# Relazione Progetto Embedded System Design: Luca Davoli

## Consegna

Obiettivo Il progetto d’esame dovrà riguardare la progettazione e realizzazione di un prototipo virtuale di sistema per l’automatizzazione della procedura di pre-somministrazione di flaconcini contenenti dosi di vaccino. Si faccia riferimento alle indicazioni riportate nel file https://www.aifa.gov.it/documents/20142/1279946/04\_Fasi\_per\_la\_vaccinazione\_con\_Comirnaty\_Poster\_IT.pdf/ Specifiche Tecniche e Funzionalità Nelle sezioni seguenti sono elencate le informazioni base necessarie allo svolgimento del progetto. Componenti principali del sistema Lo studente dovrà progettare i seguenti blocchi funzionali:

● Sistema di gestione del ciclo termico di scongelamento

● Sistema di miscelazione

● Sistema di diluizione

● Sistema di controllo della durata della fase di pre-somministrazione del vaccino

● Sistema di visualizzazione della fase di lavoro corrente

Per semplicità si faccia riferimento a un sistema automatizzato di gestione di un flaconcino alla volta.

Per ciascun componente:

● Dovranno essere identificati attuatori, sensoristica di controllo simulata utilizzando componenti scelti tra quelli disponibili in DIGITAL e PicSimLab (ad es. sensori analogici possono essere simulati utilizzando il modulo potenziometro disponibile in PicSimLab come spare part)

● Dovrà essere prodotto il codice firmware/software necessario al corretto funzionamento del sistema nel rispetto delle specifiche. Tale codice dovrà essere descritto nel report per mezzo di diagrammi di flusso e/o macchine a stati opportunamente commentati

● Laddove si ritiene necessario dovrà essere fornita una breve descrizione della parte meccanica dei componenti a complemento delle scelte fatte dal punto di vista elettronico. Le scelte progettuali dovranno essere descritte in maniera esplicita in modo da rendere immediata la comprensione dell’architettura di sistema e l’implementazione dei singoli blocchi funzionali.

Task di progetto Il sistema che verrà progettato dovrà essere in grado di eseguire i seguenti task in sequenza:

1. Controllo del ciclo termico di scongelamento del vaccino portando il flaconcino dalla temperatura di stoccaggio di -55 °C alla temperatura di +8°C seguendo il profilo indicato in Fig. 1

2. Controllo del sistema di miscelazione in grado di gestire 10 cicli completi di capovolgimento del flaconcino con durata pari 1 sec/ciclo. Per ciclo completo si intende che il flaconcino al termine di ogni ciclo sia riportato nella posizione iniziale

3. Controllo del sistema di diluizione composto dai seguenti sub-task a. Allineamento dell’ago della siringa con il centro del tappo perforabile del flaconcino b. Penetrazione di 1 cm dell’ago nel flaconcino c. Iniezione della soluzione con dosaggio controllato del solvente (vedi dosaggio qui) d. Estrazione parziale dell’ago e aspirazione di egual volume d’aria per compensazione pressione del flaconcino e. Estrazione completa dell’ago

4. Secondo ciclo di miscelazione analogo a quello del punto 2 5. Sistema di controllo del countdown sulla scadenza della soluzione diluita, con notifica di un allarme nel caso che il flaconcino non sia rimosso dal sistema entro 5 sec dalla fine della fase 4. Dovrà essere previsto, inoltre, un sistema di visualizzazione della fase di lavoro corrente attivo durante tutto il ciclo di lavoro Fig. 1 - Profilo di Temperatura per la fase di scongelamento

Vincoli di progetto

● Definizione di un protocollo di comunicazione FPGA (Digital) – Microcontrollore (PicSimLab) basato su comunicazione seriale asincrona con baud rate 9600 bps, 8 bit di dati, 1 bit di stop, nessun controllo di flusso, realizzato sfruttando porte di tipo virtual COM disponibili attraverso l’utilizzo dell’emulatore com0com.

Gradi di libertà del progetto In base alle scelte progettuali fatte per la realizzazione delle funzionalità sopra descritte è lasciato libero allo studente di decidere:

● la suddivisione delle varie funzionalità tra FPGA e microcontrollore.

● Tutto quanto non specificato, ma necessario nell’ottica di avere un prototipo realizzabile fisicamente. Nel report dovrà essere motivata ogni eventuale assunzione/semplificazione fatta. Componenti utilizzabili Il progetto sarà basato sull’utilizzo dei software di sviluppo/emulazione visti a lezione (DIGITAL e MPLABXPicSimLab).

## Descrizione Sistema e suddivisione funzioni tra FPGA e Micro

Il sistema da me progettato per soddisfare i requisiti di progetto prevede un nastro trasportatore con nastro in gomma ad alto attrito, questo nastro accoglierà la fialetta di vaccino alla stazione di INPUT, qui un sensore a Time of flight opportunamente calibrato, rivelerà quando la fialetta è posizionata.

