# Embedded systems

Progetto di programmazione bare-metal su Raspberry Pi 3B

Giuseppe Scibetta

21 settembre 2023

# 1 Una... chiamata pedonale

Il progetto, come suggerisce il titolo, rispecchia a tutti gli effetti una chiamata pedonale: il rosso fisso, una volta che il pedone preme il bottone, passa al giallo e infine al verde, con uno schermo LCD che "adorna" quest'ultimo avvertendo possibili utenti daltonici o dislessici che possono procedere! Insieme a esso, un buzzer per l'agevolazione di persone non vedenti.

A seguire, verranno prima analizzate le componenti hardware utilizzate, seguite dal software e dalla struttura del codice. Il codice si può trovare - all'interno del file compresso - nella cartella `/src`; il codice finale invece si trova nella cartella stessa del progetto, denominato `final.f`.

# 2 Hardware

## 2.1 Raspberry Pi 3B

La macchina target di riferimento per questo progetto è stato il Raspearry Pi, modello 3B, in **Errore. L'origine riferimento non è stata trovata.**.

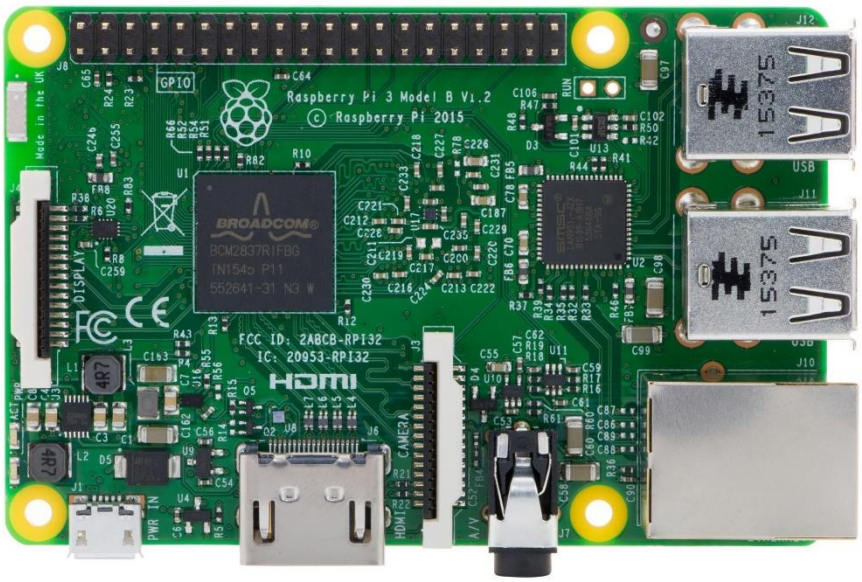


Figura 1: Raspberry Pi 3B. Si può facilmente osservare il suo SoC, il Broadcom BCM2837B0.

Mentre in Figura 2 è possibile osservarne il pinout. Per il progetto ho fortemente fatto uso di un cavo a nastro per estendere i pin e che, al contempo, etichettasse quest'ultimi. In figura è facile notare (al centro, sulla sinistra) il SoC (System on a Chip) dell'RPi, il broadcom BCM2837B0:

- questi integra un processore quad-core Cortex-A15 di terza generazione, operando a una frequenza di 1.4 GHz;
- la sua scheda grafica, Il Broadcom VideoCore 5 di quinta generazione, possiede una frequenza GPU pari a 0.4 GHz.

Altre componenti di particolare rilevanza che si possono osservare sono gli ingressi di cui è dotato: un HDMI, un Ethernet, 4 porte USB, un ingresso AUX, un'ingresso mini USB type-B per l'alimentazione e un'ingresso microSD per il sistema operativo (o, nel nostro caso, per la programmazione bare-metal).

Infine, l'header J8 che ospita i pin del Raspberry, per interagire con alcune delle GPIO del BCM2837B0.

