**Embedded System**

**: Smart Clima ;**



**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**

**d**

Sommario

[1](#_Toc95566912)

[**Descrizione del progetto** 4](#_Toc95566913)

[**Prerequisiti concettuali** 4](#_Toc95566914)

[**Componenti Hardware** 4](#_Toc95566915)

[**Schema del sistema** 5](#_Toc95566916)

[**Schema di collegamento Header** 5](#_Toc95566917)

[**Il targhet: RaspBerry PI 4 B** 6](#_Toc95566918)

[**Componenti Software** 6](#_Toc95566919)

[**Preparazione dell’ambiente di sviluppo** 6](#_Toc95566920)

[**Preparazione della SD e dell’interprete** 7](#_Toc95566921)

[**Descrizione dei componenti** 9](#_Toc95566922)

[***FTDI 232-USB Interfaccia UART*** 9](#_Toc95566923)

[***Modulo LCD 1602 con Drive I^2C PCF85741/3*** 9](#_Toc95566924)

[**Tastierino KeyPad a matrice 5x4** 12](#_Toc95566925)

[**Modulo relay HL-52s** 13](#_Toc95566926)

[**Sistema LED** 14](#_Toc95566927)

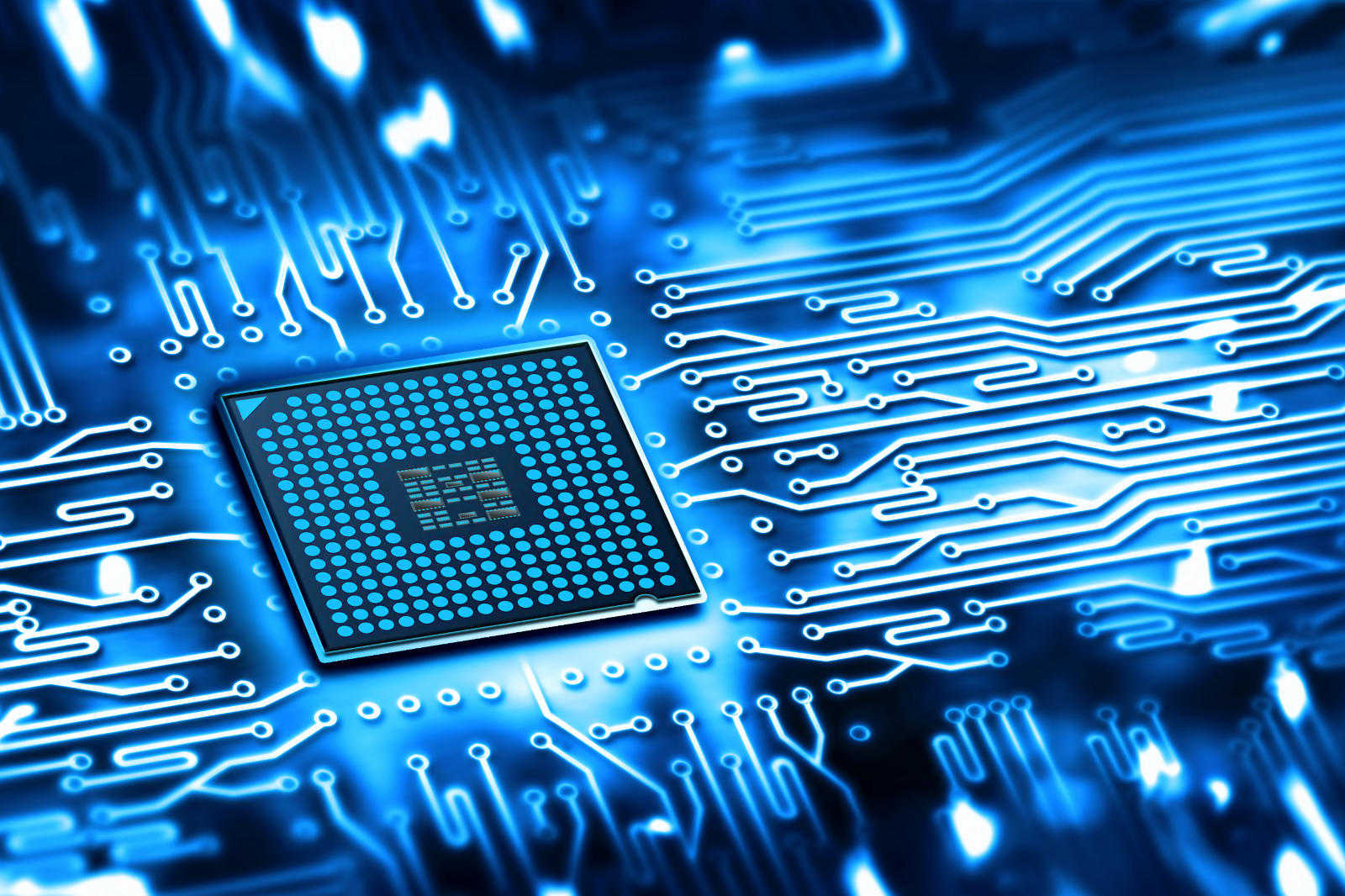
[**Flusso degli eventi** 15](#_Toc95566928)

[**Il codice** 15](#_Toc95566929)

[**Dettaglio files sorgenti** 15](#_Toc95566930)

[**Testing** 36](#_Toc95566931)

[**Conclusioni** 41](#_Toc95566932)



## **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**”, 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, 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. 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:

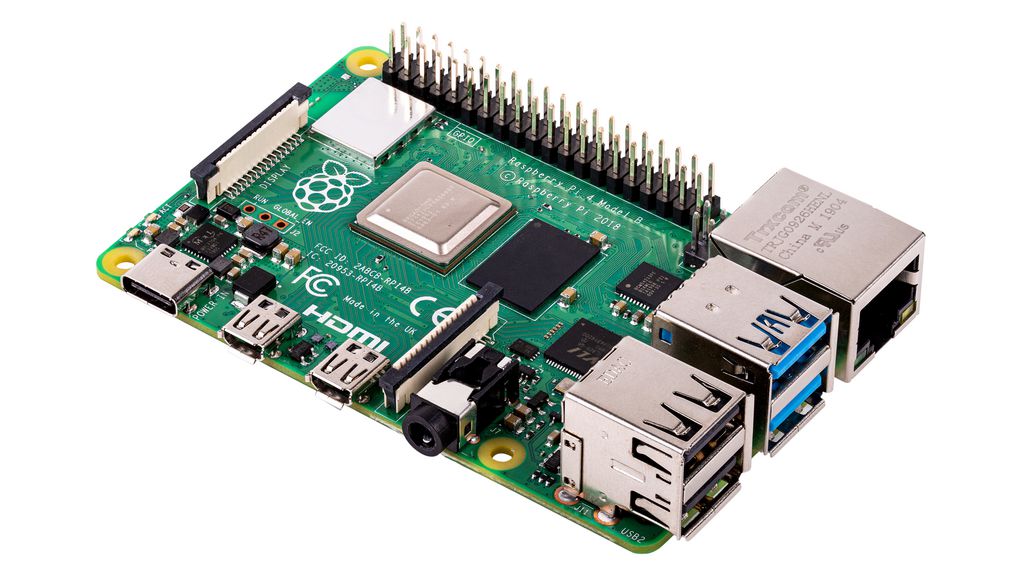
* Raspberry Pi 4B 2 GB con alimentatore;
* MicroSD 16 GB;
* FT232RL USB Interfaccia Seriale UART;
* Sharp TC1602B-01 VER:00 16x2 LCD BLU;
* Philips PCF8574AT Remote 8-bit I/O espansione per I2C-bus;
* Relay Module HL-52s;
* Lampada piatta Led 10 W 220v con interruttore;
* Ventola 12V 15\*15 cm;
* 5x4 Keypad Matrix;
* BreadBoard e Cavi connessione M/F e M/M;
* 3 Led vari colori;
* 3 Resistori ceramidi da 200 ohm;
* Personal Computer;
* Cavo USB – miniUSB M/M;
* Vaso, terra, piante e rivestimento pellicola.

## **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 |  |
|  |  |  | **37** | **38** |  |  |  |
|  |  |  | **39** | **40** |  |  |  |

## **Il targhet: RaspBerry PI 4 B**

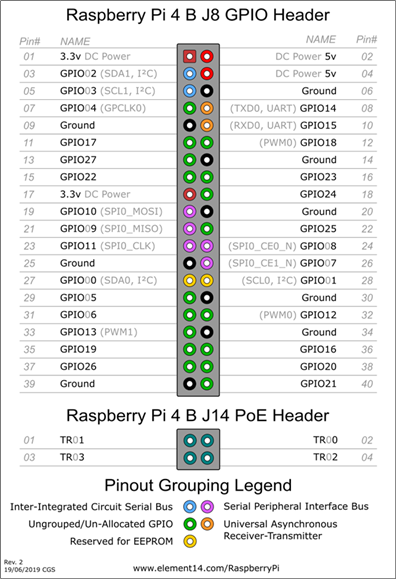
****Il Raspberry Pi 4 Model B monta un microcontrollore ARM Broadcom BCM2711 quad-core Cortex-A72 a 1.5 Ghz.

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**

• Distro linux ( Mint 20.3 LTS ) con installato G-Forth, Minicom e Picocom;

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

* 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.

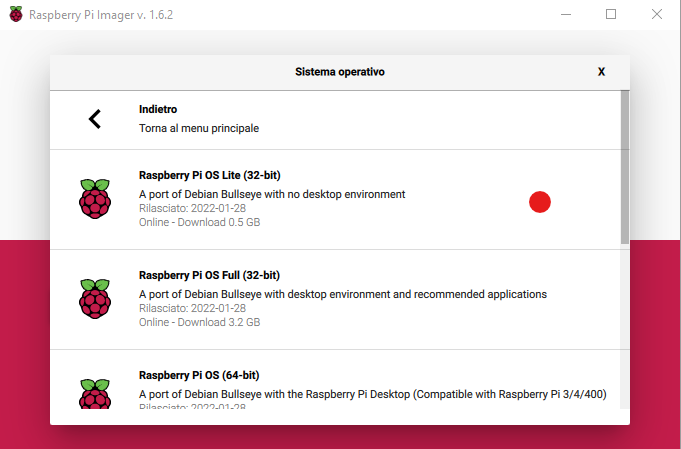
A tal scopo dobbiamo installare dei tool per la comunicazione e l’interprete Forth per comodità. Da terminale:

* 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

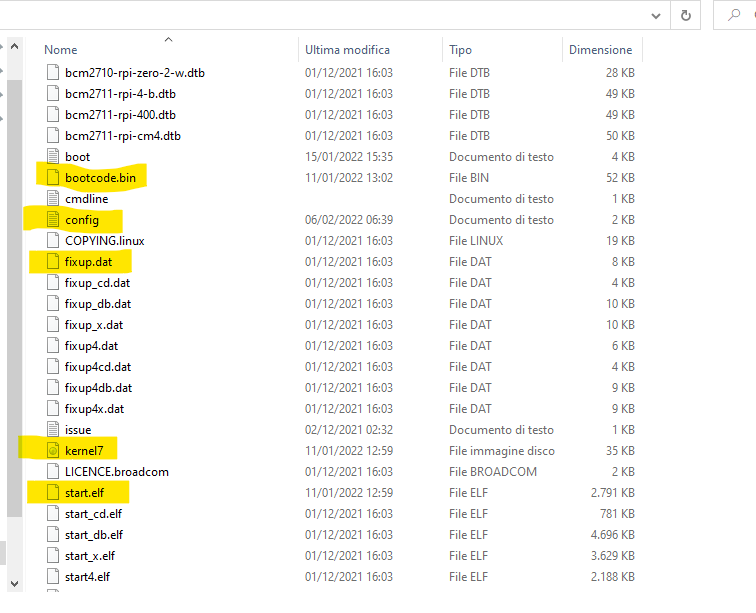
## **Preparazione della SD e dell’interprete**

****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.

I files bootcode.bin fixup.dat dobbiamo utilizzare quello generato 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

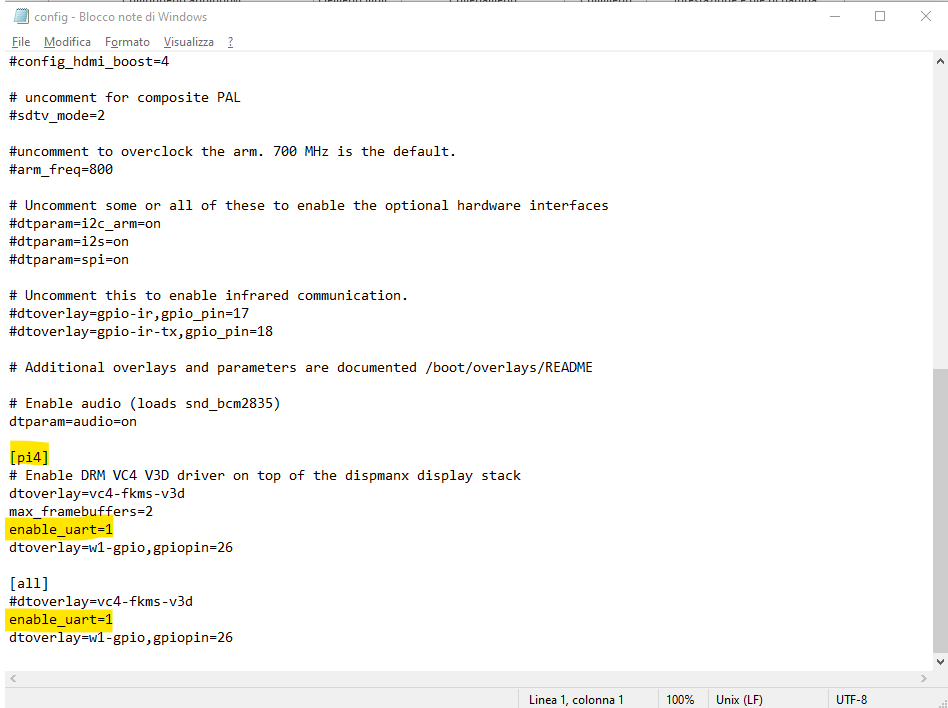
presenti nel repo <https://github.com/organix/pijFORTHos> ( o versione modificata prof D. Peri ).

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. Il 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> .

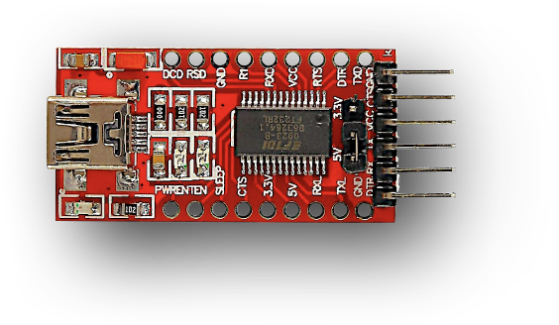
Per rendere possibile la comunicazione con l’interprete Forth con l’interfaccia di programmazione dobbiamo abilitare la UART del PI.

