Campionatore Segnale Analogico con memoria
Architettura degli Elaboratori

Università di camerino

Marcorè Leonardo
Matricola 123437
A/S 2025/2026

29 Luglio 2025

Indice

1 Analisi

1.1 Testo assegnato

Realizzare un campionatore di un segnale analogico a 10 bit, ad una frequenza di campionamento data in input e memorizzi sequenzialmente, nella memoria EPROM, i valori letti, in blocchi da 5 byte ogni 4 letture (5*8=4*10=40). Il task di analisi e scrittura dovrà operare in background mentre la funzione loop() si potrà:

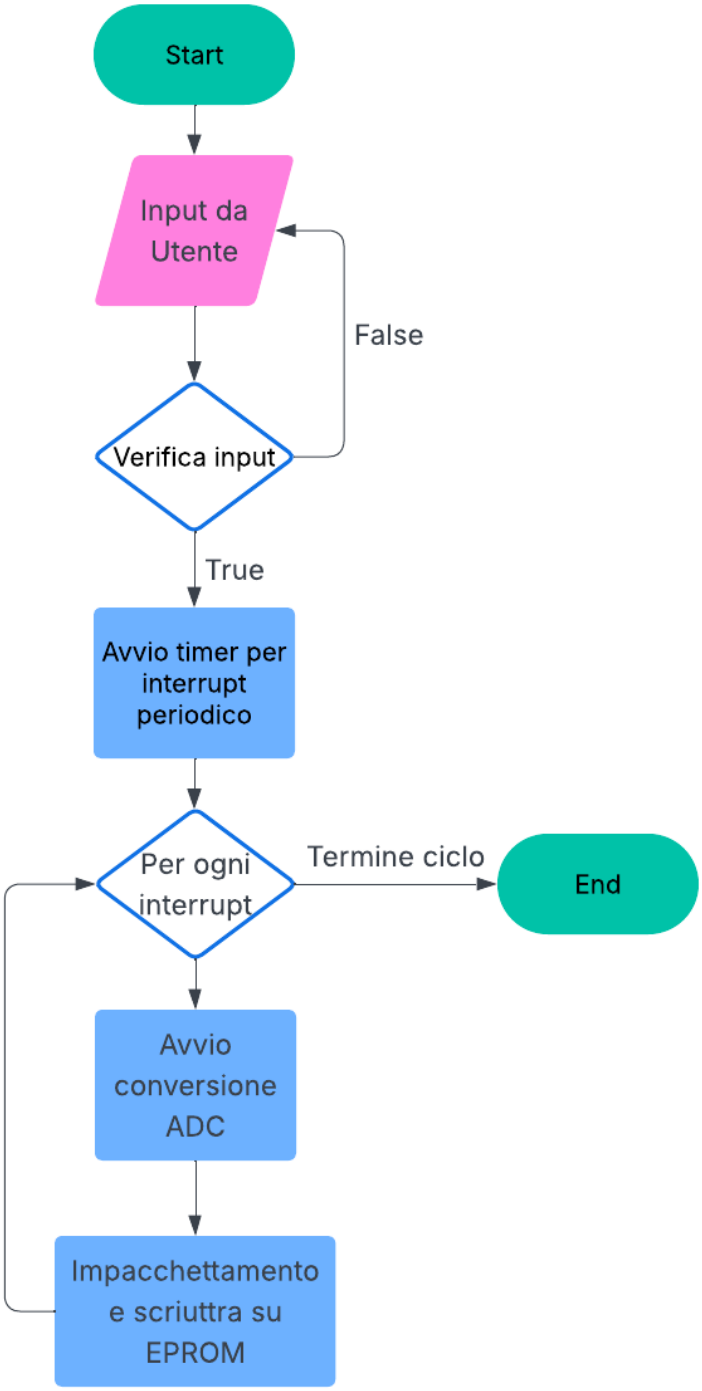
1. Avviare un ciclo di lettura, specificando frequenza e numero di campioni, controllando i limiti permessi dalla macchina.
2. Interrompere eventualmente un ciclo di rilevamento, prima della fine.
3. Segnalare la fine del campionamento.
4. Visualizzare opportunamente il contenuto della EPROM, al termine del ciclo di campionamento.

