Progettazione di Sistemi Meccatronici

Relazione progetto *Drawing Machine*

Michele Martino, Samuele Vannoni

Anno 2021-2022

UNIVERSITÀ DI PISA

1 Introduzione

Il progetto presentato in questa relazione consiste nella realizzazione di un *plotter*, o *drawing machine*, capace di interpretare comandi in Gcode e di disegnare forme poligonali. La struttura fisica si basa su una fresa *CNC 3018*, con attuatori stepper per tutti e tre gli assi cartesiani. La logica di controllo è realizzata con una scheda di sviluppo S32K144, che pilota i motori tramite una *CNC* shield realizzata per Arduino (che risulta compatibile col pinout della scheda NXP). Il software è stato realizzato su Simulink tramite l'inclusione di blocchi di configurazione proprietari della NXP per la scheda impiegata.

2 Specifiche

Inizialmente il progetto prevedeva la realizzazione di un incisore laser per disegnare sul legno. Oltre a questo, era prevista anche la lettura del Gcode salvato su una scheda SD, quest'ultima accessibile con una shield da collegare alla scheda di sviluppo. Inoltre tramite uno schermo LCD sarebbe stato possibile selezionare, tra più file, quello desiderato. Questo avrebbe richiesto inevitabilmente l'implementazione di un file system: data la scala del progetto e la complessità di implementare un tale sistema, specialmente seguendo uno stile *model based* dove non sono nativamente presenti blocchi per realizzare più o meno direttamente tale funzione, è stato deciso già dalle prime fasi di scartare questa parte del progetto. Come compromesso, è stata sfruttata la funzionalità UART del microcontrollore in quanto è anche facilmente implementabile con Simulink tramite blocchi dedicati; è stato inoltre sviluppato un piccolo script con Python capace di leggere un file Gcode e mandare opportunamente i comandi tramite seriale.

Riguardo al laser, non è stato possibile avere una facile disponibilità di tale oggetto, pertanto è stato deciso, dal momento che la fresa dispone anche dell'asse Z, di realizzare una funzionalità simile a quella inizialmente prevista, creando un plotter invece di un incisore laser struttando comunque i comandi Gcode che avrebbero controllato il laser. In questo modo rimane semplice realizzare un eventuale implementazione futura dello stesso.

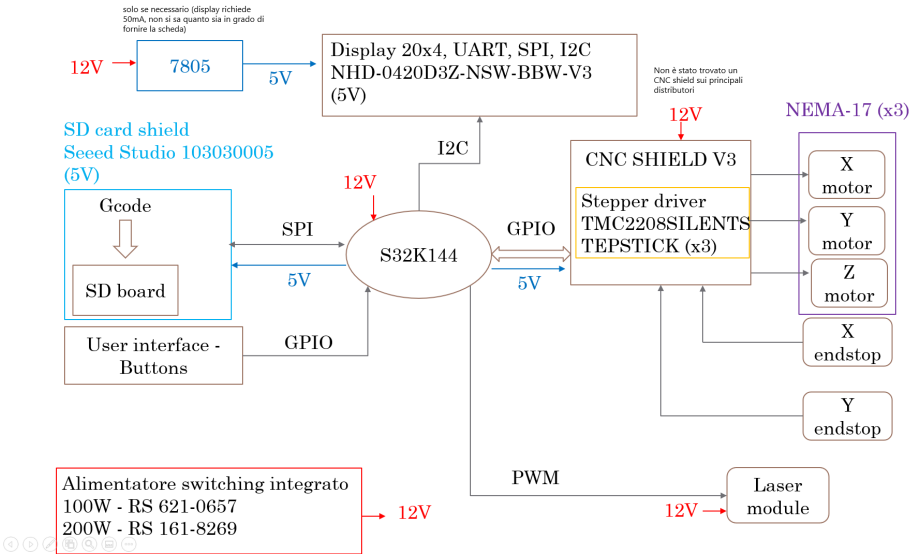


Figure 1: Struttura del progetto originale (incisore laser)

Ricapitolando dunque, le specifiche finali del progetto sono:

- Realizzazione di un plotter cartesiano,
- capace di interpretare comandi formattati come Gcode (limitandosi, nella parte di movimento, solamente a quelli lineari, ovvero descritti dal comando G1),
- i quali comandi sono mandati da un computer tramite UART, in maniera automatica o manualmente tramite terminale.

3 Descrizione progetto

3.1 Flusso di progetto

La prima parte del progetto ha riguardato lo studio dell'hardware a disposizione, in particolare l'interfacciamento tra la CNC shield e la scheda di sviluppo. Successivamente sono stati analizzati tutti i blocchi di Simulink forniti da NXP che potessero risultare utili, in particolare i blocchi dei timer e di configurazione dell' UART. Una volta fatto ciò, sono state realizzate in parallelo le parti di controllo dei motori stepper e di parsing delle stringhe di Gcode. Il problema principale riscontrato in questa fase è stato dovuto alla scelta del timer hardware da usare nel pilotaggio dei motori, infatti i driver A4988 vengono pilotati inviando una sequenza di impulsi che vengono tradotti in passi del motore in base ai microstep impostati con dei jumper sulla scheda. In una prima fase è stato scelto un timer PWM (blocco *FTM*), impostato con un duty cycle del 50%, del quale veniva variata la frequenza per far muovere il motore alla velocità necessaria. Ciò comportava un problema poichè l'aggiornamento della frequenza non veniva fatto prima che il timer partisse, ma solo dopo un certo lasso di tempo dipendente dal tick del sistema e da quanto l'interrupt del blocco *FTM* impegnava il sistema.

Questo è stato osservato in maniera empirica tramite oscilloscopio, e successivamente giustificato analizzando il codice generato da Simulink, pertanto è stato abbandonato questo sistema, sebbene fosse più semplice e immediato,perché non permetteva un controllo preciso e ripetibile. La soluzione implementata invece utilizza un timer a frequenza fissa che manda periodicamente un interrupt il quale attiva una macchina a stati che genera la forma d'onda necessaria.

Come accennato, il parser e il controllo dei motori sono stati realizzati e testati come progetti separati, per poi unirli nella fase finale del progetto in un unico elemento. Una volta unito il tutto (e risolti gli inevitabili, seppur minori, problemi dovuti all'interfacciamento di due progetti), è stato testata la capacità del sistema di disegnare un intero file Gcode realizzando uno script Python che mandasse opportunamente i comandi necessari attraverso la UART: infatti fino a questo punto per comunicare tra computer e scheda è sempre stato usato Tera Term, un programma open source per Windows capace di interfacciarsi con la scheda di sviluppo tramite comunicazione seriale.

