Embedded System
: SmartClima-2.0 ;

Project Work : Soluzione Bare Metal

per la gestione del clima ideale in un sistema "serra".

Davide Proietto - matr. 0739290

Embedded System 2021/22 – prof. D. Peri

Sommario

..... 1
Descrizione del progetto 4
Prerequisiti concettuali 4
Componenti Hardware 4
Schema del sistema 5
Schema di collegamento Header 5
Il targhet: RaspBerry PI 4 B 6
Componenti Software 6
Preparazione dell'ambiente di sviluppo 6
Procedura con Windows 10 e Zoc8 Terminal 7
Procedura con Linux Mint e Picocom 7
Preparazione della SD e dell'interprete 8
Descrizione dei componenti 10
FTDI 232-USB Interfaccia UART 10
Modulo LCD 1602 con Drive I^2C PCF85741/3 10
Tastierino KeyPad a matrice 5x4 14
Modulo relay HL-52s 15
Sistema LED 16
HDMI e FrameBuffer 17
Timer 17
Pulsante On 18
Flusso degli eventi 18
Il codice 18
Dettaglio files sorgenti 19
Testing 48
Conclusioni 58

Descrizione del progetto

Realizzazione di una interfaccia di controllo per la gestione del clima ottimale in un sistema "serra". Ovviamente il progetto è realizzato su dimensioni ridotte, ma è comunque riproducibile su larga scala con i dovuti accorgimenti. Il sistema è realizzato con il target scelto **"Raspberry PI 4 B**", che consente di gestire in maniera automatizzata il controllo della ventilazione e dell'irraggiamento luminoso alle colture presenti nella serra. Una volta impostati i parametri di temporizzazione da tastiera il sistema "bare metal" gestisce l'azione degli attuatori riportando a display le relative informazioni ed a schermo lo stato del sistema, in maniera del tutto autonoma, come vedremo nel dettaglio più avanti.

Prerequisiti concettuali

Per lo sviluppo di un progetto di questa tipologia si utilizza un approccio bottom up nella stesura del codice. Il target va scelto in funzione alle aspettative del sistema: scegliere una CPU General Purpose piuttosto che una CPU Embedded e viceversa è una scelta che deve tenere conto di tanti aspetti: la possibilità di interazione con i sensori del sistema, l'utilizzo di hardware specializzato, le caratteristiche relative al tempo medio nei confronti delle istruzioni necessarie al funzionamento, il fattore economico "costo *componenti – effetto ottenuto"*, e non da poco, l'aspetto del consumo energetico che oggi giorno è preponderante.

Per la realizzazione di un Software embedded possiamo scegliere tra tre tipologie di programmazione:

- Compilazione su una macchina target che richiede il supporto di un OS per l'utilizzo dei toolchain.
- La compilazione incrociata "Cross-compilation" che prevede l'uso di una macchina di sviluppo collegata al target o ad un emulatore con l'ausilio di software di supporto.
- Programmazione interattiva sul target: il codice sorgente viene inviato direttamente all'interprete (*PIJForthOS* nel nostro caso) che lo compila nello stesso target.

Inoltre risulta essere molto utile testare i meccanismi e le connessioni dei i sensori con un linguaggio ad alto livello, se disponibile, prima di cimentarsi allo sviluppo di codice specializzato a basso livello.

Componenti Hardware

Per la realizzazione del progetto è necessario il seguente materiale:

- Raspberry Pi 4B 2 GB con alimentatore;
- MicroSD 16 GB;
- FT232RL USB Interfaccia Seriale UART;
- Sharp TC1602B-01 VER:00 16x2 LCD BLU;
- Philips PCF8574AT Remote 8-bit I/O espansione per I2C-bus;
- Relay Module HL-52s;
- Lampada piatta Led 1 W 5v con interruttore;
- Ventola 12V 8*8 cm;
- 5x4 Keypad Matrix;
- BreadBoard e Cavi connessione M/F e M/M;
- 3 Led vari colori;
- 3 Resistori ceramidi da 200 ohm;
- Personal Computer;
- Monitor/Tv con ingresso HDMI e adattatore;
- Cavo USB – miniUSB M/M;
- Cavo HDMI
- Vaso, terra, piante e supporto attuatori.

Schema del sistema

Schema di collegamento Header

FUN	CONN	HEADER	PIN	PIN	HEADER	CONN	FUN
			1	2	5V VCC	LCD/I2C VCC	
I2C1 SDA	SERIAL DATA (SDA)	GPIO 2	3	4	5V VCC	BREADBOARD	
I2C1 SCL	SERIAL CLOCK (SCL)	GPIO 3	5	6	GND	BREADBOARD	
			7	8	UART TX	FTDI RX	
			9	10	UART RX	FTDI TX	
	PAD ROW 1	GPIO 17	11	12	GPIO 18	PAD ROW 2	
	PAD COL 3	GPIO 27	13	14	GND	LCD/I2C GND	
	PAD COL 2	GPIO 22	15	16	GPIO 23	PAD ROW 3	
			17	18	GPIO 24	PAD ROW 4	
	PAD COL 4	GPIO 10	19	20			
			21	22	GPIO 25	PAD ROW 5	
			23	24			
			25	26			
			27	28			
	LED BUSY WIND	GPIO 5	29	30			
	LED STOP	GPIO 6	31	32	GPIO 12	LED READY LIGHT	
			33	34			
			35	36	GPIO 16	PAD COL 1	
	PULSANTE ON	GPIO 26	37	38			
			39	40			

Il targhet: RaspBerry PI 4 B

Il Raspberry Pi 4 Model B monta un microcontrollore ARM Broadcom BCM2711 quad-core Cortex-A72 a 1.5 Ghz.

Questo target è molto potente poiché può essere programmato dalle applicazioni embedded più semplici come il "blinking led" ad ospitare a bordo interi sistemi operativi Linux e Windows con interfacce grafiche.

Specifiche:

- BROADCOM **BCM2711**, QUAD CORE CORTEX-A72 (ARM V8) 64-BIT SOC @ 1.5GHZ
- 2GB, 4GB OR 8GB LPDDR4-3200 SDRAM
- 2.4 GHZ AND 5.0 GHZ IEEE 802.11AC WIRELESS, BLUETOOTH 5.0, BLE
- GIGABIT ETHERNET
- 2 USB 3.0 PORTS; 2 USB 2.0 PORTS.
- **RASPBERRY PI STANDARD 40 PIN GPIO HEADER** (FULLY BACKWARDS COMPATIBLE WITH PREVIOUS BOARDS)
- 2 × MICRO-HDMI PORTS (UP TO 4KP60 SUPPORTED)
- 2-LANE MIPI DSI DISPLAY PORT
- 2-LANE MIPI CSI CAMERA PORT
- 4-POLE STEREO AUDIO AND COMPOSITE VIDEO PORT
- H.265 (4KP60 DECODE), H264 (1080P60 DECODE, 1080P30 ENCODE)
- OPENGL ES 3.1, VULKAN 1.0
- MICRO-SD CARD SLOT FOR LOADING OPERATING SYSTEM AND DATA STORAGE
- 5V DC VIA USB-C CONNECTOR (MINIMUM 3A*)
- 5V DC VIA GPIO HEADER (MINIMUM 3A*)
- POWER OVER ETHERNET (POE) ENABLED
- OPERATING TEMPERATURE: 0 – 50 DEGREES C

Componenti Software

- Pc Windows 10 con installato G-Forth, e ZOC8 Terminal;
- (In Alternativa) Distro linux (Mint 20.3 LTS) con installato G-Forth, Minicom e Picocom;
- PijFORTHOS 1. 8 (gentile concessione Prof D. Peri) un interprete Forth per soluzioni Bare Metal;

Preparazione dell'ambiente di sviluppo

L'invio del codice sorgente avviene tramite terminale con protocollo FTDI RS-232.

Procedura con Windows 10 e Zoc8 Terminal

ZOC è un software di emulazione terminale professionale per Windows e macOS.

Queta procedura verrà utilizzata durante l'esposizione del progetto ed è riportata a pagina 49.

Procedura con Linux Mint e Picocom

Installare dei tool per la comunicazione e l'interprete Forth per comodità. Da terminale:

- `sudo apt-get install -y gcc-arm-none-eabi`
- `sudo apt-get install -y picocom`
- `sudo apt-get install -y minicom`

Al fine di stabilire la connessione con il target useremo picocom digitando la stringa che ci permette di avviare la comunicazione, permettere l'invio di file con il supporto ASCII e formattare il terminale in maniera corretta:

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

Minicom è un software di emulazione di terminale per sistemi operativi simili a Unix. Viene comunemente utilizzato durante la configurazione di una console seriale remota.

Picocom è, in linea di principio, molto simile a minicom. È stato progettato come un semplice strumento manuale di configurazione, test e debug del modem.

In effetti, picocom non è un "emulatore" di per sé. È un semplice programma che apre, configura, gestisce una porta seriale (dispositivo tty) e le sue impostazioni, e ad essa collega l'emulatore di terminale già in uso.

Parametri PICOCOM:

ASCII-XFR[@ASCII] è stato scelto per inviare il file sorgente al Raspberry perché consente un ritardo tra ogni carattere e la riga inviata.

Questa caratteristica è vantaggiosa perché la comunicazione UART è asincrona: se, mentre il ricevente è impegnato a compilare o eseguire FORTH word, il mittente sta trasmettendo dati, c'è il pericolo di un errore e i caratteri in ingresso andranno persi, con conseguenze disastrose per la corretta esecuzione del programma.

Tramite il comando picocom --b 115200 /dev/tyyUSB0 --imap delbs -s "ascii-xfr -sv -l100 -c10" picocom viene lanciato sulla macchina di sviluppo con gli stessi parametri VCP del Pi UART:

--b 115200: velocità in bit 115200 bit/s.

--imap delbs: consente l'uso di backspace per eliminare un carattere.

-s "ascii-xfr -sv -l100 -c10": specifica ascii-xfr è un comando esterno da usare per trasmettere file.

-sv: modalità di invio dettagliata.

-l100: imposta un ritardo di 100 millisecondi dopo l'invio di ogni riga, questo di solito dà abbastanza tempo per eseguire un comando ed essere pronto per la riga successiva in tempo.

-c10: attende 10 millisecondi tra ogni carattere inviato.

Preparazione della SD e dell'interprete

La soluzione migliore per preparare il Raspberry è quella di formattare ed installare il Sistema operativo base di Rasbian PI OS Lite con il programma Raspberry Pi Imager https://www.raspberrypi.com/software/ . Questo ci permetterà al primo avvio, di partizionare la SD in modo corretto e nello stesso tempo verranno caricati tutti i files necessari al bootstrapping del nostro target. Scendendo nel dettaglio tra tutti i files presenti, solo alcuni sono utilizzati al nostro scopo. La prima fase di boot è svolta dal VideoCore che carica il primo strato dei file d'avvio.

Per i files bootcode.bin e fixup.dat dobbiamo utilizzare quelli generati durante la preparazione della sd. È necessario eliminare tutti gli kernel*X*.img . A questo punto scarichiamo e copiamo sulla sd i file dell'interprete Forth che sono

presenti nel repo https://github.com/organix/pijFORTHos (o versione modificata prof D. Peri).

Forth ha già definiti nel proprio dizionario un potente set di comandi standard, e fornisce dei meccanismi con cui si possono definire i nuovi comandi. Questo processo strutturale di costruzione *di definizioni su definizioni precedenti* rende Forth al pari di un linguaggio ad alto livello. Le words possono essere definite direttamente nei mnemonici assembler. Tutti i comandi sono interpretati dallo stesso interprete e compilati dallo stesso compilatore, conferendo al linguaggio estrema flessibilità. Questo interprete va caricato nel kernel.

Poiché usiamo un sistema a 32bit chiameremo il file blob kernel7.img. Per la produzione di questo file si può usare lo strumento di cross-compiling **gcc-arm-none-eabi** Toolchain https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads .

Per rendere possibile la comunicazione tra l'interprete Forth e l'interfaccia di programmazione dobbiamo abilitare la UART del PI.

