Campionatore Analogico-Digitale

Mattia Brizi

Data: 20 febbraio 2025

Questa relazione segue il modello fornito dal professore al seguente link come richiesto dal testo del progetto assegnato.

INDICE

1 Analisi

1.1 Testo assegnato

Realizzare un campionatore un segnale analogico a 10 bit in base ai seguenti valori dati in input, controllando i limiti massimi e minimi ammessi dalla macchina al reset:

1. frequenza di campionamento F in Hz
2. Livello di trigger (= segnale di avvio campionamento) V da 0 a 5 V
3. Sorgente trigger: esterno, da un ingresso analogico riservato o interno dallo stesso ingresso da campionare
4. Ritardato T nell'avvio del campionamento, rispetto al segnale di trigger
5. Numero di campioni da registrare.
   Il sistema dovrà memorizzare i valori letti nella memoria RAM, in blocchi da 5 byte dove ogni blocco contiene 4 letture da 10 bit (5*8=4*10=40).

Il task di analisi e scrittura dovrà operare in background mentre la funzione loop() svolgerà le seguenti funzioni:

a) consentire di cambiare i dati forniti inizialmente

b) Interrompere eventualmente un ciclo di rilevamento, prima della fine

c) Segnalare la fine del campionamento

d) Visualizzare il contenuto della RAM, al termine del ciclo di campionamento con una sequenza di valori esadecimali da 3 cifre (12 bit).
   Si dovrà prevedere una compressione elementare secondo la seguente regola: compressa secondo la semplice regola: se il bit più significativo del valore a 12 bit è 1, allora i restanti bit indicano il numero di repliche identiche all'ultimo valore campionato.

Per leggere i valori analogici in ingresso, si dovranno utilizzare direttamente i registri del processore, gli interrupt ed i comandi di interfaccia previsti dal datasheet ATMEGA

1.2 Analisi del problema

Descrizione dettagliata degli elementi computazionali e del flusso logico.

Per analizzare il testo fornito, dobbiamo scomporre la descrizione in vari aspetti chiave:

1. Elementi computazionali
   (a) Specifiche numeriche calcolabili
   (b) Valori in ingresso e in uscita e casi estremi
   (c) Vincoli imposti dai componenti fisici esterni
   (d) Specifiche non vincolanti (e quindi da definire meglio)
2. Flusso logico di cosa il sistema da realizzare dovrà fare in funzione degli input ed output.

3. Isolamento di potenziali componenti logici candidati ad essere sviluppati come moduli di elaborazione indipendenti che quindi operano su variabili di input e restituiscono variabili di output.

1.2.1 Specifiche numeriche calcolabili

- Per la **frequenza di campionamento (F)** va calcolato il limite massimo, in base ai vincoli hardware del microcontrollore. La frequenza della CPU è ben nota e fissata a **16MHz**. Se vogliamo evitare perdita di precisione dell'ADC la frequenza ottimale di lavoro è tra i **50KHz e i 200KHz**.
  Con un prescaler di **128** per il convertitore ADC avremo:

$$F_{ADC} = \frac{F_{CPU}}{\text{Prescaler}} = \frac{16MHz}{128} = 125KHz$$

  che rientra perfettamente nei limiti voluti.
  Inoltre l'ADC dell'ATmega328P è un convertitore **SAR** (Successive Approximation Register) a 10 bit, che opera eseguendo una sequenza di confronti per determinare il valore digitale del segnale analogico in ingresso.

  Il processo di conversione avviene in più fasi, e il datasheet specifica il numero esatto di cicli necessari:

  1. 1 ciclo di clock per avviare la conversione.
  2. 12 cicli di clock per eseguire il processo di conversione dei 10 bit, con la logica SAR.

  saranno quindi necessari un totale di **13 cicli di clock** per il completamento di una singola conversione da analogico a digitale.
  Possiamo calcolare dunque la frequenza massima di campionamento con la seguente formula:

$$F_{MAX} = \frac{125KHz}{13} \approx 9.6kSPS$$

  ossia possiamo campionare il segnale circa **9600 volte al secondo**.