1.2 Analisi del problema

1.2.1 Elenco azioni logiche

1. Riceve in input la frequenza di campionamento e il numero di campioni da acquisire.
2. Controlla se la frequenza e il numero di campioni sono compatibili con i limiti hardware del sistema.
3. Attiva un ciclo di campionamento analogico a 10bit, utilizzanod i registri del convertitore A/D dell'Atmega.
4. Campiona il segnale alla frequenza specificata.
5. Converte ciascun valore analogico letto in una parola da 10bit.
6. Accumula 4 letture dunque 40 bit.
7. Organizza i 40 bit in un blocco di 5 byte. (5*8 = 4*10 = 40 bit)
8. Scrive il blocco di 5 byte nella memoria EPROM, utilizzando i registri e comandi di interfaccia del microcontrollore.
9. Esegue l'analisi dei dati e la scrittura su EPROM in background (tramite interrupt).
10. Permette di interrompere il ciclo di campionamento in qualsiasi momento.
11. Segnala la fine del ciclo di campionamento quando viene completato o interrotto.
12. Consente di visualizzare il contenuto della memoria EPROM al termine del campionamento.

1.2.2 Flowchart Flusso logico del sistema



1.2.3 Elementi computazionali

Parametro	Tipo	Note e vincoli
Frequenza di campionamento (Hz)	Input numerico	Max tipico per ADC = 15kHz (a 10 bit, ATmega328)
Numero di campioni	Input numerico	Max memoria disponibile = (n_byte_EPROM / 5) * 4
Valore campionato (ADC)	Output numerico	10 bit, da 0 a 1023
Dimensione buffer EPROM	Calcolo	Ogni 4 campioni = 5 byte
EPROM	Vincolo fisico	Scrittura byte per byte, tempo di scrittura da gestire

1.2.4 Vincoli hardware

- **ADC dell'Atmega328p**
  Un convertitore a 10 bit, frequenza massima di campionamento circa 15kHz.
- **Timer del microcontrollore**
  Deve generare interrupt alla frequenza di campionamento specificata
- **EPROM**
  Memoria non volatile a scrittura lenta, occorre usare interrupt o buffer
- **RAM disponibile**
  Limitata (2KB su Atmega328P) dunque uso di buffer limitati
- **Numero massimo di campioni**
  EPROM è da 1KB: massimo (1024/5)*4 = 816 campioni

1.2.5 Specifiche non vincolanti da definire

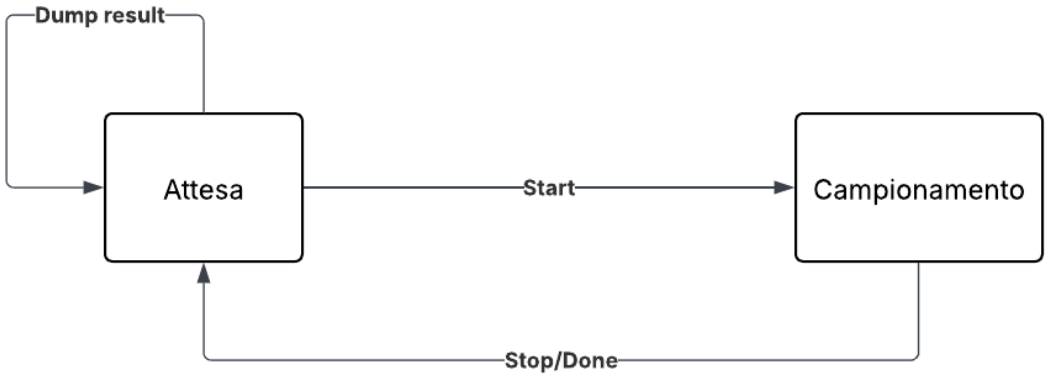
- Modalità di visualizzazione finale del contenuto EPROM
- Modalità di interazione utente
- Comportamento in caso di errore o overflow di memoria
- Codifica dei dati nella EPROM

