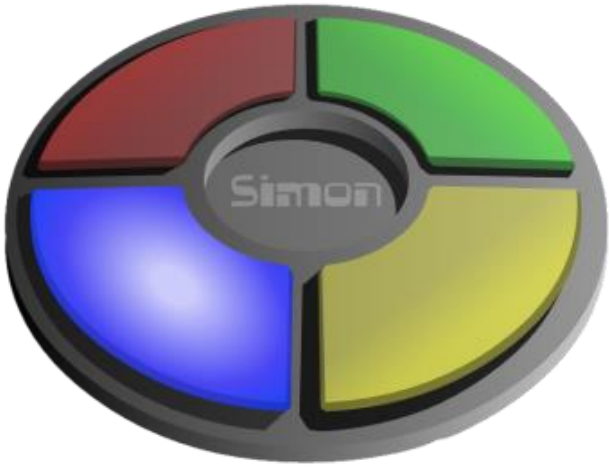
Università degli Studi di Palermo

Vincenzo Messina

0745616

Prof. Daniele Peri



Embedded Systems

Progetto su Raspberry Pi 3 B

Implementazione Simon game

Sommario

Introduzione

Il progetto consiste nell'implementazione del gioco elettronico "Simon" con programmazione bare-metal sul dispositivo Raspberry Pi 3B, utilizzando il linguaggio FORTH e vari componenti elettronici di base.

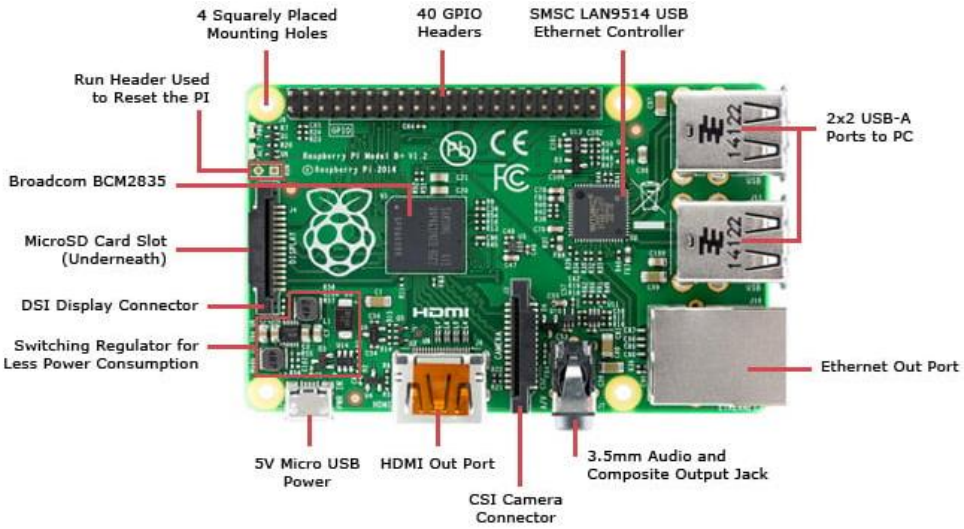
Il gioco originale è composto da quattro pulsanti in grado di illuminarsi, ognuno di diverso colore per il feedback visivo e associato a una nota musicale per il feedback sonoro. L'obiettivo del gioco è di memorizzare e riprodurre una sequenza di pulsanti a cui, a ogni livello superato, si aggiunge un nuovo pulsante casuale da premere.

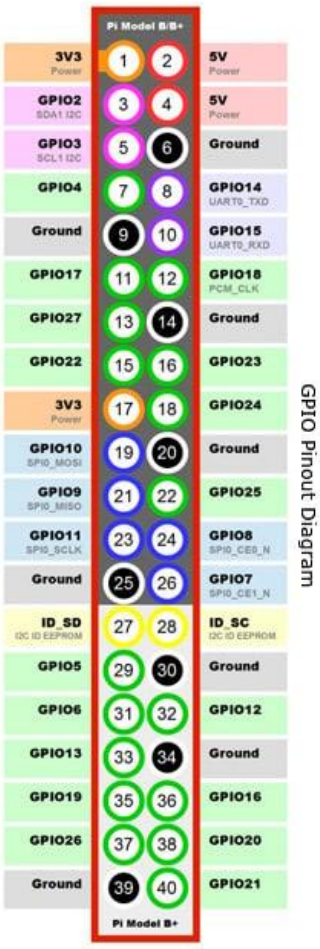
In questa implementazione, per rendere il gioco più arduo, la sequenza di pulsanti da premere è totalmente casuale per ogni livello.