- Il **livello di trigger (V)** deve essere confrontato con la soglia di 0V e 5V. L'ADC restituirà, una volta campionato, un valore **da 0 a 1023** (in base alla sua risoluzione a 10 bit, $2^{10} = 1024$) che andrà poi convertito per essere confrontato con il valore in Volt, tramite le seguenti formule:

$$V = \frac{ADC}{1023} \times 5$$

  Esempio: Se il valore dell'ADC è 512, la tensione risultante sarà:

$$V = \frac{512}{1023} \times 5 \approx 2.5\,\text{V}$$

  Per convertire una tensione da 0V a 5V in un valore ADC da 0 a 1023, si usa la formula:

$$ADC = \frac{V}{5} \times 1023$$

Esempio: Se la tensione in ingresso è 3.3V, il valore ADC sarà:

$$ADC = \frac{3.3}{5} \times 1023 \approx 678$$

- **Memoria necessaria**: Poiché ogni blocco è di 5 byte e contiene 4 letture da 10 bit, il numero totale di byte necessari sarà legato al numero di campioni da registrare. La grandezza della SRAM è ben nota e in base al datasheet è di **2KB** (2048 byte). Considerando lo spazio necessario al codice compilato (informazione che verrà calcolata staticamente una volta finito il codice), possiamo ricavare la formula che ci interessa. Il numero massimo di campioni che possiamo memorizzare nella RAM è dato da:

$$N_{\text{MAX}} = \frac{\text{Memoria disponibile}}{\text{Spazio per campione}}$$

Caso ideale (tutta la RAM disponibile)

Se tutta la RAM fosse disponibile per i campioni:

$$N_{\text{MAX}} = \frac{2048}{5} \cdot 4 = 409,6 \cdot 4 \approx 1638 \text{ campioni}$$

dividiamo la memoria per la grandezza di ogni singolo blocco di campioni (in byte), che a sua volta contiene al suo interno 4 campioni, quindi moltiplichiamo per 4.

Caso realistico (memoria occupata dal codice)

Se assumiamo che il codice occupi una parte della RAM, ad esempio 1 KB (1024 byte), allora la memoria rimanente per i campioni sarà:

$$\text{Memoria disponibile per campioni} = 2048 - 1024 = 1024 \text{ byte}$$

Quindi il numero massimo di campioni memorizzabili sarà:

$$N_{\text{MAX}} = \frac{1024}{5} \cdot 4 = 204,8 \cdot 4 \approx 819 \text{ campioni}$$

Calcolo dinamico della grandezza dell'array

Possiamo dunque ricavare una formula dato, il numero di campioni voluti (N), per calcolare l'occupazione della memoria in byte. Utilizzando un array di valori da 8bit ciascuno, questa risulta inoltre essere anche la grandezza dell'array stesso.

$$\text{Spazio Occupato} = \left\lceil \frac{N}{4} \right\rceil \times 5 \quad \textit{Byte}$$

Mentre la formula inversa per calcolare N (numero campioni) sapendo la loro occupazione in byte è:

$$N = 4 \times \left\lfloor \frac{M}{5} \right\rfloor$$

dove M è il numero di byte usati.

Blocchi di memoria

Come specificato nel testo del progetto, la memorizzazione dei campioni dovrà avvenire all'interno di blocchi da 5 byte (4 campioni ciascuno), quindi si dovrà considerare il fatto che se il numero effettivo di campioni è inferiore al numero di campioni totali, calcolati in base di 5byte, l'ultimo blocco non verrà riempito lasciando degli spazi vuoti.

Ad esempio: se il numero di campioni (N) voluti è 34

$$\text{Byte necessari} = \left\lceil \frac{N \times 10}{8} \right\rceil$$

$$\text{Numero di blocchi} = \left\lceil \frac{\text{Byte Necessari}}{5} \right\rceil = \left\lceil \frac{\left\lceil \frac{N \times 10}{8} \right\rceil}{5} \right\rceil$$

$$\frac{34 \times 10}{8} = 42.5 \quad \Rightarrow \quad \lceil 42.5 \rceil = 43 \text{ byte}$$

$$\frac{43}{5} = 8.6 \quad \Rightarrow \quad \lceil 8.6 \rceil = 9 \text{ blocchi}$$

Queste approssimazioni (necessarie) fanno si che possa esistere questo leggero spreco di spazio(ovviamente questo accade se e solo se si vuole ragionare con questa logica a blocchi).