Nome	Ultima modifica	Tipo	Dimensione
bcm2710-rpi-zero-2-w.dtb	01/12/2021 16:03	File DTB	28 KB
bcm2711-rpi-4-b.dtb	01/12/2021 16:03	File DTB	49 KB
bcm2711-rpi-400.dtb	01/12/2021 16:03	File DTB	49 KB
bcm2711-rpi-cm4.dtb	01/12/2021 16:03	File DTB	50 KB
boot	15/01/2022 15:35	Documento di testo	4 KB
bootcode.bin	11/01/2022 13:02	File BIN	52 KB
cmdline		Documento di testo	1 KB
config	06/02/2022 06:39	Documento di testo	2 KB
COPYING.linux	01/12/2021 16:03	File LINUX	19 KB
fixup.dat	01/12/2021 16:03	File DAT	8 KB
fixup_cd.dat	01/12/2021 16:03	File DAT	4 KB
fixup_db.dat	01/12/2021 16:03	File DAT	10 KB
fixup_x.dat	01/12/2021 16:03	File DAT	10 KB
fixup4.dat	01/12/2021 16:03	File DAT	6 KB
fixup4cd.dat	01/12/2021 16:03	File DAT	4 KB
fixup4db.dat	01/12/2021 16:03	File DAT	9 KB
fixup4x.dat	01/12/2021 16:03	File DAT	9 KB
issue	02/12/2021 02:32	Documento di testo	1 KB
kernel7	11/01/2022 12:59	File immagine disco	35 KB
LICENCE.broadcom	01/12/2021 16:03	File BROADCOM	2 KB
start.elf	11/01/2022 12:59	File ELF	2.791 KB
start_cd.elf	01/12/2021 16:03	File ELF	781 KB
start_db.elf	01/12/2021 16:03	File ELF	4.696 KB
start_x.elf	01/12/2021 16:03	File ELF	3.629 KB
start4.elf	01/12/2021 16:03	File ELF	2.188 KB

Nel file config.txt inseriamo la stringa di abilitazione come in figura:

enable_uart=1

```
#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800

# Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

# Uncomment this to enable infrared communication.
#dtoverlay=gpio-ir,gpio_pin=17
#dtoverlay=gpio-ir-tx,gpio_pin=18

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on

[pi4]
# Enable DRM VC4 V3D driver on top of the dispmanx display stack
dtoverlay=vc4-fkms-v3d
max_framebuffers=2
enable_uart=1
dtoverlay=w1-gpio,gpiopin=26

[all]
#dtoverlay=vc4-fkms-v3d
enable_uart=1
dtoverlay=w1-gpio,gpiopin=26
```

Descrizione dei componenti

FTDI 232-USB Interfaccia UART

La "USB to Serial Adapter with FT232RL" permette di collegare i PC con qualsiasi sistema a microcontrollore attraverso la porta USB. Usa l'integrato FT232RL della FTDI, dotato di buffer in ricezione da 128 byte e buffer in trasmissione da 256 byte che garantiscono robustezza in trasmissioni ad alta velocità fino a 3Mbaud/s. Oltre ai segnali TX e RX, sono presenti anche le linee CTS, RTS e le altre linee di handshaking. Collegando la scheda alla porta USB, il PC la riconoscerà come una VirtualCOM Port seriale(VCP) attraverso la quale possiamo stabilire la connessione con il targhet emulando la porta seriale RS232, senza bisogno di alcuna modifica. Per il collegamento è sufficiente fornire la linea di alimentazione dalla breadboard e collegare le linee segnale **TX ed RX** alle rispettive **RX e TX** del PI come da tabella. Dato che si tratta di un collegamento SPI Asincrono doppiamo specificare i parametri di sincronizzazione, come la velocità di trasmissione che sarà 115200 e la dimensione del Frame che sarà di un byte.

Modulo LCD 1602 con Drive I^2C PCF85741/3

Questo progetto utilizza un LCD 16x2 (16 righe x 2 colonne) con un modulo I2C integrato. L'LCD è in grado di visualizzare caratteri, lettere, numeri e simboli ASCII. Infatti, comunichiamo con il display LCD semplicemente inviando il codice ASCII del carattere che vogliamo mostrare, in HEX.

L'utilizzo di un'interfaccia I2C che collega l'ingresso seriale e la modalità di uscita parallela all'LCD consente di farlo utilizzare solo 4 linee per comunicare con il display LCD. Il chip IC utilizzato è PCF8574AT e per poter ricavare l'indirizzo dello slave si può utilizzare il sistema operativo Raspbian digitando da terminale il comando *i2cdetect -y 1* per recuperare le due cifre in esadecimale.

Il Bus I2C è un:

- sincrono
- multimaster
- multislave
- commutazione di pacchetto
- single-ended

SCHEMA PARALLELO I2C<->LCD	
On board PC8574T	**Attached LCD 2004A**
P0	RS
P1	RW
P2	E
P3	Backlight
P4	D4
P5	D5
P6	D6
P7	D7

Questo tipo di bus seriale è generalmente utilizzato per collegare circuiti integrati periferici a bassa velocità a processori/microcontrollori a breve distanza.

I cavi Serial Data (SDA) e Serial Clock (SCL) trasportano i dati in un bus I2C.

Usando il meccanismo di Open-Drain per la comunicazione bidirezionale possiamo trasferire il livello logico 0 tirando giù la linea data, il livello logico 1 lasciandola "fluttuante", grazie al resistore di pull-up.

Descrizione del funzionamento del bus I2C:

- Il master inizia la comunicazione inviando:
  - Start Condition
  - L' indirizzo slave (7 bit)
  - 0 per la scrittura R/W (1 bit)

- Lo slave invia l' ACK per confermare la ricezione
- Il master invia l'indirizzo del registro in cui scrivere
- Lo slave invia l' ACK per confermare la ricezione del registro
- Il master inizia a inviare i dati effettivi
- Il master invia la Stop Condition per terminare la comunicazione

In questo caso il master è il Raspberry Pi 4 e lo slave è il modulo LCD I2C.

Mentre l'SCL è alto, una transizione da alto a basso sulla linea SDA con SCL alto definisce una "Start Condition" condizione d'avvio, e alla transizione da basso ad alto sulla linea SDA con SCL alto si definisce "Stop Condition" condizione di arresto. Durante ogni impulso di clock del SCL, un bit di dati viene trasmesso tramite SDA. È possibile trasferire un numero qualsiasi di byte di dati tra la Condizioni di avvio e arresto. I dati vengono trasferiti inviando per primo il bit più significativo.

Ogni byte di dati riceve una risposta ACK (riconoscimento) dal ricevente. Per ricevere l'ACK, il mittente deve rilasciare la linea SDA, in modo che il destinatario possa tirare la linea SDA verso il basso che diventa stabilmente basso durante la fase alta del periodo di clock ACK.

Il collegamento seriale è gestito direttamente dal bus i2c.

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Code	0	0	0	0	1	DL	N	F	X	X

X: Do not care (0 or 1)

DL: It sets interface data length.

DL = 1: Data transferred with 8-bit length (DB7 - 0).

DL = 0: Data transferred with 4-bit length (DB7 - 4).

It requires two times to accomplish data transferring.

N: It sets the number of the display line.

N = 0: One-line display.

N = 1: Two-line display.

F: It sets the character font.

F = 0: 5 x 8 dots character font.

F = 1: 5 x 10 dots character font.

Prima di poter visualizzare i caratteri sull'LCD, dobbiamo inviare diversi comandi per la configurazione dell'LCD. Questo è il passaggio in cui vogliamo inviare un comando al display LCD:

Impostare l'interruttore RS su logico basso (non premuto).

Impostare il dip switch sul comando desiderato.

Premere il pulsante Abilita (che “abilita” il chip ad accettare il comando).

Il primo comando che invieremo è function set. Questo comando serve per impostare la modalità interfaccia, la modalità linea e il tipo di carattere.

Utilizzeremo l'interfaccia a 8 bit, la modalità a 2 righe e il tipo di carattere 5 × 7 punti. Pertanto impostiamo il comando su 0b00111000 o 0x38 in esadecimale. Impostare il pin RS su basso e quindi premere il pulsante di abilitazione.

Dopo aver premuto il pulsante di abilitazione, il modello predefinito sul display LCD verrà cancellato.

Il secondo comando da inserire è display on/off e cursore. Con questo comando accendiamo il display LCD e impostiamo lo stile del cursore (sottolineatura e cursore lampeggiante on/off).

Pertanto impostiamo il comando su 0b00001111 o 0x0F per attivare il display LCD, il cursore sottolineato e il cursore lampeggiante. Impostare il pin RS su basso e quindi premere il pulsante di abilitazione. Dopo aver premuto il pulsante di abilitazione, il display LCD visualizzerà la sottolineatura e il cursore lampeggiante sulla prima riga.

Successivamente, possiamo iniziare a inserire i dati dei caratteri. Per inserire un carattere nell'LCD dobbiamo usare questo passaggio:

1. Impostare l'interruttore RS su logico alto (premuto).
2. Impostare il dip switch sui dati ASCII desiderati.
3. Premere il pulsante Abilita (che “abilita” il chip ad accettare i dati).

Ad esempio, vogliamo visualizzare il carattere "A" sull'LCD. Innanzitutto, impostiamo i dati su 0b01000001 o 0x41 in esadecimale. Quindi imposta il pin RS su alto e infine premi il pulsante di abilitazione. Dopo aver premuto il pulsante di abilitazione, verrà visualizzato il carattere "A" e il cursore si sposta sulla colonna successiva.

Per cancellare il display LCD, possiamo inviare il comando 0b00000001 o 0x01 impostare il pin RS su basso (perché vogliamo inviare il comando), quindi premere il pulsante di abilitazione.

Nel nostro caso utilizziamo la modalità a 4 bit e usiamo una funzione per definire che tipo di informazione stiamo inviando all’LCD.

La WORD **>I2C** del nostro codice scrive un byte relativo all’indirizzo del bus I2C nel master Broadcom Serial Controller (BSC). In questo progetto utilizziamo il secondo indirizzo master, che otteniamo sommando l'offset 804008 all'indirizzo del dispositivo di base.

La WORD **>LCD** controlla se vogliamo inviare un comando o una parte di dati, il bus decodifica e serializza un byte di cui i bit sono posizionati in modo significativo.

Il comando ha un 1 in più nel bit più significativo rispetto ai dati. Infatti un input come 101 >LCD sarebbe considerato un COMANDO per cancellare lo schermo 0000 0001, altrimenti se 0 (ad esempio un input come 41 >LCD) sarebbe considerato un DATA per inviare A CHAR (41 in esadecimale) allo schermo

Per ad esempio, il comando 2C >LCD produce (0x2C = 0010 1100) sul bus I2C:

per la comunicazione seriale all'LCD avremo i 4 bit di selezione e le impostazioni

D7=0, D6=0, D5=1, D4=0, Retroilluminazione=1, Abilita=1, R/W'=0, RS=0

Il comando 28 >LCD produce (0x2C = 0010 1000) sul bus I2C:

D7=0, D6=0, D5=1, D4=0, Retroilluminazione=1, Abilita=0, R/W'=0, RS=0

Questa sequenza di comandi viene interpretata come il comando Function Set (0x20 = 0010 0000) con il parametro DL=0. Di conseguenza, possiamo passare il bus alla modalità a 4 bit. Utilizzando la modalità a 4 bit, per inviare 1 byte a LCD dobbiamo scrivere 4 volte sul bus I2C:

- 4 bit più significativi con Enable = 1
- 4 bit più significativi con Enable = 0
- 4 bit meno significativi con Enable = 1
- 4 bit meno significativi con Enable = 0

Dopo questa configurazione si può inviare qualsiasi carattere ASCII digitando il suo codice HEX e chiamando >LCD word; ad esempio 41 >LCD (che invia A all'LCD).

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

Tastierino KeyPad a matrice 5x4

Questo Keypad è composto da una matrice di linee circuitali 5 righe x 4 colonne. Utilizzando uno dei metodi di scansione delle righe o un metodo di scansione della colonna possiamo rilevare se il tasto è stato premuto.