Visto che il sistema prevede diverse fasi di esecuzione(ricezione dati, interpretazione stringa, elaborazione dei comandi...) è stato scelto di far comunicare i vari sottosistemi utilizzando delle variabili globali condivise che segnalano quali azioni eseguire. Infine per facilitare la fase di debug sono stati impiegati i LED disponibili sulla scheda, oltre che alla possibilità fornita dalla UART di mandare i contenuti delle varie *Data Store Memory*, che contengono le informazioni sullo stato del sistema, al terminale di Tera Term.

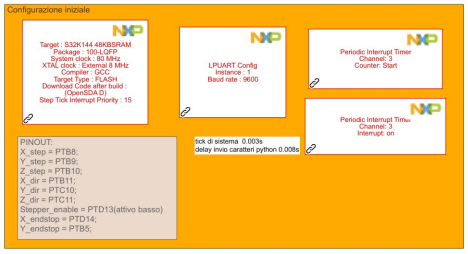
Una volta soddisfatte le specifiche di progetto, è stato notato che il movimento dei motori stepper generava un suono variabile opportunamente con la velocità impostata. Come semplice variante del progetto è stato quindi deciso di creare un programma alternativo, in python, per generare note musicali da

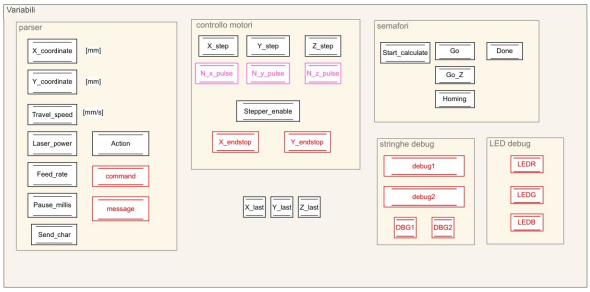
spartiti personalizzati, creando un Gcode apposito da mandare alla macchina, così da riprodurre la melodia desiderata.

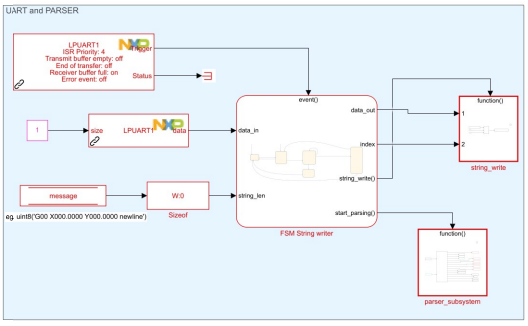
3.2 Panoramica

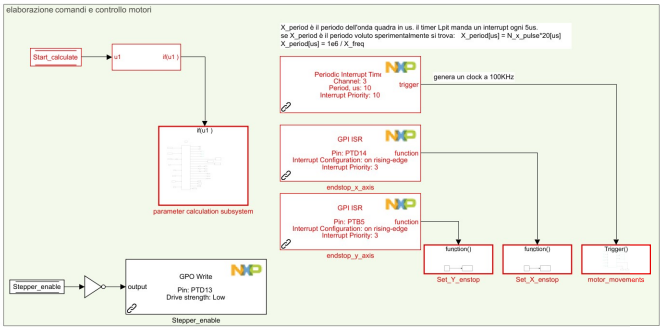
La versione finale del top level grafico è costituita da due blocchi di controllo e due di debug:

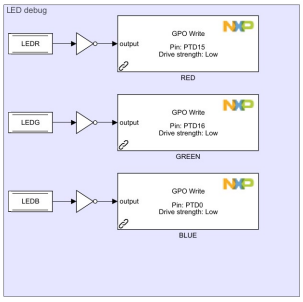
- **UART and PARSER**: legge i singoli caratteri mandati tramite porta seriale e attraverso una macchina a stati li salva su una stringa. Quest'ultima viene poi analizzata, una volta completata, da una funzione dedicata per estrarre informazioni sul comando da effettuare.
- **Elaborazione comandi e controllo motori**: con una funzione Matlab e le informazioni fornite dal parser aggiorna i parametri necessari per il controllo dei motori; questi vengono poi pilotati tramite una macchina a stati attuata dall'interrupt di un timer hardware.
- **LED debug**: a ogni LED è associata una *data store memory*, che contiene un valore booleano che determina se accendere o meno il particolare colore. Questo è utile per determinare in che stato si trova il sistema.
- **UART debug**: tramite i due pulsanti presenti sulla scheda di sviluppo, vengono mandati i contenuti delle *data store memory* delle quali si vuole avere informazioni visualizzandoli su Tera Term.

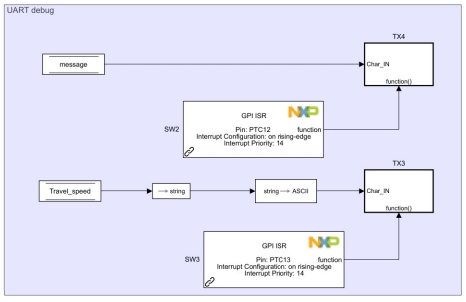

Configurazione iniziale
Target : S32K144 48KBSRAM
Package : 100-LQFP
System clock : 80 MHz
XTAL clock : External 8 MHz
Compiler : GCC
Target Type : FLASH
Download Code after build : (OpenSDA.D)
Step Tick Interrupt Priority : 15
LPUART Config
Instance : 1
Baud rate : 9600
Periodic Interrupt Timer
Channel: 3
Counter: Start
Periodic Interrupt Timer
Channel: 3
Interrupt: on
PINOUT:
X_step = PTB8;
Y_step = PTB9;
Z_step = PTB10;
X_dir = PTB11;
Y_dir = PTC10;
Z_dir = PTC11;
Stepper_enable = PTD13(attivo basso)
X_endstop = PTD14;
Y_endstop = PTB5;
tick di sistema 0.003s
delay invio caratteri python 0.008s


Variabili
parser
X_coordinate
[mm]
Y_coordinate
[mm]
Travel_speed
[mm/s]
Laser_power
Feed_rate
Pause_millis
Send_char
Action
command
message
controllo motori
X_step
Y_step
Z_step
N_x_pulse
N_y_pulse
N_z_pulse
Stepper_enable
X_endstop
Y_endstop
X_last
Y_last
Z_last
semafori
Start_calculate
Go
Done
Go_Z
Homing
stringhe debug
debug1
debug2
DBG1
DBG2
LED debug
LEDR
LEDG
LEDB