Hardware

Componenti

1. Raspberry Pi 3 Model B





Il Raspberry Pi 3 Model B è un Single Board Computer dotato del System on a Chip (SoC) Broadcom **BCM2837**. Esso monta un processore quad-core Cortex-A15 che opera a una frequenza di 1.4 GHz e una scheda grafica Broadcom VideoCore IV che opera a 0.4 GHz.

Parte fondamentale del dispositivo è la sua dotazione di 40 pin GPIO (General Purpose Input/Output). Questi pin possono essere utilizzati sia come pin di digital input che di digital output permettendo la comunicazione coi componenti elettronici esterni (come i sensori) o altre periferiche. Includono, infatti, pin dedicati alle comunicazioni seriali I^2C e SPI.

Inoltre, è dotato di:

1. 1 x GB di RAM
2. 4 x porte USB
3. 1 x porta HDMI, Ethernet, micro-USB per l'alimentazione, ingresso AUX
4. 1 x slot MicroSD su cui caricare il sistema operativo

2. FT232RL USB to TTL Serial Adapter UART

Componente necessario per poter comunicare in modo seriale con il Raspberry Pi e, di conseguenza, per caricare le istruzioni FORTH. Il collegamento tra l'adattatore e il Raspberry Pi richiede che i pin TX e RX dell'adattatore vengano rispettivamente collegati al pin GPIO 14 (UART0_TXD) e al pin GPIO 15 (UART0_RXD), i quali sono dedicati appositamente per questo scopo. Infine, il pin GND può essere collegato a qualsiasi pin GND del Raspberry.