A tale scopo si configurano:

- i pin GPIO che controllano le righe come output (pin GPIO 17, 18, 23, 24, 25)
- i pin GPIO che controllano le colonne come input (pin GPIO 16, 22, 27, 10)
- Si abilita il rilevamento del fronte di discesa per i pin che controllano le righe al fine di verificare il valore corrente del segnale rispetto al valore che aveva un passo temporale precedente ovvero il registro GPFEN0
- Il registro GPFSEL1 viene utilizzato per definire il funzionamento dei pin GPIO-10 - GPIO-19
- Il registro GPFSEL2 viene utilizzato per definire il funzionamento dei pin GPIO-20 - GPIO-29
- Ogni riga può essere cancellata utilizzando il registro GPCLR corrispondente.

Ad ogni iterazione

- Impostiamo HIGH sulla riga che vogliamo controllare utilizzando il registro GPSET corrispondente
- Premiamo un tasto qualsiasi della riga impostata su HIGH
- Il ciclo di scansione controlla il valore del pin GPIO che controlla la colonna del tasto premuto, tramite il registro GPLEV corrispondente: se il valore è HIGH l'evento press è stato rilevato correttamente.

La configurazione dei pin non è vincolante, basta essere certi di controllare la resistenza Pull Up/Pull Down dei pin selezionati.

Per avviare queste configurazioni oltre ad aver definito i registri con costanti simboliche adatte, basta chiamare la WORD **SETUP_KEYPAD** per abilitare la tastiera.

Modulo relay HL-52s

Possiamo controllare i dispositivi elettronici ad alta tensione usando i relè. Un relè è in realtà un interruttore che è azionato elettricamente da un elettromagnete. L'elettromagnete viene attivato con una bassa tensione, nel nostro caso 5 volt da dal microcontrollore del PI che chiude un contatto per fare erogare o interrompere un circuito ad alta tensione.

Il modulo relè HL-52S a 2 canali ha 2 relè con portata di 10A @ 250 e 125 V AC e 10A @ 30 e 28 V DC. Il connettore di uscita ad alta tensione ha 3 pin, quello centrale è il pin comune e come si vede dalle marcature uno degli altri due pin è per la connessione normalmente aperta e l'altro per la connessione normalmente chiusa.

Dall'altro lato del modulo abbiamo 2 gruppi di pin. Il primo ha 4 pin, una massa e un pin VCC per alimentare il modulo e 2 pin di ingresso In1 e In2 ai rispettivi relè. Il secondo set di pin ha 3 pin con un ponticello tra il JDVcc e il pin Vcc.

In questa configurazione l'elettromagnete del relè è alimentato direttamente dal Raspberry. I carichi di alimentazione degli attuatori, **lampada e ventola**, arrivano da sorgenti esterne, che in uno scenario ecosostenibile potrebbero derivare direttamente da fonti rinnovabili.

Sistema LED

Il sistema led è composto da 3 led di colore diverso e da 3 resistori ceramici da 200 ohm che servono da protezione agli stessi led.

Ogni led è collegato ad un GPIO specifico che ne abilita l'emissione luminosa e lo stesso GPIO gestisce il NC dell'attivazione di un relè.

I GPIO 5, 6, e il 12 sono configurati come OUT.

Quando lo stato passa ad alto, il circuito si chiude a massa, ed il led si accende. Se ciò avviene il contatto NC del relè si interrompe e l'attuatore collegato viene disattivato.

Per cui abbiamo il seguente schema:

FUNZ	GPIO	LED	STATO	ATTUATORE
STOP SISTEMA	6	ROSSO	ON	STOP
ATTIVA LUCE	5	GIALLO	ON	VENTOLA
ATTIVA VENTO	12	VERDE	ON	LAMPADA LED

Per utilizzare in maniera corretta i sensori sono state create delle WORD che rappresentano la configurazione a semaforo dei tre led:

```
: GO_LIGHT
 REDLED GPOFF!
 STOPWIND
 CLEAR
 SYSTEM
 LIGHT
 SYSTEMLIGHT
 ;
```

```
: GO_WIND
 REDLED GPOFF!
 STOPLIGHT
 CLEAR
 SYSTEM
 WIND
 SYSTEMWIND
 ;
```

```
: STOP_DISP
 ALL_LED_ON
 CLEAR
 SYSTEM
 STOP
 ;
```

HDMI e FrameBuffer

Il display Pi HDMI è generato dal Videocore. Un pixel (picture element) si trova attraverso le sue coordinate xy (da (0,0) a (-1,-1)). Il colore di un pixel è controllato da un valore codificato in bit color_depth. Quando si utilizza 32 bit per pixel (bpp), il colore del pixel è codificato come valore ARGB : 8 bit per ciascuno dei componenti red, green, blu colors 8 bit per il canale alfa utilizzato per gli effetti di trasparenza.

I valori dei pixel sono memorizzati in un framebuffer, che è una regione di memoria condivisa dal VC e dalla CPU. Il framebuffer è organizzato come la riga principale della matrice di color_depth.

Il nostro interprete configura il display HDMI per 1024x768x32bpp all'avvio e il colore del pixel x,y è controllato dal valore ARGB nella parola all'offset del byte 4*(1024*y+x) dall'indirizzo di base del framebuffer (ad esempio per il primo pixel bianco 00FFFFFF 3E8FA000).

Il FrameBuffer è inizializzato con i parametri:

```
framebuffer at 0x3E8FA000 width 00000400 height 00000300 depth 00000020 pitch 00001000
```

- framebuffer at 0x3E8FA000 — indirizzo base del framebuffer
- width 00000400 height 00000300 — dimensione del'immagine
- depth 00000020 pitch 00001000 — profondità colore

Al fine di disegnare i simboli necessari al sistema definiamo:

1. Le costanti dei colori necessari in esadecimale.
2. Delle variabili contatori per i cicli al fine disegnare le linee orizzontali e per tenere traccia del numero di riga corrente.
3. Le words necessarie per spostarsi nelle 4 direzioni partendo da CENTER e per colorare ciascun pixel con il colore passato.

Le Words di alto livello per disegnare i simboli dei vari eventi sono:

- DRAWSTARTWIND — Simbolo di avvio ciclo di Ventilazione
- DRAWSTARTLIGHT — Simbolo di avvio ciclo di Illuminazione
- DRAWSTOP — Simbolo di arresto del sistema
- CLEAN — Cancella lo schermo
- DRAWITAFLAG — Simbolo della bandiera Italiana (Benvenuto)

Timer

Il Timer di sistema fornisce quattro canali timer a 32 bit e un singolo contatore a corsa libera a 64 bit. L'indirizzo di base fisico (hardware) per i timer di sistema è 0x7e003000.

Quando i due valori corrispondono, la periferica timer di sistema genera un segnale per indicare una corrispondenza per l'appropriato canale. Il segnale di corrispondenza viene quindi inviato al controller di interruzione. La routine del servizio di interruzione legge quindi l'output confronta il registro e aggiunge l'offset appropriato per il prossimo tick del timer.

Per la gestione del tempo nel nostro sistema useremo il registro a 32 bit CLO. In particolare, definita la costante di un secondo (F4240 in HEX) la logica è che viene registrato il valore di CLO, e a questo viene aggiungo un secondo. Quando il valore attuale di CLO sarà uguale a CLO+1 secondo (ovvero CLO + F4240), allora effettivamente sarà passato un secondo e si potrà prendere questo nuovo valore come riferimento.

Pulsante On

Il sistema di accensione è gestito da un interruttore miniatura da pannello con Pulsante Rosso:

Tensione Nominale: 250 VAC

Corrente Nominale: 1 Ampere

Resistenza di contatto: ≤0.02Ω

Diametro foro di montaggio: 10mm.

Alla pressione del pulsante viene rilevato il fronte di salita asincrono sul pin GPIO dove è collegato tale pulsante: ovvero al GPIO26.

Flusso degli eventi

All'avvio il codice sorgente lancia un ciclo con la word "START" e rimane in attesa dell'evento asincrono di pressione del pulsante POWER ON e il sistema si inizializza visualizza il a schermo il logo Italia e sul display il messaggio di benvenuto, visualizza il nome del dispositivo e lo stato attuale.

A questo punto sarà necessario l'immissione dei dati di temporizzazione espresso in secondi da 0 a 99: le prime due cifre per il SISTEMA ILLUMINAZIONE e altre due per quello di VENTILAZIONE. Si tratta di definire così i tempi relativi agli attuatori "Illuminazione" e "Ventilazione" tutto ciò sarà reso possibile attraverso il keypad.

Una volta inserito questo dato il sistema inizia un ciclo di 4 scambi, controllato da una variabile COUNTER che alterna l'utilizzo del SYSTEM LIGHT al SYSTEM WIND e da una variabile FLAG che viene settata al termine degli scambi.

Il sistema alternerà l'utilizzo degli attuatori in base al delay inserito. Questo processo è inserito in un ciclo infinito.

Nel caso di anomalie o manutenzione il sistema potrà essere disabilitato premendo il tasto ESC in fase di digitazione tempi.

Il codice

Il codice sorgente è suddiviso e commentato in singoli file per area tematica. Ma in fase di caricamento è opportuno unificare e pulire il codice dal commento per aumentare la velocità di caricamento. Ovvero verrà caricato in un unico file con nome final.f realizzato dal comando make:

```
EMBEDDED: ans.f gpio.f i2c.f lcd.f led.f pad.f hdmi.f timer.f button.f main.f
    rm -f embedded.f
    rm -f final.f
    cat ans.f >> embedded.f
    cat gpio.f >> embedded.f
    cat i2c.f >> embedded.f
    cat lcd.f >> embedded.f
    cat led.f >> embedded.f
    cat pad.f >> embedded.f
    cat hdmi.f >> embedded.f
    cat timer.f >> embedded.f
    cat button.f >> embedded.f
    cat main.f >> embedded.f
    grep -v '^ *\\' embedded.f > final.f
```

