RELAZIONE PROGETTO DSO

Corso di Integrazione di Sistemi Embedded a.a. 2020/2021

Professore: Massimo Ruo Roch

Studente: Bononi Andrea **Matricola: 287628**

INTRODUZIONE

Il progetto consiste nella realizzazione di un oscilloscopio digitale che soddisfi una serie di specifiche. Il microcontrollore deve campionare i valori in ingresso sui due canali CH0 e CH1 e allo stesso tempo ricevere dei comandi per modificare la modalità di lavoro; questi comandi sono inviati dall'utente per mezzo di una interfaccia grafica.

Le periferiche necessarie sono le seguenti:

- **ADC**: attua la conversione dei segnali analogici in ingresso
- **UART**: necessaria per trasmettere i campioni e per ricevere i comandi
- **TIMER**: è impiegato per temporizzare le conversioni del ADC, in modo da ottenere la frequenza di campionamento desiderata

In aggiunta, nel progetto si è anche impiegato un LED nel seguente modo:

- ogni trasmissione corrisponde ad un lampeggiamento del LED, che permette quindi di avere un riscontro visivo sulla velocità del sistema (oltre all'interfaccia grafica)
- in caso di errore, il LED viene mantenuto acceso e il sistema si porta in un loop infinito vuoto (siamo quindi in grado di accorgerci quando il sistema è in errore)

I parametri di funzionamento delle periferiche sono i seguenti:

- $\mathbf{F_{CLK}}$ **= 84 MHz** per tutte le periferiche.
- $\mathbf{PSC_{TIMER}}$ **= 168** quindi considerando di voler lavorare a frequenze di campionamento nel range (10 Hz ÷ 100 kHz) il *reload value* del timer andrà da 5 a 50000 (la frequenza di default è 1 kHz quindi *reload value* = 500).
- Il convertitore lavora con un **regular channel** (CH0) ed un **injected channel** (CH1) in modalità **auto-injected**: i due canali vengono campionati uno subito dopo l'altro e i risultati sono memorizzati in due data register diversi (ogni injected channel ha un data register dedicato) in modo da scongiurare eventuali overrun. Ogni coppia di conversioni è attivata dal raggiungimento del reload value da parte del timer (single conversion mode, external trigger).