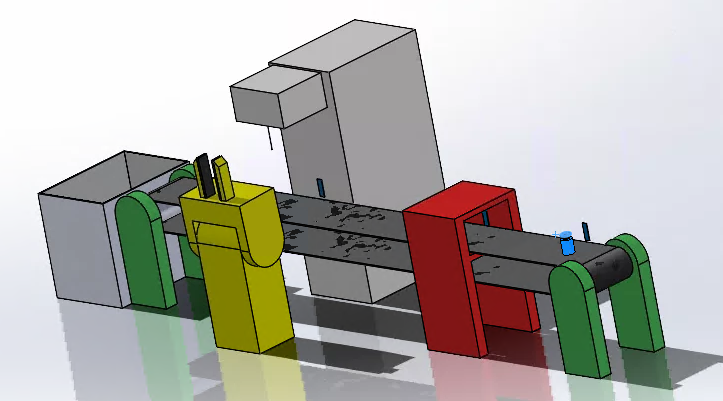


Figura 1 Fialetta alla postazione iniziale

Una volta posizionata e rilevata, il nastro partirà, il nastro è guidato da un motore DC, il sistema capirà quando la fiala è alla seconda stazione, il forno, una volta che un secondo sensore TOF la rivelerà, quando questo accade il motore viene fermato, in modo che la fiala si fermi dove è trovata dal sensore.

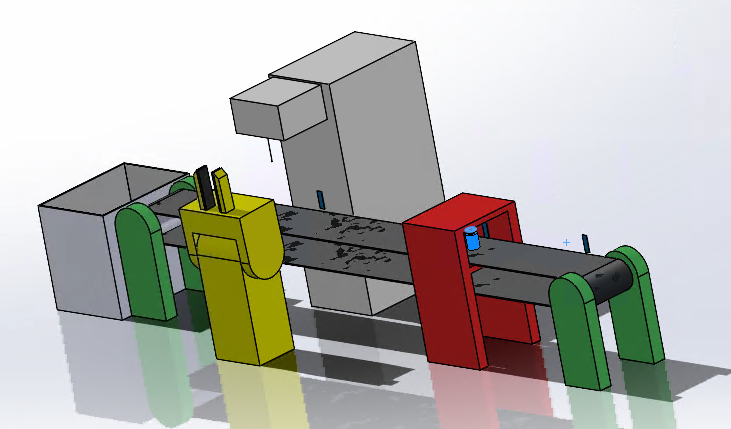


Figura Fialetta nel forno

Una volta nel forno, parte un timer di 5 secondi, nel frattempo con un sensore di temperatura quale una termocamera o una termocoppia a contatto con la fialetta produce un feedback di temperatura, grazie al quale al termine dei 5 secondi si controlla che mediamente la curva di temperatura si sia comportata come da consegna. Al termine dei 5 secondi se il controllo non va a buon fine, il sistema va in errore, l’operatore dovrà dunque rimuovere la fiala.