- **Compressione dei dati**: La compressione dei dati può avvenire:

  1. durante la stampa dei valori in modo da ridurre il quantitativo effettivo di stampe e rendere il campionamento più leggibile. Non andando dunque a inficiare sulla memoria e non modificando l'array di valori campionati, ma limitandosi a dei calcoli "in loco".
  2. calcolando i valori compressi per poi riscriverli all'interno dell'array, in modo da avere un potenziale risparmio di memoria in quanto questi ultimi necessitano di meno spazio.

  In ogni caso essa dipende dal numero di valori identici consecutivi. Se non ci sono repliche consecutive, ogni valore a 12 bit verrà scritto separatamente.

  Trattandosi nel primo caso solamente di una funzione di lettura dei valori e di stampa, qui di seguito viene riportata l'analisi della soluzione numero 2.

La memoria richiesta senza compressione è data da:

$$\text{Memoria senza compressione} = \frac{N}{4} \times 5$$

dove:

- N è il numero totale di campioni.
- Ogni blocco contiene 4 campioni e occupa 5 byte.

Calcolo della Memoria con Compressione

La memoria richiesta con compressione è:

$$\text{Memoria con compressione} = 3 \times \left(\frac{M}{4}\right) + 5 \times \left(\frac{N-M}{4}\right)$$

dove:

- M è il numero di campioni ripetuti.
- I blocchi compressi occupano 3 byte invece di 5.

Esempio di Calcolo

Supponiamo di avere $N = 1000$ campioni e che $M = 800$ di essi siano ripetuti.

$$\text{Memoria con compressione} = 3 \times \left(\frac{800}{4}\right) + 5 \times \left(\frac{1000-800}{4}\right)$$

$$= 3 \times 200 + 5 \times 50$$

$$= 600 + 250 = 850 \text{ byte}$$

Senza compressione, la memoria richiesta sarebbe:

$$\text{Memoria senza compressione} = \frac{1000}{4} \times 5 = 1250 \text{ byte}$$

Calcolo del Compression Ratio

Il Compression Ratio è dato da:

$$\text{Compression Ratio} = \frac{\text{Memoria con compressione}}{\text{Memoria senza compressione}}$$

Sostituendo i valori:

$$\text{Compression Ratio} = \frac{850}{1250} = 0.68$$

Espresso in percentuale:

$$\text{Compressione ottenuta} = (1 - 0.68) \times 100 = 32\%$$

Conclusioni

- **Alta efficienza:** Se i dati sono molto ripetitivi, la compressione offre un risparmio significativo. Ad esempio, il risparmio può essere superiore al 30% e può arrivare fino all'80% se la ripetizione dei campioni è quasi totale.
- **Bassa efficienza:** Se i campioni sono molto variabili e ci sono poche ripetizioni, la compressione non darà un gran risparmio. Anzi, potrebbe non essere vantaggiosa.

- **Ritardo dell'avvio di campionamento**

  Supponiamo di voler configurare un ritardo di 100 ms prima di avviare il campionamento. Se il microcontrollore ha un clock a 16 MHz, possiamo utilizzare un timer che imposta il ritardo.

  - Un tick del timer corrisponde a un intervallo di $\frac{1}{16\,\text{MHz}} = 62.5\,\text{ns}$.
  - Per ottenere un ritardo di 100 ms, dobbiamo determinare quanti ticks sono necessari:

$$\text{Numero di ticks} = \frac{100\,\text{ms}}{62.5\,\text{ns}} = 1.6 \times 10^6\,\text{ticks}$$

  Quindi, si configura il timer per generare il ritardo corrispondente a circa 1,6 milioni di ticks, dopodiché avviare la conversione ADC.

Simboli

- f_{clk}: frequenza del clock del microcontrollore (16 MHz per l'ATmega328P).
- T: tempo di ritardo desiderato (nel nostro esempio, 100 ms).
- N: prescaler applicato al timer.
- Conteggio del timer: numero di conteggi che il timer deve fare prima di fare il *compare match* o essere azzerato.

Formula Generale

La relazione che lega il tempo di ritardo, il prescaler e il conteggio del timer è la seguente:

$$T = \frac{N \times \text{Conteggio del timer}}{f_{\text{clk}}}$$

Calcolo del Prescaler N

Per ottenere un ritardo di 100 ms, dobbiamo calcolare il prescaler N. La formula per determinare il prescaler minimo che ci consente di ottenere un ritardo di 100 ms senza superare il limite massimo di 65.535 conteggi è:

$$N \geq \frac{f_{\text{clk}} \times T}{65.535}$$

Sostituendo i valori:

$$N \geq \frac{16.000.000 \times 0.1}{65.535} \approx 24.4$$

Il prescaler minimo che possiamo utilizzare è quindi $N = 64$, poiché N deve essere un valore tra quelli disponibili (1, 8, 64, 256, 1024).