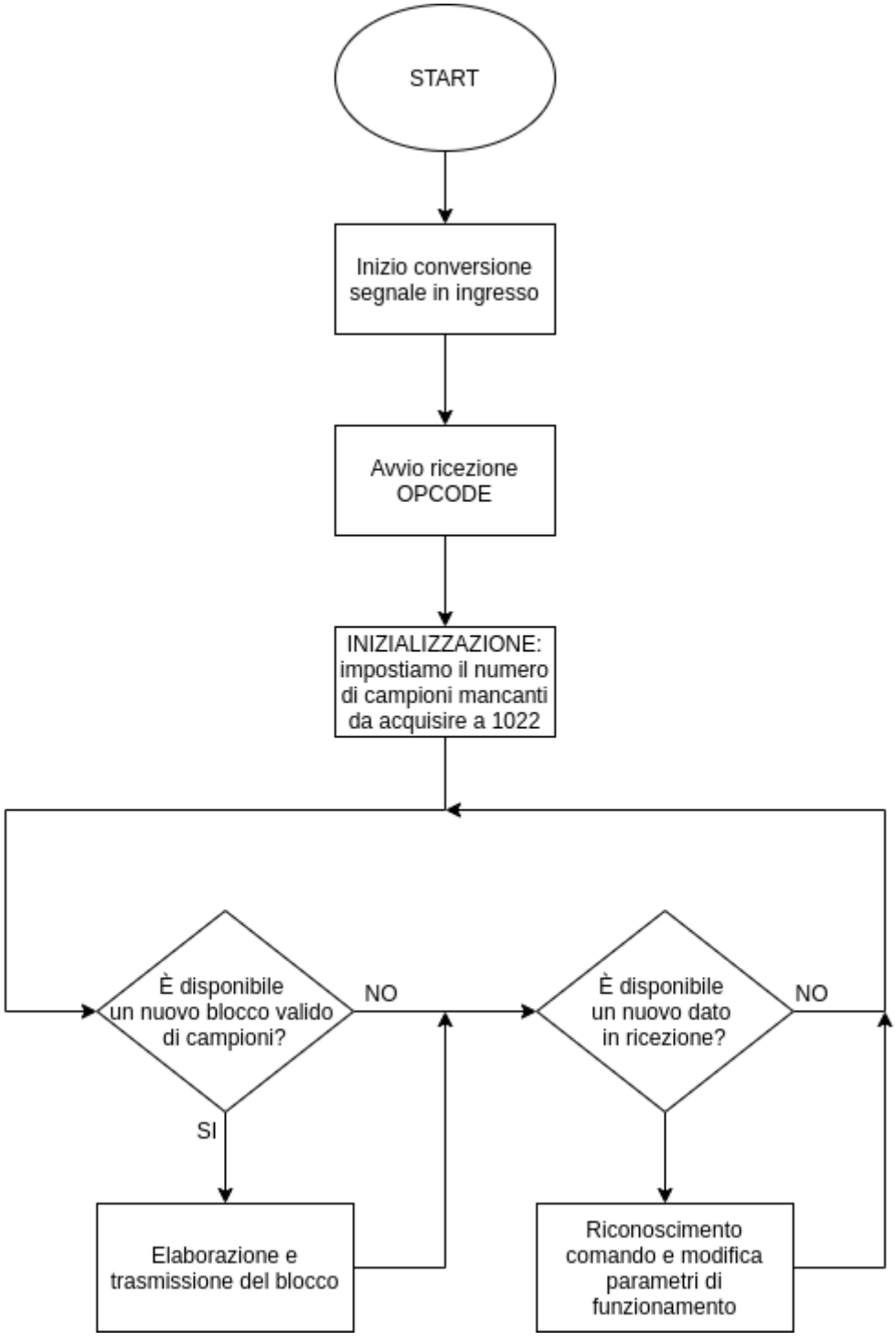
Il sistema funziona interamente in interrupt, con le seguenti priorità:

- ADC: 0
- UART: 10
- TIMER: 5

Il convertitore ha la priorità più elevata, in quanto vogliamo essere sicuri di non perdere dei dati.