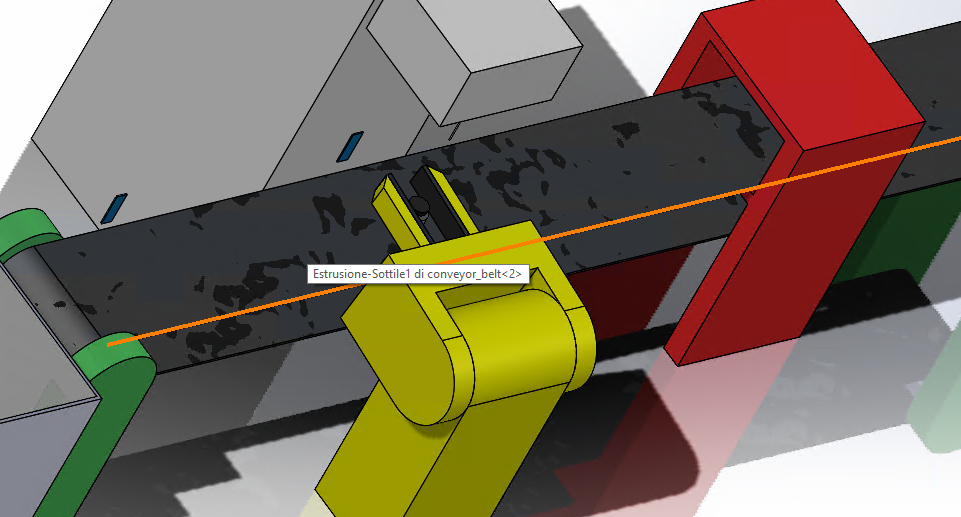


Figura 3 Braccio robotico afferra la fiala

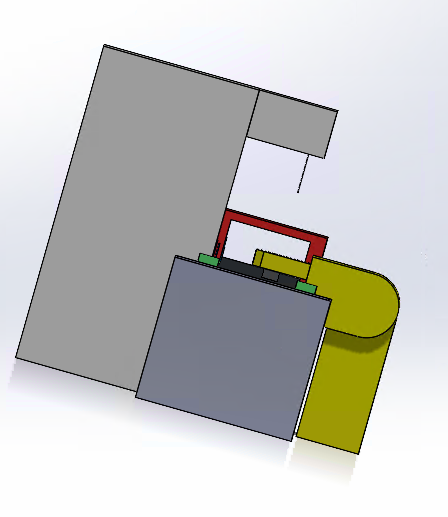


Figura Vista laterale dell'afferraggio

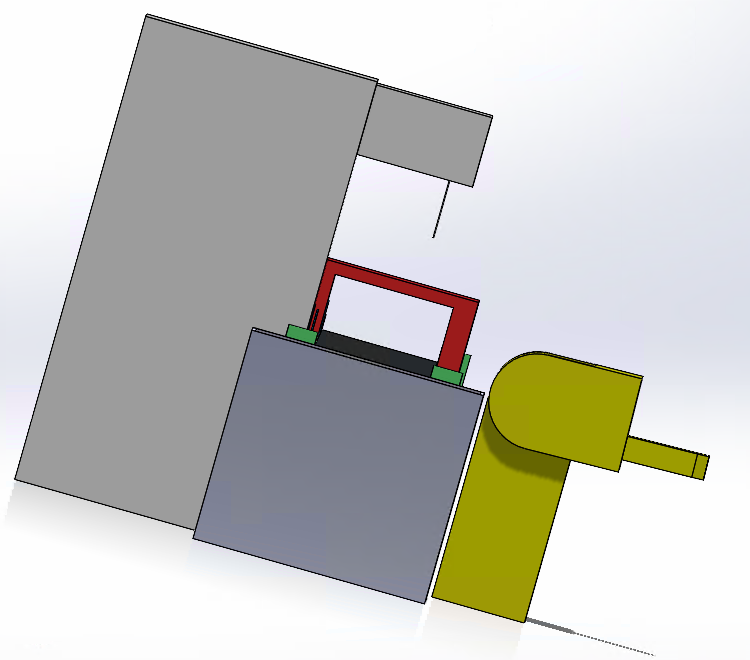


Figura 5 Miscelazione, braccio robotico al suo estremo

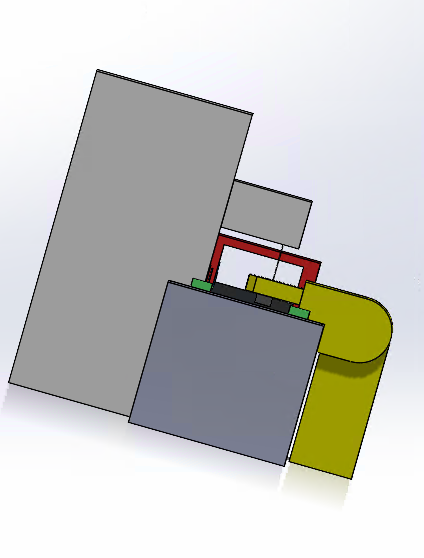


Figura 6 Diluizione

Altrimenti il nastro viene riattivato e fiala giunge così alla 3 stazione di miscelazione e diluzione, il suo arrivo viene ancora una volta rilevato da un TOF sensor. A quel punto un braccio robotico dalla posizione di home si muove verso la fiala la centra e afferra, dopodichè si procede alla miscelazione, dunque il braccio con la fialetta in presa, inizierà a compiere rotazioni di +-180°, fatte un totale di 20 rotazioni di 180 gradi la fialetta è di nuovo sul nastro e perfettamente centrata, dato che ancora in presa, il diluitore può dunque abbassarsi, forare la membrana sulla fialetta con il suo ago e per gradiente di pressione riempirla di soluzione. Terminata la diluizione, il diluitore si rialza e lo stesso processo di miscelazione si ripete in modo da mischiare il vaccino alla soluzione, terminate le 20 rotazioni il braccio robotico rilascia finalmente la fialetta esattamente dove l’ha trovata e il nastro la può dunque portare verso l’ultima stazione.