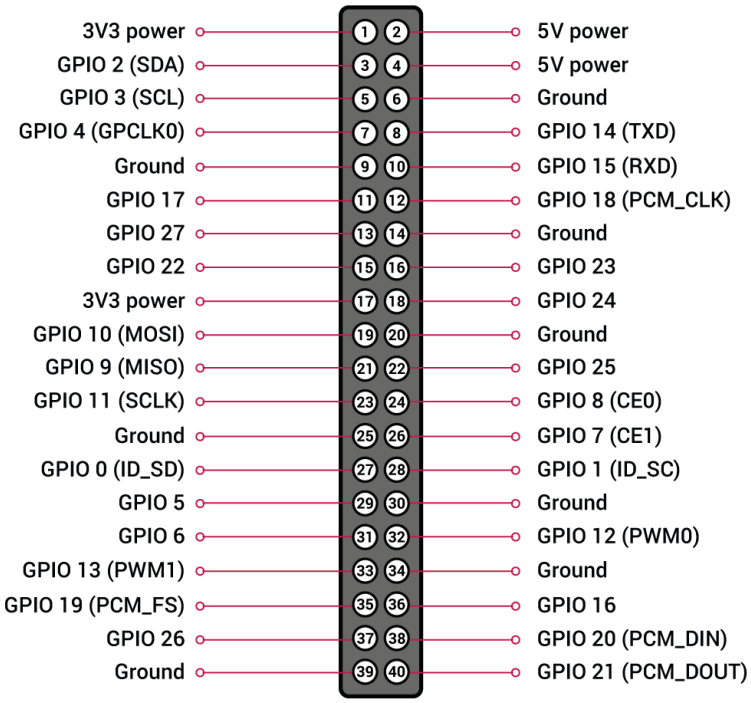


Figura 2: pinout del Raspberry Pi modello 3.

## 2.2 LED, pulsante e resistori

Per la segnalazione luminosa, ho fatto uso dei tipici colori di un semaforo - e nel tipico ordine - e di resistenze da 220 Ω per ogni LED.
Per la chiamata da parte del pedone, si fa uso di un pulsante come quello in Figura 3, a cui sono collegati due resistori da 10 $k\Omega$ ciascuno. In base ai collegamenti effettuati, il pulsante opera su logica hardware negativa: ovvero, HIGH (3.3 $V$) quando non viene pressato, e LOW (0 $V$) quando viene pressato.

## 2.3 Adattatore UART-USB CH340G

I computer moderni non espongono le proprie interfacce seriali per questioni di sicurezza: infatti, le porte UART (Universal Asynchronous Receiver-Transmitter) sono le migliori in ambito embedded per il **debugging**, di conseguenza nel momento in cui un dispositivo entra in commercio, ogni sua porta UART viene rimossa. Per poter dunque consentire la comunicazione tra

il nostro calcolatore e la macchina target, è necessario fare uso dell'adattatore UART-USB CH340G (una versione più "economica" dell'FTDI).

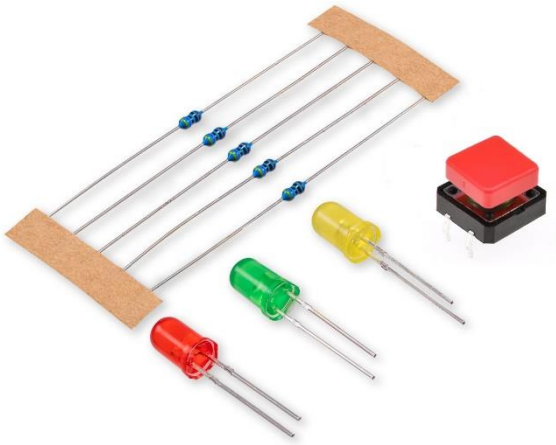

Figura 3: LED, resistori e pulsante utilizzati per il progetto.

Dell'adattatore si farà uso di tre cavi: il GND (ground), TX (trasmissione dati) e RX (ricezione dati); di conseguenza, il collegamento tra il CH340G è l'header J8 dell'RPi è molto intuitivo.

- CH340G TX trasmetterà i dati alla GPIO 14 (RXD), appositamente specializzata nella ricezione dati UART;
- CH340G RX riceverà dati dalla GPIO 15 (RXD), appositamente specializzata nella trasmissione dati UART

Figura 4: adattatore UART-USB CH340G.