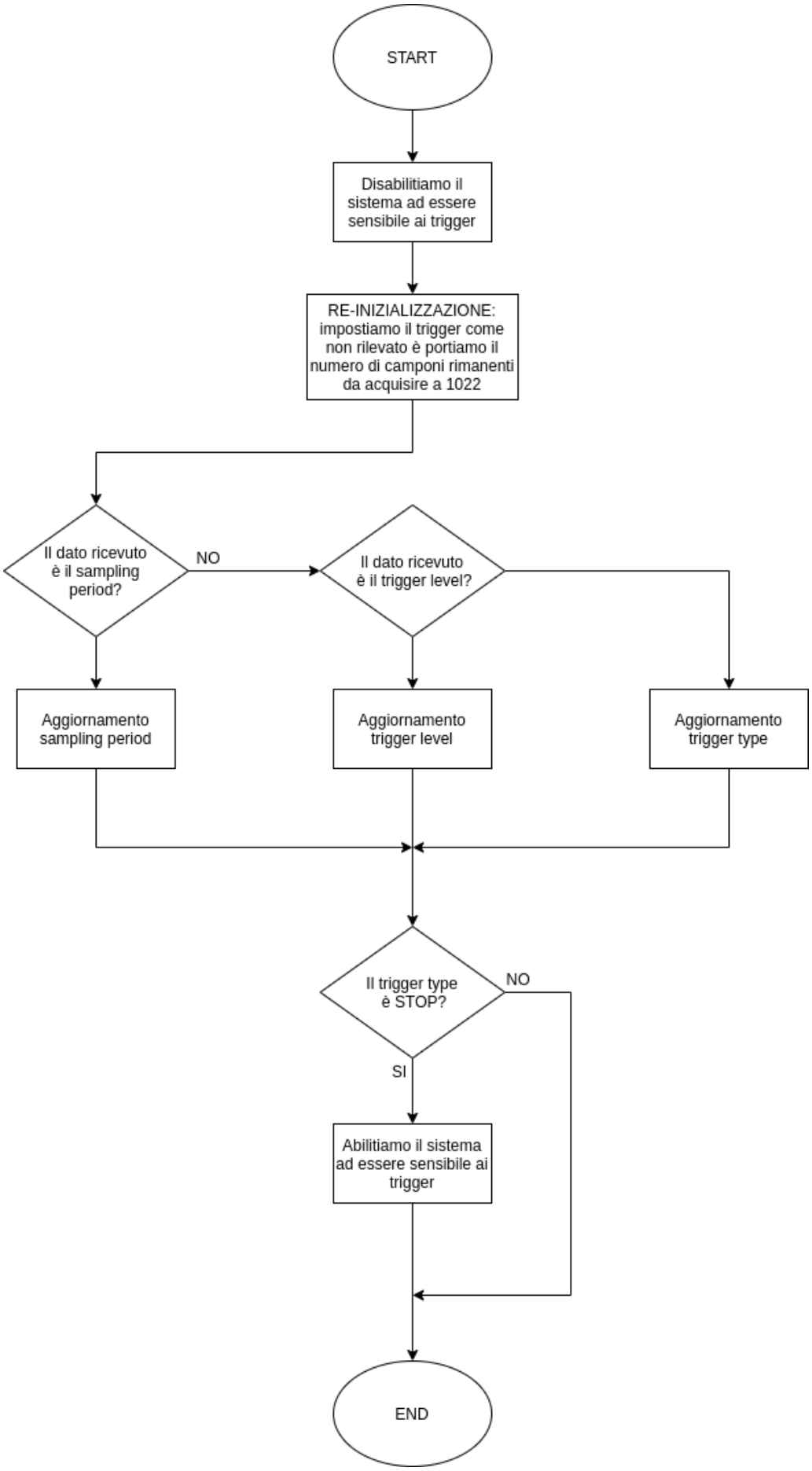
L’invio dei dati dal controllore all’interfaccia grafica avviene a blocchi di 511 campioni per canale, 255 precedenti al trigger e 255 successivi (in totale 1022 campioni + 2 byte di inizio frame '*' e fine frame '#').