A picture containing LEGO

Description automatically generated

Figura 7 Fialetta alla stazione di prelievo

Una volta che la fialetta raggiunge la stazione di prelievo, posizione sempre rivelata da un TOF sensor, il nastro ancora una volta si ferma in modo che la fialetta possa essere prelevata in sicurezza, sempre alla rilevazione un timer di 5s viene avviato, se al termine del timer la fiala non è stata prelevata, questa verrà portata, sempre tramite il nastro, al cestino che una volta pieno deve essere vuotato manualmente; una volta cestinata la fialetta scaduta il sistema torna in attesa di una nuova fiala, se invece la fialetta è stata prelevata, il timer viene disattivato e si torna in attesa immediatamente.

Tutti questi passaggi sono accompagnati da messaggi che spiegano all’operatore cosa stia succedendo.

I vantaggi di un design di questo tipo sono la semplicità, la completa automazione e la robustezza, durante la fase di sviluppo concettuale ho validato più idee (un braccio robotico che muove la fiala in varie stazioni, un sistema integrato in un'unica stazione con cella di temperatura) e questa è emersa come la migliore.

## Hardware: selezione e design

Dal lato HW ho cercato di limitare la complessità il più possibile riducendo al minimo i componenti e la diversità di questi, ad esempio riutilizzando i TOF sensors in tutte e 4 le stazioni e usando lo stepper motor sia nel braccio meccanico che nel diluitore.

L’assieme creato con sw serve in primo luogo a validare e chiarire, ma vuole anche essere rappresentativo di un design reale, che a meno di cablaggi e supporti potrebbe rispecchiare una prima implementazione del sistema.

Dal punto di vista elettrico sempre per omogeneizzare opterei per una linea logica a basso voltaggio per dare pwr a fpga e micro e una a 24 volt per i motori con emergency button che la può interrompere disposto ad inizio e fine linea.

Il forno è pensato come un forno di linea classico per industria alimentare, per aumentare l’isolamento aggiungerei una sorta di tenda isolante in entrata e ingresso.

Questo design si presta bene alla parallelizzazione, se infatti più fiale messe ad esempio in una matrice fossero poste alla posizione iniziale, il solo limite sarebbe avere più aghi nel diluitore ma il design rimarrebbe pressochè invariato sia nel codice che fisicamente.

Si potrebbe anche aggiungere un robot che automatizza la presa in carico della fiala così da rendere il processo completamente automatico.

La frequenza di lavoro del processore è stata settata a 4MHz, mentre per la FPGA purtroppo sebbene frequenze più alte sarebbero più opportune mi son ritrovato a usare 10Hz per far sì che la simulazione e la comunicazione funzionassero a dovere.

A livello elettrico tutti gli schematici sono trovabili su <https://lcgamboa.github.io/picsimlab_docs/stable/> per ciascun dispositivo connesso al micro. Serve poi una doppia linea per il segnale tra FPGA e micro per implementare la comunicazione USART. Solo i TOF sono connessi alla fpga, gli altri dispositivi devono essere connessi agli opportuni pin del micro.

## State Machine: spiegazione

Di seguito la state machine che descrive e riassume quanto introdotto a parole nel primo capitolo.

A picture containing text, diagram, circle, font

Description automatically generated

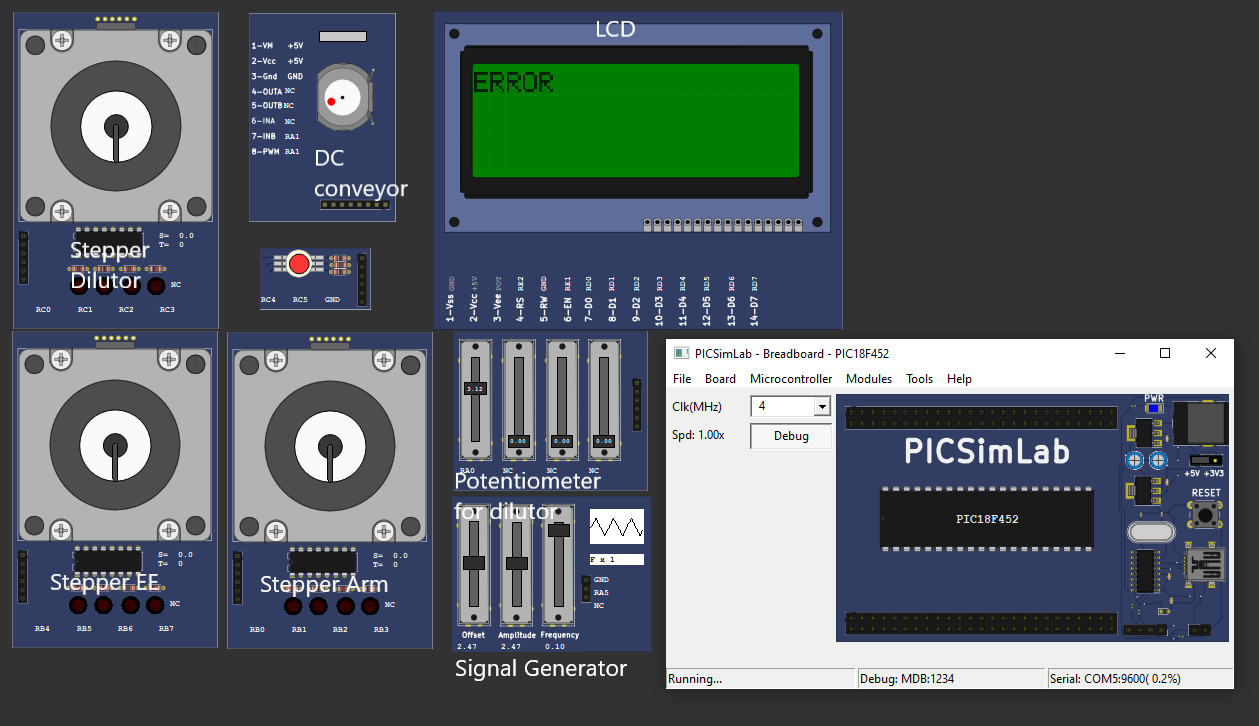
Gli stati mostrati presentano tutti variabili di stato interne che vengono aggiornate e che quando raggiungono una certo valore definito dalle specifiche innescano il cambiamento di stato, gli stati che non hanno tale comportamento sono quelli che dipendono solo da input esterni, nel nostro caso cambiamenti dei valori rilevati dai TOF sensor, quindi IDLE non ha nessuna variabile interna, ne MOVE, tutti gli altri stati sono da intendersi come ciclici fintanto che la variabile di stato non raggiunge il valore designato, introdurrò di seguito le variabili in modo da demistificarle, cercando di seguire ancora una volta il flusso logico standard della fialetta.