UART and PARSER
LPUART1
ISR Priority: 4
Transmit buffer empty: off
End of transfer: off
Receiver buffer full: on
Error event: off
trigger
Status
event()
data_out
index
string_write()
start_parsing()
function()
string_write
FSM String writer
parser_subsystem
size
LPUART1
data
data_in
message
W:0
Sizeof
string_len
eg. uint8('G00 X000.0000 Y000.0000 newline')


elaborazione comandi e controllo motori
X_period è il periodo dell'onda quadra in us. Il timer Lpit manda un interrupt ogni 5us.
se X_period è il periodo voluto sperimentalmente si trova: X_period[us] = N_x_pulse*20[us]
X_period[us] = 1e6 / X_freq
Start_calculate
u1
if(u1)
parameter calculation subsystem
Periodic Interrupt Timer
Channel: 3
Period, us: 10
Interrupt Priority: 10
trigger
genera un clock a 100KHz
GPI ISR
Pin: PTD14
Interrupt Configuration: on rising-edge
Interrupt Priority: 3
function
endstop_x_axis
GPI ISR
Pin: PTB5
Interrupt Configuration: on rising-edge
Interrupt Priority: 3
endstop_y_axis
function()
Set_Y_enstop
Set_X_enstop
Trigger()
motor_movements
Stepper_enable
GPO Write
Pin: PTD13
Drive strength: Low
output
Stepper_enable


LED debug
LEDR
LEDG
LEDB
GPO Write
Pin: PTD15
Drive strength: Low
output
RED
GPO Write
Pin: PTD16
Drive strength: Low
GREEN
GPO Write
Pin: PTD0
Drive strength: Low
BLUE


UART debug
message
TX4
Char_IN
function()
GPI ISR
Pin: PTC12
Interrupt Configuration: on rising-edge
Interrupt Priority: 14
SW2
Travel_speed
--> string
string --> ASCII
TX3
Char_IN
function()
GPI ISR
Pin: PTC13
Interrupt Configuration: on rising-edge
Interrupt Priority: 14
SW3

3.3 Controllo e attuazione dei motori

Questo macroblocco (fig 2) viene attivato dal semaforo *Start_calculate*, che viene settato dal parser una volta finita la sua esecuzione. Quindi partendo dalle coordinate assolute e dal comando corrente, attua il movimento dei motori. Inizialmente la funzione *Calculate_step* determina i parametri per pilotare i driver degli stepper, e successivamente mediante i semafori *Go, Go_z, Homing* attiva la macchina a stati *FSM_motor* la quale genera la sequenza di impulsi a frequenza opportuna da inviare al driver.

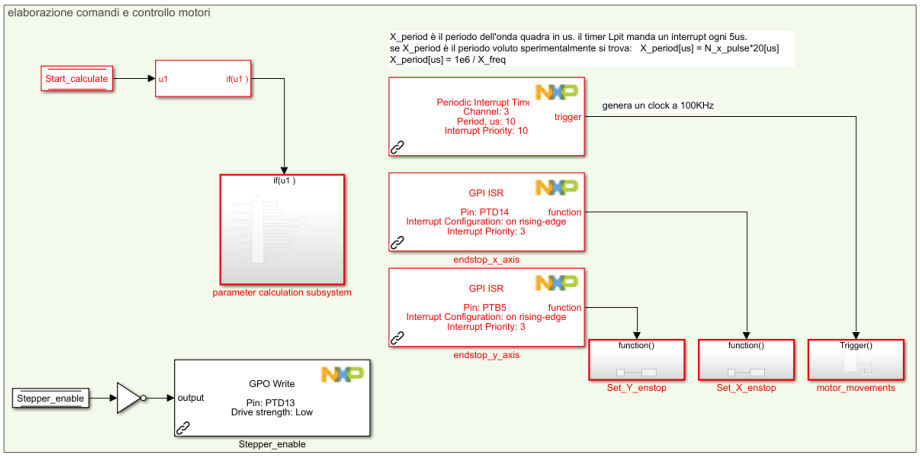


Figure 2: macroblocco controllo motori

La funzione *Calculate_step*, realizzata in Matlab, calcola la frequenza e il numero di impulsi da inviare ai due motori per andare nella posizione voluta. Questo è stato fatto ipotizzando di avere una velocità di avanzamento del laser/pennarello costante a prescindere dal movimento, la quale viene decisa dal parametro *Travel_speed* proveniente dal Gcode. Partendo dai parametri *Step_per_revolution* e *Revolution_per_mm* dipendenti dall'hardware, e tenendo conto della scelta di usare *Half_microstep* per i driver, si ottiene:

$$X_{rel} = X - X_{last}$$
$$Y_{rel} = Y - Y_{last}$$

$$X_{step} = 2\,|X_{rel}| * Step\_per\_revolution * Revolution\_per\_mm$$

Dovendo spostarsi a velocità costante:

$$Travel\_time = \frac{\sqrt{X\_rel^2 + Y\_rel^2}}{Travel\_speed}$$

$$X\_period = \frac{Travel\_time}{X\_step}$$

E utilizzando il timer *LPIT* impostato per generare un interrupt ogni 10μs, il numero di interruzioni da contare per ottenere il periodo voluto sarà:

$$N\_x\_pulse = \frac{X\_period}{2 * 10e - 6}$$

Una volta determinati tali valori, tramite il semaforo *Go*, viene fatta partire la macchina a stati *FSM_motor* (fig.3), la quale esegue in maniera parallela due sequenze di stati per muovere contemporaneamente i motori X e Y e riuscire così a disegnare linee di angolazione arbitraria. Infine una volta terminato il movimento, viene inviato sulla UART il carattere '*' che indica allo script Phyton di inviare la successiva riga di Gcode.
Questa macchina a stati ha come trigger il segnale di overflow del timer *LPIT*, che pertanto determina la quantizzazione temporale della forma d'onda generata. Per avere una buona risoluzione frequenziale tale intervallo deve essere il più piccolo possibile, tuttavia deve essere sufficientemente grande da permettere al sistema di effettuare le operazioni previste all'interno dei vari stati prima che arrivi il trigger successivo. La scelta finale è ricaduta su un periodo di 10us, che permette teoricamente di avere una frequenza max di 50Khz (N_x_pulse = 1), e una risoluzione temporale di $20/mus$.