FLOWCHART LOOP



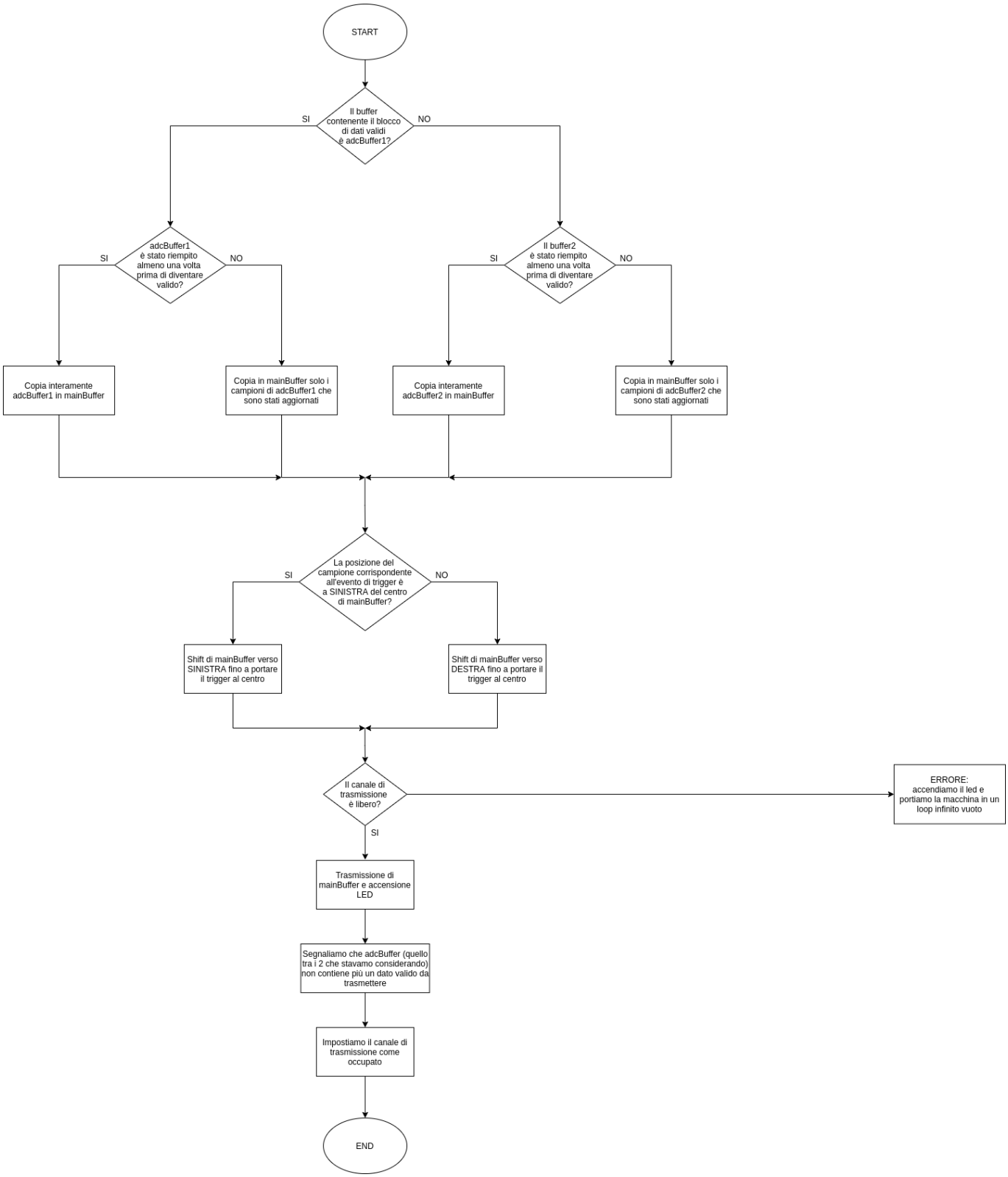
Il loop principale eseguito dal programma controlla in **polling** la presenza di un nuovo blocco di campioni da trasmettere o di un dato in ricezione. Inizialmente è necessario accendere il convertitore e la porta UART.
Gli stati "Elaborazione e trasmissione" e "riconoscimento comando e modifica parametri" sono sviluppati in seguito.

Riconoscimento comando e modifica parametri



Elaborazione e trasmissione del blocco

Il sistema prevede l'utilizzo di due generici buffer **adcBuffer1** e **adcBuffer2** per la memorizzazione dei campioni (indipendentemente da come questi siano realizzati in linguaggio C).

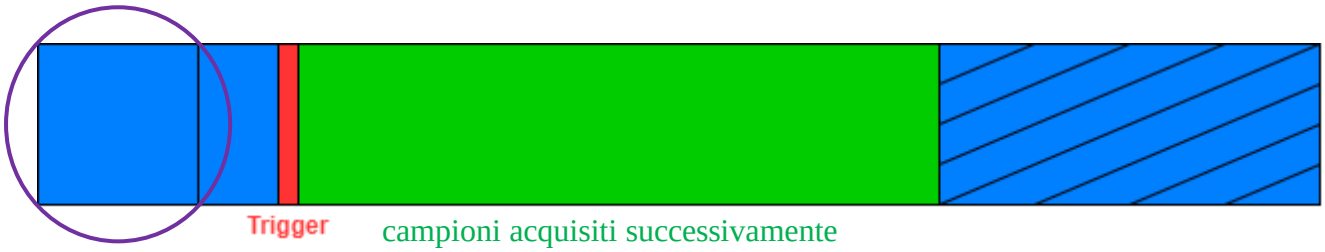


Supponiamo che i dati siano inizialmente salvati in adcBuffer1. Quando il blocco di campioni in esso contenuto diventa valido è necessario un certo tempo per elaborarlo e trasmetterlo, durante il quale non può essere modificato; il sistema deve però continuare a campionare (nonostante non sia più sensibile ai