IDLE cambia a MOVE se TOF1, il primo sensore TOF, viene triggerato, da MOVE ci si muove poi a OVEN solo una volta TOF2=1, dal forno si esce invece solo una volta terminato il timer e constatata la correttezza del ciclo di scioglimento, stato caratterizzato da T reached, si torna dunque a MOVE da cui si esce quando TOF3 rivela la fialetta di fronte a sé, TOF3=1 e si passa dunque a MIX. Lo stato cambia una volta finite le 20 rotazioni, ovvero una volta Mixed la soluzione, si passa poi a DILUTE, solo una volta che la soluzione è stata iniettata e il diluitore retratto si triggera il segnale Diluted e si procede a mixare di nuovo, terminate ancora le rotazioni, ora lo stato di MIX è Diluted and Mixed si passa dunque a MOVE nuovamente stato che solo una volta che TOF4=1 cambia passa a PICK-UP, qui vi sono due casi, se TOF4=0 mentre in stato PICK-UP 🡪 fialetta prelevata 🡪 si ritorna a IDLE, mentre se il Timer scade Timer overflow ci porta allo stato TRASH, qui tramite un counter, lo stato si assicura che la fialetta tramite il nastro venga buttata nel cestino, quando il counter ha superato la soglia di sicurezza 🡪 Trashed e anche qui si ritorna a IDLE.

Ho introdotto una gestione errori differenziata, per i tre tipi di errori considerati ovvero, più di una fialetta alla volta sul nastro TOFx && TOFy, T not reached nel forno e soluzione nel diluitore non sufficiente. Nei primi due casi l’utente dovrà risolvere manualmente la situazione e resettare il device una volta ritirata e cestinata la fialetta, nel terzo caso se durante la fase di IDLE si riscontra che il livello di soluzione nel diluitore non è sufficiente si va in errore, ma siccome si è ancora in fase di IDLE non vi sono grossi rischi ed è facile riprendere a seguito dell’errore nel caso la soluzione venga aggiunta al diluitore. Ci potrebbero essere dubbi riguardo la bontà della fialetta dopo l’attesa, ma dato che passera ugualmente il check nel forno riguardo la temperatura, mi sembra opportuno non buttare il vaccino nel caso l’operatore sia a sufficienza veloce a rabboccare la soluzione.

# MPLABX e Picsimlab

La macchina a stati è implementata con una semplice struttura if else, ci sono tuttavia più stati di quelli mostrati nella macchina a stati vista in precedenza perché nel codice ho anche introdotto stati di transizione sia per visualizzare il raggiungimento della stazione di lavoro, sia per inizializzare chiaramente lo stato che le segue. Procederò ora a descrivere come ho implementato le principali fasi del processo



### Init state

Non appena acceso il processore le porte sono inizializzate e inizia il while(1), si entra dunque nella condizione di idle, solo quando dalla fpga arriva il messaggio 0X01, un messaggio viene mostrato sul LCD che comunica la presenza della fialetta sul nastro nella posizione di partenza, in automatico si triggera poi il movimento del nastro.