## 2.4 LCD1602

Si è anche fatto uso dell'LCD1602, che come suggerisce il nome non è altro che uno schermo LCD $16 \times 2$, dove ogni cella è costituita a sua volta $5 \times 8$ "puntini" per consentire la visualizzazione di qualunque carattere ASCII.

In base al suo datasheet, è possibile notare che può essere pilotato in due modalità: 4/8 bit, e fa uso di 3 segnali di controllo (RS, RW, E). Inoltre, per leggere o scrivere correttamente dati sul bus, è necessaria una corretta temporizzazione: basti osserva il grafico in Figura 6.

Figura 5: schermo LCD1602.

Si può notare che, a priori dal fatto che si stia scrivendo o leggendo (cioè che WR sia un segnale HIGH o LOW) o che l'input sia un dato o un comando (cioè che RS sia HIGH o LOW), è necessario che dopo un tempo $t_{sp_1} = 30\ ns$ venga abilitato il segnale E e che questi abbia il tempo necessario per tornare LOW e campionare correttamente i dati D7-D0 in input: questo tempo è minimo di $t_c = 1000\ ns$, altrimenti la campionatura dei dati subirà degli errori.

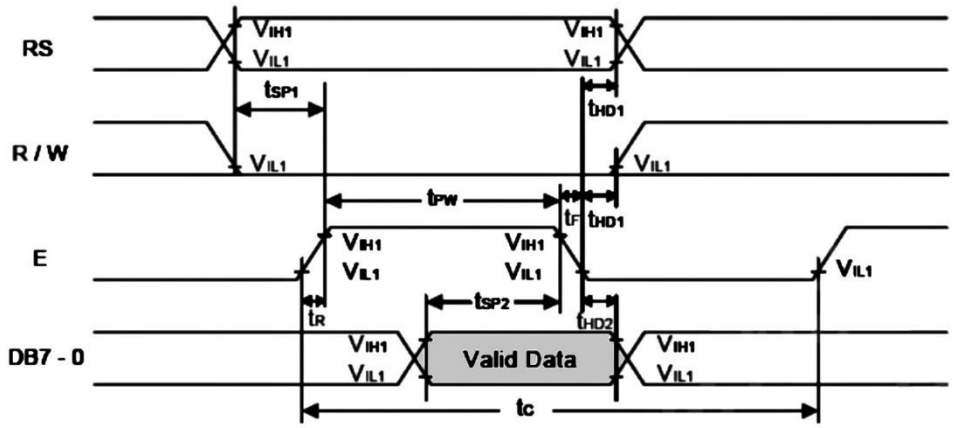


Figura 6: temporizzazione (in fase di scrittura). In fase di lettura sono molto simili.

### 2.4.1 PCF8574

Per agevolare fortemente la comunicazione, si è fatto uso di uno schermo LCD1602 che avesse già integrato un modulo I2C, il PCF8574. Questi è caratterizzato da 6 ingressi, come si può vedere dalla Figura 7.

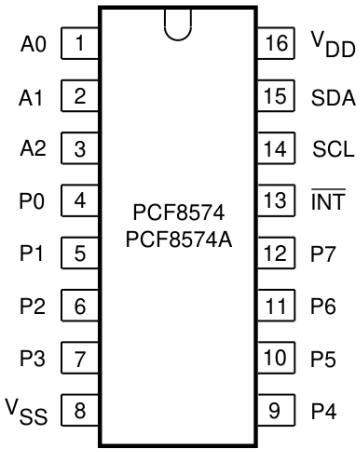


Figura 7: A2-A0 determinano l'indirizzo I2C del dispositivo slave, mentre SDA è la linea utilizzata per la comunicazione I2C: la scrittura (da parte del master) consente di trasmettere serialmente dei dati sulle linee parallele di output P7-P0, la lettura consente di leggere dati da parte dello slave. SCL consente di generare segnali di clock da parte del master sul bus.