Dettaglio files sorgenti

ans.f: Inserisce nel dizionario dell'interprete forth alcune WORDS necessarie al funzionamento dell'interprete PiJForthOs.

```
\ Embedded Systems - Sistemi Embedded - 17873
\ definizione di alcune word per l'interprete pijFORTHos
\ modificated by Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

\ '\n'  newline character (10)
: '\n' 10 ;
\ BL    blank character (32)
: BL 32 ;

\ STANDARD CHAR
: ':' [ CHAR : ] LITERAL ;
: ';' [ CHAR ; ] LITERAL ;
: '(' [ CHAR ( ] LITERAL ;
: ')' [ CHAR ) ] LITERAL ;
: '"' [ CHAR " ] LITERAL ;
: '.' [ CHAR . ] LITERAL ;
: 'A' [ CHAR A ] LITERAL ;
: '0' [ CHAR 0 ] LITERAL ;
: '-' [ CHAR - ] LITERAL ;

\ ?IMMEDIATE    ( entry -- p )  get IMMEDIATE flag from dictionary entry
\ ( comment text )  ( -- )  comment inside definition
: ( IMMEDIATE 1 BEGIN KEY DUP '(' = IF DROP 1+ ELSE ')' = IF 1- THEN THEN DUP 0=
UNTIL DROP ;

\ SPACES    ( n -- )    print n spaces
: SPACES BEGIN DUP 0> WHILE SPACE 1- REPEAT DROP ;

\ WITHIN    ( a b c -- p )  where p = ((a >= b) && (a < c))
```

```
: WITHIN -ROT OVER <= IF > IF TRUE ELSE FALSE THEN ELSE 2DROP FALSE THEN ;

\ ALIGNED   ( addr -- addr' )   round addr up to next 4-byte boundary
: ALIGNED 3 + 3 INVERT AND ;

\ ALIGN ( -- )  align the HERE pointer
: ALIGN HERE @ ALIGNED HERE ! ;

\ C,    ( c -- )    write a byte from the stack at HERE
: C, HERE @ C! 1 HERE +! ;

\ S" string"    ( -- addr len ) create a string value
: S" IMMEDIATE ( -- addr len )
    STATE @ IF
        ' LITS , HERE @ 0 ,
        BEGIN KEY DUP '"'
                <> WHILE C, REPEAT
        DROP DUP HERE @ SWAP - 4- SWAP ! ALIGN
    ELSE
        HERE @
        BEGIN KEY DUP '"'
                <> WHILE OVER C! 1+ REPEAT
        DROP HERE @ - HERE @ SWAP
    THEN
;

\ ." STRING"    ( -- )  print string
: ." IMMEDIATE ( -- )
    STATE @ IF
        [COMPILE] S" ' TELL ,
    ELSE
        BEGIN KEY DUP '"' = IF DROP EXIT THEN EMIT AGAIN
    THEN
;

: DICT WORD FIND ;
: VALUE ( n -- ) WORD CREATE DOCOL , ' LIT , , ' EXIT , ;
: TO IMMEDIATE ( n -- )
        DICT >DFA 4+
    STATE @ IF ' LIT , , ' ! , ELSE ! THEN
;
: +TO IMMEDIATE
        DICT >DFA 4+
    STATE @ IF ' LIT , , ' +! , ELSE +! THEN
;
: ID. 4+ COUNT F_LENMASK AND BEGIN DUP 0> WHILE SWAP COUNT EMIT SWAP 1- REPEAT
2DROP ;
: ?HIDDEN 4+ C@ F_HIDDEN AND ;
: ?IMMEDIATE 4+ C@ F_IMMED AND ;
```

```
: WORDS LATEST @ BEGIN ?DUP WHILE DUP ?HIDDEN NOT IF DUP ID. SPACE THEN @ REPEAT CR
;
: FORGET DICT DUP @ LATEST ! HERE ! ;
: CFA> LATEST @ BEGIN ?DUP WHILE 2DUP SWAP < IF NIP EXIT THEN @ REPEAT DROP 0 ;
: SEE
    DICT HERE @ LATEST @
    BEGIN 2 PICK OVER <> WHILE NIP DUP @ REPEAT
    DROP SWAP ':' EMIT SPACE DUP ID. SPACE
    DUP ?IMMEDIATE IF ." IMMEDIATE " THEN
    >DFA BEGIN 2DUP
        > WHILE DUP @ CASE
        ' LIT OF 4 + DUP @ . ENDOF
        ' LITS OF [ CHAR S ] LITERAL EMIT '"' EMIT SPACE
            4 + DUP @ SWAP 4 + SWAP 2DUP TELL '"' EMIT SPACE + ALIGNED 4 -
        ENDOF
        ' 0BRANCH OF ." 0BRANCH ( " 4 + DUP @ . ." ) " ENDOF
        ' BRANCH OF ." BRANCH ( " 4 + DUP @ . ." ) " ENDOF
        ' ' OF [ CHAR ' ] LITERAL EMIT SPACE 4 + DUP @ CFA> ID. SPACE ENDOF
        ' EXIT OF 2DUP 4 + <> IF ." EXIT " THEN ENDOF
        DUP CFA> ID. SPACE
    ENDCASE 4 + REPEAT
    ';' EMIT CR 2DROP
;
: :NONAME 0 0 CREATE HERE @ DOCOL , ] ;
: ['] IMMEDIATE ' LIT , ;
: EXCEPTION-MARKER RDROP 0 ;
: CATCH ( xt -- exn? ) DSP@ 4+ >R ' EXCEPTION-MARKER 4+ >R EXECUTE ;
: THROW ( n -- ) ?DUP IF
    RSP@ BEGIN DUP R0 4-
        < WHILE DUP @ ' EXCEPTION-MARKER 4+
        = IF 4+ RSP! DUP DUP DUP R> 4- SWAP OVER ! DSP! EXIT THEN
    4+ REPEAT DROP
    CASE
        0 1- OF ." ABORTED" CR ENDOF
        ." UNCAUGHT THROW " DUP . CR
    ENDCASE QUIT THEN
;
: ABORT ( -- ) 0 1- THROW ;

: JF-HERE   HERE ;
: JF-CREATE   CREATE ;
: JF-FIND   FIND ;

\ JF-WORDS  ( -- )  print all the words defined in the dictionary
: JF-WORD   WORD ;

: HERE   JF-HERE @ ;
: ALLOT   HERE + JF-HERE ! ;
```

```
\ ['] name  ( -- xt )   compile LIT
: ']    ' LIT , ; IMMEDIATE
: '    JF-WORD JF-FIND >CFA ;

: CELL+  4 + ;

: ALIGN JF-HERE @ ALIGNED JF-HERE ! ;

: DOES>CUT   LATEST @ >CFA @ DUP JF-HERE @ > IF JF-HERE ! ;

: CREATE   JF-WORD JF-CREATE DOCREATE , ;
: (DODOES-INT)  ALIGN JF-HERE @ LATEST @ >CFA ! DODOES> ['] LIT ,  LATEST @ >DFA ,
;
: (DODOES-COMP)  (DODOES-INT) ['] LIT , , ['] FIP! , ;
: DOES>COMP   ['] LIT , HERE 3 CELLS + , ['] (DODOES-COMP) , ['] EXIT , ;
: DOES>INT   (DODOES-INT) LATEST @ HIDDEN ] ;
: DOES>   STATE @ 0= IF DOES>INT ELSE DOES>COMP THEN ; IMMEDIATE

: PRINT-STACK-TRACE
    RSP@ BEGIN DUP R0 4-
        < WHILE DUP @ CASE
        ' EXCEPTION-MARKER 4+ OF ." CATCH ( DSP=" 4+ DUP @ U. ." ) " ENDOF
        DUP CFA> ?DUP IF 2DUP ID. [ CHAR + ] LITERAL EMIT SWAP >DFA 4+ - . THEN
    ENDCASE 4+ REPEAT DROP CR
;

: UNUSED ( -- n ) PAD HERE @ - 4/ ;

\ Control Structures
\ Word  Stack   Description
\ EXIT  ( -- )  restore FIP and return to caller
\ BRANCH offset ( -- )  change FIP by following offset
\ 0BRANCH offset    ( p -- )    branch if the top of the stack is zero
\ IF true-part THEN ( p -- )    conditional execution
\ IF true-part ELSE false-part THEN ( p -- )    conditional execution
\ UNLESS false-part ... ( p -- )    same as NOT IF
\ BEGIN loop-part p UNTIL   ( -- )  post-test loop
\ BEGIN loop-part AGAIN ( -- )  infinite loop (until EXIT)
\ BEGIN p WHILE loop-part REPEAT    ( -- )  pre-test loop
\ CASE cases... default ENDCASE ( selector -- ) select case based on selector value
\ value OF case-body ENDOF  ( -- )  execute case-body if (selector == value)

\ Ritorna informazioni sull'autore delle modifiche
: AUTHOR
    S" TEST-MODE" FIND NOT IF
        ." AUTHOR DAVIDE PROIETTO " VERSION . CR
        UNUSED . ." CELLS REMAINING" CR
        ." OK "
    THEN
```

```
;
```

gpio.f: Questo è il file di configurazione principale, dove vengono settatele costanti dei registri necessari al funzionamento dei vari GPIO e vi sono definite alcune WORDS necessarie a modificare e testare questi dati.

```
\ Embedded Systems - Sistemi Embedded - 17873
\ Configurazione dei registri General Purpose e funzioni sui GPIO
\ Università degli Studi di Palermo
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

\ Includere dopo il flie ans.f

\ GPIO Mapping
HEX

\ Indirizzo base dei registri GPIO in VMU FE200000
FE000000 CONSTANT BASE

\ I registri di selezione della funzione vengono utilizzati per definire il
funzionamento dei pin di I/O generici e delle funzioni
\ alternative relative.
BASE 200000 + CONSTANT GPFSEL0
BASE 200004 + CONSTANT GPFSEL1
BASE 200008 + CONSTANT GPFSEL2

\ Registro dello stato di rilevamento degli eventi, utilizzato per registrare gli
eventi di livello e
\ di onda sui pin GPIO.
BASE 200040 + CONSTANT GPEDS0

\ Registro del set di output utilizzato per impostare un pin GPIO. Ogni bit
definisce il comportamento del pin GPIO da impostare ( bit 1 = GPIO0 )
\ Se il pin GPIO viene utilizzato come input (per impostazione predefinita), il
valore nel campo SETn è ignorato.
\ Tuttavia, se il pin viene successivamente definito come uscita, il bit verrà
impostato in base all'ultimo operazione di set/cancellazione.
BASE 20001C + CONSTANT GPSET0

\ Registro di cancellazione dell'uscita viene utilizzati per cancellare un pin
GPIO. Ogni bit corrisponde ad un pin GPIO da cancellare ( bit 1 = GPIO0 )
\ Se il pin GPIO viene utilizzato come input (per impostazione predefinita), il
valore nel campo CLRN è ignorato.
\ Tuttavia, se il pin viene successivamente definito come uscita, il bit verrà
impostato in base all'ultimo operazione di set/cancellazione.
BASE 200028 + CONSTANT GPCLR0
```

```
\ Registro del livello del pin:  restituisce il valore effettivo del pin. Il campo
LEVn fornisce il valore del rispettivo pin GPIO.
BASE 200034 + CONSTANT GPLEV0

\ Registro di abilitazione del rilevamento del fronte di discesa definisce i pin
per i quali una transizione del fronte di discesa imposta
\ un bit nell'evento di rilevamento registri di stato (GPEDSn). Il registro GPFENn
utilizza il rilevamento del fronte sincrono. Questo
\ significa che il segnale di ingresso viene campionato utilizzando il clock di
sistema e quindi cerca un pattern "100" sul segnale.
BASE 200058 + CONSTANT GPFEN0

\ Registro di abilitazione del rilevamento del fronte di salita asincrono sui pin
GPIO dove è abilitato
\ il controllo di stato di rilevamento eventi.
BASE 20007C + CONSTANT GPAREN0

\ Applica lo spostamento logico sinistro di 1 bit sul valore dato
\ e restituisce il valore spostato
\ Utilizzo: 2 MASK
   \ 2( BIN 0010 ) -> 4( BIN 0100 )
: MASK 1 SWAP LSHIFT ;

\ Imposta il pin GPIO specificato su HIGH se configurato come output
\ Utilizzo: 12 ( pin fisico ) HIGH -> Imposta il GPIO-18 su HIGH
: HIGH
  MASK GPSET0 ! ;

\ Resetta il pin GPIO specificato se configurato come output
\ Utilizzo: 12 ( pin fisico ) LOW -> Resetta il GPIO-18
: LOW
  MASK GPCLR0 ! ;

\ Verifica il valore effettivo dei pin GPIO 0..31
\ 0 -> Il pin GPIO n è LOW
\ 1 -> Il pin GPIO n è HIGH
\ Utilizzo: 12 TPIN (Test GPIO-18)
: TPIN GPLEV0 @ SWAP RSHIFT 1 AND ;

\ Crea un tempo di attesa in millisecondi
\ Utilizzo: 1000 DELAY
: DELAY
  BEGIN 1 - DUP 0 = UNTIL DROP ;

DECIMAL

\ GPIO ( n -- n ) prende il numero pin GPIO e verifica se è inferiore a 27,
altrimenti interrompe
: GPIO DUP 30 > IF ABORT THEN ;
```

```
\ MODE ( n -- a b c) prende il numero del pin GPIO e lascia nello stack il numero
del bit di spostamento a sinistra (a) richiesto per impostare i corrispondenti bit
di controllo GPIO di GPFSELN,
\ dove N è il numero del registro, insieme all'indirizzo del registro GPFSELN (b) e
al valore corrente memorizzato in (c) cancellato da una MASCHERA;
\ N ad (a) sono calcolati dividendo il numero GPIO per 10; N è il quoziente
moltiplicato per 4 mentre a è il promemoria. Quindi GPFSELN viene calcolato da
GPFSEL0 + N
\ (es. GPIO 21 è controllato da GPFSEL2 quindi 21 / 10 --> N = 2 * 4, a = 1 -->
GPFSEL0 + 8 = GPFSEL2 )
\ MASK viene utilizzato per cancellare i 3 bit del registro GPFSEL che controlla
gli stati GPIO utilizzando INVERT AND e il valore (a)
\ La maschera si ottiene spostando a sinistra 7 (111 binary ) di 3 * (resto di 10
divisioni), ad esempio 21 / 10 -> 3 * 1 -> 7 spostato a sinistra di 3 ).
: MODE 10 /MOD 4 * GPFSEL0 + SWAP 3 * DUP 7 SWAP LSHIFT ROT DUP @ ROT INVERT AND
ROT ;

\ OUTPUT (a b c -- ) attiva l'uscita di MODE e quindi imposta il registro GPFSELN
del GPIO corrispondente come uscita.
\ Il bit GPFSELN che controlla l'uscita GPIO è impostato dall'operazione OR tra il
valore corrente di GPFSELN, cancellato dalla maschera, e un 1 spostato a sinistra
dal promemoria di 10
\ divisione moltiplicata per 3. (il valore 001 nella posizione del bit
corrispondente di GPFSELN imposta GPIO come OUTPUT)
\ es. con GPIO 21 AND @GPFSEL2: 011010--> 111000 011010 INVERT AND --> 000010
001000 OR --> 001010
: OUTPUT 1 SWAP LSHIFT OR SWAP ! ;

\ INPUT (a b c -- ) attiva l'uscita di MODE e quindi imposta il registro GPFSELN
del GPIO corrispondente come input.
\ Uguale a OUTPUT ma elimina il valore di spostamento non necessario e il bit
GPFSELN che controlla l'ingresso GPIO è impostato dal
\ INVERTE AND operazione tra il valore corrente di GPFSELN, azzerato dalla
maschera,
: INPUT 1 SWAP LSHIFT INVERT AND SWAP ! ;

\ GPIO On e Off
\ : ON ( pin -- ) 1 SWAP LSHIFT GPSET0 ! ;
\ : OFF ( pin -- ) 1 SWAP LSHIFT GPCLR0 ! ;


\ ON ( n -- ) prende il numero pin GPIO, sposta a sinistra 1 per questo numero e
imposta il bit corrispondente del registro GPCLR0
: ON 1 SWAP LSHIFT GPSET0 ! ;

\ OFF ( n -- ) prende il numero pin GPIO, sposta a sinistra 1 per questo numero e
imposta il bit corrispondente del registro GPSET0
: OFF 1 SWAP LSHIFT GPCLR0 ! ;
```

```
\ LEVEL ( n -- b ) prende il numero pin GPIO, sposta a sinistra 1 di questo numero,
ottiene il valore corrente del registro GPLEV0 e lascia nello stack il valore del
corrispondente
\ Bit numero pin GPIO
: LEVEL 1 SWAP LSHIFT GPLEV0 @ SWAP AND ;

\ GPFSELOUT! scorciatoia per impostare gpio come output
: GPFSELOUT! GPIO MODE OUTPUT ;

\ GPFSELIN! scorciatoia per impostare gpio come input
: GPFSELIN! GPIO MODE INPUT ;

\ GPFSELOUT! scorciatoia per impostare gpio HIGH
: GPON! GPIO ON ;

\ GPFSELOUT! scorciatoia per impostare gpio low
: GPOFF! GPIO OFF ;

\ GPFSELOUT! scorciatoia per ottenere il livello gpio
: GPLEV@ GPIO LEVEL ;

\ GPAFEN ( n -- ) imposta il registro GPAFEN0 per il pin gpio n per l'evento di
caduta asincrono
: GPAFEN! GPIO 1 SWAP LSHIFT GPFEN0 ! ;

HEX
```