1.2.6 Componenti logici candidati a moduli indipendenti

Modulo	**Input**	**Output**	**Note**
Setup Timer	Frequenza desiderata	Configurazione registri timer	
ADC Init + Lettura	Canale analogico	Valore a 10 bit	Usa registri ADMUX, ADCSRA
Encoder 4×10bit → 5×8bit	4 valori da 10 bit	5 byte	Gestisce pacchettizzazione
Scrittura su EPROM	5 byte	Conferma scrittura	Deve attendere tempo di scrittura tipico
Controllo da `loop()`	Comandi utente	Stati macchina (RUN, STOP, DONE)	Parser comandi, invoca azioni
Visualizzazione contenuto EPROM	Comando utente	Dati letti	Via seriale (es. Serial.print())

1.3 Analisi della soluzione adottata

1.3.1 Diagramma di stato del sistema



1.3.2 Moduli di elaborazione indipendenti

Il progetto è stato suddiviso in una serie di moduli logici indipendenti, ciascuno responsabile di una funzione ben definita, ma capaci di interagire tra loro per ottenere il comportamento complessivo desiderato.

- **Modulo Timer**

  Gestisce la generazione di eventi periodici tramite il Timer1 hardware del microcontrollore. Timer1 è uno dei tre timer hardware interni del microcontrollore ATmega328P. È un timer/counter a 16 bit, che può contare da 0 a 65535, ed è in grado, tra le altre cose, di generare interrupt periodici. Nella soluzione adottata Timer1 viene usato in modalita CTC, dunque il timer inizia a contare da 0 e arriva ad un valore di confronto preso dall'utente (in pratica la frequenza di campionamento scelta). Arrivato al valore di confronto il timer viene azzerato e scatta l'interrupt ISR(TIMER1_COMPA_vect) che ha il compito di effettuare il campionamento.

- **Modulo ADC**

  Tale modulo si occupa della conversione del segnale analogico in digitale. Ogni volta che il timer genera un evento, viene avviata una nuova conversione tramite l'ADC interno dell'ATmega328P. Il modulo ADC nello specifico si limita a leggere il valore di tensione sul pin A0, lo converte in un numero da 0 a 1023(10bit). Una volta letto il valore scatta un interrupt, per la precision ADC_vect, che il sistema legge e di conseguenza salva il valore nel buffer, salvando i valori gia' presenti in esso all'interno della EPROM nel caso fosse pieno.

- **Modulo EPROM writer**

  Questo modulo riceve blocchi da 4 campioni e li comprime in 5 byte, per sfruttare al massimo la capacità della memoria EPROM interna. Una volta completato ogni blocco, i byte vengono scritti sequenzialmente in EPROM. La logica di scrittura evita operazioni superflue, utilizzando EEPROM.update() per ridurre l'usura della memoria.

- **Modulo Interfaccia seriale**

  Gestisce l'interazione con l'utente tramite porta seriale. Interpreta i comandi digitati nel monitor seriale (start, stop, status, dump) ed esegue le azioni corrispondenti. Questo modulo rappresenta il punto di controllo del sistema ed è responsabile dell'avvio del campionamento, dell'interruzione anticipata e della visualizzazione dei dati acquisiti.

- **Logica di coordinamento**

  Usa variabili globali condivise, alcune delle quali volatile, per sincronizzare l'attività dei moduli attivati da interrupt con il flusso logico principale. Questa scomposizione in moduli permette di semplificare lo sviluppo, il debug e l'evoluzione futura del sistema.

1.3.3 Componenti fisiche e analisi delle specifiche tecniche rilevanti

La realizzazione fisica del sistema si basa su una scheda Arduino UNO, equipaggiata con un microcontrollore ATmega328P, le cui risorse hardware sono state sfruttate in modo diretto per ottenere massime prestazioni. Inoltre viene usato anche un potenziometro che fornisce il segnale analogico da convertire.

- **Convertire Analogico-Digitale (ADC)**

  Il microcontrollore integra un ADC a 10 bit, sufficiente per acquisire con buona precisione segnali analogici come quello generato da un potenziometro. È stato configurato per utilizzare AVcc come riferimento di tensione, ed è abilitato per lavorare in modalità interrupt-driven: ogni conversione è avviata dal timer e completata automaticamente, senza bloccare il programma principale.