### 2.4.2 Comunicazione tra PCF8574 e LCD1602

I collegamenti tra il PCF8574 è l'LCD1602 sono i seguenti.

Tabella 1: collegamenti tra i pin dei due dispostivi.

| PCF8574 | LCD1602 |
|---|---|
| P0 | RS |
| P1 | RW |
| P2 | E |
| P3 | BackLight |
| P4 | D4 |
| P5 | D5 |
| P6 | D6 |
| P7 | D7 |

La prima volta che lo schermo viene avviato, il dispositivo deve ricevere la seguente sequenza di 2 byte che configura l'LCD1602 in modalità 4 bit (nibble):

Tabella 2: setting della FUNCTION SET a modalità 4 bit.

| D7 | D6 | D5 | D4 | BL | E | RW | RS |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

| D7 | D6 | D5 | D4 | BL | E | RW | RS |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |

Si osservi che il nibble di BL, E, RW e RS è, come standard, sempre 1100 seguito dal nibble 1000: questi verranno concatenati al nibble più significativo riguardante i dati D7, D6, D5, D4.
Fatto ciò, per mandare correttamente un dato (per esempio il carattere ASCII "A", in binario 0100 0001) allo schermo, si dovrà fare come segue: in primis, il byte viene suddiviso nei suoi due nibble 0100 e 0001, e questi verranno trasmessi "separatamente" - a partire dal **più significativo** - come segue:

Tabella 3: invio del carattere ASCII "A" allo schermo LCD1602. Per inviare un comando, è sufficiente settare RS a 0.

| | D7 | D6 | D5 | D4 | BL | E | RW | RS |
|---|---|---|---|---|---|---|---|---|
| 1° byte | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 2° byte | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

| | D7 | D6 | D5 | D4 | BL | E | RW | RS |
|---|---|---|---|---|---|---|---|---|
| 1° byte | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 2° byte | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Il MSB data nibble dell'input si concatena con il nibble di configurazione e si ritrasmette con il secondo nibble di configurazione: la stessa operazione si esegue con il LSB data nibble.

## 2.5 Schema del progetto

Il precedente hardware è connesso nel seguente modo.
La differenza con ciò che si è visto e quel che si può osservare in Figura 8 è il modello di adattatore UART-USB che nel software di progettazione (Fritzing)

non risultava presente. Per quanto concerne l'LCD1602, nel mio caso il modulo PCF8574 risulta già integrato allo schermo, ma essendo anch'esso assente ho "rimediato" inserendo un dispositivo che ne facesse uso.

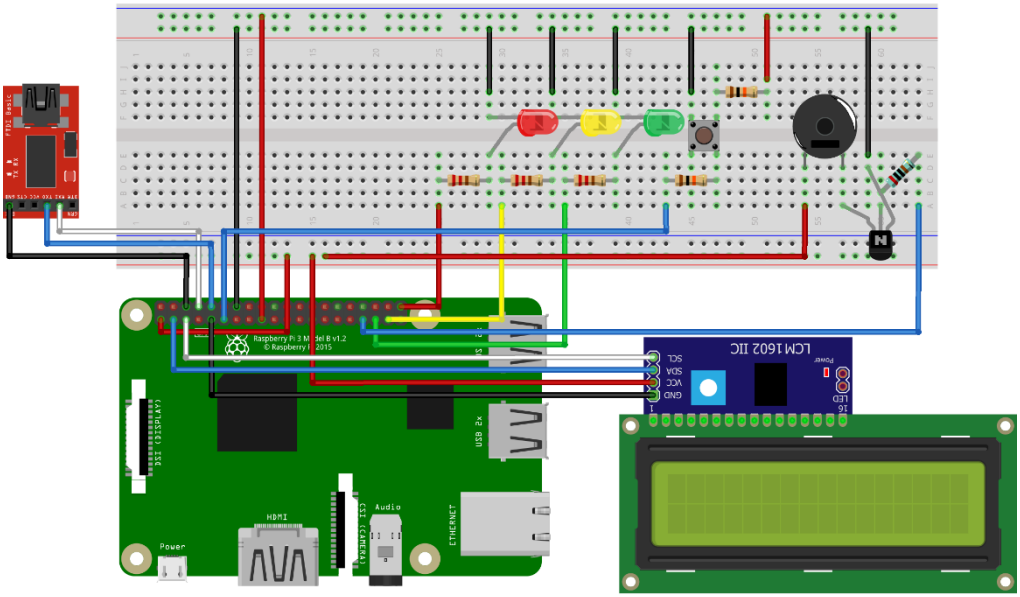


Figura 8: schema del progetto.