I2c.f: Questo è il file che definisce la comunicazione seriale tra il Rasberry PI 4 e il modulo i2c collegato all'LCD 1602.

```
\ Embedded Systems - Sistemi Embedded - 17873
\ I2C Driver
\ Università degli Studi di Palermo
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

\ Includere dopo gpio.f

\ Ci sono otto master Broadcom Serial Controller (BSC) all'interno di BCM2711,
\ BSC2 e BSC7 sono dedicati all'uso da parte delle interfacce HDMI, qui utilizziamo
BSC1 all'indirizzo: 0xFE804000

\ Per utilizzare l'interfaccia I2C, basta aggiungere i seguenti offset
all'indirizzo del registro BCS1.
\ Ogni registro è lungo 32 bit
```

```
\ 0x0 -> Control Register ( usato per abilitare gli interrupt, cancellare il FIFO,
definire un'operazione di lettura o scrittura e avviare un trasferimento )
\ 0x4 -> Status Register ( usato per registrare lo stato delle attività, gli errori
e le richieste di interruzione )
\ 0x8 -> Data Length Register ( definisce il numero di byte di dati da trasmettere
o ricevere nel trasferimento I2C )
\ 0xc -> Slave Address Register ( specifica l'indirizzo slave e il tipo di ciclo )
\ 0x10 -> Data FIFO Register ( utilizzato per accedere al FIFO )
\ 0x14 -> Clock Divider Register ( usato per definire la velocità di clock della
periferica BSC )
\ 0x18 -> Data Delay Register ( fornisce un controllo accurato sul punto di
campionamento/lancio dei dati )
\ 0x1c -> Clock Stretch Timeout Register ( fornisce un timeout su quanto tempo il
master può attende lo slave, così da allungare il timeout, prima di decretarne la
caduta )

\ I2C REGISTER ADDRESSES
\ BASE 804000 + -> I2C_CONTROL_REGISTER_ADDRESS
\ BASE 804004 + -> I2C_STATUS_REGISTER_ADDRESS
\ BASE 804008 + -> I2C_DATA_LENGTH_REGISTER_ADDRESS
\ BASE 80400C + -> I2C_SLAVE_ADDRESS_REGISTER_ADDRESS
\ BASE 804010 + -> I2C_DATA_FIFO_REGISTER_ADDRESS
\ BASE 804014 + -> I2C_CLOCK_DIVIDER_REGISTER_ADDRESS
\ BASE 804018 + -> I2C_DATA_DELAY_REGISTER_ADDRESS
\ BASE 80401C + -> I2C_CLOCK_STRETCH_TIMEOUT_REGISTER_ADDRESS

\ I pin GPIO-2 (SDA) e GPIO-3 (SCL) devono prendere la ALTERNATIVE FUNCTION 0
\ quindi dobbiamo configurare il GPFSEL0 che viene utilizzato per definire il
funzionamento dei primi 10 pin GPIO.
\ ogni 3-bit del gpfsel rappresentano un pin GPIO, così per indirizzare GPIO-2 e
GPIO-3
\ nel campo GPFSEL0 (32-bits), dobbiamo operare sui bit in posizione 8-7-6 (GPIO-2)
e 11-10-9 (GPIO-3)
\ di conseguenza dobbiamo scrivere (0000 0000 0000 0000 0000 1001 0000 0000) ovvero
in HEX (0x00000900)
\ su GPFSEL0 per settare la ALTERNATIVE FUNCTION 0

: SETUP_I2C
  900 GPFSEL0 @ OR GPFSEL0 ! ;

\ Ripristina lo Status Register utilizzando I2C_STATUS_REGISTER (BASE 804004 +)
\ HEX (0x00000302) è (0000 0000 0000 0000 0000 0011 0000 0010) in BIN
\ Il bit 1 è 1 -> Cancella campo DONE
\ Il bit 8 è 1 -> Cancella campo ERR
\ Il bit 9 è 1 -> Cancella campo CLKT
: RESET_S
  302 BASE 804004 + ! ;

\ Ripristina FIFO utilizzando I2C_CONTROL_REGISTER (BASE 804000 +)
```

```
\ HEX (0x00000010) è (0000 0000 0000 0000 0000 0000 0001 0000) in BIN
\ Il bit 4 è 1 -> Cancella FIFO
: RESET_FIFO
  10 BASE 804000 + ! ;

\ Imposta l'indirizzo SLAVE 0x00000027 ( perché il nostro modello DRIVE è PCF8574T
)
\ in I2C_SLAVE_ADDRESS_REGISTER (BASE 80400C +)
: SET_SLAVE
  27 BASE 80400C + ! ;

\ Memorizza i dati in I2C_DATA_FIFO_REGISTER_ADDRESS (BASE 804010 +)
: STORE_DATA
  BASE 804010 + ! ;

\ Avvia un nuovo trasferimento utilizzando I2C_CONTROL_REGISTER (BASE 804000 +)
\ (0x00008080) è (0000 0000 0000 0000 1000 0000 1000 0000) in BINARIO
\ Il bit 0 è 0 -> Scrivi trasferimento pacchetti
\ Il bit 7 è 1 -> Avvia un nuovo trasferimento
\ Il bit 15 è 1 -> Il controller BSC è abilitato
: SEND
  8080 BASE 804000 + ! ;

\ La parola principale per scrivere 1 byte alla volta
: >I2C
  RESET_S
  RESET_FIFO
  1 BASE 804008 + !
  SET_SLAVE
  STORE_DATA
  SEND ;

\ Invia i 4 bit più significativi rimasti di TOS
: 4BM>LCD
  F0 AND DUP ROT
  D + OR >I2C 1000 DELAY
  8 OR >I2C 1000 DELAY ;

\ Invia 4 bit meno significativi rimasti di TOS
: 4BL>LCD
  F0 AND DUP
  D + OR >I2C 1000 DELAY
  8 OR >I2C 1000 DELAY ;

: >LCDL
 DUP 4 RSHIFT 4BL>LCD
 4BL>LCD ;

: >LCDM
```

```
  OVER OVER F0 AND 4BM>LCD
  F AND 4 LSHIFT 4BM>LCD ;

\ Per verificare che si tratti di un comando controllo se il 9 bit è 1 (lo shift
elimina gli otto precedenti)
: IS_CMD
  DUP 8 RSHIFT 1 = ;

\ Controlla se stiamo inviando un comando o un dato a I2C
\ Il comando rispetto al dato ha un 1 in più nel bit più significativo rispetto ai
\ dati. Un input come 101 >LCD sarebbe considerato un COMANDO per cancellare lo
\ schermo  0000 0001
\ altrimenti se 0 ( ad esempio un input come 41 >LCD ) sarebbe considerato un DATA
\ per inviare A CHAR (41 in esadecimale)
\ allo schermo
: >LCD
  IS_CMD SWAP >LCDM
;
```

lcd.f: In questo file sono definite le WORDS che, tramite successione di esadecimali corrispondenti a caratteri ASCII, costruiscono le stringhe di testo che verranno visualizzate a display, e quelle per intercettare i comandi all'LCD.

```
\ Embedded Systems - Sistemi Embedded - 17873)
\ LCD Setup paraole per la compilazione di messaggi su LCD 1602 )
\ Università degli Studi di Palermo )
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22 )

\ includere dopo i2c.f

\ Set di words che genera le parole da visualizzare su LCD
\ I due esadecimali rappresentano gli otto bit del blocco data per visualizzare la
lettera
\ così come devinira nella CGRAM
\ Stampa "welcome" a display
: WELCOME
    57 >LCD
    45 >LCD
    4C >LCD
    43 >LCD
    4F >LCD
    4D >LCD
    45 >LCD
    20 >LCD ;

\ Stampa "SMART" a display
: SMART
```

```
    53 >LCD
    4D >LCD
    41 >LCD
    52 >LCD
    54 >LCD
    20 >LCD ;

\ Stampa "CLIMA" a display
: CLIMA
    43 >LCD
    4C >LCD
    49 >LCD
    4D >LCD
    41 >LCD
    20 >LCD ;

\ Stampa "SYSTEM"  a display
  : SYSTEM
    53 >LCD
    59 >LCD
    53 >LCD
    54 >LCD
    45 >LCD
    4D >LCD
    20 >LCD ;

\ Stampa "READY" a display
: READY
    52 >LCD
    45 >LCD
    41 >LCD
    44 >LCD
    59 >LCD
    20 >LCD ;

\ Stampa "BUSY" a display
: BUSY
    42 >LCD
    55 >LCD
    53 >LCD
    59 >LCD
    20 >LCD ;

\ Stampa "LIGHT" a display
: LIGHT
    4C >LCD
    49 >LCD
    47 >LCD
    48 >LCD
```

```
    54 >LCD
    20 >LCD ;

\ Stampa "WIND" a display
: WIND
    57 >LCD
    49 >LCD
    4E >LCD
    44 >LCD
    20 >LCD ;

\ Stampa "STOP" a display
: STOP
    53 >LCD
    54 >LCD
    4F >LCD
    50 >LCD
    20 >LCD ;

\ Stampa "INSERT" a display
: INSERT
    49 >LCD
    4E >LCD
    53 >LCD
    45 >LCD
    52 >LCD
    54 >LCD
    20 >LCD ;

\ Stampa "TIME" a display
: TIME
    54 >LCD
    49 >LCD
    4D >LCD
    45 >LCD
    20 >LCD ;

\ Set di words che accede al blocco funzioni del LCD
\ Il bit più significativo impostato a 1 identifica una funzione anzicchè una dato
\ 0x80 → Indirizzo di memoria della DDRAM del display LCD per la linea 1
\ 0xC0 → Indirizzo di memoria della DDRAM del display LCD per la linea 2

\ Cancella il display
: CLEAR
  101 >LCD ;

\ Muove il cursore sulla seconda linea
: >LINE2
  1C0 >LCD ;
```

```
\ Visualizza il cursore sulla prima linea
: SETUP_LCD
  102 >LCD ;
```