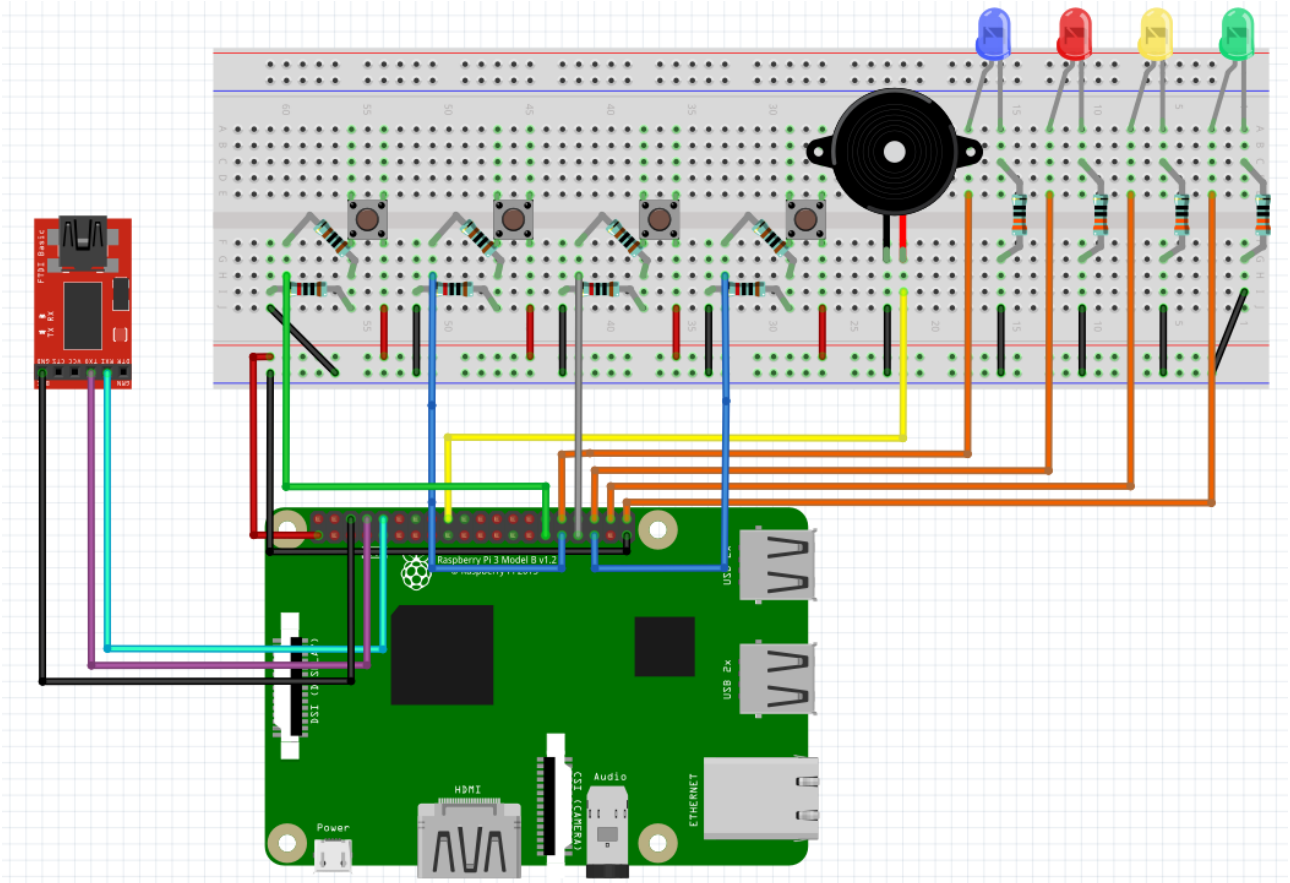
3. Componenti elettronici

- Breadboard
- 4 x Diodi LED 5mm (verde, giallo, rosso e blu)
- 4 x Push Button
- 1 x Buzzer Attivo

  Dispositivo elettronico che utilizza un oscillatore interno che emette un suono fisso quando viene alimentato da una tensione continua.

- 4 x Resistenze 330 Ohm
- 4 x Resistenze 1k Ohm
- 4 x Resistenze 10k Ohm
- Vari Jumper wires

Schema circuitale



Configurazione GPIO

Breadboard	3v3	5v	/
/	GPIO 2	5v	/
/	GPIO 3	GND	GND UART
/	GPIO 4	GPIO 14	TX UART
/	GND	GPIO 15	RX UART
/	GPIO 17	GPIO 18	/
/	GPIO 27	GND	/
/	GPIO 22	GPIO 23	/
/	3v3	GPIO 24	Buzzer
/	GPIO 10	GND	/
/	GPIO 9	GPIO 25	/
/	GPIO 11	GPIO 8	/
/	GND	GPIO 7	/
/	GPIO 0	GPIO 1	/
BUTTON3	GPIO 5	GND	/
BUTTON2	GPIO 6	GPIO 12	LED3 blu
BUTTON1	GPIO 13	GND	/
BUTTON0	GPIO 19	GPIO 16	LED2 rosso
/	GPIO 26	GPIO 20	LED1 giallo
Breadboard	GND	GPIO 21	LED0 verde

Software

Ambiente di sviluppo

Per poter implementare il software del progetto, si è utilizzato il sistema operativo PERIpijFORTHos, una versione dell'ambiente pijFORTHos che si basa sull'implementazione dell'interprete FORTH JonesForth, scritto in Assembly x86.

Il caricamento delle istruzioni, tramite la connessione seriale tra PC e Raspberry, avviene con l'utilizzo della componente software picocom in un terminale Linux o di un qualsiasi sistema Unix-like.
Picocom è un tool per il testing, il debugging e la configurazione di modem che offre anche un programma per la comunicazione seriale, utilizzabile per accedere ai circuiti integrati.

Il comando utilizzato per avviare la connessione è:
"picocom --b 115200 /dev/ttyUSB0 --send "ascii-xfr -sv -l100 -c10" --imap delbs"
in cui viene definito il bitrate (--b 115200) della Virtual COM Port UART del Raspberry, il dispositivo USB corrispondente all'adattatore e vari parametri relativi alla trasmissione dei caratteri.

Funzionalità

N.B.: Dato che a ogni led è associato un bottone, il numero/indice del led (led# nel codice) e il numero/indice del bottone (btn#) coincidono: la scelta di un termine rispetto all'altro è dovuta al contesto. Allo stesso modo, entrambi sono associati anche a un colore, quindi ogni colore è una coppia led/bottone.