### Oven

Quando dalla fpga arriva il messaggio 0x02 significa che la fiala è al centro del forno, il nastro si ferma, il forno è modellato come un timer (timer 0) di 5s che al termine controlla che il gradiente di temperatura nei 5s sia entro una certa tolleranza uguale a quello desiderato: 12.6, derivante da (8 – (-55) )/5.

Configurazione timer 0

|  |  |  |
| --- | --- | --- |
| **Config Bit/REGISTRO** | **Valore** | **Motivo** |
| INTCONbits.TMR0IE | 1 | Enable interrupt per timer overflow |
| T0CONbits.T08BIT | 0 | 16 bit ci servono per i 5s |
| T0CONbits.T0CS | 0 | Usa instruction cycle (1MHz) |
| T0CONbits.PSA | 0 | Use prescaler |
| T0CONbits.T0PS<2:0> | 110 | Prescaler a 128, minimo valore di prescaler per i 5s |
| TMR0L | 0x69 | Calcoli a seguire |
| TMR0H | 0x67 | Calcoli a seguire |
| T0CONbits.TMR0ON | 1 | Per farlo partire |

Il timer viene usato due volte in pick up station e nel forno come da sopra, in entrambi i casi utilizzo una funzione che setta esattamente i registri come da tabella precedente, perché il periodo richiesto è lo stesso. Dato l’instruction cycle di 1MHz il tempo di incremento, cioè ogni quanto il counter fa +1 è dato dà il prescaler moltiplicato per 2.5\*10^-7s (1/instruction cycle) si ottiene dunque 3.2\*10^-5s il che significa che in 5s ci stanno 39062 o in hex 0x9896 viene dunque facile trovare il valore da cui iniziare a contare per avere l’overflow a 5s 🡪 0Xfff – 0x9896 = 0x6769 che vengono caricati nei rispettivi registri. T0CONbits.TMR0ON viene azzerato nella gestione dell’interrupt, ma vi sono casi in cui questo non si verifica, allora va azzerato manualmente in modo che non si raggiunga l’overflow, ad esempio quando la fialetta viene prelevata in tempo dalla pick up station.

La temperatura viene simulata da un generatore di segnale con una onda a dente di sega con f 0.1Hz in modo che il tempo di salita sia esattamente 5 s e come ampiezza 2.5 con un offset di 2.5 così che si abbia una tensione variabile tra 0 e 5V su AN4 pin configurato come analog input, questo mappa dunque gli 0V alla temperatura più bassa -55C e 5V alla più alta e 5V sono rappresentati con 255 mentre con 0 e rappresentato -55C. Serve dunque una funzione che riscali il valore binario dal sensore e lo offsetti (val \* scala + offset). Vi è poi un’altra funzione che serve per far si che il generatore di funzioni simuli l’andamento di temperatura, questa fa si che si aspetti l’inizio di un onda triangolare, un fronte di salita, o più semplicemente uno zero e poi inizializza il timer e fa partire il countdown di 5s.

### Stepper Motors and DC motor & PORTS

|  |  |  |
| --- | --- | --- |
| **Config Bit/REGISTRO/pin** | **Valore** | **Motivo/Uso** |
| TRISA | 0Xf1 | RA0 e RA5 input, RA1 output |
| RA0/AN0 |  | Analogico per livello soluzione |
| RA1 |  | Output: Input for dc motor |
| RA5/AN4 |  | Analogico per temperature forno |
| TRISB | 0x00 | Output dunque input per stepper motor del braccio robotico |
| TRISC | 0xA0 | RC7 come input per Usart, <0:3> stepper motor per diluitore, RC4 e RC5 usate per error management |

Gli stepper motor a 4 coils si muovono di 1.8° ogni step a patto che si segua il corretto ordine di accensione delle coil (accensione in sequenza), ho implementato una struct stepperMotor in modo da riutilizzare quanto più codice possibile nei tre casi in cui è utilizzato. I metodi prinicipali permettono di gestire un cambio di direzione, il raggiungimento di un obiettivo e l’inizializzazione del motore.

Il DC motor viene controllato dallo stato 2 (MOVE) che setta a 1 RA1 e lo fa girare a velocita costante fintanto che un messaggio dalla fpga non lo interrompe. Viene anche utilizzato per muovere nel pattume una fiala se lasciata troppo a lungo alla pick up station.

### Misuratore di livello diluitore

Semplicemente un input analogico AN0 collegato a un potenziometro lineare che rappresenta quanti litri ci sono, ho poi implementato una funzione get liters che converte il valore del registro a 8 bit a litri moltiplicando per 0.01 supponendo il serbatoio al massimo possa contenere 2.55l. Se durante la fase di idle questo scende sotto 0.01l si va in errore, un errore che può però essere risolto.