trigger), quindi è necessario disporre di un altro buffer (in questo caso adcBuffer2) per memorizzare i campioni. Dopo la trasmissione di adcBuffer1, la sensibilità ai trigger viene nuovamente attivata ed il ciclo ricomincia (invertendo però il ruolo dei due buffer). Nel flowchart precedente non è presente la parte di riattivazione del sistema, si trova nel flowchart che fa riferimento all'evento di fine trasmissione (interrupt).

In realtà, adcBuffer1 viene copiato in un terzo buffer (chiamato **mainBuffer**) il quale sarà elaborato e trasmesso; è comunque necessario del tempo per effettuare la copia, bisogna comunque disporre di 2 adcBuffer diversi.
Quando il contenuto di adcBuffer1 diventa valido, adcBuffer2 inizia a riempirsi. Lavorando ad una frequenza di campionamento elevata, durante l'elaborazione e la trasmissione il convertitore può ottenere un gran numero di campioni, arrivando più volte a riempire adcBuffer2 (**buffer circolare**).
Lavorando a basse frequenze è invece possibile che adcBuffer2 arrivi nella condizione di validità senza essersi riempito nemmeno una volta: i campioni nelle locazioni che non sono mai state toccate (che dovrebbero contenere una parte dei dati temporalmente precedenti il trigger) non sono validi. In queste condizioni, i campioni validi si trovano nelle locazioni corrispondenti di mainBuffer, che contiene i dati immediatamente precedenti a quelli di adcBuffer2.
Non possiamo pensare di prendere i campioni validi da adcbuffer1, in quanto questo inizia ad essere modificato nell'esatto istante in cui adcBuffer2 diventa valido (il suo contenuto non è più affidabile).

Il disegno seguente rappresenta adcBuffer2: in verde sono rappresentati i campioni temporalmente successivi al trigger, in blu quelli precedenti. La parte in blu sbarrata rappresenta i dati non validi, che dovrebbero far parte del gruppo precedente il trigger ma di fatto non sono i campioni che vorremmo.