led.f: In questo file sono assegnati i GPIO ai LED, e definite le WORDS per l'attivazione e la disattivazione degli attuatori.

```
\ Embedded Systems - Sistemi Embedded - 17873
\ Led Drive e Relè
\ Università degli Studi di Palermo
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

\ Includere dopo il flie gpio.f e ans.f

\ Il sistema led è composto da 3 led di colore diverso e da 3 resistori ceramici da
200 ohm che servono da protezione agli stessi led.
\ Ogni led è collegato ad un GPIO specifico che ne abilita l'emissione luminosa e
lo stesso GPIO gestisce il NC dell'attivazione di un relè.
\ I GPIO 5, 6, e il 12 (in HEX C) sono configurati come OUT.
5 CONSTANT YELLOWLED
6 CONSTANT REDLED
C CONSTANT GREENLED

\ Setup Led abilita i GPIO come output
: SETUP_LED
    REDLED GPFSELOUT!
    YELLOWLED GPFSELOUT!
    GREENLED GPFSELOUT! ;

\ Accende tutti i led disattivando tutti gli attuatori ( NC interdetto )
: ALL_LED_ON
  REDLED GPON!
  YELLOWLED GPON!
  GREENLED GPON!
;

\Questa WORD attiva il led giallo
: SYSTEMLIGHT YELLOWLED GPON! ;

\Questa WORD attiva il led verde
: SYSTEMWIND GREENLED GPON! ;

\Questa WORD disattiva un pin
: TURNOFF ( pin -- ) GPOFF! ;

\Questa WORD disattiva il led giallo
```

```
: STOPLIGHT YELLOWLED GPOFF! ;

\Questa WORD disattiva il led verde
: STOPWIND GREENLED GPOFF! ;

\ Variabili temporali
VARIABLE LIGHTIME
VARIABLE WINDTIME

\ Le variabili temporali vengono inizializzate a 1
1 LIGHTIME !
1 WINDTIME !
```

pad.f: In questo file sono descritte ed implementate le logiche per il controllo input dalla tastiera e il meccanismo per memorizzare il valore delle variabili temporali.

```
\ Embedded Systems - Sistemi Embedded - 17873
\ Keypad
\ Università degli Studi di Palermo
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

\ Includere dopo led.f

\ Per ogni riga inviare un output
\ Per ogni colonna controllare i valori
\ Se viene letto il bit di rilevamento dell'evento, abbiamo trovato il tasto
premuto
   \ nel formato RIGA-COLONNA

   \ MATRIX 5x4
\ GPIO-17 -> Riga-1 (F1-F2-#-*)
\ GPIO-18 -> Riga-1 (1-2-3-SU)
\ GPIO-23 -> Riga-2 (4-5-6-GIU)
\ GPIO-24 -> Riga-3 (7-8-9-ESC)
\ GPIO-25 -> Riga-4 (SX-0-DX-ENT)
\ GPIO-16 -> Colonna-1 (*-SU-GIU-ESC-ENT)
\ GPIO-22 -> Colonna-2 (#-3-6-9-DX)
\ GPIO-27 -> Colonna-3 (F2-2-5-8-0)
\ GPIO-10 -> Colonna-4 (F1-1-4-7-SX)

\ Abilita il rilevamento del fronte di discesa per i pin che controllano le RIGHE
   \ scrivendo 1 nelle posizioni dei pin corrispondenti (GPIO-18, 23, 24, 25)
\ HEX (0x03840000) che è (0000 0011 1000 0100 0000 0000 0000 0000) in BIN
: SETUP_ROWS
  3840000 GPFEN0 ! ;

\ I pin RIGA sono impostati come output , i pin colonna sono impostati come input
```

```
\ Il campo GPFSEL1 viene utilizzato per definire il funzionamento dei pin GPIO-10 -
GPIO-19
\ Il campo GPFSEL2 viene utilizzato per definire il funzionamento dei pin GPIO-20 -
GPIO-29

\ Ogni 3 bit di GPFSEL rappresenta un pin GPIO
\ Per indirizzare GPIO-10, GPIO-17, GPIO-16 e GPIO-18 dovremmo operare sulla
posizione dei bit
   \ 2-1-0(GPIO-10), X-Y-Z(GPIO-17), 20-19-18(GPIO-16) e 26-25-24(GPIO-18) che
memorizzano il valore in GPFSEL1

\ Per indirizzare GPIO-22, GPIO-23, GPIO-24, GPIO-25 e GPIO-27 dovremmo operare
sulla posizione dei bit
   \ 8-7-6(GPIO-22), 11-10-9(GPIO-23), 14-13-12(GPIO-24), 17-16-15(GPIO-25) e 23-
22- 21(GPIO-27)
   \ memorizzazione del valore in GPFSEL2

\ GPIO-17 settato in output -> 001
\ GPIO-18 settato in output -> 001
\ GPIO-23 settato in output -> 001
\ GPIO-24 settato in output -> 001
\ GPIO-25 settato in output -> 001
\ GPIO-16 settato in input -> 000
\ GPIO-22 settato in input -> 000
\ GPIO-27 settato in input -> 000
\ GPIO-10 settato in input -> 000

\ Di conseguenza dovremmo scrivere
\ (0001 0000 0000 0000 0000 0000 0000) in GPFSEL1_REGISTER_ADDRESS che è in
HEX(0x1000000)
\ (0000 1001 0010 0000 0000) in GPFSEL2_REGISTER_ADDRESS che è in HEX(0x9200)
: SETUP_IO
  1000000 GPFSEL1 @ OR GPFSEL1 !
  9200 GPFSEL2 @ OR GPFSEL2 ! ;

\ Cancella GPIO-17, GPIO-18, GPIO-23, GPIO-24 e GPIO-25 utilizzando il registro
GPCLR0
   \ scrivendo 1 nelle posizioni corrispondenti
\ HEX (0x3840000) che è (0011 1000 0100 0000 0000 0000 0000) in BIN
: CLEAR_ROWS
  3840000 GPCLR0 ! ;

\ Definizione della WORD per inizializzare la tastiera
: SETUP_KEYPAD
  SETUP_ROWS
  SETUP_IO
  CLEAR_ROWS ;

\ Testa un pin, se viene premuto lascia 1 sullo stack altrimenti 0
```

```
: PRESSED
  TPIN 1 = IF 1 ELSE 0 THEN ;

3 CONSTANT RANGE

\ Variabile gestisce la terminazione del ciclo.
VARIABLE FLAG
\ Variabile per memorizzare il tempo decine di secondi.
VARIABLE CAS
VARIABLE COS
\ Variabile per memorizzare il tempo unità di secondi.
VARIABLE CASS
VARIABLE COSS

1 FLAG !

\ Questo flag permette di gestire l'avvio del ciclo.
: FLAGOFF  0 FLAG ! ;

\ Questo flag permette di gestire l'arresto del ciclo.
: FLAGON  1 FLAG ! ;

\ Variabile Contatore
CREATE COUNTER

\ Incrementa di 1 la variabile COUNTER
: COUNTER++
  COUNTER @ 1 + COUNTER ! ;

 \ Memorizza il valore in decimale di un numero nell'array D_CMDS e lo emette su
LCD
\ Duplica il TOS ed emettilo
\ Lascia l'indirizzo D_CMDS su TOS
\ Lascia il valore COUNTER su TOS
\ Lasciare l'indirizzo del COUNTER'esimo indice dell'array D_CMDS su TOS
\ Infine memorizzare il valore DEC emesso a quell'indirizzo
\ Esempio: 30 EMIT_STORE -> Stampa 0 su LCD e lo memorizza in
D_CMDS[COUNTER_current_value]
: EMIT_STORE
  COUNTER @ 0 = IF LIGHT 1000 DELAY ELSE
  COUNTER @ 2 = IF >LINE2 WIND 1000 DELAY
  THEN THEN
  DUP 500 DELAY >LCD
  DUP 30 -  \ . CONSUMA LO STACK
  DUP .
\ Termina Programma con la pressione del tasto ESC
  DUP -15 = IF CLEAR ALL_LED_ON SYSTEM STOP 30000 DELAY ." EXIT TO END PROGRAM "
FLAGOFF CR AUTHOR CR CR EXIT ELSE
```

```
  DUP COUNTER @ 0 = IF DUP CAS ! DUP 4 LSHIFT LIGHTIME ! ELSE
  DUP COUNTER @ 1 = IF DUP CASS ! DUP LIGHTIME @ + LIGHTIME ! ELSE
  DUP COUNTER @ 2 = IF DUP COS ! DUP 4 LSHIFT WINDTIME !  ELSE
  DUP COUNTER @ 3 = IF DUP COSS ! DUP WINDTIME @ + WINDTIME !
  THEN THEN THEN THEN THEN
  ;

\ Stampa uno dei caratteri trovati nella Colonna 1 controllando il numero di riga
specificato con un ciclo condizionale
\ Pin fisico Riga -> EMIT-Colonna
\ Esempio: 12 EMTC1 stampa A (41 in HEX) su lcd
\ 19 EMTC1 stampa D (44 in HEX) su lcd
\ IL RIFERIMENTO DEL GPIO LO ABBIAMO IN HEX ES. GPIO17 = 11
: EMITC1
  DUP 11 = IF 2A DUP EMIT_STORE DROP ELSE
  DUP 12 = IF 5E DUP EMIT_STORE DROP ELSE
  DUP 17 = IF 5F DUP EMIT_STORE DROP ELSE
  DUP 18 = IF 1B DUP EMIT_STORE DROP ELSE
  19 = IF D DUP EMIT_STORE
  THEN THEN THEN THEN THEN ;

\ Stampa uno dei caratteri trovati sulla Colonna 2 controllando il numero di riga
specificato con un ciclo condizionale
\ Pin fisico Riga -> EMIT-Colonna
\ Esempio: 32 EMTC2 stampa # (23 in HEX) su lcd
\ 17 EMTC2 stampa 6 (36 in HEX) su lcd
\ IL RIFERIMENTO DEL GPIO LO ABBIAMO IN HEX ES. GPIO18 = 12
: EMITC2
  DUP 11 = IF 23 DUP EMIT_STORE DROP ELSE
  DUP 12 = IF 33 DUP EMIT_STORE DROP ELSE
  DUP 17 = IF 36 DUP EMIT_STORE DROP ELSE
  DUP 18 = IF 39 DUP EMIT_STORE DROP ELSE
  19 = IF 3E DUP EMIT_STORE
  THEN THEN THEN THEN THEN ;

\ Stampa uno dei caratteri trovati sulla Colonna 3 controllando il numero di riga
specificato con un ciclo condizionale
\ Pin fisico Riga -> EMIT-Colonna
\ Esempio: 18 EMTC2 stampa 8 (38 in HEX) su lcd
\ 19 EMTC2 stampa 0 (30 in HEX) su lcd
\ IL RIFERIMENTO DEL GPIO LO ABBIAMO IN HEX ES. GPIO17 = 11
: EMITC3
  DUP 11 = IF 25 DUP EMIT_STORE DROP ELSE
  DUP 12 = IF 32 DUP EMIT_STORE DROP ELSE
  DUP 17 = IF 35 DUP EMIT_STORE DROP ELSE
  DUP 18 = IF 38 DUP EMIT_STORE DROP ELSE
  19 = IF 30 DUP EMIT_STORE
  THEN THEN THEN THEN THEN ;
```

```
\ Stampa uno dei caratteri trovati sulla Colonna 4 controllando il numero di riga
specificato con un ciclo condizionale
\ Pin fisico Riga -> EMIT-Colonna
\ Esempio: 12 EMTC2 stampa 1 (31 in HEX) su lcd
\ 18 EMTC2 stampa 7 (37 in HEX) su lcd
\ IL RIFERIMENTO DEL GPIO LO ABBIAMO IN HEX ES. GPIO18 = 12
: EMITC4
  DUP 11 = IF 24 DUP EMIT_STORE DROP ELSE
  DUP 12 = IF 31 DUP EMIT_STORE DROP ELSE
  DUP 17 = IF 34 DUP EMIT_STORE DROP ELSE
  DUP 18 = IF 37 DUP EMIT_STORE DROP ELSE
  19 = IF 3C DUP EMIT_STORE
  THEN THEN THEN THEN THEN ;

\ Stampa la combinazione di caratteri Riga-Colonna specificata utilizzando la
corrispondente WORD EMTC1/C2/C3/C4
\ Esempio: 12 10 EMIT_R
: EMIT_R
  DUP 10 = IF DROP EMITC1 ELSE
  DUP 16 = IF DROP EMITC2 ELSE
  DUP 1B = IF DROP EMITC3 ELSE
  A = IF EMITC4
  THEN THEN THEN THEN ;

\ Verifica se un tasto della riga data è premuto, attende il suo rilascio,
\ e stampa il valore esadecimale corrispondente sull'LCD
: CHECK_CL
  DUP DUP
    PRESSED 1 = IF 1000 DELAY
    PRESSED 0 = IF 1000 DELAY
      EMIT_R
      COUNTER++
    ELSE DROP DROP
    THEN
    ELSE DROP DROP DROP
  THEN ;

\ TODO
\ Controlla la riga data impostandola su HIGH, controllandone le colonne e infine
impostandola su LOW
\ Esempio -> 32 CHECK_ROW (Controlla la prima riga)
\         -> 12 CHECK_ROW (Controlla la seconda riga)
\         -> 17 CHECK_ROW (Controlla la terza riga)
\         -> 18 CHECK_ROW (Controlla la quarta riga)
\         -> 19 CHECK_ROW (Controlla la quinta riga)
: CHECK_ROW
  DUP DUP DUP DUP DUP
  HIGH
    10 CHECK_CL
```

```
    16 CHECK_CL
    1B CHECK_CL
    A CHECK_CL
  LOW ;

: ?CTR
  COUNTER @ 4 = ;

: RES_CTR
  0 COUNTER ! ;

: ?CTF
  FLAG @ 0 = ;

\ La main WORD per rilevare qualsiasi evento di PRESS/RELASE ed eventualmente
stampa il
\ carattere corrispondente su LCD
\ Questa WORD deve essere chiamata all'avvio del SETUP,
\ quindi, a meno che non si impostino le righe su LOW, non è necessario utilizzare
questa WORD
: DETECT
  CLEAR
  0 COUNTER !
  BEGIN
    11 CHECK_ROW
    12 CHECK_ROW
    17 CHECK_ROW
    18 CHECK_ROW
    19 CHECK_ROW
  ?CTR UNTIL
  0  CR \ LIGHTIME @ . ." <- LIGHTIME " CR WINDTIME @ . ." <- WINDTIME " CR
  ." RUN SYSTEM . . . " CR
  ." SETTING TIME LIGHT >>>    "  CAS @ . CASS @ . ."    SECONDS " CR
  ." SETTING TIME WIND >>>    "  COS @ . COSS @ . ."    SECONDS " CR CR ;
```