I collegamenti tra le GPIO sono rappresentati nella seguente tabella.

Tabella 4: collegamenti tra le GPIO dello progetto in **Errore. L'origine riferimento non è stata trovata.**. Le sigle "R", "Y" e "G" stanno rispettivamente per RED, YELLOW e GREEN. I **pin del pulsante** sono etichettati, in senso orario e a partire dal pin in alto a sinistra, come $B_A$, $B_B$, $B_C$ e $B_D$. Con "E", "B" e "C" si intendono l'emettitore, la base e il collettore del transistor NPN utilizzato nel progetto, mentre "1(+)" fa riferimento al pin positivo del buzzer.

| 3B GPIO | LED rosso | LED giallo | LED verde | Pulsante |
|---|---|---|---|---|
| 21 | + | | | |
| 26 | | + | | |
| 19 | | | + | |
| 17 | | | | $B_C$ |
| 15 | | | | |
| 14 | | | | |
| 3.3 $V$ | | | | $B_B$ |
| GND | − | − | − | $B_A$ |

| 3B GPIO | CH340G | PCF8574 | Buzzer | NPN |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| 14 | RX | | | |
| 15 | TX | | | |
| 2 | | SDA | | |
| 3 | | SCL | | |
| 13 | | | | B |
| 3.3 *V* | | | | |
| 5 *V* | | VCC | 1(+) | |
| GND | GND | GND | | E |

| Buzzer | NPN |
|---|---|
| 2(−) | C |

# 3 Software

## 3.1 Ambiente di sviluppo

### 3.1.1 pijFORTHos

Per programmare in bare-metal sulla macchina target si è fatto uso dell'ambiente pijFORTHos, che come suggerisce il nome è basato sul linguaggio procedurale a stack **Forth**: questi fa uso di definizioni di "word", simili a delle funzioni, che ampliano il dizionario base dell'ambiente Forth.

Tutte le versioni di Forth si basano sul concetto di "programmazione interattiva": quest'ambiente, infatti, è basato su un'implementazione assembly - scritta su un unico file sorgente - dell'interprete **JonesForth**, quest'ultimo scritto invece in assembly x86 da Richard W.M. Jones, e adottato in tanti altri ambienti per la sua semplicità. Per l'interfaccia utente, JonesForth necessita di sole due subroutine di basso livello per leggere dallo `stdin` e scrivere sullo `stdout` un singolo carattere, cioè `getchar` e `putchar`; essendo però poco efficiente, è poco utilizzato. L'unico "inconveniente" di questo ambiente è che opera solamente sul primo modello di RPi. Si ringrazia il professore Daniele Peri per averci permesso di lavorare sulla nuova versione da egli modificata, **PERIpijFORTHos**, che a differenza del suo predecessore è capace di operare su tutti i modelli di RPi usciti fino a oggi (fino al modello 4).

### 3.1.2 Console di comando

Per interagire con la macchina target mediante UART, è necessario installare i seguenti software:

- **minicom**: software di emulazione di terminale per sistemi operativi simili a Unix, comunemente utilizzato per connettere dispositivi GNU/Linux tramite porte seriali;
- **picocom**: è molto simile a minicom, ma progettato in origine come un semplice strumento manuale di configurazione, test e debug di modem. Infatti, picocom non è un "emulatore", ma più propriamente un programma che apre, configura e gestisce una porta seriale e ad essa collega l'emulatore di terminale già in uso.