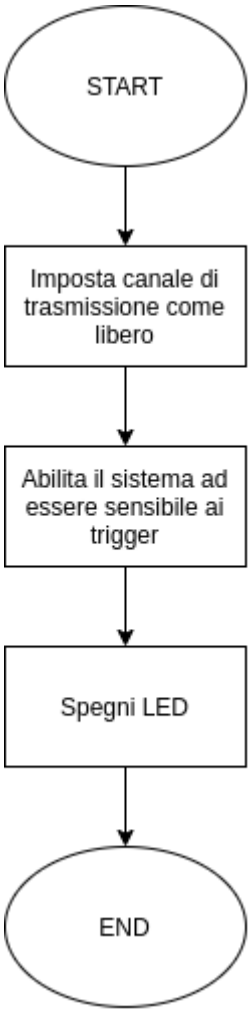


campioni acquisiti durante la trasmissione dei dati di adcBuffer1

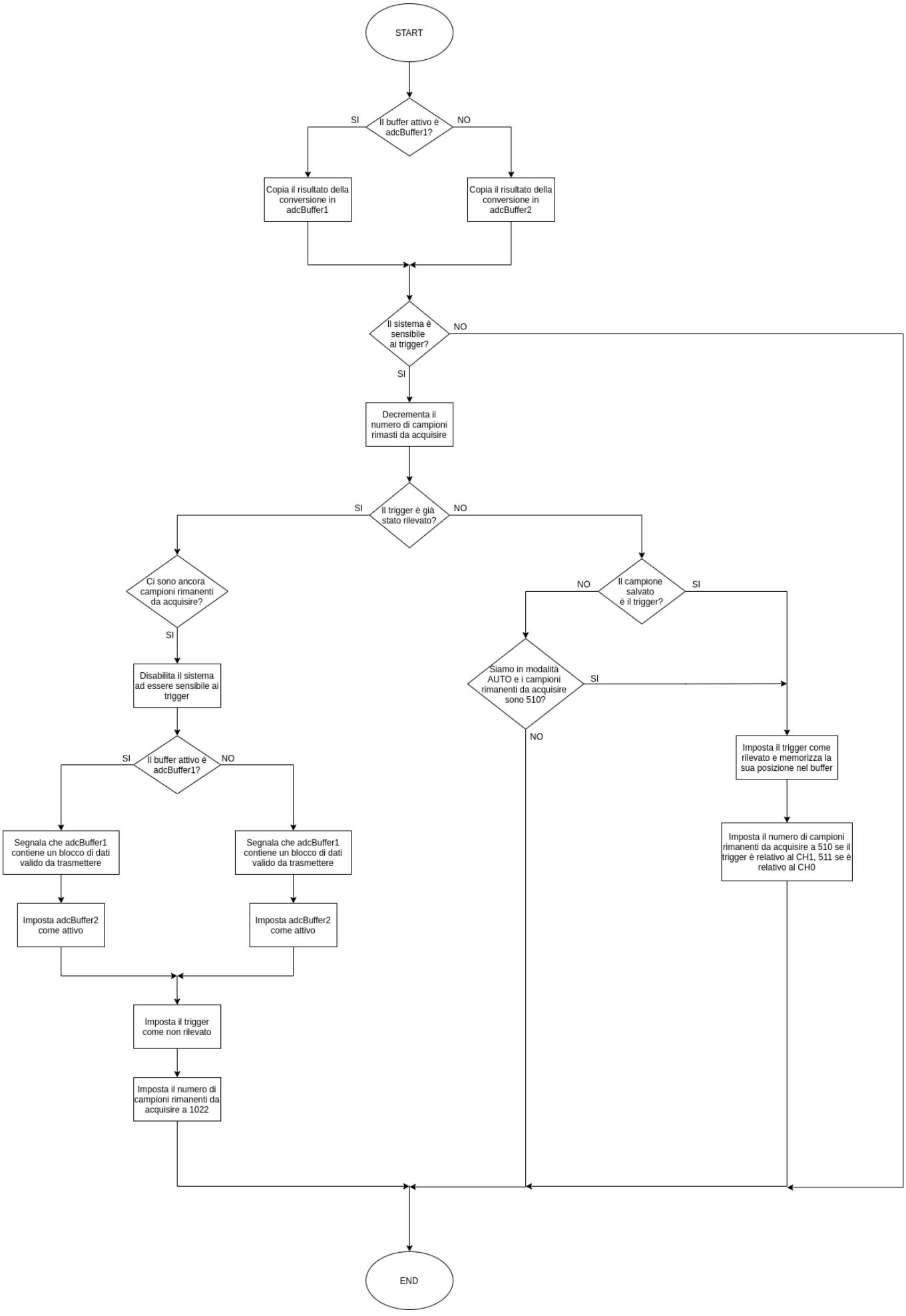
campioni acquisiti successivamente all'evento di trigger (510 o 511)

FLOWCHART DI FINE TRASMISSIONE (interrupt)

Come anticipato, all'arrivo dell'evento di fine trasmissione è necessario riattivare il sistema.
Viene anche spento il LED che era stato acceso all'inizio della trasmissione, creando un lampeggiamento la cui velocità dipende dalla frequenza della trasmissione (baud rate). La frequenza dei vari lampeggiamenti dipende invece dal tempo che trascorre tra una trasmissione e la successiva, che dipende a sua volta dal periodo di campionamento e dal tempo necessario per l'elaborazione del blocco di dati validi. Diminuendo il periodo di campionamento la frequenza di lampeggiamento aumenta, fino a quando il contributo principale diventa il tempo di elaborazione (quindi diminuire il periodo di campionamento non causa più variazioni apprezzabili nella frequenza di lampeggiamento).