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

*****enable\_uart=1*

## **Descrizione dei componenti**

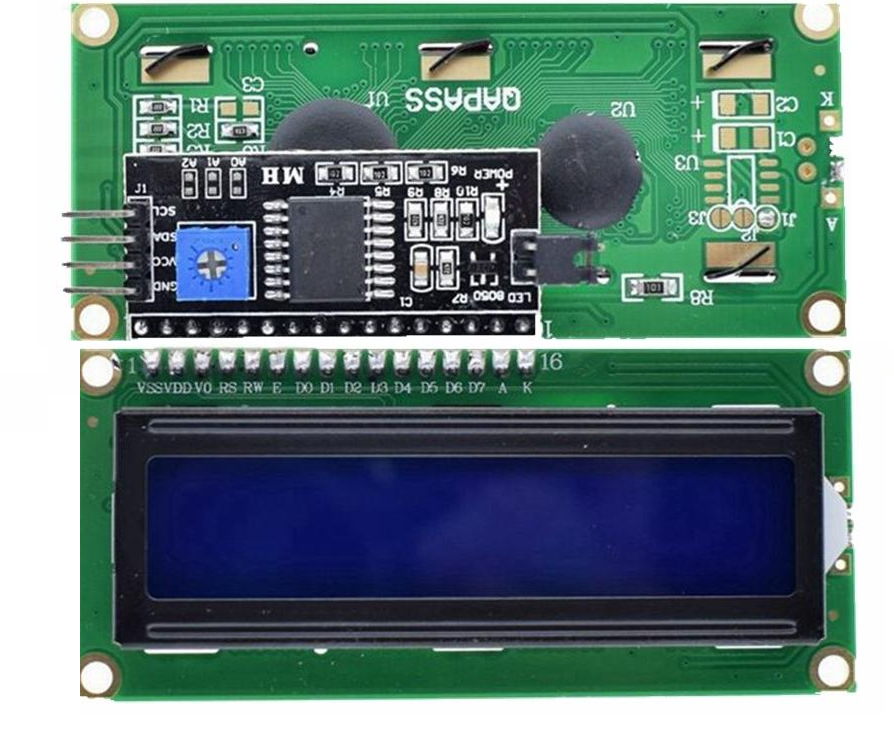
## **FTDI 232-USB Interfaccia UART**

 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 fornite la linea di l’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

|  |  |
| --- | --- |
| **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.

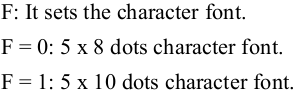
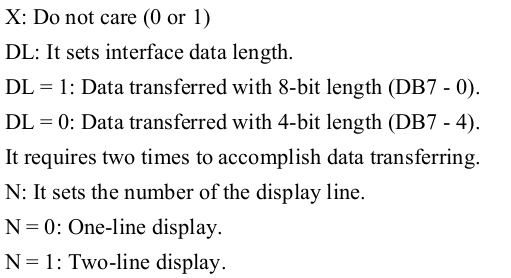
Usando il meccanismo di Open-Drain per la comunicazione bidirezionale possiamo trasferire con un minimo tirando ottimizzando l’uso del canale ovvero lasciandolo “fluttuante”, grazie al resistore di pull-up.

Descrizione del funzionamento del bus I2C:

* Il master inizia la comunicazione inviando:
  + - Start Condition
    - L’ indirizzo slave (7 bit)
    - 0 per la scrittura (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 definisce una “Start Condition” condizione d’avvio, e alla transizione da basso ad alto sulla linea SDA 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.

[](https://user-images.githubusercontent.com/33685811/125796047-7c024fdb-3333-44a1-b8ce-67d84374d436.png)[](https://user-images.githubusercontent.com/33685811/125796435-ea23d4fa-6e44-44d9-b65b-2a2e8b13f876.png)[](https://user-images.githubusercontent.com/33685811/125796234-cc9f899c-035d-4a70-87cd-5848cb6b4f32.png)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.

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 parola **>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. 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

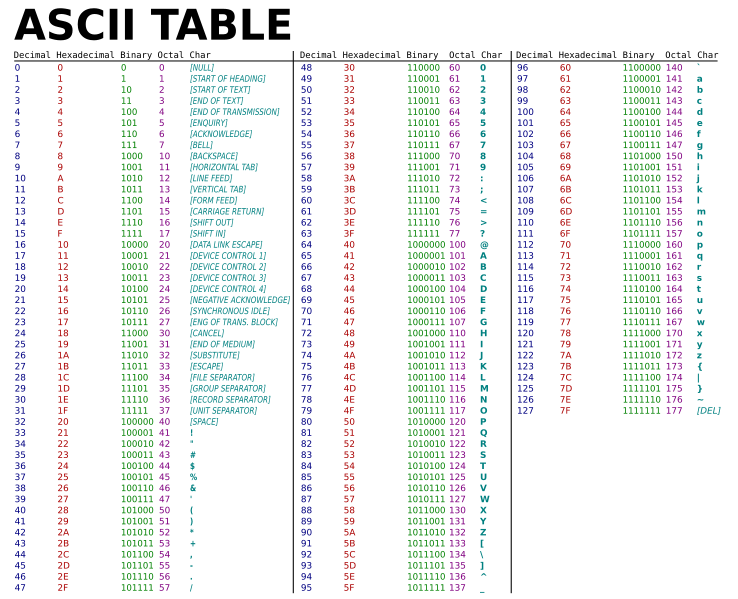
D7=0, D6=0, D5=1, D4=0, Retroilluminazione=1, Abilita=1, R/W’=0, RS=0

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, in per inviare 1 byte a LCD dobbiamo scrivere 4 volte sul bus I2C:

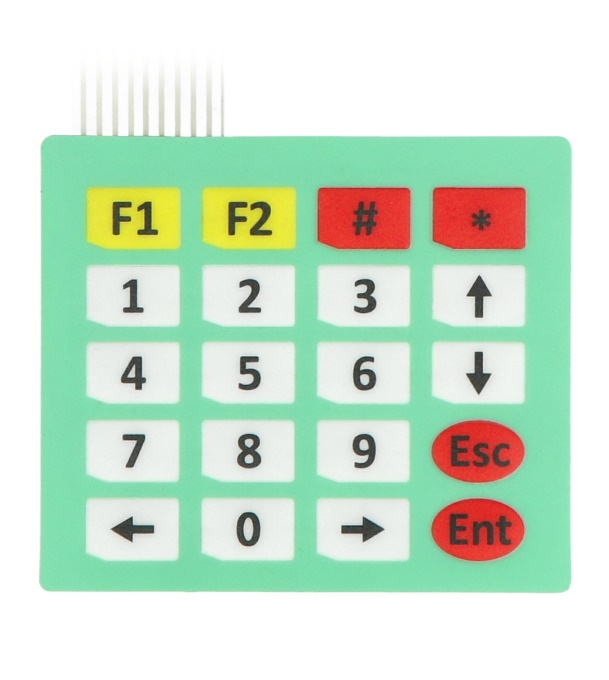
* 4 bit più significativi con Enable = 1
* 4 bit più significativi con Enable = 0
* 4 bit meno significativi con Enable = 1
* 4 bit meno significativi con Enable = 0

Dopo questa configurazione si può inviare qualsiasi carattere ASCII digitando il suo codice HEX e chiamando >LCD word; ad esempio 45 >LCD (che invia A all'LCD).