Svolgimento della partita

Digitando "START" dal terminale, avrà inizio la partita.
La prima sequenza da riprodurre sarà composta da un solo colore, verrà generata e verrà mostrata utilizzando sia i led che il buzzer. Una volta finita la dimostrazione, il giocatore dovrà premere i bottoni, corrispondenti ai led precedentemente illuminati, nell'ordine corretto.

- Se il giocatore inserirà la sequenza corretta, si sentirà un unico segnale acustico e si passerà alla prossima sequenza, la quale conterrà un colore in più e sarà estratta nuovamente in maniera casuale.
- Se il giocatore inserirà la sequenza sbagliata, si sentiranno due segnali acustici. La sequenza successiva verrà ridotta di un colore e sarà estratta nuovamente in maniera casuale.

La partita sarà vinta dopo aver inserito correttamente una sequenza di dieci colori.
Qualora il giocatore sbagliasse la sequenza di un singolo colore, la partita sarà persa.

Scalabilità

Qualora si volesse, è possibile cambiare facilmente alcune caratteristiche del gioco come il numero di colori utilizzabili, il livello massimo e il livello minimo e di partenza.

Per cambiare i livelli basta avviare la partita anteponendo il numero del livello massimo e del livello minimo alla word 'START', ad esempio scrivendo su terminale '5 15 START' si avvierà una partita partendo dal livello 5 (5 colori da ricordare) e si concluderà al livello 15. La partita verrà persa inserendo la sequenza errata al livello 5.

Per diminuire il numero di colori utilizzabili basta digitare sul terminale 'n CHANGE_COLORS' con n numero di colori desiderato. Il massimo è di quattro colori ma è possibile modificare il limite dal codice modificando la word 'CHANGE_COLORS' nel file gamelogic.f

Generatore di numeri pseudocasuali

Per generare le sequenze che vanno premute, si utilizza il generatore di numeri pseudocasuali XORSHIFT ideato da George Marsaglia.
XORSHIFT, come suggerisce il nome, utilizza esclusivamente spostamenti di bit e operazioni XOR al posto di polinomi, risultando molto veloce e implementabile con poco codice.

I coefficienti utilizzati per gli spostamenti (13, 17, 5) sono una delle combinazioni valide, scelte da Marsaglia, in base ai suoi esperimenti sulle probabilità statistiche delle sequenze generate da esse.

```
\ Generatore di numeri pseudocasuali che inserisce nello stack un numero 32-bit
\ Il valore iniziale del seed è arbitrario e cambia a ogni generazione ma,
\ scegliendo il valore del clock di sistema al momento della generazione, si
\ assicura che sia sempre diverso
VARIABLE SEED   NOW SEED !
: XORSHIFT ( -- u )
    SEED @
    DUP 13 LSHIFT XOR
    DUP 17 RSHIFT XOR
    DUP 5  LSHIFT XOR
    DUP SEED !
;
```

Si utilizzano quindi generazioni successive per popolare l'array di soluzioni

```
\ Genera la sequenza soluzione utilizzando l'algoritmo XORSHIFT e la memorizza
\ nell'array apposito. Utilizza il livello corrente come indice
: GEN_SOL ( -- )
        LEVEL @
        BEGIN
                1- DUP DUP
                XORSHIFT                \ Genera il numero casuale
                COLORS MOD              \ Modulo del numero generato per ridurlo
                                        \ al range dei bottoni presenti
                SWAP GET_SOL !
        0= UNTIL
        DROP
;
```

Suoni diversi col buzzer attivo

Nonostante il buzzer attivo emetta un suono fisso sui 2-2.5 kHZ, è possibile generare suoni con diverse frequenze modulando l'ampiezza via codice.