- **Timer1**

  Sempre interno al microcontrollore ATmega328P viene utilizzato per generare eventi precisi in base alla frequenza di campionamento desiderata. La modalita' CTC permette di mantener una cadenza stabile. Il prescaler e' configurato in base alla frequenza richiesta, garantendo la compatibilita' con un ampio range di intervalli.

- **Memoria EPROM interna**

  Utilizzata per memorizzare i campioni acquisiti in modo persistente, anche in caso di spegnimento. Per ottimizzare l'uso dello spazio disponibile, è stato scelto di salvare ogni 4 campioni (40 bit) in un blocco di 5 byte, evitando sprechi. La scrittura avviene tramite la funzione EEPROM.update(), che riduce il numero di cicli di scrittura non necessari.

- **Interfaccia seriale**

  La porta seriale del microcontrollore è utilizzata per interagire con l'utente via monitor seriale. Il baud rate è fissato a 9600 bps, valore stabile e compatibile con la maggior parte dei terminali. La comunicazione avviene in modo asincrono e non interferisce con le attività a tempo reale.

1.3.4 Diagramma dei moduli e delle dipendenze

1. **Timer**

   Innesca un interrupt (TIMER_COMPA_vect) ogni 1/frequenza secondi.
   Dipendenza: Attiva il modulo ADC

2. **Modulo ADC**

   Avvia una conversione analogico-digitale su A0.
   A fine conversione lancia un interrupt (ADC_vect) e il dato viene salvato, se c'è spazio, nel buffer.
   Dipendenza: Viene attivato dal timer e scrive nel buffer condiviso.

3. **EPROM writer**

   Quando il buffer ha 4 campioni, li compatta in 5 byte.
   Scrive in EPROM in modo sequenziale
   Dipendenza: Viene attivato dal modulo ADC.

4. **Interfaccia seriale**

   Riceve comandi da monitor seriale (start, stop, dump etc).
   Scrive sulle variabili condivise
   Dipendenza: Controlla lo stato globale del sistema.

5. **Logica principale**

   Coordina gli altri moduli secondo gli stati.
   Dipendenza: Dipende da comandi seriali e variabili di stato.