## **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.

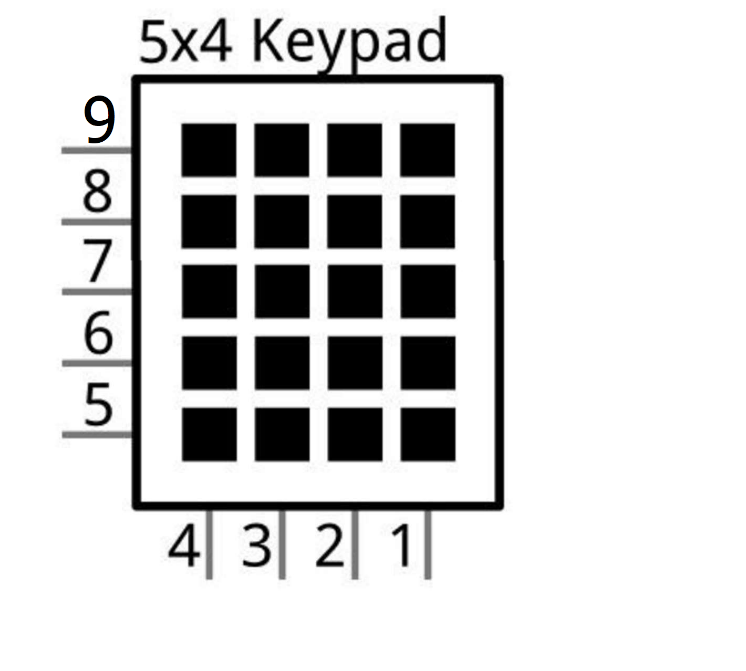
A tale scopo si configurano:

* ****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

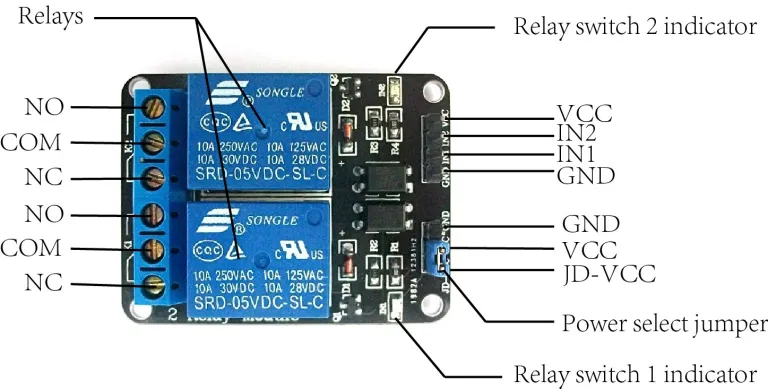
Ad ogni iterazione

* Impostiamo HIGH sulla riga che vogliamo controllare utilizzando il registro GPSET corrispondente
* Premere 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 questi 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.



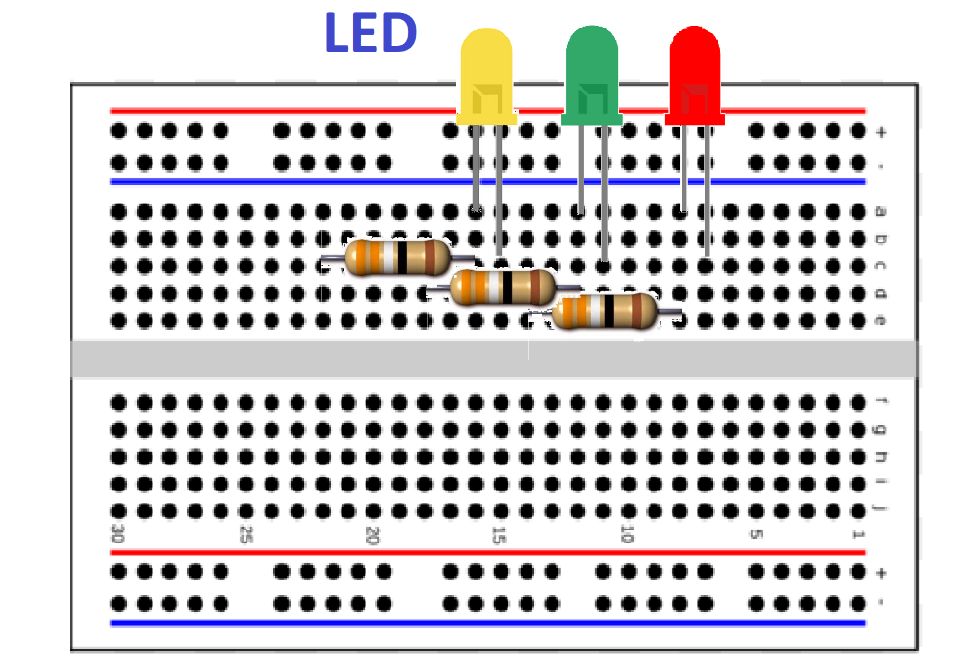
In questa configurazione l’elettromagnete del relè è alimentato direttamente dal Raspberry. I carichi di alimentazione degli attuatori, **lampada e ventola**, arrivano da sorgenti esterne, che in uno scenario ecosostenibile potrebbero derivare direttamente da fonti rinnovabili.