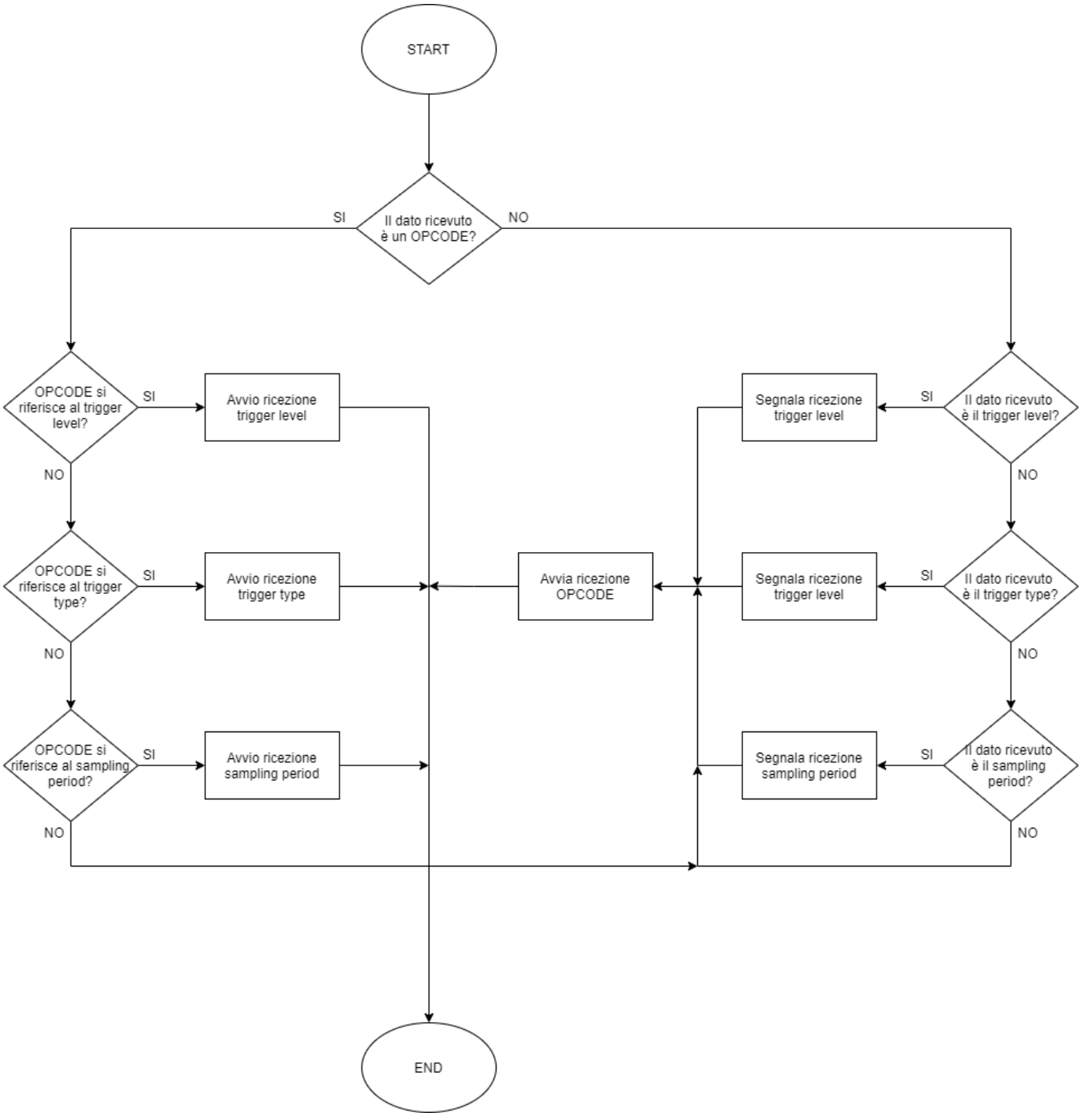


FLOWCHART DI FINE CONVERSIONE (interrupt)