hdmi.f: In questo file sono descritti i parametri di funzionamento del framebuffer la definizione delle immagini da visualizzare nei vari stati del sistema.

```
\ Embedded Systems - Sistemi Embedded - 17873)
\ Gestione dell'uscita HDMI del FrameBuffer)
\ Università degli Studi di Palermo )
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22 )

\ includere dopo pad.f

HEX
```

```
\ IMPOSTAZIONI  DEL FRAMEBUFFER SU INTERPRETE
\ Initializing frame buffer
\ framebuffer at 0x3E8FA000 width 00000400 height 00000300 depth 00000020 pitch
00001000

\ Svuota tutto
: 4DROP DROP DROP DROP DROP ;

\ Dichiarazione dei colori in esadecimale (formato ARGB)
00FFFFFF CONSTANT WHITE
00000000 CONSTANT BLACK
00FF0000 CONSTANT RED
00FFFF00 CONSTANT YELLOW
0000FF00 CONSTANT GREEN
000000FF CONSTANT BLUE

\ Dichiarazione del base address del framebuffer ( definito nel kernel )
3E8FA000 CONSTANT FRAMEBUFFER

VARIABLE DIM

\ Contatore utilizzato nei cicli per disegnare le linee orizzontali
VARIABLE COUNTERH


\ RESETCOUNTERH
: RESETCOUNTERH 0 COUNTERH ! ;

\ Contatore utilizzato nei cicli per tenere traccia del numero di riga corrente
: +COUNTERH COUNTERH @ 1 + COUNTERH ! ;

\ RESETNLINE
\ Restituisce l'indirizzo del punto centrale dello schermo. Coordinata ((larghezza-
1)/2, (altezza-1)/2)
VARIABLE NLINE
: RESETNLINE 1 NLINE ! ;
: +NLINE NLINE @ 1 + NLINE ! ;

( -- addr )
: CENTER FRAMEBUFFER 200 4 * + 180 1000 * + ;
\ Colora, con il colore presente sullo stack, il pixel corrispondente all'indirizzo
\ presente sullo stack,
\ dopodichè punta al pixel a destra

( color addr -- color addr_col+1 )
: RIGHT 2DUP ! 4 + ;

\ Colora, con il colore presente sullo stack, il pixel corrispondente all'indirizzo
\ presente sullo stack,
```

```
\ dopodichè punta al pixel in basso

( color addr -- color addr_row+1 )
: DOWN 2DUP ! 1000 + ;

\ Colora, con il colore presente sullo stack, il pixel corrispondente all'indirizzo
\ presente sullo stack,
\ dopodichè punta al pixel a sinistra

( color addr -- color addr_col-1 )
: LEFT 2DUP ! 4 - ;

\ Ripristina il valore di partenza dell'indirizzo a seguito di COUNTERH * 4
spostamenti a
\ destra

( addr_endline_right -- addr )
: RIGHTRESET COUNTERH @ 4 * - ;

\ Ripristina il valore di partenza dell'indirizzo a seguito di COUNTERH * 4
spostamenti a
\ sinistra

( addr_endline_left -- addr )
: LEFTRESET COUNTERH @ 4 * + ;


\ Disegna una linea verso destra di dimensione pari a 48 pixel
: RIGHTDRAW
    BEGIN COUNTERH @ DIM @ < WHILE +COUNTERH RIGHT REPEAT RIGHTRESET RESETCOUNTERH
;

\ Disegna una linea verso sinistra di dimensione pari a 48 pixel
: LEFTDRAW
    BEGIN COUNTERH @ DIM @ < WHILE +COUNTERH LEFT REPEAT LEFTRESET RESETCOUNTERH ;


\ Disegna o il simbolo di stop o pulisce la porzione di schermo su cui disegnamo.
\ Partendo da CENTER-32px-48px, quindi CENTER-80px, e poichè ogni spostamento di
1px su
\ una riga
\ vale 4, abbiamo 320 in dec e cioè 140 in hex
: DRAWSQUARE
    80 DIM !
    CENTER 140 - RIGHTDRAW
\ Ciclo che disegna un simbolo di stop di altezza 105 pixel
    BEGIN NLINE @ 70 <
        WHILE
            DOWN RIGHTDRAW
```

```
            +NLINE
        REPEAT
    2DROP RESETNLINE
;

\ Disegna il simbolo di start.
: DRAWSTARTWIND
    GREEN CENTER 80 -
    BEGIN NLINE @ 70 <=
        WHILE
\ Permette di rappresentare le linee del triangolo superiore del simbolo start
            NLINE @ 37 <= IF
            NLINE @ DIM !
\ Permette di rappresentare le linee del triangolo inferiore del simbolo start
                ELSE
                    70 NLINE @ - DIM !
            THEN
\ Disegna una linea verso destra di dimensione variabile e dipendente dal numero di
riga \ memorizzato in NLINE.
            DOWN RIGHTDRAW
            +NLINE
        REPEAT
    2DROP RESETNLINE
;

\ Disegna il simbolo di start.
: DRAWSTARTLIGHT
    YELLOW CENTER 80 -
    BEGIN NLINE @ 70 <=
        WHILE
\ Permette di rappresentare le linee del triangolo superiore del simbolo start
            NLINE @ 37 <= IF
            NLINE @ DIM !
\ Permette di rappresentare le linee del triangolo inferiore del simbolo start
                ELSE
                    70 NLINE @ - DIM !
            THEN
\ Disegna una linea verso destra di dimensione variabile e dipendente dal numero di
riga \ memorizzato in NLINE.
            DOWN RIGHTDRAW
            +NLINE
        REPEAT
    2DROP RESETNLINE
;

: DRAWSTOP RED DRAWSQUARE ;

: CLEAN BLACK DRAWSQUARE ;
```

```
\ Disegna la bandiera
: DRAWITAFLAG

    WHITE DRAWSQUARE
    30 DIM !
\ Disegna la prima linea della seconda pipe
    RED CENTER RIGHTDRAW
\ Disegna la prima linea della prima pipe, spostandosi di 32 pixel a sinistra
    GREEN CENTER 80 - LEFTDRAW
\ Ciclo che disegna due pipe, distanziate, di altezza 105 pixel
    BEGIN NLINE @ 70 <
        WHILE
\ Disegna l'n-esima linea della prima pipe
            DOWN LEFTDRAW
            2SWAP
\ Disegna l'n-esima linea della seconda pipe
            DOWN RIGHTDRAW
            2SWAP
            +NLINE
        REPEAT
    4DROP RESETNLINE
;
```

timer.f: Il file contiene il codice per la gestione del registro CLO che ci permette di effettuale un conteggio in secondi utilizzando un campione " secondo" in esadecimale.

```
\ Embedded Systems - Sistemi Embedded - 17873)
\ Timer di Sistema e definizione di unità temporale )
\ Università degli Studi di Palermo )
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22 )

\ includere dopo hdmi.f

HEX

\ Clock Register
BASE 003004 + CONSTANT CLO

\ Dichiarazione della costante che indica un secondo e che ha valore 1 000 000 usec
in decimale o
\ F4240 in hex
F4240 CONSTANT SEC

\ Variabile che memorizza il valore attuale del CLO + 1 secondo
VARIABLE COMP0

VARIABLE TIME_COUNTER
```

```
0 TIME_COUNTER !

\ Inizializzazione delle variabili utilizzate
0 COMP0 !

\ Restituisce il valore attuale del registro CLO
( -- clo_value )
: NOW CLO @ ;
\ Setta un ritardo corrispondente al valore presente sullo stack
( delay_sec -- )
: DELAY_SEC NOW + BEGIN DUP NOW - 0 <= UNTIL DROP ;

\ Converte il valore decimal in secondi a hexadecimal per darlo in pasto al tic. La
word pone sullo stack il resto
\ e il quoto della divisione per 10 moltiflica il quoto per A (10 in HEX) somma il
resto ed effettua successivamente uno swap.
: PARSE_DEC_HEX ( n1 -- n3 n2 ) 10 /MOD A * SWAP + DUP . ." >> SECONDS " ;

\ Memorizza il valore attuale del CLO + 1 secondo in COMP0
: INC NOW SEC + COMP0 ! ;


\ Segnala ogni qual volta e passato un secondo confrontando CLO con COMP0

: SLEEPS DECIMAL INC BEGIN NOW COMP0 @ < WHILE REPEAT DUP U. ." | " DROP DECIMAL ;

\ Words di incremento e decremento contatore
: DECCOUNT TIME_COUNTER @ 1 - TIME_COUNTER ! ;
: INCCOUNT TIME_COUNTER @ 1 + TIME_COUNTER ! ;

\ Word che imposta un conto alla rovescia in secondi a partire dal n passato fino a
zero.

DECIMAL : TIMER PARSE_DEC_HEX TIME_COUNTER ! CR begin TIME_COUNTER @ SLEEPS
DECCOUNT TIME_COUNTER @ 0 = until CR
 ." END EROGATION " CR CR DROP ;

HEX
```