Di seguito la tabella per la configurazione del AD converter AN0 nel tank del diluitore e per AN4 per la termocoppia nel forno.

|  |  |  |
| --- | --- | --- |
| **Config Bit/REGISTRO/pin** | **Valore** | **Motivo/Uso** |
| ADCON0bits.CHS<0:2> | 100 | Per fare conversione dell’input su AN4 |
| ADCON0bits.CHS<0:2> | 000 | Per fare conversione dell’input su AN0 |
| ADCON0bits.ADON | 1 | Fa partire la conversione |
| ADCON1 | 0X00 |  |

### Pick up state

Fa partire un timer0, con stessi settings di quello del forno. Se fialetta prelevata prima arriva il messaggio 0x20 dalla fpga che fa si che si esca da questo stato e si ritorni in IDLE in attesa di una nuova fiala.

### State update and USART

Ogni qualvolta si cambia di stato internamente al codice C una variabile chiamata state\_changed viene portata a true, questa viene anche portata a true ogni volta arriva un messaggio dalla fpga, vi è dunque una routine di traduzione messaggi e aggiornamento stato che avviene ogni volta è settata a true, alla fine della routine viene poi posta a false nuovamente.

|  |  |  |
| --- | --- | --- |
| **Config Bit/REGISTRO/pin** | **Valore** | **Motivo/Uso** |
| SPBRG | 0x25 | Baudrate 9600bps |
| Config Bit/REGISTRO/pin | Valore | Motivo/Uso |
| TXSTAbits.TX9 | 0 | 8 bit Tx |
| TXSTAbits.TXEN | 1 | Enable Tx |
| TXSTAbits.SYNC | 0 | Async |
| TXSTAbits.BRGH | 1 | High baud rate per ridurre l’errore in trasmissione |
| RCSTAbits.SPEN | 1 | Enable Rx |
| RCSTAbits.RX9 | 0 | 8bit |

I messaggi tra fpga e micro sono codificati in modo siano un char ad ogni stato corrisponde quindi un char di seguito la tabella, queste conversioni da char a state e viceversa vengono svolte da funzioni poste nel file utils.c.

Da FPGA a micro

|  |  |  |  |
| --- | --- | --- | --- |
| **Stato** | **Messaggio** | **Stato** | **Messaggio** |
| TOF1 | 0x01 | ERR | 0x40 |
| TOF2 | 0x02 | IDLE | 0x00 o 0x09 |
| TOF3 | 0x04 | PROCESS | 0x80 |
| TOF4 | 0x08 | PICKED | 0x20 |
| MV | 0X10 |  |  |

Da Micro a FPGA

|  |  |  |  |
| --- | --- | --- | --- |
| **Stato** | **Messaggio** | **Stato** | **Messaggio** |
| 2 (MV) | 0x10 | 14 (ERR) | 0x40 |
| 0 (IDLE) | 0x00 | Else (PROCESS) | 0x80 |

### LCD

Ogni stato ha il suo messaggio che è salvato in un array 14x4x20, perché ciascuna riga del display può contenere al massimo 20 char e vi sono 4 righe; quindi, ogni messaggio può al più essere 80 char. Per la gestione dello schermo ho usato una libreria per il LCD di picsim trovata <https://github.com/lcgamboa/picsimlab_examples/blob/master/docs/board_Breadboard/PIC18F4550/test_b0/src/teste_b0.X/lcd.c> o meglio l’ho rivisitata e vi ho aggiunto funzioni come lcd\_update che semplificano e rendono più leggibile il codice.

LCD lavora su PORTE e PORTD quindi entrambi i registri TRISD e TRISDE devono essere settati a 0x00 così che siano output.

### Error management

Ci sono tre fonti di errore e due comportamenti diversi del sistema; in caso di errore un messaggio ERROR viene mostrato sul LCD, ma al contempo si accende un LED RGB, giallo se l’errore è risolvibile, ROSSO se richiede non solo l’intervento dell’utente per rimuovere la fiala, ma anche il reset del sistema.

Il primo caso si ha quando il livello della soluzione scende a un valore critico, qui infatti l’utente può aggiungere altro liquido e una volta risuperata la soglia minima, si riparte dalla condizione di IDLE.

Il secondo caso si ha quando vi è un errore proveniente dalla FPGA, 0x40, o quando la curva di temperatura non corrisponde. In questo caso il led sarà rosso e l’utente dovra intervenire resettando il device e buttando via il vaccino.

## FPGA

Per la descrizione di quanto implementato in fpga procederò ancora componente per componente.