Considerando il segnale del buzzer come segnale portante, si introduce un segnale modulante per ottenere un segnale modulato con frequenza diversa. Il segnale modulante è dato dalla frequenza di attivazione del buzzer, nonché i delay introdotti.

```
\ Genera note diverse col buzzer attivo sfruttando
\ il principio di modulazione di ampiezza con
\ l'introduzione di un delay, preso dallo stack,
\ che modifica la durata degli impulsi del buzzer
: BUZZ ( us -- )
        500 MS                  \ durata del suono: 500ms, sommato al
        NOW +                   \ valore attuale del clock di sistema
        BEGIN
                BUZZER ON
                OVER DELAY
                BUZZER OFF
                OVER DELAY
                DUP
                NOW -
        0 <= UNTIL
        DROP DROP
;
```

Il valore dei delay introdotti vengono calcolati automaticamente con la word 'BUZZ_DELAY'.
La legge utilizzata è: $(100 + 2^{numero\_led + 4})$ us

```
\ In base al colore che è stato scelto, e quindi al numero del bottone premuto,
\ fornisce i microsecondi di delay utilizzati nella word BUZZ per produrre il suono associato.
\ verde: 116 us, giallo: 132 us, rosso: 164 us, blu: 228 us
: BUZZ_DELAY ( led#/btn# -- us_delay )
        3 +                     \ in modo da rendere l'operazione
        2 SWAP LSHIFT           \ 100 + 2^(led#+4)
        100 +
        US
;
```

Files

I file che compongono la parte software sono:

- jonesForth.f – Alcune delle word utili fornite da Richard W.M. Jones
- init.f – Definisce le costanti e le variabili utilizzate in tutto il progetto
- utils.f – Definisce gli strumenti per la temporizzazione e la generazione di numeri casuali
- components.f – Definisce le funzionalità dei led, dei bottoni e del buzzer
- gamelogic.f – Definisce la gestione dei livelli
- main.f

Vanno caricati sul dispositivo in quest'ordine.

Indice delle word

INIT.F

CELLS	(n_blocks -- tot_size)	Converte celle da allocare in bytes per ALLOT
PIN_TO_WORD	(pin# -- pin_word)	Converte pin GPIO in codifica hex 32-bit

GET_LED	(led# -- led_pin)	Restituisce elemento led# di LEDS
GET_BTN	(btn# -- btn_pin)	Restituisce elemento btn# di BUTTONS
GET_SOL	(index -- led#)	Restituisce i-esimo elem. di SOL (soluzione i)
GET_SEQ	(index -- led#)	Restituisce i-esimo elem. di SEQ (btn premuto i)
SET_ARRAY	(--)	Popola gli array LEDS e BUTTONS
INIT_GPIO	(--)	Set di GPFSEL1 e GPFSEL2

UTILS.F

NOW	(-- clk_value)	Restituisce valore clock attuale
MS	(us -- ms)	Converte microsecondi in millisecondi
SEC	(us -- s)	Converte microsecondi in secondi
DELAY	(us_delay --)	Delay in microsecondi di us_delay
XORSHIFT	(-- u)	Genera numero 32-bit casuale
GEN_SOL	(--)	Genera la soluzione casuale e popola SOL

COMPONENTS.F

ON	(pin --)	Accende/attiva dispositivo dato pin
OFF	(pin --)	Spegne/disattiva disp. dato pin
BUZZ	(us_delay --)	Emette suono con dei ritardi di us_delay
BUZZ_DELAY	(led#/btn# -- us_delay)	Calcola il ritardo associato al led/btn
?ON	(led# -- led_state)	Controlla se led# è acceso
SWITCH_LED	(led# --)	Cambia lo stato di led#
BLINK	(led# --)	Illumina led# e produce suono corrispondente
LED_SEQ	(--)	Mostra soluzione
PRESS	(n --)	Set di ?PRESSED a n
RESET	(--)	Reset di ?PRESSED
WAIT_BUTTON	(--)	Busy wait pressione di un bottone

GAMELOGIC.F

+SCORE	(--)	Incrementa livello attuale / Vittoria
-SCORE	(--)	Decrementa livello attuale / Sconfitta
COMPARE	(-- comparisons)	Confronto di SOL e SEQ
SUM	(comparisons -- sum)	Somma dei confronti
CHECK	(--)	Controllo somma dei confronti e modifica livello
SETTINGS	({n1 n2} --)	Set del livello massimo e minimo
CHANGE_COLORS	(n --)	Cambia il numero di coppie led/btn utilizzate

MAIN.F

LOOP	(--)	Main loop del gioco
START	({n1 n2} --)	Inizia la partita

Sviluppi futuri

- Installazione di un LCD 1602 per visualizzare il livello corrente
- Installazione di un buzzer passivo in grado di generare note precise utilizzando la modulazione di larghezza d’impulso