Un sistema analogo è utilizzato per muovere l'asse Z, partendo però da comandi diversi.

Per implementare la funzionalità di *Homing*, necessaria per riferire la macchina a delle coordinate assolute, viene effettuato il movimento dei due assi in direzione negativa finchè viene toccato il pulsante di finecorsa, letto in maniera asincrona tramite interrupt, che ferma il movimento. Avendo scelto di attivare l'interrupt sul fronte del segnale, per evitare problemi meccanici legati all'eventuale esecuzione di due *Homing* successivi, è stata utilizzata l'accortezza di spostare i due assi di qualche mm per scostarsi dal pulsante. Dal momento che non è stato previsto un endstop sull'asse Z, è previsto che all'avvio di un disegno la punta del pennarello tocchi il foglio sul quale disegnare. Infatti la *data store memory* della posizione dell'asse Z è già inizializzata a 255 (valore massimo consentito, essendo un `uint8`), così che un comando per alzare il pennarello (per esempio `M03 S0`). L'escursione completa sull'asse Z è di circa 2mm.

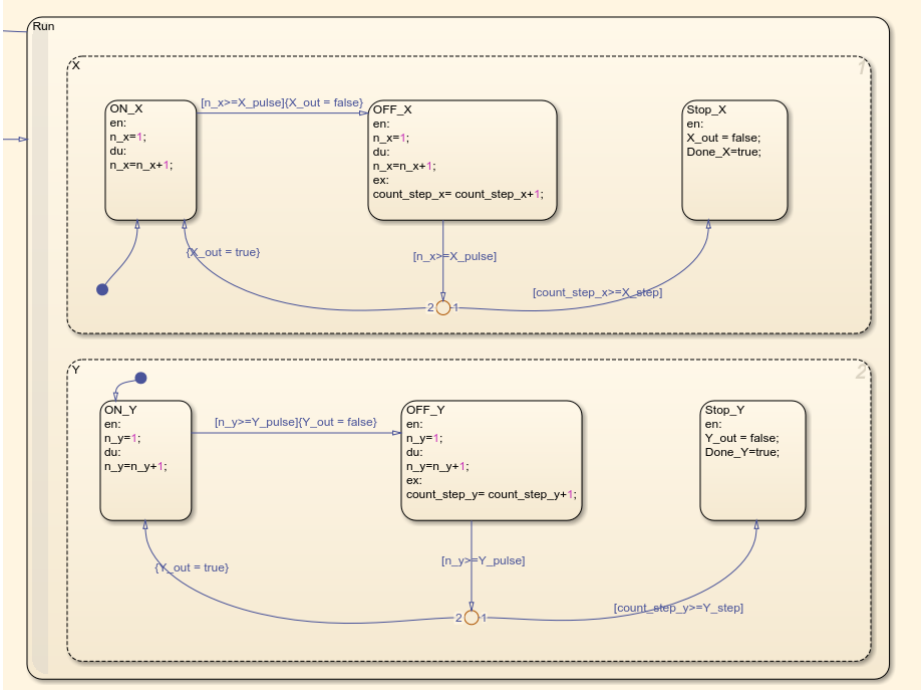


Figure 3: Parte di *FSM_motor*

3.4 Interfaccia di comunicazione (UART and parser)

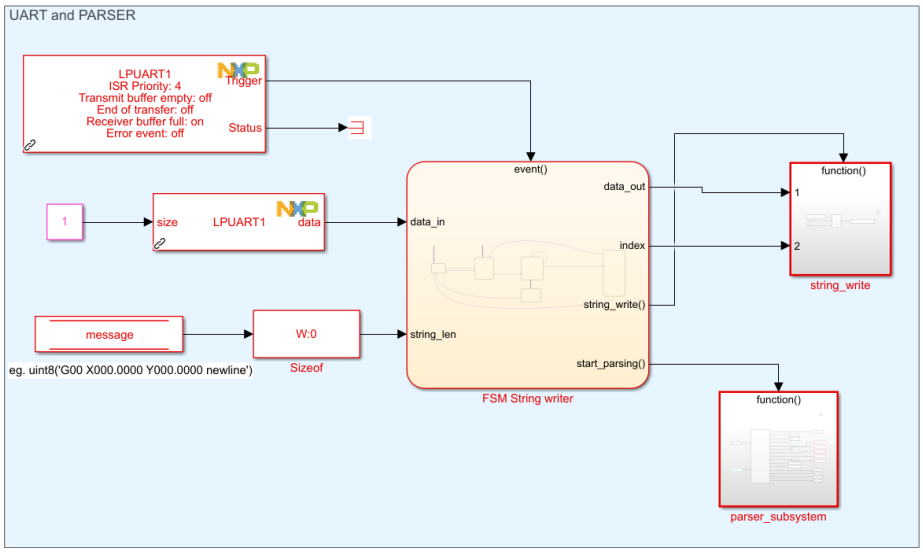


Figure 4: Panoramica del blocco di ricezione ed elaborazione del Gcode.

3.4.1 Introduzione

Avendo dovuto scartare, perché troppo complesso rispetto alla scala del progetto, la lettura del Gcode da scheda SD, è stato necessario trovare un modo alternativo per comunicare con la scheda di sviluppo che fosse semplice e già supportato dai blocchi forniti dalla NXP disponibili su Simulink. La soluzione naturale consisteva nell'uso della porta seriale, in quanto è stata già affrontata anche a lezione (e quindi tutto il software necessario lato computer era già presente).

3.4.2 Configurazione UART

Per implementare la comunicazione seriale, sono stati impiegati tre blocchi NXP:

- **LPUART_Config**: è il blocco di configurazione della UART, dove è possibile specificare quale istanza delle tre disponibili usare, il baud rate (impostato al valore standard di 9600, relativamente basso, ma che non costituisce il bottleneck del sistema come spiegato in seguito), e altri parametri del protocollo quali il numero di bit dati, bit di parità e di stop.
- **LPUART_Receive**: prende in ingresso una costante (*size*) che imposta il numero di caratteri da ricevere per comunicazione, e scrive in uscita (*data*) i valori ricevuti. Avendo voluto implementare la possibilità di poter leggere un Gcode generato automaticamente (così come lo è quello creato

da numerosi script realizzati per Inkscape, dedicati agli incisori laser), si è reso necessario poter leggere righe di lunghezza variabile. Di conseguenza è stato deciso di leggere un carattere alla volta, impostando *size* pari a 1, così da non avere il vincolo di ricevere un comando di lunghezza fissa. Una volta che il buffer riceve *size* caratteri, viene abilitata la richiesta di interrupt, che verrà eseguita da `LPUART_RxTx_ISR`.