Calcolo del Conteggio del Timer

Per il prescaler $N = 64$, possiamo calcolare quanti conteggi sono necessari:

$$\text{Conteggio del timer} = \frac{16.000.000 \times 0.1}{64} = 25.000$$

Quindi, con $N = 64$, il timer farà 25.000 conteggi prima di essere azzerato, che è inferiore al limite massimo di 65.535 conteggi. Così facendo abbiamo trovato la formula per il calcolo del valore da inserire all'interno del registro di confronto, che potrà essere utilizzata nel codice per generare un interrupt, inserendo la logica necessaria al suo interno. Ad esempio il set di un flag per indicare che è decorso il tempo di delay impostato.

Ritardo Massimo con $N = 64$

Ora possiamo calcolare il ritardo massimo che possiamo ottenere con il prescaler $N = 64$:

$$T_{\text{MAX}} = \frac{64 \times 65.535}{16.000.000} \approx 0.262\,\text{s} = 262\,\text{ms}$$

Con il prescaler $N = 64$, possiamo ottenere un ritardo massimo di circa 262 ms, senza dover utilizzare un contatore per gli overflow del TCNT1, ma rimanendo nel tetto massimo dei valori rappresentabili con i 16 bit del TIMER1.
Analogamente si possono calcolare i vari valori massimi di ritardo in base ai differenti prescaler utilizzati e al tempo di ritardo desiderato.

1.2.2 Valori in ingresso

- **Frequenza di campionamento (F)**: è il numero di campioni acquisiti ogni secondo (in Hz). Il sistema deve essere in grado di gestire la frequenza passata in input, controllando limiti fisici della macchina.
- **Livello di trigger (V)**: il segnale che avvia il campionamento. Deve essere un valore compreso tra 0V e 5V. Questo segnale determina il momento in cui il campionamento inizia. Una volta superato il voltaggio impostato in input il campionamento può avere inizio.

- **Sorgente del trigger**: Il trigger può provenire da diverse fonti:
  - Esterno: una sorgente dedicata che provoca un trigger, come ad esempio un bottone.
  - Ingresso riservato: un segnale proveniente da un ingresso analogico dedicato esterno.
  - Ingresso interno: segnale proveniente dallo stesso ingresso da campionare.
- **Ritardo (T)**: Un ritardo nell'avvio del campionamento, rispetto al segnale di trigger. Questo parametro introduce una variazione temporale tra il trigger e l'effettivo inizio della lettura.
- **Numero di campioni da registrare**: Quanti campioni il sistema deve registrare prima di terminare il campionamento.

1.2.3 Valori in uscita

- **Memorizzazione in RAM**: I valori letti devono essere memorizzati in blocchi da 5 byte, con ciascun blocco che contiene 4 letture da 10 bit. Ogni lettura da 10 bit occupa 2 byte per la memorizzazione.
- **Compressione dei dati**: I dati devono essere compressi in base a una regola che dipende dal bit più significativo del valore a 12 bit. Se il bit MSB è 1, i restanti bit indicano quante repliche consecutive dello stesso valore sono presenti.

1.2.4 Casi estremi

Problemi relativi alla Frequenza di campionamento

Abbiamo potuto constatare la frequenza massima dell'ADC che è risultata essere circa 9,6 kSPS con un prescaler di 128 e un clock dell'ADC di 125 KHz. Se si imposta una frequenza di campionamento superiore al limite consentito, potrebbero verificarsi errori o perdita di dati.

Problemi relativi al voltaggio di Trigger

Se il voltaggio di trigger è impostato troppo vicino al massimo (5V nel caso di ATmega328P), potrebbe non esserci mai un segnale che supera questa soglia, impedendo quindi l'attivazione del campionamento. In altre parole, non verrà mai generato il segnale di trigger, e il campionamento non inizierà.

Problemi relativi alla memoria

Se il numero di campioni richiesti supera la memoria RAM disponibile, si possono verificare diversi problemi:

- **Sovrascrittura dei dati:** Quando la memoria è piena, non ci sarà più spazio per memorizzare nuovi campioni. Se il programma non è progettato per gestire correttamente questa situazione, si potrebbe sovrascrivere la memoria, perdendo quindi i dati precedentemente registrati. Questo porterebbe a dati incompleti o corrotti.