A picture containing diagram, text, plan, technical drawing

Description automatically generated

### Sensori TOF (tof\_mimic)

Implementati come pulsanti, 4, per rendere però più verosimile il comportamento di questi ho aggiunto in cascata ai pulsanti un blocco che prende le 4 uscite di questi e le restituisce a valle, se però in input arriva un segnale di MOVE 0x10 dal uC allora le uscite vengono azzerate, per simulare il fatto che una volta fatto ripartire il nastro la fiala si sposta e il sensore non avendo più nulla davanti torna basso.

### Decoder

Il decoder è un’entità che riceve in input l’uscita dei sensori TOF, nella realtà sarebbe fisicamente l’interfaccia con questi, mentre qui sono simulati e necessitano l’accortezza implementata da tof\_mimic.

Oltre a questi riceve anche il messaggio ricevuto dal micro.

In base a questi input il decoder restituisce un output a 8 bit che costituisce il messaggio che sarà poi mandato al micro.

Se il messaggio è MV 0x10 allora il decoder è trasparente per i segnali TOF in modo che si po ssa mandare il messaggio di raggiunta posizione al micro e interrompere il nastro, se però due TOF vengono triggerati o più allora il messaggio mandato è 0x40 ERR.

Nel caso il messaggio in input sia IDLE 0, ovvero quando si inizia, allora il decoder diventa reattivo ad eventi su TOF1 così che rilevi l’arrivo della fialetta.

Se si è in process 0x80 e TOF4 è alta, ovvero la fialetta è alla pick up station allora nel caso di un fronte di discesa su TOF4, prelievo della fialetta, la linea picked viene portata alta, questo componente gestisce dunque la fase di pick up oltre ai sensori ed il comportamento in funzione dello stato-msg corrente.

### Contatore (counter)

Quando dal decoder la linea picked è alta al primo ciclo di clock somma 1 a una variabile di stato interna, ai successivi, mentre picked è alta, invece la variabile interna a 8 bit non viene aggiornata, nella pratica in questo caso si tratta infatti sempre dello stesso evento picked, non avrebbe senso contarlo più volte. La variabile di stato viene poi portata in uscita e ripartita in due da 4bit così che possano essere mostrate su display a 7 segmenti, si ottiene quindi una visualizzazione in esadecimale di quante fialette sono arrivate sino alla fine e sono state prelevate in tempo.

### Comunicazione (write\_uart\_sm, registroIncDec8bit e COM6)

Write uart assicura che solo se il messaggio a 8 bit proveniente dal decoder è cambiato allora si scriva portando alta la linea data write bool collegata al l’ingresso write della porta COM6.

Mentre in ricezione il messaggio viene salvato in un buffer, implementato da un registro a 8 bit.

## Github e scelte progettuali

Ho scelto di caricare ogni file necessario a riprodurre il progetto su github ed anche la documentazione di questo, principalmente per aiutarmi durante lo sviluppo ed avere un sistema di versioning che non solo salvasse nel cloud il progetto, ma gestisse le versioni man mano che cresceva.

Lo trovo anche il miglior modo per documentare il processo di sviluppo che altrimenti avrebbe trovato poco spazio nella relazione. Processo che è partito da un’idea base lo scheletro, ovvero la comunicazione USART micro-fpga, e dunque la detection della posizione della fialetta e man mano incorpora le varie funzionalità di ciascuna stazione, per poi raffinarle e metterle alla prova con test di integrazione (nonché poi debuggare).

Procedendo per step incrementali e validando in principio l’idea alla base, non mi sono ritrovato costretto a fare grossi refactoring del progetto.

Una scelta progettuale che si può notare dalla macchina a stati nel micro è che ho sempre cercato, con le limitazioni di picsimlab, di ciclare nel while il più possibile tenendo brevissimi i processi negli stati, in modo che il sistema fosse reattivo ai cambiamenti.

### Possibili Sviluppi

Aggiungo un paio di idee che potrebbero essere aggiunte:

* Camera che fa la detection di eventuali impurità, si potrebbe implementare in due modi o nella fpga facendo il deployment di una rete neurale classificatrice in essa o con un computer terzo ed integrato nella camera, che abbia già integrata la detection feature, messo poi in comunicazione con il micro
* Gestione automatica dell’errore in modo che i due errori rossi siano risolvibili automaticamente ad esempio gettando sempre la fialetta tramite accensione nastro e controllo del passaggio della fialetta al tof4.
* Homing dei motori stepper, una fase iniziale al reset del micro che porti i motori alla posizione di homing, ad esempio con homing sensor esterni, in picsimlab si potrebbe aggiungere un interrupt su un pulsante che potrebbe simulare il raggiungimento della posizione e fermare il motore
* Sequenza di recupero, nel caso venga premuto l’emergency button mentre si è nella fase di mixaggio o diluizione, si può implementare una sequenza che recuperi lo stato di homing rilasciando la fiala però sul nastro in modo safe