- **LPUART_RxTx_ISR**: permette l'esecuzione di sottoblocchi specifici tramite interrupt. Può essere configurato per quattro eventi possibili, quello impiegato in questa configurazione è quello di *Receiver buffer full*. La sua attivazione manda avanti la macchina a stati che si occupa di salvare la stringa da successivamente analizzare, *FSM string writer*.

3.4.3 FSM string writer

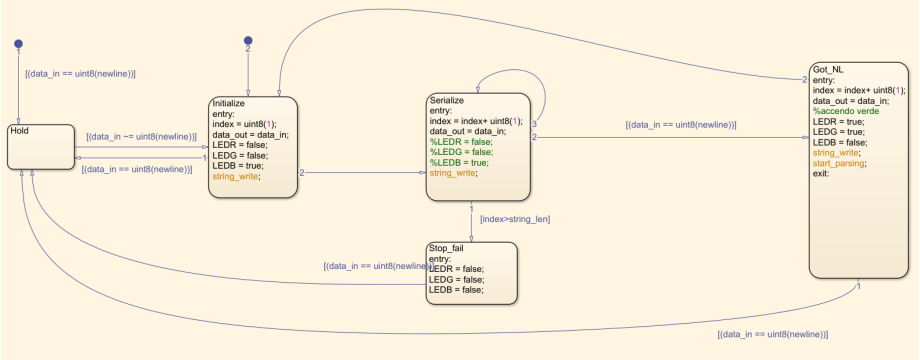


Figure 5: FSM string writer

Data la decisione di ricevere un carattere alla volta, si è rivelato necessario implementare un sistema di controllo di scrittura del messaggio, così da permetterne una corretta analisi successiva. Dato che ogni riga di Gcode termina con un carattere di newline(`\n`), questo viene usato come segnalatore per il completamento dell'operazione di scrittura.

Il messaggio viene scritto in una variabile, `message`, che è un vettore di lunghezza predeterminata (25 caratteri, salvati come `uint8`). Nel caso il messaggio in trasmissione risultasse più lungo di ciò, è previsto uno stato di errore, che porta a scartare l'intero risultato della stringa. Per il salvataggio in memoria viene usato un blocco simulink, `assignment`, che permette di scrivere il dato voluto all'indice desiderato di un vettore, ovvero `message`.

Riassumendo dunque, la scrittura del messaggio inizia con il primo carattere utile, e termina con la ricezione di una *newline*, controllando che non venga scritto un messaggio più lungo del buffer atto a contenerlo. Quest'ultima azione porta l'avvio del parsing della stringa.

3.4.4 Python

Volendo poter processare un intero file in Gcode, avendo scartato la soluzione tramite scheda SD, si è reso necessario automatizzare il processo di scrittura delle istruzioni tramite porta seriale, non facendo più affidamento su una soluzione manuale tramite Tera Term. Per fare ciò è stato creato un programma in Python, `python_scribe.py`, che si occupa di mandare i singoli caratteri tramite porta seriale (tramite il modulo `pyserial`).

Per mandare il carattere di newline a fine di una riga, che comporta l'inizio dell'elaborazione del comando appena mandato, il programma aspetta che l'ultima azione sia terminata. Per fare ciò il microcontrollore manda, alla fine di ogni movimento o cambio di impostazione sempre tramite seriale, un asterisco (`*`). Tramite questo script è stato inoltre possibile apprezzare a pieno il maggiore lim-

ite della soluzione impiegata per leggere la stringa, ovvero la lettura dei singoli caratteri tramite UART e successiva scrittura su una variabile tramite macchina a stati: dal momento che il blocco di lettura da porta seriale (`LPUART_Receive`) è aggiornato con il tick di sistema e non dentro un blocco di interrupt (come verrà approfondito tra poco), non è possibile mandare i caratteri a una frequenza maggiore di quella del tick, in quanto ciò avvierebbe la `FSM string writer` tramite interrupt, ma quest'ultima leggerebbe un valore, proveniente dal blocco di ingresso seriale, non ancora aggiornato dal tick di sistema. Con un metodo *trial and error* è stato appurato che la lettura viene consistentemente eseguita in maniera corretta con un tick di sistema pari a 0.005 secondi (200 Hz), e un tempo di attesa sullo script python di 0.007 secondi (143 Hz) tra un carattere e il successivo. Collaudando, è risultato un tempo di attesa tra un comando e il successivo relativamente accettabile, specialmente considerando che nel caso di un movimento lungo c'è una buona probabilità che la stringa sia già stata inviata e manchi solo da mandare l'invio (che corrisponde al carattere `\n`).

Il principale inconveniente si è manifestato nel voler impiegare la CNC come strumento musicale, usando la frequenza udibile dei motori e modulando la velocità di movimento per ottenere le note desiderate (e calcolando opportunamente la distanza da percorrere per ottenere la durata desiderata delle note): infatti in questa modalità è spesso richiesto di cambiare, come intuibile, la velocità di movimento, pertanto per ogni nota vanno inviati due comandi, uno per la tonalità (usando `G1 F...`) e uno per la durata (impiegando `G1 X... Y...`); questo comporta il non poter mai avere tutto il messaggio inviato prima dell'esaurimento del movimento, portando a una maggiore durata delle pause tra un comando e un altro. Questo ha permesso di realizzare comunque melodie, anche se solo quelle con un tempo necessariamente rilassato. Per realizzare questa funzione è stato realizzato un ulteriore programma in Python, che dato un file che codifica uno spartito con una convenzione realizzata *ad hoc* per questo sistema, genera sul momento il codice Gcode da mandare alla CNC, aspettando a mandare i comandi dove sono previste le pause.

3.5 Interpretazione Gcode (UART and parser)

Il Gcode è il linguaggio standard utilizzato nelle macchine a controllo numerico, viene generato dai software CAM(Computer Aided Manufacturing), e contiene una serie di istruzioni eseguite in sequenza dalla macchina che ne controllano il comportamento.
Queste si dividono in comandi istantanei, preceduti dalla lettera 'G', e comandi modali, identificati dalla lettera 'M' i quali attivano un comportamento che persiste finché tale comando non viene modificato.
Di seguito alcuni dei comandi supportati dal sistema:

- `G1  X104.5242 Y22.7892       ;comando istantaneo`

  Effettua uno spostamento con la velocità preimpostata(feedrate) verso le coordinate indicate.