- **Eccessivo uso della memoria:** Se vengono memorizzati troppi campioni senza alcun controllo, la RAM potrebbe essere esaurita prima che il sistema abbia la possibilità di terminare correttamente il campionamento. Questo potrebbe causare un fallimento nel ciclo di acquisizione dei dati.

1.2.5 Vincoli imposti dai componenti fisici

Il microcontrollore ATMEGA ha delle specifiche riguardanti:

- **Range di tensione del segnale**: Tensione di ingresso (V_{IN}): Il microcontrollore ha come riferimento di tensione predefinito 5V (V_{REF}) con un range di ingresso per l'ADC che va da 0V a 5V.
- **Tensione di Trigger**: Il livello di trigger deve essere compreso tra 0V e 5V. Un segnale esterno o interno che superi questo range non attiverà correttamente l'inizio del campionamento. Inoltre se il trigger è configurato come un segnale digitale (ad esempio, un ingresso da un sensore o un segnale da un altro dispositivo), potrebbe essere necessario un circuito di condizionamento del segnale per garantire che il trigger sia sempre compreso nell'intervallo di 0-5V.
- **Memoria e RAM**: Il microcontrollore ha un limite di memoria di 2KB, quindi bisogna ottimizzare l'uso della RAM minimizzando il codice e l'uso di variabili per immagazzinare più campioni possibili.
- **Tempi di risposta e frequenza di campionamento**: Il microcontrollore avrà una limitata capacità di operare a frequenze molto alte, quindi la frequenza di campionamento F dovrà essere progettata tenendo conto delle capacità di elaborazione del processore. La frequenza di campionamento effettiva dipenderà dal clock del sistema ($F_{\text{CPU}} = 16MHz$), dalla configurazione del prescaler dell'ADC e dai tempi di acquisizione del campione richiesti.

1.2.6 Specifiche non vincolanti

Tra le specifiche non vincolanti troviamo:

1. Segnalazione di fine campionamento
2. Segnali analogici in ingresso
3. Trigger di inizio e di fine campionamento anticipato
4. Cambiare i dati forniti inizialmente
5. isualizzare il contenuto della RAM

Possibile soluzione di sviluppo:

1. Accensione di un **led** per indicare che il campionamento è terminato
2. Utilizzo di **potenziometri** come simulazione di un segnale analogico in ingresso e quindi variabili manualmente
3. Utilizzo di **bottoni** come trigger per segnale di avvio campionamento e di fine campionamento anticipato

4. Utilizzare la **porta seriale** per effettuare un interscambio di dati tra l'utente e il programma, in modo da prendere in input i valori scritti dall'utente
5. Utilizzare anche in questo caso la **porta seriale** per stampare tutti i valori campionati e compressi

1.2.7 Flusso logico del sistema

1. **Inizio campionamento**:
   - Il sistema riceve i valori di input (F, V, sorgente del trigger, T, numero di campioni).
   - Verifica che i valori passati non eccedano i valori consentiti e in caso contrario li modifica al valore massimo supportato
   - Il trigger viene monitorato.
   - Se il trigger è attivo, il sistema può introdurre un ritardo (T) prima di iniziare il campionamento.
   - In qualsiasi momento dell'escuzione si può interrompere il campionamento prima della sua durata naturale
2. **Acquisizione dei campioni**:
   - Il sistema acquisisce i campioni in base alla frequenza di campionamento F.
   - I dati letti vengono memorizzati nella RAM in blocchi di 5 byte (contenenti 4 letture da 10 bit).
   - Se il numero di campioni è raggiunto, il sistema segna la fine del campionamento.
3. **Uscita dei dati**:
   - Al termine, il sistema visualizza i dati compressi in formato esadecimale.
4. **Funzioni di gestione**:
   - Modifica dei parametri iniziali.
   - Interruzione del ciclo di campionamento in corso.
   - Segnalazione della fine del campionamento.

1.2.8 Componenti logici da sviluppare come moduli indipendenti

Alcuni moduli che potrebbero essere sviluppati separatamente includono:

- **Modulo di campionamento**:
  - Funzione: Campionare il segnale analogico e convertirlo in un valore digitale a 10 bit.
  - Ingressi:
    * Segnale analogico da campionare (0-5 V).
    * Segnale di trigger (esterno o interno).