1.3.5 Flowchart di algoritmi presenti nel sistema

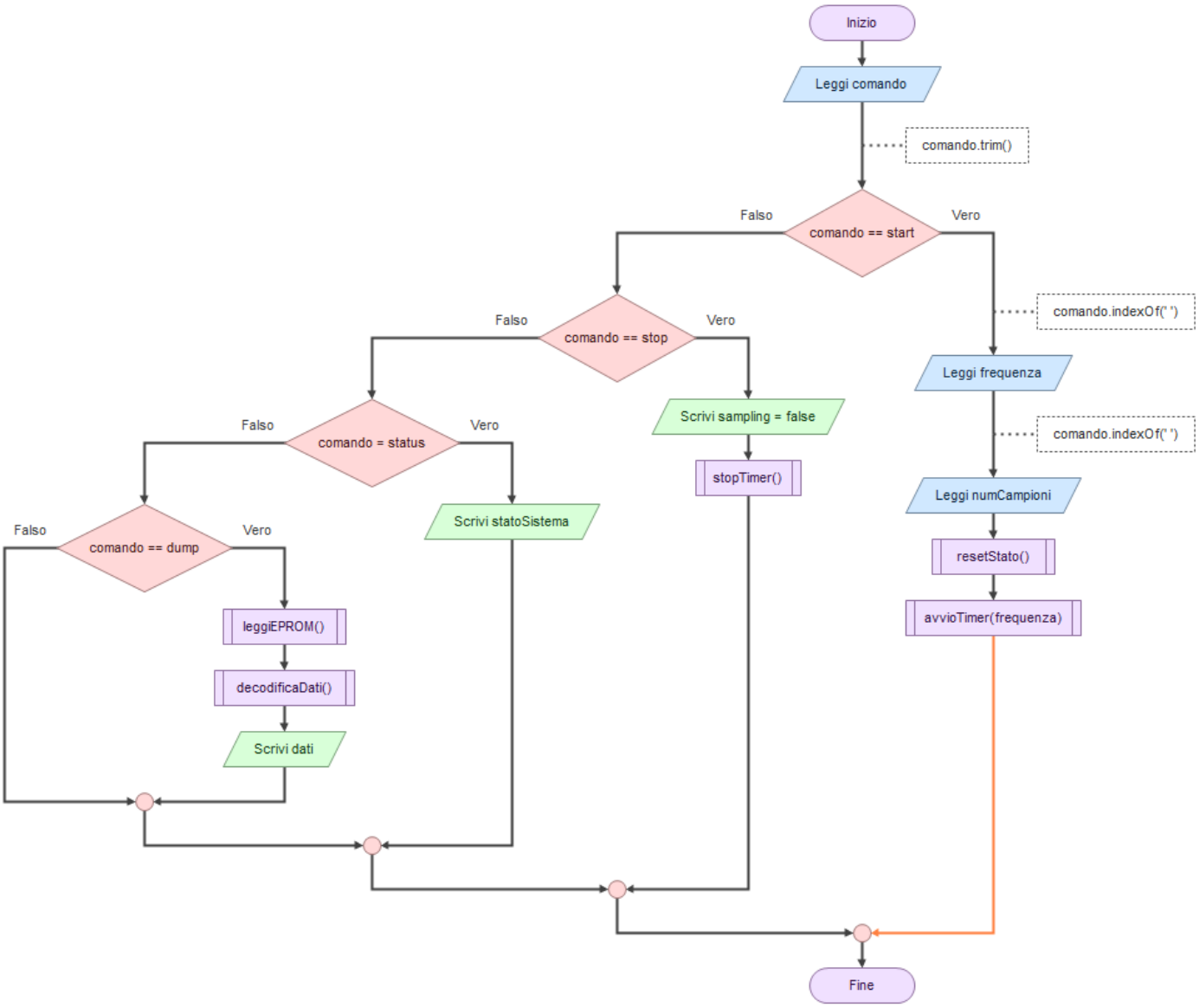


Figura 1: Loop principale

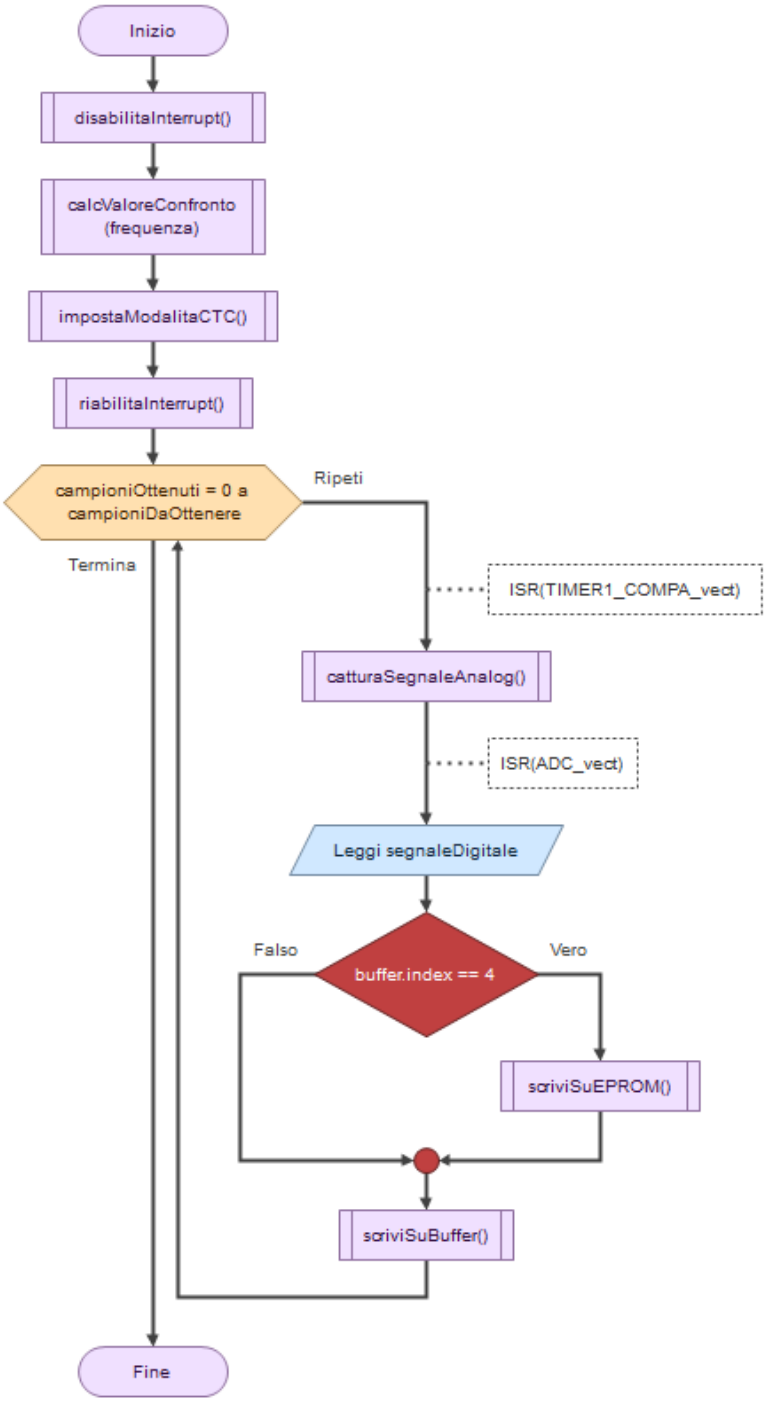


Figura 2: Avvio Timer e gestione interrupt

1.4 Analisi delle soluzioni alternative

1. **Uso della funzione delay() per scandire il campionamento**
   Opzione scartata data la natura della funzione delay() che non permette campionamenti precisi e non bloccanti.

2. **Uso della funzione millis() per gestione del tempo** Soluzione migliorativa rispetto a delay(), ma meno precisa e affidabile rispetto all'uso dei timer hardware.

2 Sintesi

2.1 Sintesi componente ME1 – Modulo Timer

Questo modulo gestisce il temporizzatore hardware Timer1 in modalità CTC (Clear Timer on Compare Match), per generare un interrupt a intervalli regolari definiti dall'utente (frequenza di campionamento).

Realizzazione: Implementato direttamente tramite registri del microcontrollore ATmega328P

- TCCR1A, TCCR1B per la configurazione.
- OCR1A per il valore di confronto, viene calcolato dinamicamente in base al valore della frequenza scelta dall'utente.
- TIMSK1 per abilitare l'interrupt.

Accorgimenti:

- Utilizzo del prescaler a 8 per miglior bilanciamento precisione/overflow
- Calcolo automatico del valore OCR1A in funzione della frequenza
- Uso di uint32_t per evitare overflow nei calcoli. Il tipo di variabile uint32 si tratta di un tipo utilizzato nella programmazione embedded, u sta per unsigned, int per intero e 32 sono i bit di cui viene composta la variabile.

2.2 Sintesi componente ME2 – Modulo ADC

Questo modulo effettua la conversione del segnale analogico in un valore digitale a 10 bit, leggendo il canale analogico A0 ogni volta che il Timer1 lo attiva.

Realizzazione: Il modulo ADC è configurato usando i registri ADMUX, ADCSRA, ADCL/ADCH. L'interrupt ADC_vect viene abilitato per la gestione asincrona del risultato.