- `G1 F10.00`

  Modifica la velocità di avanzamento(feedrate) impostandola a 10 mm/s.

- `M03 S255             ;comando modale`

  Attiva il laser con potenza massima (nel nostro caso abbassa l'asse Z per disegnare).

Nell'ottica di rendere il sistema compatibile con un generatore universale di Gcode, è stato fatto in modo che ulteriori comandi presenti in un Gcode generico (G90, G4, G21...) vengano comunque accettati dal parser senza che il funzionamento della macchina si interrompa. Tuttavia vista la flessibilità del parser questi potranno essere sempre implementati in un secondo momento con poca difficoltà.

3.5.1 Parser mastermind

L'interpretazione è affidata alla funzione *parser mastermind*, interna al blocco *parser_subsystem*. Data una generica stringa come una di quelle riportate precedentemente, la funzione si occupa di dividere il messaggio salvando i contenuti in tre variabili, ovvero `parsed_command`, `parsed_first_message` e `parsed_second_message`:

- `parsed_command` contiene la stringa del comando da eseguire (per esempio `G1`
- `parsed_first_message` può essere, a seconda del comando in questione, la coordinata X da raggiungere, la velocità di movimento desiderata, o la posizione sull'asse Z
- `parsed_second_message` è impiegato solamente se il comando è `G1`, e contiene la coordinata Y voluta.

La lettura sfrutta due puntatori, uno riferito a un carattere del messaggio salvato da leggere, e un altro che indica la posizione del carattere da scrivere su una delle tre variabili sopra citate. Sapendo che in una riga i tre parametri sono separati tra di loro da uno spazio, la scrittura dei parametri si riduce a scorrere il primo puntatore lungo i caratteri della stringa salvata, e salvare opportunamente i valori nelle tre variabili. Una volta che viene letto una newline (ovvero `\n`) il primo puntatore cessa di avanzare, e inizia l'analisi dei tre *data store memory*. Dato che `G1` può indicare sia uno spostamento che un aggiornamento della velocità di movimento, viene effettuato un controllo sul primo carattere di `parsed_first_message` che permette di discriminare correttamente l'azione da eseguire. In seguito, i valori registrati vengono salvati come `double` che saranno poi usati dalla funzione `calculate_step`.

4 Collaudo e validazione

Durante l'avanzamento del progetto i vari moduli sono stati testati separatamente utilizzando dei testbench e dei metodi di debug per verificarne la correttezza funzionale e individuare eventuali casi critici che avrebbero portato a mal funzionamenti.

4.0.1 Salvataggio stringa e interpretazione

La verifica della parte di lettura da UART e interpretazione dei comandi ha fatto grande affidamento sui due blocchi di debug inseriti nel sistema, basati sull'impiego dei LED presenti sulla scheda e della comunicazione seriale. La macchina a stati (*FSM string writer*) è stata verificata accendendo una combinazione univoca dei LED in ogni stato, permettendo la conferma del corretto funzionamento una volta ultimata. Inoltre mandando i comandi, inseriti manualmente su Tera Term, è stato possibile verificare che tutte le stringhe inviate venissero correttamente salvate nella variabile `message` e che venissero correttamente parsate nella funzione `parser mastermind` interna al blocco `parser_subsystem` (andando, di volta in volta, a variare i *data store memory* da visualizzare alla pressione dei due pulsanti.
Lo svantaggio principale della soluzione impiegata risiede nel dover ricompilare il progetto ogni volta che si desidera cambiare quale variabile deve essere visualizzata tramite seriale: questo ha portato a una maggiore lentezza in fase di validazione, ma ha permesso di realizzare il sistema in poco tempo, e prevedendo di non dover variare molto spesso i comandi da visualizzare, è stato preferita una realizzazione più veloce della soluzione di debug a scapito della facilità di modificarne le uscite. A fine progetto questa scelta è risultata valida, anche se una realizzazione più complessa del sistema di debug non avrebbe richiesto una quantità di tempo notevole, nel caso fosse stato necessario visualizzare più variabili nella stessa esecuzione del programma.

4.0.2 Limite lettura da seriale

Come già detto, la principale limitazione consiste nell'avere la frequenza di lettura limitata superiormente dalla frequenza del tick di sistema in quanto il blocco `LPUART_Receive` è nel livello gerarchico superiore, mentre la `FSM string writer` è comandata tramite interrupt di ricezione da seriale. La soluzione più immediata sarebbe quella di creare un blocco comandato dall'interrupt già accennato, nella quale includerci sia la macchina a stati sia il blocco di ricezione, così da aggiornare la variabile in ingresso contemporaneamente con l'interrupt. Purtroppo non è stato possibile implementare questa soluzione naturale, semplice, veloce, immediata e intuitiva, in quanto l'inserimento del blocco NXP dentro il sottosistema comandato da interrupt comporta un fallimento totale della lettura da seriale. Per confermare ciò, sono state realizzati due programmi di esempio dove la ricezione di un 1 o di un 2 tramite seriale comportava l'accensione di un LED blu o rosso, ed è stata tentata un'analisi del codice sintetizzato nelle

due configurazioni, ovvero con la lettura della UART dentro o fuori dal blocco azionato dall'interrupt.