button.f: Questo sorgente contiene le word per implementare il comportamento del pulsante di accensione asincrono.

```
\ Embedded Systems - Sistemi Embedded - 17873)
\ Bottone POWER ON )
\ Università degli Studi di Palermo )
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22 )
```

```
\ includere dopo timer.f

\ Word per il controllo del bottone d'avvio

HEX

\ Impostiamo il GPIO26 come output ( 001 ) sul bit 20:18 con 40000 in HEX a cui è
associati il pulsante di accenzione.
: SETUP_BUTTON
  40000 GPFSEL2 @ OR GPFSEL2 !
;

\ I registri di abilitazione del rilevamento del fronte di salita asincrono
definiscono i pin per i quali è verificata la presenza di
\ un fronte di salita asincrono: la transizione imposta un bit nei registri di
stato di rilevamento eventi (GPEDSn).
\ Asincrono significa che il segnale in ingresso non è campionato dall'orologio di
sistema. Tali fronti di salita si possono verificarsi
\ in tempi molto brevi, in qualunque istante ed è possibile rilevare la durata.

\ Rendiamo sensibile ai fronti di salita il GPIO26 a cui è associati il pulsante,
quindi
\ avremo 1000 0000 0000 0000 0000 0000 0000 che equivale a 4000000 in decimale o
0x63 in esadecimale
\ l'OR logico ci permette di aggiornare e di non cancellare precedenti settaggi.

: GPAREN! 4000000 GPAREN0 @ OR GPAREN0 ! ;
```

main.f: Questo è il file che contiene le WORDS in linguaggio comprensivo che permettono di inizializzare il sistema **SETUP**, di avviare il ciclo principale **RUN**, oltre a quelle che descrivono i vari comportamenti. Alla fine del caricamento viene lanciata la word **START** che mette il sistema in ascolto con un ciclo infinito: alla pressione del tasto **POWER ON** il sistema si avvia.

```
\ Embedded Systems - Sistemi Embedded - 17873)
\ Main Words )
\ Università degli Studi di Palermo )
\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22 )

\ includere dopo tutti gli altri *.f


HEX
\ Aziona il Sistema Illuminazione
```

```
: GO_LIGHT
  REDLED GPOFF!
  STOPWIND
  CLEAR
  CLEAN
  DRAWSTARTLIGHT
  SYSTEM
  LIGHT
  SYSTEMLIGHT
  ;

\ Aziona il Sistema Ventilazione
: GO_WIND
  REDLED GPOFF!
  STOPLIGHT
  CLEAR
  CLEAN
  DRAWSTARTWIND
  SYSTEM
  WIND
  SYSTEMWIND
  ;

\ Sistema Arrestato
: STOP_DISP
  ALL_LED_ON
  CLEAR
  CLEAN
  DRAWSTOP
  SYSTEM
  STOP
  ;

\ Esecuzione attuatori per 4 cicli. Il Flag di fine procedura viene modificato al 4
ciclio.
\ Al termine il programma si rimette in configurazione d'immissione dati.
: RUN 0 COUNTER !
  BEGIN
    FLAG @ 1 = WHILE
      GO_LIGHT ." SYSTEM LIGHT "  LIGHTIME @ TIMER
      GO_WIND ." SYSTEM WIND " WINDTIME @ TIMER
      COUNTER++
      COUNTER @ 4 = IF  FLAGOFF THEN
      ." Cycle n. "
      COUNTER @ .
      ." Flag setting "
      FLAG @ .
      CR CR
    REPEAT
```

```
  ?CTF UNTIL FLAGON STOP_DISP 10000 DELAY ." END AUTOMATIC CYCLE " CR CLEAR INSERT
TIME 10000 DELAY ; \ Riutilizzo di flag per gestire il ciclo principale.


\ Main WORD che contiene settaggi di base e avvio del ciclo principale:
\ Vengono avviati i setup per il Bus I2C per inizializzare l'LCD, la Keypad, led e
gli attuatori.
\ A questo punto parte il messaggio di benvenuto e inizia il ciclo infinito:
\ Sarà necessario introdurre il tempo di esecuzione degli adattatori espresso in
secondi ( 2 cifre per illuminazione e 2 cifre per il vento);
: SETUP
  CLEAN
  DRAWITAFLAG
  SETUP_I2C
  SETUP_LCD
  SETUP_KEYPAD
  SETUP_LED
  CLEAR
  WELCOME
  >LINE2
  SMART
  CLIMA
  50000 DELAY
  CLEAR
  STOP_DISP
  20000 DELAY
  CLEAR
  INSERT TIME
  40000 DELAY
    BEGIN
    FLAGON
    DETECT
    1
    RUN 20000 DELAY
    ?CTF UNTIL
    ;


\ Solo setup Hardware per testing.
  : ONLY_SETUP
  CLEAN
  DRAWITAFLAG
  SETUP_I2C
  SETUP_LCD
  SETUP_KEYPAD
  SETUP_LED
  CLEAR
  WELCOME
  >LINE2
```

```
  SMART
  CLIMA
  30000 DELAY
  CLEAR
  STOP_DISP
  ;

\ Viene letto il valore del registro GPEDS0, che se ha valore 4000000 ( ovvero
rilevato fronte sul GPIO26 ) fa partire il setup.
: POWER_ON GPEDS0 @ 4000000 = IF ." PREMUTO TASTO AVVIO DEL SISTEMA " CR SETUP THEN
0 GPEDS0 ! ;

\ Ciclo d'avvio che mette il sistema in ascolto della premuta del tasto di
accensione asincrono
: START
    SETUP_BUTTON
    GPAREN!
    BEGIN
      WHILE
        POWER_ON
      REPEAT
    0 GPEDS0 !
;

START
```

Testing

GPIO
DISPLAY
RUN GLOBAL_EN
FCC ID: 2ABCB-RPI4B
IC: 20953-RPI4B
Made in the UK
HDMI
CAMERA

Passiamo alla configurazione del TTY in ZOS8 impostando la connessione seriale come da immagini:

È possibile fare la scansione delle porte com disponibili per trovare quella alla quale è collegata la nostra interfaccia UART.

Settiamo anche i paramentri per le line del terminale Carrage Return e Line Feed.

Session Profile (Standard.zoc)
Connection Type
Terminal
Layout/Font
Colors
Emulations
Text Sending
Logging
Window Parts
User Buttons
Keyboard
Auto-Action
Auto-Highlight
File Transfer
File Handling
Trace/Debug
Advanced
Terminal
Terminal-Parameters
Local echo (half duplex)
Don't CR/LF for local echo
Incoming end of line -> CR/LF
Soft Shift-Lock for letters
Scroll jumps:
Frog
Advanced
Answer to ENQ
ZOC^9
Disable URL highlighting
Send echo (host mode)
Strip high bit from incoming data
Software 7E1
Ignore remote beep
Ignore character translation profile
Cursor Appearance
Format:
Block
Mode:
Blinking
Idle Timer
After
0
sec. idle (0=disable)
Send
^@
Help
Save As...
Save
Cancel

Per la programmazione una volta avviato il terminale e attivata la connessione con l'interfaccia seriale, possiamo dare alimentazione al Raspberry PI 4.

```
[SERIAL/DIRECT] CONNECTED TO PORT COM10 (115200-8N1)
```

Dopo pochi secondi comparirà la console a riga di comando dell'interprete.

```
[SERIAL/DIRECT] CONNECTED TO PORT COM10 (115200-8N1)


PURE324ES
Palermo University 32 bit Research Environment for Embedded Systems
(2022) - Daniele Peri
v. 202201182136
PERI BASE FE000000
TIME      0000000000593BC8
CPUID     410FD083
CPSR      000001DA
BOARDREV  00B03114
Board model       0x00000000
Board revision    0x00B03114
Board MAC address 0x0000E45F017EDB5F
Board serial      0x1000000093DE1DA8
ARM memory base address 0x00000000, size 0x3B400000
 VC memory base address 0x3B400000, size 0x04C00000
Clock rates (Hz):
UART:    48000000
CPU:     600000000
CPU MAX: 1500000000
CORE:    500000000
ISP:     250000000
Initializing frame buffer
framebuffer at 0x3E8FA000 width 00000400 height 00000300 depth 00000020 pitch 00001000

Changing clock rate
clock max rate: 1500000000, clock current rate: 1500000000
Clock rates (Hz):
UART:    48000000
CPU:     1500000000
CPU MAX: 1500000000
CORE:    500000000
ISP:     500000000
pijFORTHos 0.1.8-se-20220113 sp=0x3B400000
```

Dalla barra dei pulsanti selezioniamo il caricamento di un file sorgente.

Si aprirà la selezione file e a questo punto carichiamo il nostro final.f.

Il codice verrà inviato al target alla velocità stabilita nei parametri di connessione.

Terminato il caricamento viene lanciata automaticamente la WORD “START” che mette il sistema in ascolto alla pressione del tasto POWER ON

All'inizio il sistema sarà in fase di SYSTEM STOP e quindi sarà acceso il led rosso e tutti gli attuatori saranno disattivati.

A questo punto si potrà digitare dalla tastiera il valore di temporizzazione espresso in secondi da 0 a 99, con due cifre decimali per il tempo di illuminazione e altre due per quello di ventilazione.

D'ora in avanti il sistema entra in un ciclo composto da quattro scambi di alternanza tra le funzioni. Il tempo è scandito in secondi.

```
START
PREMUTO TASTO AVVIO DEL SISTEMA
1 0 1 1
RUN SYSTEM . . .
SETTING TIME LIGHT >>>    1 0    SECONDS
SETTING TIME WIND >>>    1 1    SECONDS

SYSTEM LIGHT A >> SECONDS
10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
END EROGATION

SYSTEM WIND 11 >> SECONDS
11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
END EROGATION

Cycle n. 1 Flag setting 1

SYSTEM LIGHT 10 >> SECONDS
10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
END EROGATION

SYSTEM WIND 11 >> SECONDS
11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
END EROGATION

Cycle n. 2 Flag setting 1

SYSTEM LIGHT 10 >> SECONDS
10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
END EROGATION
```

Quando sarà in fase di SYSTEM LIGHT sarà acceso il led giallo e il sistema di illuminazione sarà in funzione.

Quando sarà in fase di SYSTEM WIND sarà acceso il led verde e il sistema di ventilazione sarà in funzione.

In caso di anomalia o manutenzione può essere premuto il tasto ESC per interrompere il programma.

Conclusioni

Nella realizzazione di questo progetto le difficoltà iniziali sono state legate al fatto di avere a che fare con componenti che solitamente siamo già messi in condizione di utilizzarli senza troppi fronzoli. In realtà quando si crea un software che deve descrivere i comportamenti di un hardware così a basso livello si comincia ad apprezzare in maniera concreta la validità dello stesso.

Questo progetto potrebbe evolversi: ad esempio si potrebbero applicare numerosi sensori ed attuatori per rendere veramente autonomo lo stesso sistema; come dei meccanismi di irrigazione controllati, piuttosto che dei sensori crepuscolari o altro.

Potrebbe benissimo trovare applicazione in altri ambiti semplicemente modificando gli attuatori e le WORDS di controllo degli stessi.

Inoltre la programmazione così gestita "*a blocchi*" rende riutilizzabile il codice anche su altri target con le dovute pre-configurazioni.

Davide Proietto