Il flowchart rappresentato è generale, l'algoritmo è lo stesso per il regular channel (CH0) e per l'injected channel (CH1). Bisogna però gestire in modo corretto la copia dei risultati della conversione in adcBuffer, in modo che CH0 salvi i campioni nelle locazioni dispari, CH1 in quelle pari.

FLOWCHART DI RICEZIONE (interrupt)



La ricezione è suddivisa in 2 parti:

- ricezione OPCODE, lunghezza fissa (3 byte)
- ricezione comando, la cui lunghezza dipende dall'opcode precedentemente ricevuto

La ricezione di un dato in formato non valido viene ignorata, il sistema lo scarta e si prepara alla ricezione successiva.

IMPLEMENTAZIONE IN LINGUAGGIO C

Il programma è basato su 4 moduli (source file + header file) in aggiunta a quelli di libreria:

- **DSO_application**: contiene le definizioni di tutte le **callback** di interrupt e del **loop** principale, e sfrutta *funzioni* e *define* contenute nei moduli successivi
- **data_events**: modulo per la gestione degli **eventi** associati ai dati (abilitazione/disabilitazione della sensibilità ai trigger, segnalazione di buffer valido, segnalazione rilevamento trigger…)
- **serial**: modulo per gestire lo stato della trasmissione e della ricezione
- **data_operation**: modulo per la gestione delle **operazioni** di elaborazione sui dati (shift del buffer per portare il trigger al centro)

Il file *main.c* generato da MX include *DSO_application.h* mentre *DSO_application.c* include *data_events.h*, *serial.h* e *data_operation.h*

Esiste anche un file chiamato *hwHandles.h* che include *main.h* ed è incluso in *DSO_application.h* in modo da poter utilizzare gli handles.

INTERFACCIA GRAFICA PYTHON

L'interfaccia grafica è stata realizzata utilizzando i moduli **tkinter** (per l'interfaccia vera e propria) e **matplotlib** (per la gestione del grafico), in aggiunta al modulo **pyserial** per gestire trasmissione e ricezione.

Oltre al grafico, l'interfaccia contiene 4 pulsanti per modificare separatamente i valori di trigger level, trigger type, trigger position e sampling period (associati a 4 caselle di ingresso in cui inserire il valore desiderato). Inserendo dei valori non consentiti, il sistema mostra un messaggio di errore e riporta i parametri ai valori precedenti (di fatto non trasmette nessun comando al microcontrollore).
Per quanto riguarda il sampling period, è possibile selezionare l'unita di misura desiderata (s, ms, us, ns) ma sono consentiti solo valori interi (se per esempio volessimo impostare 1.25 ms dovremmo inserire 1250 us).

NOTA: è possibile che, dopo aver collegato il microcontrollore al PC, avviando per la prima volta il programma python si crei un conflitto sul bus di trasmissione/ricezione, che porta il controllore nello stato di errore (il LED rimane costantemente acceso) e l'interfaccia grafica a non visualizzare nessun segnale nel grafico. Se questo dovesse verificarsi, sarebbe sufficiente chiudere il programma python, resettare il controllore tramite l'apposito pulsante e riavviare l'interfaccia grafica. In generale, se il programma python non visualizza nulla è sufficiente riavviarlo per risolvere il problema (la probabilità che questo problema si verifichi non è comunque troppo alta).

NOTA2: a causa della mancanza di un generatore di segnali, non è stato possibile testare il sistema se non con alcuni segnali molto semplici (costanti, transitori di carica di un condensatore).