EMBEDDED SYSTEMS

FINAL PROJECT – SAFEFORTH

Docente: Daniele Peri

Alessandro Di Giovanni

Ingegneria Informatica
Anno Accademico 2021/22

Indice

Obbiettivo del progetto

Il progetto finale realizzato è un semplice gioco chiamato “Safeforth”, idea nata partendo dall’hardware disponibile e cercando di coprire quanti più argomenti possibili trattati durante il corso.

Il gioco consiste nel trovare tutti e tre i tasti di una sequenza casuale per aprire una cassaforte. Il giocatore muove, tramite un joystick, un puntatore all’interno di un tastierino, e una volta selezionato il tasto desiderato, un led si illuminerà di verde se è il tasto corretto della combinazione o in caso contrario di rosso. Una volta individuati tutti i tasti della combinazione il led lampeggerà di verde e la cassaforte si aprirà.

La sfida consiste nell’ aprire la cassaforte nel minor numero di tentativi, infatti a schermo verrà mostrato il numero di mosse effettuate dal giocatore e il punteggio migliore fino a quel momento.

Il codice del gioco è scritto interamente in Forth, con l’utilizzo di qualche subroutine scritta in ARM Assembly.

Possibili miglioramenti futuri

Partendo dalla base di questo progetto nascono una serie di possibili implementazioni future, anche con un maggiore supporto hardware a disposizione:

- Classifica dei migliori punteggi
- Tasto fisico di avvio e reset
- Possibilità di mettere in pausa il gioco
- Tastiera per inserire il proprio nome in classifica
- Dispositivo di output esterno per inserire suoni nel gioco

Requisiti necessari

Per poter avviare e giocare sono necessari i seguenti requisiti software e hardware:

Software

- pijFORTHOS-SE (implementazione del Prof.Peri)
- Emulatore di terminale nella macchina di sviluppo (minicom)

Hardware

- Raspberry Pi
- Joystick o analogo in grado di fornire in input UP, DOWN, LEFT, RIGHT, MID
- RGB LED
- Monitor con HDMI
- Connettore seriale
- Alimentatore

Il codice del gioco è stato scritto per essere quanto più possibile indipendente dall'architettura hardware su cui viene eseguito.

Infatti, aldilà degli strumenti hardware necessari, a livello software basterà cambiare semplicemente poche costanti nel codice per poter adattare il gioco alle proprie configurazioni.

Inoltre, è necessario, copiare nella scheda SD il file binario *bg_font_open* della cartella *util* è aggiungere a *config.txt*:

initramfs bg_font_open "indirizzo base"

per poterlo caricare in memoria al momento del boot.

File forniti

All'interno della cartella principale è presente:

assembly

Cartella che contiene i codici delle routine assembly utilizzate:

- *fontASSEMBLY.s* → per caricare il font bitmap;
- *loadBG.s* → per caricare il background;
- *loadOPEN.s* → per caricare l'immagine della cassaforte aperta;

util

Cartella che contiene file di diversa natura utilizzati durante il progetto:

- *bg_font_open* → binario dell'immagine di sfondo, cassaforte aperta e font;
- *time.f* → codice Forth utilizzato per dimostrare le diverse complessità temporali per caricare lo sfondo di gioco;
- *toHEX.ipynb* → script Python per convertire un'immagine jpg o png in binario;

src

Cartella che contiene i codici FORTH del gioco:

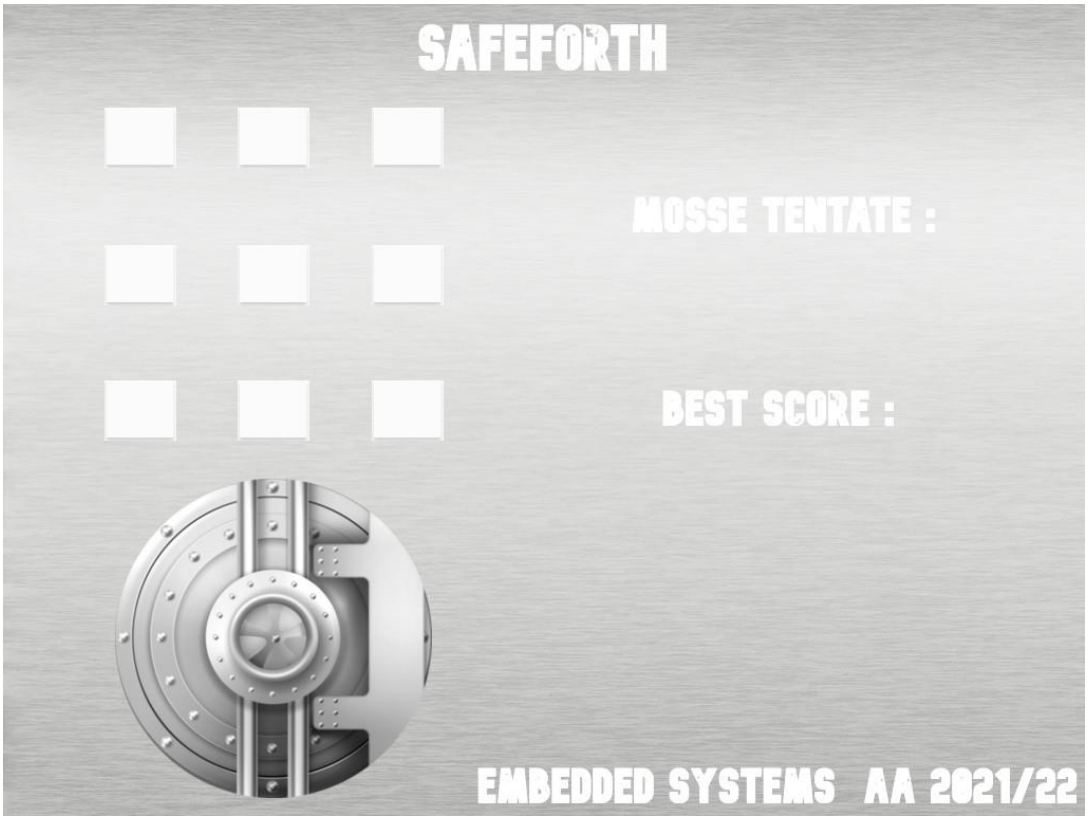
- *0se-ans.f*
- *1initialization.f*
- *2display.f*
- *3memory.f*
- *4output.f*
- *5input.f*
- *6game.f*
- *7start.f*
- *ALL.f*

Istruzioni di avvio

Dopo aver configurato correttamente l'hardware necessario, per poter avviare il gioco bisogna aprire dalla cartella *src* il file *1initialization.f* e cambiare secondo i propri valori le costanti di default degli indirizzi dei registri, del file binario e dei pin di input e output .

Una volta fatto ciò, con il comando del proprio emulatore di terminale, si può decidere di mandare al Pi o il file *ALL.f* che è l'unione di tutti i sorgenti o i singoli file sorgente secondo l'ordine previsto dal numero nel loro nome.

Se tutto è andato correttamente, digitando *START STARTGAME*, il risultato nel monitor è il seguente e il sistema entra in modalità gioco cioè inizia la ricerca della combinazione esatta.



Infine, aperta la cassaforte, quindi terminata la partita, è possibile avviarne un'altra digitando semplicemente *RESET STARTGAME.*

Caricamento dello sfondo

Riguardo il caricamento dello sfondo, quindi di fatto dell'accesso in memoria, è stata prestata maggiore attenzione in quanto esistono più modi per svolgere questo compito.

In particolare le alternative sono tre:

- Utilizzare un loop interamente scritto in Forth che legge i valori 4 byte alla volta dalla memoria, a partire dall'indirizzo noto dello sfondo, e li assegna al rispettivo pixel del framebuffer;
- Utilizzare una word built-in di Forth, *CMOVE*, il cui scopo è proprio quello di copiare una regione di memoria di dimensione x byte, un byte alla volta in un'altra regione di memoria;
- Sfruttare le word implementate dal Prof. Peri per utilizzare in Forth una subroutine assembly, scritta per svolgere esclusivamente questo compito.

Nel file *time.f* della cartella *util* è possibile leggere la dimostrazione che l'ultima alternativa è la migliore in termini di tempo.

Infatti facendo una media del tempo di esecuzione su 10 esecuzioni emerge che le ultime due alternative sono estremamente efficienti e quindi utilizzabili entrambe ma i risultati sono:

- Caricamento tramite CMOVE → **t esecuzione medio = 0,846 secondi;**
- Caricamento tramite Assembly → **t esecuzione medio = 0,212 secondi;**

Discorso opposto invece per le performance del caricamento tramite loop interamente scritto in Forth:

- Caricamentro tramite loop Forth → **t esecuzione medio = 17,27 secondi;**

Infatti utilizzando questo metodo è possibile vedere a occhio nudo il progressivo riempimento dello sfondo.

L'interpretazione personale data ai risultati è che, anche se CMOVE utilizza sicuramente codice assembly più efficiente, nasce per copiare un byte alla volta, mentre per gli scopi del progetto è sufficiente copiarne 4 alla volta, di conseguenza riducendo il numero di operazioni di lettura/scrittura.