## **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

;

## **Flusso degli eventi**

All’avvio (digitando la WORD “SETUP”) il sistema si inizializza e 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 un 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 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 dal commento per aumentare la velocità di caricamento. Ovvero verrà inviato tramite picocom con la scorciatoia CTRL +A +S 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 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 main.f >> embedded.f

grep -v '^ \*\\' embedded.f > final.f

## **Dettaglio files sorgenti**

Ans.f: Inserisce nel dizionario dell’interprete forth alcune WORDS necessarie alla compilazione dei sorgenti

\ Embedded Systems - Sistemi Embedded - 17873

\ some dictionary definitions

\ from  pijFORTHos and prof. D. Peri

\ modificated by Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

\ '\n'  newline character (10)

: '\n' 10 ;

\ BL    blank character (32)

: BL 32 ;

: ':' [ CHAR : ] LITERAL ;

: ';' [ CHAR ; ] LITERAL ;

: '(' [ CHAR ( ] LITERAL ;

: ')' [ CHAR ) ] LITERAL ;

: '"' [ CHAR " ] LITERAL ;

: '.' [ CHAR . ] LITERAL ;

\ ?IMMEDIATE    ( entry -- p )  get IMMEDIATE flag from dictionary entry

\ ( comment text )  ( -- )  comment inside definition

\ SPACES    ( n -- )    print n spaces

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

\ ALIGNED   ( addr -- addr' )   round addr up to next 4-byte boundary

\ ALIGN ( -- )  align the HERE pointer

\ C,    ( c -- )    write a byte from the stack at HERE

\ S" string"    ( -- addr len ) create a string value

: ( IMMEDIATE 1 BEGIN KEY DUP '(' = IF DROP 1+ ELSE ')' = IF 1- THEN THEN DUP 0= UNTIL DROP ;

: SPACES BEGIN DUP 0> WHILE SPACE 1- REPEAT DROP ;

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

: ALIGNED 3 + 3 INVERT AND ;

: ALIGN HERE @ ALIGNED HERE ! ;

: C, HERE @ C! 1 HERE +! ;

: 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

;

: 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

: 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)

DROP

\ 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 settati i registri necessari al funzionamento dei vari GPIO e vi sono definite alcune WORDS necessarie a modificare e testare i tali registri.

\ Embedded Systems - Sistemi Embedded - 17873

\ Settaggi GPIO

\ Università degli Studi di Palermo

\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

\ Includere dopo il flie ans.f

\ GPIO Mapping

HEX

FE000000 CONSTANT BASE  \ Indirizzo base dei registri

BASE 200000 + CONSTANT GPFSEL0  \ Spazio dei registri GPIO FE200000

BASE 200004 + CONSTANT GPFSEL1

BASE 200008 + CONSTANT GPFSEL2

BASE 200040 + CONSTANT GPEDS0

BASE 20001C + CONSTANT GPSET0

BASE 200028 + CONSTANT GPCLR0

BASE 200034 + CONSTANT GPLEV0

BASE 200058 + CONSTANT GPFEN0

\ 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 ! ;

\ 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 definisce la comunicazione tra il PI 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 ;

: IS\_CMD

  DUP 8 RSHIFT 1 = ;

\ Decide se stiamo inviando un comando o un dato a I2C

\ Commands 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

\ dove 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 rappresentazione di esacecimali corrispondenti a caratteri ASCII costruiscono le stringhe di testo che verranno visualizzate a display.

\ 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

\ 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 ;

\ 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 disattivazione degli attuatori.

\ Embedded Systems - Sistemi Embedded - 17873

\ Led Drive

\ Università degli Studi di Palermo

\ Davide Proietto matr. 0739290 LM Ingegneria Informatica, 21-22

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

\ LED GPIO SETTING IN HEX

5 CONSTANT YELLOWLED

6 CONSTANT REDLED

C CONSTANT GREENLED

\ GPIO On e Off

: ON ( pin -- ) 1 SWAP LSHIFT GPSET0 ! ;

: OFF ( pin -- ) 1 SWAP LSHIFT GPCLR0 ! ;

\ 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

\Settaggi di default luce e vento in ms 800000 37SEC  400000 19SEC  200000 9SEC

420000 LIGHTIME !

420000 WINDTIME !

pad.f: In questo file descritte ed implementate le logiche per il controllo input dalla tastiera e il meccanismo per immagazzinare le 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 ABORT ELSE

  DUP COUNTER @ 0 = IF DUP CAS ! DUP 2 \* 4 LSHIFT LIGHTIME ! ELSE

  DUP COUNTER @ 1 = IF DUP CASS ! DUP LIGHTIME @ + 8 LSHIFT 8 LSHIFT LIGHTIME ! ELSE

  DUP COUNTER @ 2 = IF DUP COS ! DUP 2 \* 4 LSHIFT WINDTIME !  ELSE

  DUP COUNTER @ 3 = IF DUP COSS ! DUP WINDTIME @ + 8 LSHIFT 8 LSHIFT 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 . . . " CR

  ." SETTING TIME LIGHT >>>    "  CAS @ . CASS @ . ."    SECONDS " CR

  ." SETTING TIME WIND >>>    "  COS @ . COSS @ . ."    SECONDS " CR ;

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** e quelle che descrivono i vari comportamenti.

HEX

\ Aziona il Sistema Illuminazione

: GO\_LIGHT

  REDLED GPOFF!

  STOPWIND

  CLEAR

  SYSTEM

  LIGHT

  SYSTEMLIGHT

  ;

\ Aziona il Sistema Ventilazione

: GO\_WIND

  REDLED GPOFF!