- ADMUX: Seleziona quale pin leggere (La soluzione va a leggere sul pin A0) e quale riferimento di tensione usare (AVcc, pin di alimentazione del convertitore analogico collegato quindi a +5V che quindi rappresenta il valore massimo che il convertitore ADC usera per calcolare il valore digitale corrispondente alla tensione letta)
- ADCSRA: Controlla l'avvio, l'abilitazione e lo stato del convertitore ADC. Bit utilizzati: ADEN per abilitare l'ADC, ADIE per abilitare l'interrupt e ADPS2 imposta la velocita di conversione.
- ADCL e ADCH: Registri dove si trova il risultato della conversione analogico-digitale. L'ADC restituisce un valore a 10 bit, i bit meno significativi sono in ADCL e i piu significativi sono in ADCH. (Nel codice viene usato uint16_t valore = ADC; una macro che combina i risultati dei due registri).

Accorgimenti:

- Impostazione della sorgente del riferimento analogico su AVcc.
- Avvio della conversione solo se sampling == true.
- Variabile volatile per samplesCaptured, usata tra interrupt e loop().

2.3 Sintesi componente ME3 – EEPROM Writer

Questo modulo riceve blocchi da 4 campioni da 10 bit e li codifica in 5 byte, per poi scriverli nella memoria EEPROM interna in modo compatto.

Realizzazione: I dati vengono memorizzati in un array uint16_t buffer[4], e compressi in un array da 5 byte usando operazioni bitwise. La scrittura in EEPROM avviene con EEPROM.update() per evitare usura inutile.

Accorgimenti:

- Codifica compatta 4x10 bit → 5 byte per risparmiare spazio
- Uso di EEPROM.update() per evitare scritture ridondanti
- Salvataggio sequenziale fino a esaurimento dei campioni richiesti

2.4 Sintesi componente ME4 – Interfaccia seriale

Gestisce i comandi seriali da monitor seriale per avviare (start), fermare (stop), controllare lo stato (status) e leggere la EEPROM (dump).

Realizzazione: Analisi del comando ricevuto via Serial.readStringUntil('/n') e parsing dei parametri (frequenza, numero campioni). Imposta le variabili globali necessarie.

Accorgimenti:

- Lettura non bloccante dei comandi

- Parsing sicuro con verifica parametri validi
- Segnalazione di errori e fine acquisizione via Serial.println()

2.5 Sintesi componente ME5 – Stato del sistema

Gestisce lo stato globale del sistema (IDLE, SAMPLING) e coordina i moduli tra loro. Controlla se il numero totale di campioni richiesto è stato raggiunto, e termina il campionamento.

Realizzazione: La logica a stati è implementata all'interno della funzione loop(), ma guidata da flag condivisi settati negli interrupt.
Accorgimenti:

- Uso di volatile per le variabili modificate dagli interrupt. La keyword volatile serve a dire al compilatore "Questa variabile puo' cambiare in modo asincrono, anche se tu (compilatore) non la vedi cambiare nel codice principale." Nella soluzione analizzata ci sono due contesti separati che accedono alle stesse variabili: Il programma principale eseguito in modo sequenziale e gli interrupt eseguiti fuori dal fluso normale (appena scatta un evento hardware).
- Separazione tra eventi hardware (interrupt) e logica principale.
- Messaggi seriali a fine ciclo per segnalazione.

3 Sistema complessivo

Di seguito il link al progetto wokwi : `https://wokwi.com/projects/436557469413140481`

3.1 Manuale utlizzo

1. Avvio simulazione wokwi
2. Usare uno dei comandi evidenziati (start (freq) (num), stop, status, dump). start Viene utilizzato se si vuole iniziare un campionamento, il parametro freq rappresenta la frequenza di campionamento desiderata (massimo 1000Hz e num il numero di campioni (massimo 816). stop Viene utilizzato se si vuole interrompere il campionamento in anticipo. status Viene utilizzato quando si vuole conoscere lo stato attuale del sistema. dump Viene utilizzato per stampare il contenuto della memoria EPROM.

3.2 Considerazioni finali

Il progetto ha permesso di approfondire concetti fondamentali legati all'acquisizione di segnali analogici, alla gestione della memoria EPROM e alla programmazione di microcontrollori. Sono stati implementati con successo moduli per la configurazione del timer, la lettura dell'ADC, la codifica compatta dei dati e la scrittura su memoria persistente. In conclusione, il progetto ha raggiunto gli obiettivi previsti e rappresenta una buona base per estensioni più complesse in ambito embedded.