Per lanciare picocom dalla macchina di sviluppo, bisogna digitare:

```
picocom --b 115200 /dev/ttyUSB0 --send "ascii-xfr -sv -l100 -c10" -
-imap delbs
```

Analizzando i singoli parametri:

- `--b 115200` stabilisce semplicemente il bitrate, in modo che si abbiano gli stessi parametri VCP (Virtual COM Port) UART del Pi;
- `/dev/ttyUSB00` è il dispositivo USB con cui iniziare la comunicazione. Di default è sempre `/ttyUSB00`;
- `--send` stabilisce i parametri di tramissione:
    - `ascii-xfr` consente la trasmissione di caratteri ASCII con un ritardo tra il singolo carattere e la riga trasmessa, rendendo di conseguenza la comunicazione più "sicura": se il ricevente è impegnato nella compilazione o esecuzione di una word Forth e al contempo il mittente trasmette a questi una serie di dati, con molta probabilità i caratteri in input andranno persi (essendo il ricevente impegnato);
    - `-sv` è la combinazione delle opzioni `-s`, che sta per "silenzia la trasmissione" (disabilita la visualizzazione di caratteri trasmessi durante la sessione) e `-v`, che sta per "verbose" (cioè aumenta il livello di dettaglio della sessione, quali porta seriale, opzioni di linea di comando e operazioni in corso);
    - `-l100` impone un ritardo di 100 ms dopo l'invio di ogni riga, garantendo più tempo per l'esecuzione di una word e rendendo di conseguenza il terminale (quasi) sempre disponibile per l'esecuzione di un nuovo comando subito dopo il precedente;
    - `-c10` impone un attesa di 10 ms tra ogni carattere inviato;
- `--imap delbs` rende disponibile l'uso del backspace per eliminare un carattere. `--imap` serve a mappare, lato input, l'opzione di cancellazione `del` (Delete) nel tasto `bs` (Backspace);
- non è possibile utilizzare le frecce in quanto, nello standard ASCII, queste non erano inizialmente contemplate, essendo introdotte in un periodo relativamente "recente"; per tale motivo, non esiste uno standard (variano tra Unix e Windows).

Per uscire dal terminale, si utilizza la sequenza di comandi `[C-a] [C-x]`, ovvero Ctrl + A seguito da Ctrl + X. Per caricare un file (nel nostro caso, il file `.f`

contenente le word da dover essere caricate nel dizionario Forth), si utilizza la sequenza di comandi `[C-a] [C-s]`.

## 3.2 Bootloader e kernel

Per poter lavorare su pijFORTHos, è necessario installare correttamente i seguenti file sulla propria microSD (che si suppone esser stata precedentemente formattata in formato FAT32):

- `bootcode.bin`;
- `config.txt` (file di configurazione, da modificare per abilitare l'UART e l'I2C);
- `fixup.dat`;
- `kernel7.img` (file di immagine contente pijFORTHos);
- `start.elf` (`start4.elf` per un RPi 4);
- (`bcm2711-rpi-4-b.dtb` per un RPi 4).

Se non si dovessero avere questi file, la procedura per ricavarli è molto semplice:

1. installare Raspberry Pi OS sulla stessa microSD, tramite il software ufficiale Raspberry Pi Imager;
2. una volta installato, rimuovere tutti i file nella cartella principale e nella cartella `/firmware` tranne quelli necessari per il boot;
3. cancellare tutti i `kernel*.img`, e sostituirlo con il `kernel7.img` contenente pijFORTHos.

Di `config.txt` è necessario decommentare le righe rigurdanti le connessioni UART e I2C nel seguente modo:

```
dtparam=i2c_arm=on
enable_uart=1
```

In questo modo, le suddette connessioni possono essere utilizzate.