  STOPLIGHT

  CLEAR

  SYSTEM

  WIND

  SYSTEMWIND

  ;

\ Sistema Arrestato

: STOP\_DISP

  ALL\_LED\_ON

  CLEAR

  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 @ DELAY

      GO\_WIND ." SYSTEM WIND " WINDTIME @ DELAY

      COUNTER++

      COUNTER @ 4 = IF  FLAGOFF THEN

      ." Cycle n° "

      COUNTER @ .

      ." Flag setting "

      FLAG @ .

      CR

    REPEAT

  ?CTF UNTIL FLAGON STOP\_DISP 10000 DELAY ." FINE PROGRAMMA " 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

  SETUP\_I2C

  SETUP\_LCD

  SETUP\_KEYPAD

  SETUP\_LED

  CLEAR

  WELCOME

  >LINE2

  SMART

  CLIMA

  30000 DELAY

  CLEAR

  STOP\_DISP

  10000 DELAY

  CLEAR

  INSERT TIME

  30000 DELAY

    BEGIN

    FLAGON

    DETECT

    1

    RUN 20000 DELAY

    ?CTF UNTIL

    ;

\ Solo setup Hardware per testing

  : ONLY\_SETUP

  SETUP\_I2C

  SETUP\_LCD

  SETUP\_KEYPAD

  SETUP\_LED

  CLEAR

  WELCOME

  >LINE2

  SMART

  CLIMA

  30000 DELAY

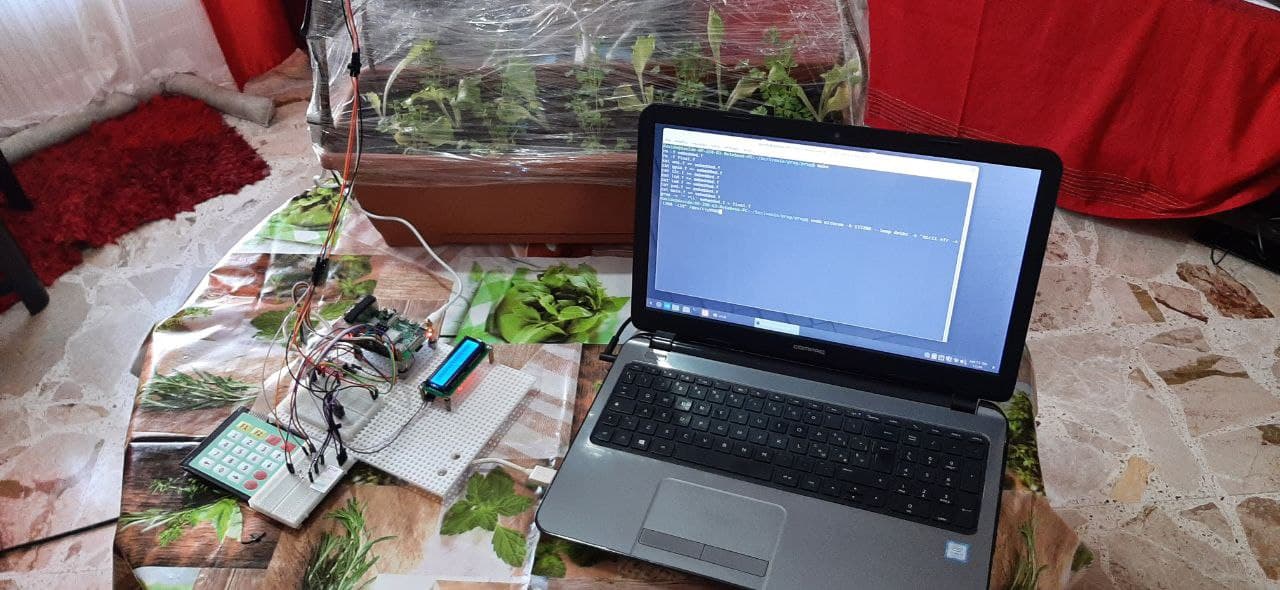
  CLEAR

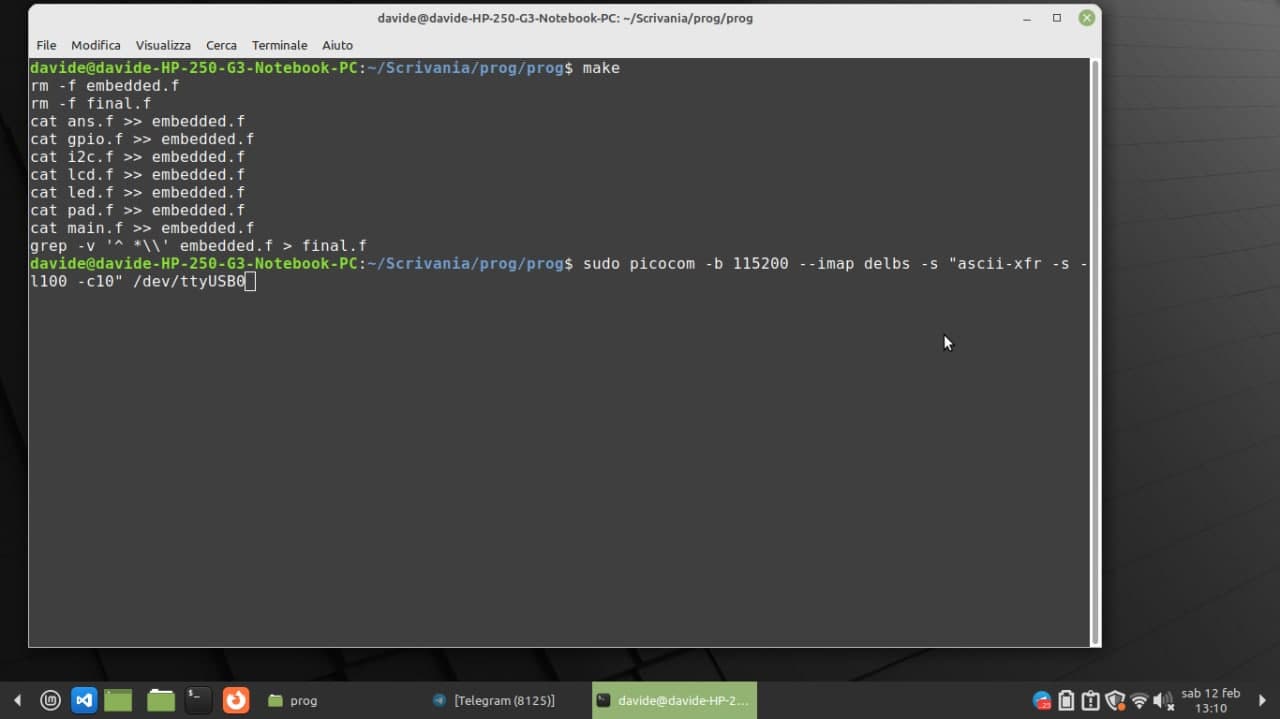
  STOP\_DISP

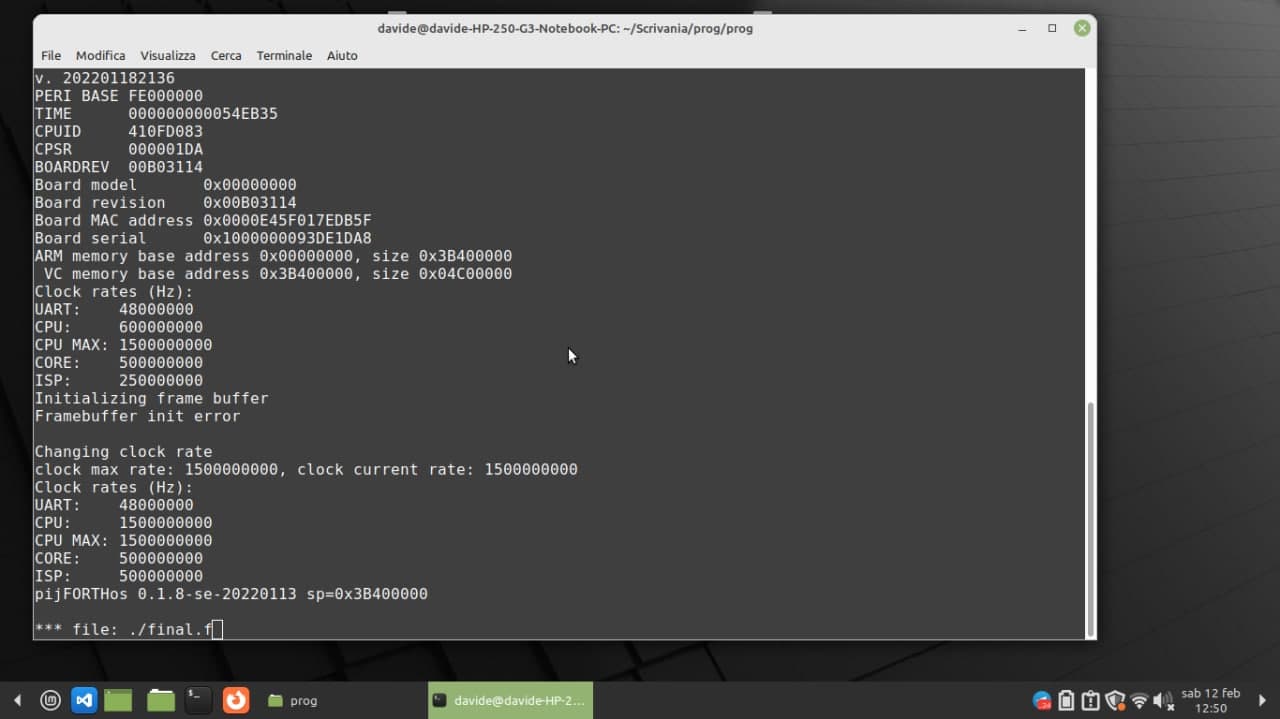
  ;

## **Testing**

Per la programmazione una volta avviato il terminale e attivata la connessione con l’interfaccia seriale possiamo dare alimentazione al Raspberry PI 4. Dopo pochi secondi comparirà la console a riga di comando dell’interprete.







Digitando la combinazione di tasti per inviare il file al target CTRL +a +s , basterà digitare il nome del file sorgente e premere invio.

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



A questo punto digitando la WORD SETUP il sistema si configura attivando tutto l’hardware e presentando il messaggio di benvenuto.



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 il valore di temporizzazione espresso in millisecondi di 4 cifre decimali (è sufficiente per la dimostrazione) dalla tastiera.







D’ora in avanti il sistema entra in un ciclo perpetuo di alternanza tra le funzioni.

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.



## **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 blocch*i” rende riutilizzabile il codice anche su altri target con le dovute pre-configurazioni.

*Non è possibile non sottolineare quanto, alla fine della realizzazione e della stesura di questo progetto, sono rimasto affascinato da questo ambito dell’ingegneria informatica.*

*Davide Proietto*