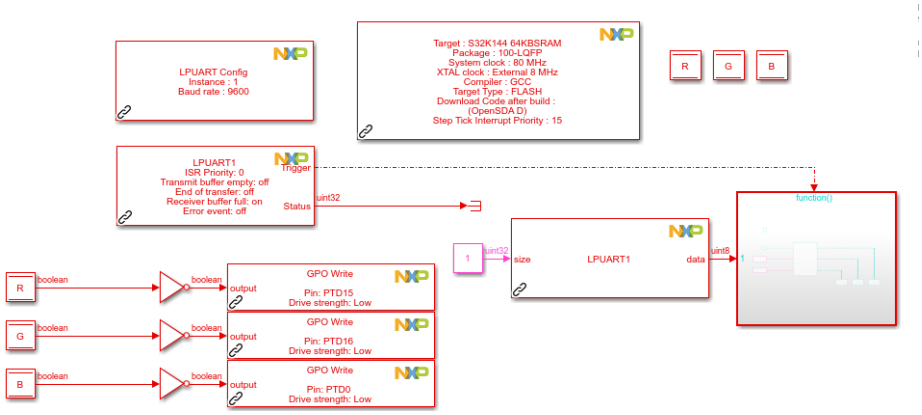


Figure 6: blocco fuori da sottosistema comandato da interrupt

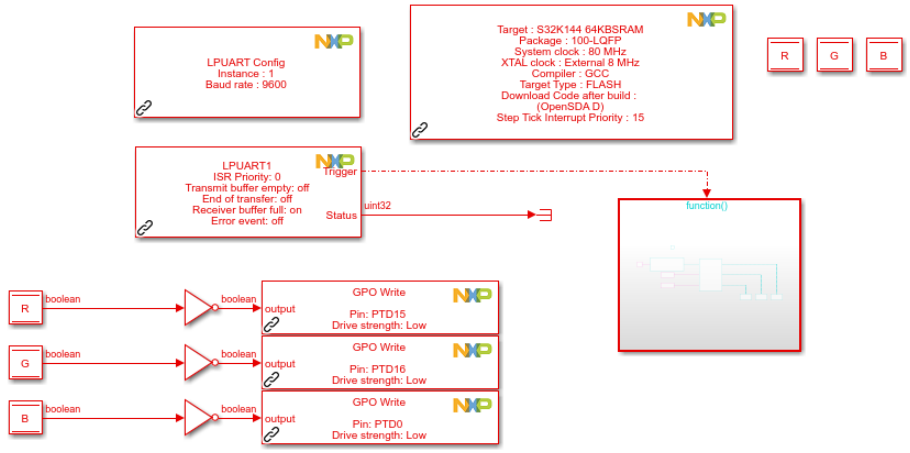


Figure 7: blocco dentro sottosistema comandato da interrupt

La differenza principale risiede nel collocamento della funzione `LPUART_DRV_ReceiveData()` che in un caso viene collocata[1] dentro la funzione `LPUART1_RxTx_callback()` (ovvero la routine di interrupt dell'uart), nell'altro[2] è interna a `test_led_step()`, che è la funzione comandata dal tick di sistema. Dal momento che, almeno apparentemente, la sintesi sembra sia stata eseguita correttamente, risulterebbe necessario approfondire il funzionamento della funzione `LPUART_DRV_ReceiveData()`,

[1] riga 33 di *led UART dentro interrupt.c*
[2] riga 118 di *led UART fuori interrupt.c*

cosa che non è stata valutata necessaria da compiere in quanto non è di banale difficoltà e non avrebbe comunque migliorato la soluzione attuale.

Il corretto funzionamento di `python_scribe.py` è stato verificato mandando l'uscita dello script direttamente a schermo su Tera Term. Per fare ciò, è stato installato un driver per creare coppie di porte seriali virtuali, `com0com`[3]. In questo modo, è stato relativamente facile emulare la conclusione delle azioni come lo farebbe il microcontrollore, mandando l'asterisco come già descritto in precedenza.

4.0.3 Generazione impulsi di pilotaggio dei motori

Per la parte relativa al controllo motori è stato utilizzato un oscilloscopio per osservare gli impulsi generati, a tale scopo è stata creata una piccola macchina a stati che alla pressione del pulsante sulla scheda NXP inviasse a *FSM_motor* dei comandi corrispondenti alla generazione di burst di lunghezza e frequenza nota.
In tale occasione è stato osservato che per piccoli valori di *N_x_period* la frequenza generata differiva da quella teorica, tuttavia questo non è stato un problema in quanto analizzando l'andamento della frequenza al variare di *N_x_period*, fig 8, si vede che è per avere un errore relativo minore del 2.5% tra due frequenze successive, è necessario utilizzare valori di *N_x_period* maggiori di 40. Questo è inevitabile se si vuole riuscire a disegnare linee diagonali con una certa risoluzione, tuttavia comporta avere l'eccitazione degli stepper in una gamma frequenziale all'interno della banda audio, con conseguente rumore generato.

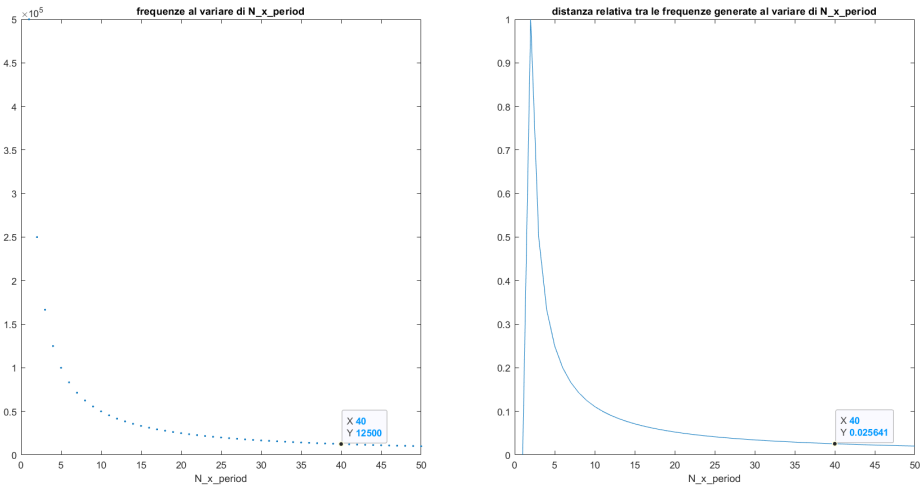


Figure 8:

[3]http://com0com.sourceforge.net/

4.0.4 Codice di prova

Per testare il funzionamento della macchina sono sono stati creati vari disegni vettoriali, utilizzando il programma open source *Inkscape*, il quale tramite il plugin *J Tech Photonic Laser Tool* trasforma un tracciato nel Gcode contenente le varie istruzioni.