## 3.3 Avvio della comunicazione

Per avviare la comunicazione tra l'host e la macchina target, è necessario digitare quanto segue. Con `./merge` si eseguirà il merge dei sorgenti `.f`

modularizzati generando `final.f` e, con il secondo comando, viene avviata la comunicazione:

```
$ ./merge
$ picocom --b 115200 /dev/ttyUSB0 --send "ascii-xfr -sv -l100 -c10"
--imap delbs
picocom v3.1

port is        : /dev/ttyUSB0
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
hangup is      : no
nolock is      : no
send_cmd is    : ascii-xfr -sv -l100 -c10
receive_cmd is : rz -vv -E
imap is        : delbs,
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none
initstring     : none
exit_after is  : not set
exit is        : no

Type [C-a] [C-h] to see available commands
Terminal ready
```

A questo punto, dopo aver acceso la macchina target, apparirà su schermo:

```
Palermo University 32 bit Research Environment for Embedded Systems
(2022) - Daniele Peri
v. 202201182136
PERI BASE 3F000000
TIME      0000000000190543
CPUID     410FD034
CPSR      600001DA
BOARDREV  00A02082
Board model       0x00000000
Board revision    0x00A02082
Board MAC address 0x0000B827EB626AC3
Board serial      0x000000009B626AC3
ARM memory base address 0x00000000, size 0x08000000
 VC memory base address 0x08000000, size 0x08000000
Clock rates (Hz):
UART:    48000000
```

```
CPU:     600000000
CPU MAX: 1200000000
CORE:    250000000
ISP:     250000000
Initializing frame buffer
framebuffer at 0x0E8FA000 width 00000400 height 00000300 depth
00000020 pitch 00001000

Changing clock rate
clock max rate: 1200000000, clock current rate: 1200000000
Clock rates (Hz):
UART:    48000000
CPU:     1200000000
CPU MAX: 1200000000
CORE:    250000000
ISP:     300000000
pijFORTHos 0.1.8-se-20220113 sp=0x08000000
```

E digitando `[C-a] [C-s]` si potrà caricare `final.f`, contenente l'insieme di word che rendono funzionante il progetto mostrato. Digitando "`START`" sarà possibile avviare il semaforo.

## 3.4 Struttura del codice

Il codice è composto di 7 file:

- `jonesforth.f`: contiene alcune definizioni standard di JonesForth, per esempio per la gestione di stringhe o la conversione di numeri. Del file originale, scaricabile all'URL github.com/jdavidberger/pijFORTHos, ho preso solo un frammento per la word `S"`;
- `utils.f`, contenente le dichiarazioni iniziali quali la `PERIBASE` (indirizzo base delle periferiche) e word ausiliarie di alto livello utilizzate in tutto il codice, quali `SET-FSEL ( fsel pin -- )` e `ON ( pin -- )` e `OFF ( pin -- )` capaci di accettare in input qualunque pin;
- `i2c.f`, contenente le word che manipolano le operazioni sull'I2C;
- `lcd1602.f`, contenente le word che trasmettono dati e comandi allo schermo LCD;
- `buzzer.f`: contenete qualche word per il funzionamento del buzzer;
- `lights.f`, contenente le word che consentono di effettuare la chiamata pedonale;
- `main.f`, contenente la word `START` e le inizializzazioni di I2C e LCD.

Il merge in un unico file, dato che Forth non prevede dei veri e propri "import", avviene mediate il file eseguibile `merge`, strutturato nel seguente modo:

```
#!/bin/bash

if [ -e "final.f" ]; then
    rm final.f
fi
cat src/jonesforth.f \
    src/utils.f      \
    src/i2c.f        \
    src/lcd1602.f    \
    src/buzzer.f     \
    src/lights.f     \
    src/main.f       > final.f
```

Il codice in sé e per sé è molto semplice.
Consiste in un loop dove, ciclicamente, viene mantenuto il rosso acceso (dato che in un semaforo pedonale il rosso è fisso) e lo schermo LCD che mostra "FERMO!": in questo ciclo, si controlla costantemente se il pulsante viene premuto; in quel caso, viene avviata la chiamata pedonale, passando dal LED rosso, al LED giallo e infine a quello verde, con intervalli di tempo differenti e mostrando sullo schermo LCD parole differenti. Il LED verde, verso la fine, inizierà a lampeggiare e un buzzer avvertirà eventuali utenti non vedenti.
Infine, si ritorna allo stato iniziale, in attesa di un'altra chiamata pedonale.