salvare il proprio *score* in una leaderboard, la cui lunghezza può essere scelta all'interno del file *constants.py*. Al fine di implementare una classifica, accessibile e gestita mediante il programma, alcune modifiche sono state effettuate al codice e all'interfaccia grafica per:

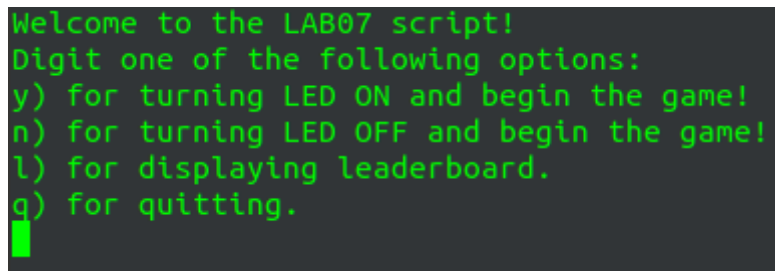
- Dare l'opportunità di **aggiungere** il proprio punteggio alla *leaderboard*.
- **Visualizzare** la classifica.

Nel caso l'utente voglia aggiungere lo *score* ottenuto, deve identificarlo con un *nickname*, richiesto dal programma, che verrà memorizzato e ordinato in modo crescente in base al punteggio. Se il nome scelto è già presente in classifica, lo score verrà aggiornato solo se minore rispetto al precedente. Se non viene inserito il nome nessuna modifica viene effettuata.

Non è possibile azzerare la classifica mediante programma. Unico modo è quello di eliminare a mano il file di salvataggio contenuto nel path relativo *./mem/MemCard1.txt*.

4.1 Esecuzione da terminale

L'esecuzione da terminale si presenta come segue:



```
Welcome to the LAB07 script!
Digit one of the following options:
y) for turning LED ON and begin the game!
n) for turning LED OFF and begin the game!
l) for displaying leaderboard.
q) for quitting.
█
```

Figure 1: Menù programma da terminale

Digitando l'opzione *l* è possibile stampare a schermo la classifica. Nel caso invece si avvii il gioco e si ottenga un punteggio valido, il programma chiede se si vuole salvare. In caso affermativo viene richiesto il *nickname* da associare allo score.

4.2 Esecuzione con GUI

L'esecuzione con GUI si presenta come segue:

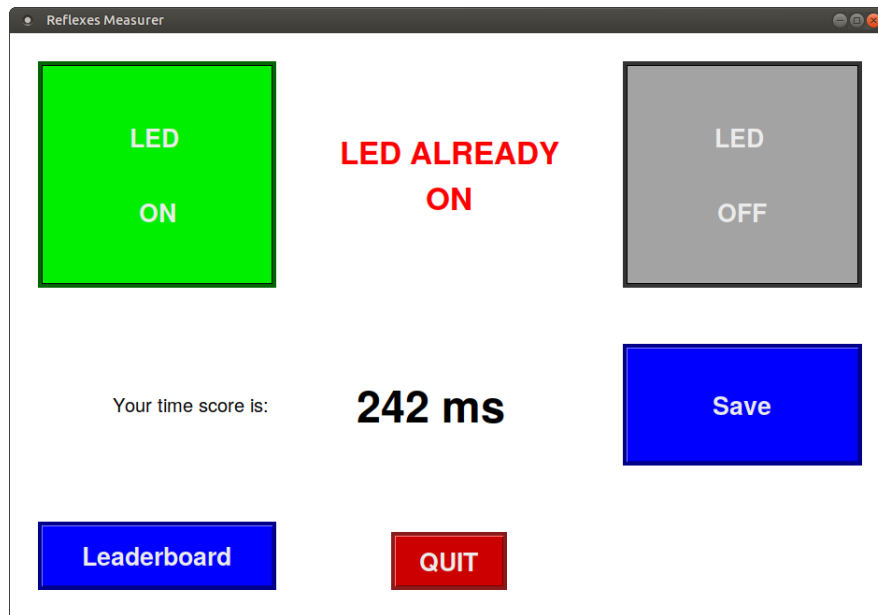


Figure 2: Interfaccia grafica del programma

Dove, rispetto alla precedente versione, l'errore è stato posizionato tra i due pulsanti di avvio. Al suo posto è stato inserito il *Button* che permette all'utente di salvare il punteggio. Una volta premuto farà aprire la seguente finestra:

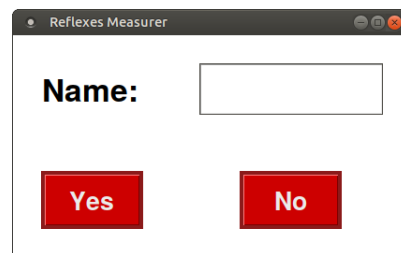


Figure 3: Finestra di salvataggio

Dove se si preme il pulsante "Yes" il *Name*: inserito viene salvato insieme al punteggio nel file di memoria, e poi la finestra viene chiusa. Diversamente, premendo "No" nessuna operazione viene effettuata se non il ritorno alla interfaccia principale.

Secondo *Button* inserito è quello relativo alla visualizzazione della classifica vera e propria. Se viene premuto la finestra di dialogo che si aprirà è la seguente:

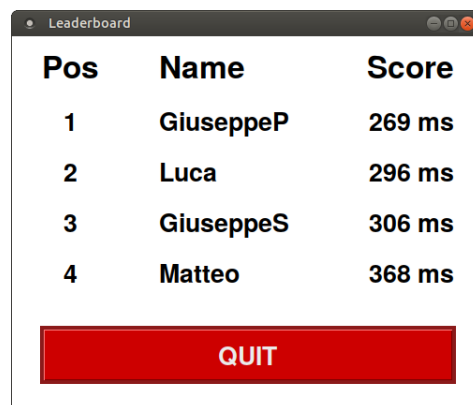


Figure 4: Leaderboard

5 Limiti del progetto

Principali limiti del progetto sono maggiormente due.

- Il primo è relativo alla UART la quale, al primissimo avvio del gioco dopo il reset o il caricamento del programma, a volte invia un dato inesatto che non si aspetta. Dalla seconda esecuzione tutto funziona correttamente.
- Altro limite del programma è il fatto che al suo avvio si presuppone il LED sia spento. Ma se all'esecuzione precedente, si è chiusa l'interfaccia con il LED ancora acceso, all'avvio successivo del programma bisognerà premere il bottone *LED ON*, attendere i 100s di timeout e poi tutto riprende normalmente. Alternativa, in tale caso, è premere il tasto nero di reset sul micro controllore.