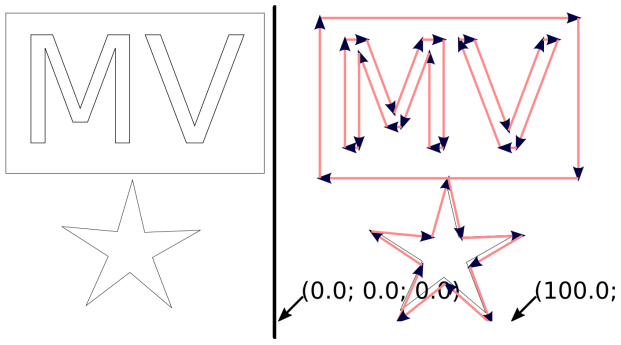


Figure 9: A sinistra il disegno vettoriale di partenza. A destra il percorso utensile scelto dal plugin per generare il Gcode

Gcode corrispondente al disegno in figura 9 :

```
M03 S0

G90
G21
G1 F15
G1  X51.5938 Y0.3214
G4 P0
M03 S255
G4 P0
G1 F10.000000
G1  X61.9764 Y24.2779
G1  X40.2558 Y38.7667
G1  X66.2481 Y36.2952
G1  X73.3158 Y61.4299
G1  X78.9973 Y35.946
G1  X105.0859 Y36.9914
G1  X82.605 Y23.713
G1  X91.661 Y-0.7758
G1  X72.0855 Y16.5017
G1  X51.5938 Y0.3214
G4 P0
M03 S0
G1 F15
G1  X95.6354 Y74.5937
G4 P0
M03 S255
G4 P0
G1 F10.000000
G1  X77.962 Y120.8855
G1  X84.5043 Y120.8855
G1  X99.17 Y81.9111
G1  X113.8668 Y120.8855
G1  X120.378 Y120.8855
G1  X102.7357 Y74.5937
G1  X95.6354 Y74.5937
G4 P0
M03 S0
G1 F15
G1  X28.9107 Y120.8855
G4 P0
M03 S255
G4 P0
G1 F10.000000
G1  X38.2435 Y120.8855
G1  X50.0567 Y89.3835
G1  X61.932 Y120.8855
G1  X71.2648 Y120.8855
G1  X71.2648 Y74.5937
G1  X65.1566 Y74.5937
G1  X65.1566 Y115.2424
G1  X53.2193 Y83.4924
G1  X46.9252 Y83.4924
G1  X34.9879 Y115.2424
G1  X34.9879 Y74.5937
G1  X28.9107 Y74.5937
G1  X28.9107 Y120.8855
G4 P0
M03 S0
G1 F15
G1  X18.1429 Y130.1562
G4 P0
M03 S255
G4 P0
G1 F10.000000
G1  X129.0788 Y130.1562
G1  X129.0788 Y61.5536
G1  X18.1429 Y61.5536
G1  X18.1429 Y130.1562
G4 P0
M03 S0
G1 F15
G1 X0 Y0
```

Risultato ottenuto dal plotter:

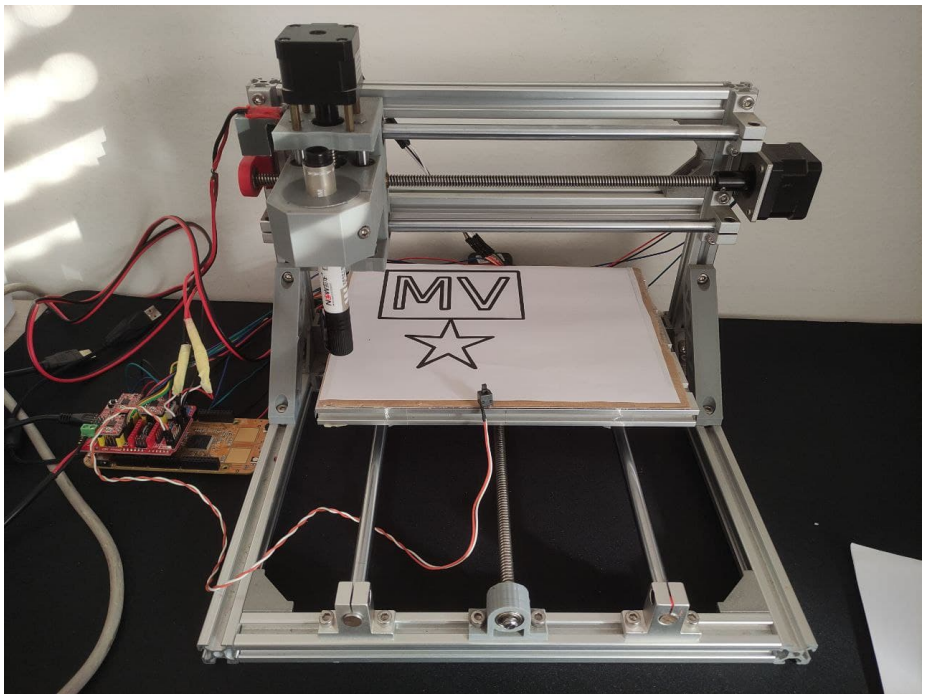


Figure 10:

5 Conclusioni

5.1 Punti di forza

- L'impiego del un timer hardware *LPIT* invece di quello dedicato al PWM ha permesso di collocare i pin di uscita a piacere, rendendo possibile piazzare la CNC shield sopra la scheda di sviluppo.
- Anche con l'assenza di bassi microstep nel controllo dei motori, la qualità dei disegni finali risulta ottima (essendo praticamente limitata dallo spessore del tratto della penna o pennarello impiegato).

5.2 Criticità attuali

- Una velocità maggiore del timer dedicato al controllo dei motori avrebbe permesso di impiegare i microstep più piccoli, aumentando la risoluzione ed riducendo il rumore dei motori.
- La soluzione attuale di leggere i singoli caratteri tramite UART rappresenta l'elemento più critico alla velocità complessiva del sistema.

5.3 Sviluppi futuri

- Introdurre la possibilità di eseguire i comandi `G02` e `G03` utilizzati nei Gcode per descrivere archi, potendo così disegnare figure più complesse.
- Introdurre il display controllato tramite i2c (del quale NXP fornisce i blocchi di configurazione necessari), anche solo per mostrare la stringa in esecuzione o altri parametri del sistema, e successivamente implementare la lettura della scheda SD utilizzando del codice dedicato scritto in c, cosa nativamente supportata da Simulink.
- In un vicino orizzonte temporale risulta più naturale proseguire col miglioramento della comunicazione seriale: una soluzione relativamente semplice per arrivare alla massima velocità di lettura possibile consiste nel far mandare dalla scheda un altro carattere a ogni tick di sistema.In questo modo non è più necessario determinare empiricamente la frequenza massima possibile con il quale mandare il codice tramite python, ma questo compito direttamente alla scheda, che segnala quando è pronta a ricevere un nuovo carattere. Inoltre, modificando maggiormente il sistema e il codice da mandare, l'implementazione della lettura dell'intera stringa piuttosto che del singolo carattere mitigherebbe in maniera praticamente totale il problema.