  - Uscite:
    - Valore digitale a 10 bit (0-1023).
    - Segnale di fine conversione (flag o interrupt).

- **Gestione della Memoria**:
  - Funzione: Memorizzare i campioni in blocchi da 5 byte, con compressione dei dati.
  - Ingressi:
    - Valori digitali a 10 bit dall'ADC.
    - Comando di scrittura/lettura.
  - Uscite:
    - Dati memorizzati in formato compresso.
    - Segnale di fine scrittura/lettura.

- **Gestione del Trigger**:
  - Funzione: Gestire il segnale di avvio del campionamento.
  - Ingressi:
    - Segnale di trigger esterno.
    - Segnale di trigger interno al valore da campionare.
    - Segnale di trigger da un ingresso analogico riservato .
  - Uscite:
    - Segnale di avvio campionamento.
    - Segnale di ritardo (T) impostato.

- **Interazione con l'utente**:
  - Funzione: Consentire all'utente di modificare i parametri e visualizzare i risultati.
  - Ingressi:
    - Comandi dall'utente (ad esempio, pulsanti o interfaccia seriale).
    - Dati dalla memoria.
  - Uscite:
    - Visualizzazione dei dati (ad esempio, su display o interfaccia seriale).
    - Segnale di interruzione del campionamento.

1.3 Analisi

1.3.1 Diagramma di stato complessivo

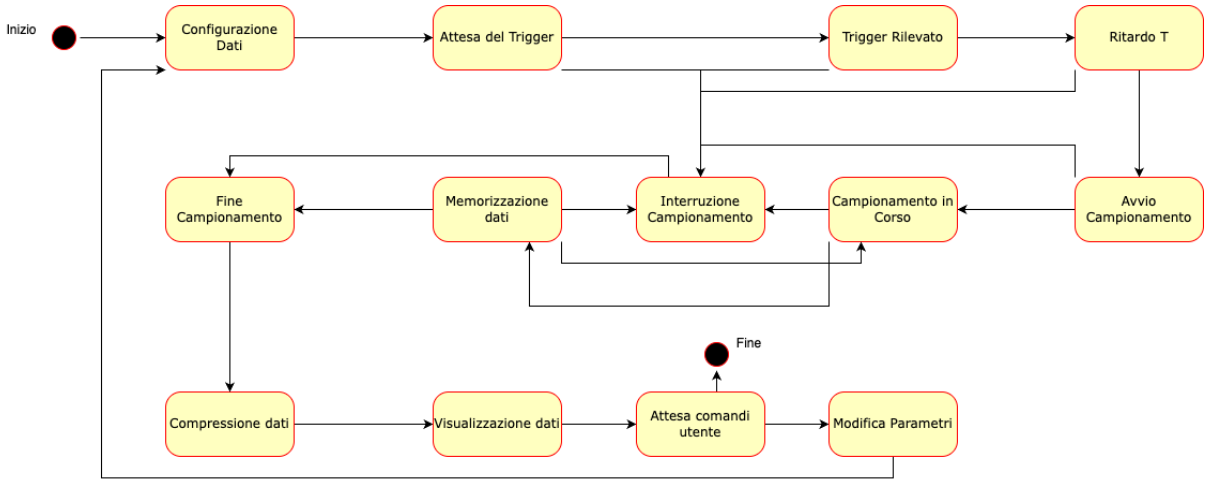


Figura 1: Generic State Chart

1.3.2 Moduli di elaborazione indipendenti

Modulo di Campionamento

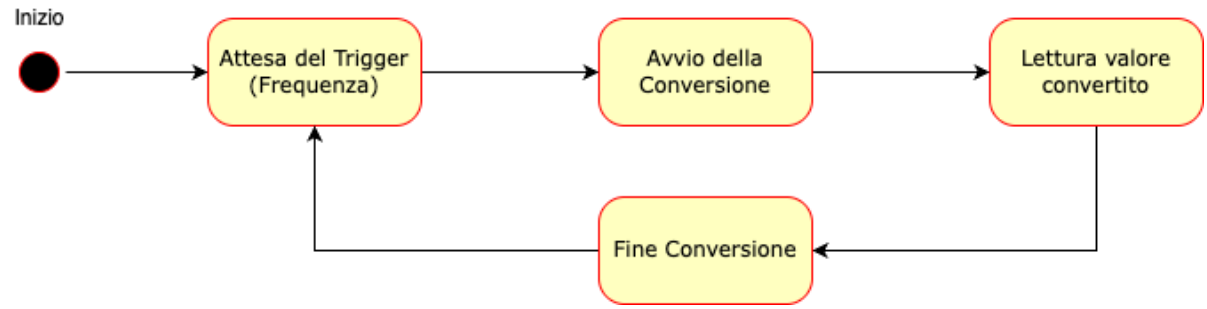


Figura 2: ADC State Chart

Componente fisico associato: ADC integrato nell'ATMEGA 328p
Specifiche tecniche utili per il progetto:

1. Risoluzione a 10 bit (0-1023).
2. Tensione di riferimento 0-5V.
3. Tempo di conversione → vedere la sezione Frequenza di campionamento(consultabile qui).
4. Funzioni, Ingressi e Uscite(consultabile qui)

Gestione della Memoria

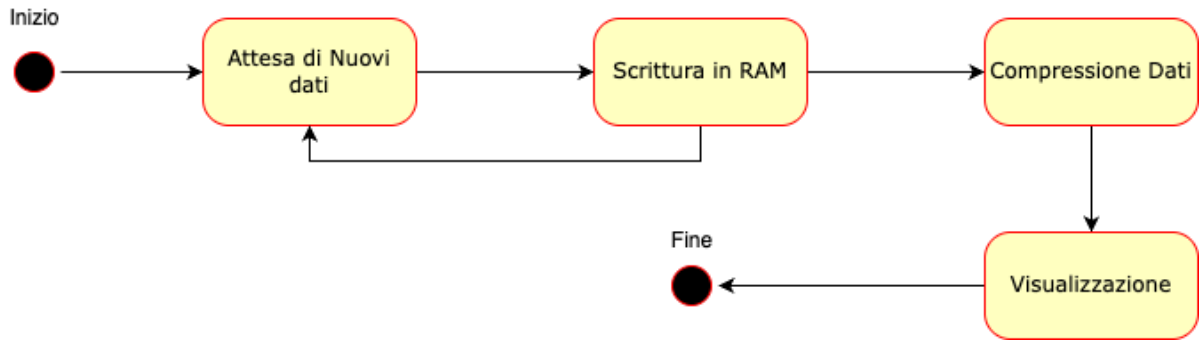


Figura 3: Memory State Chart

Componente fisico associato: Memoria RAM dell'ATMEGA 328p
Specifiche tecniche utili per il progetto:

1. Dimensioni: 2KB.
2. Organizzazione: blocchi da 5 byte.
3. Velocità: è sufficientemente veloce per memorizzare i dati in tempo reale durante il campionamento.
4. Funzioni, Ingressi e Uscite(consultabile qui)

Gestione del Trigger

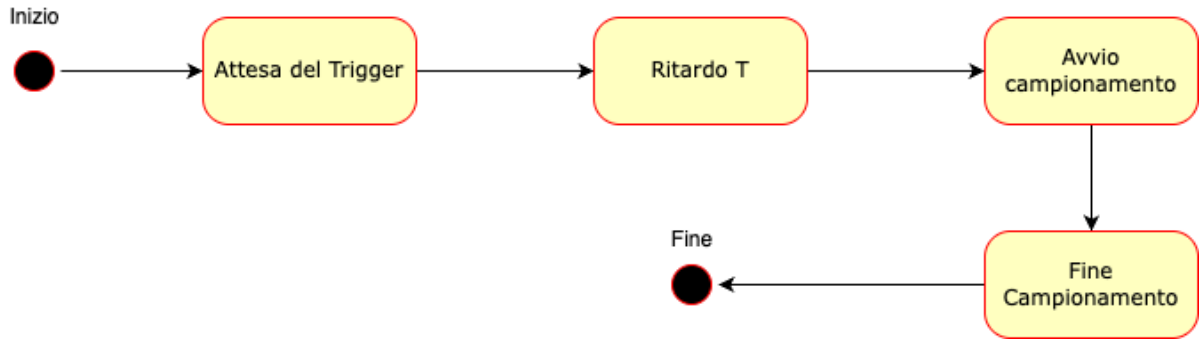


Figura 4: Trigger State Chart

Componente fisico associato: Pin di ingresso analogico/digitale dell'ATMEGA 328P.
Specifiche tecniche utili per il progetto:

1. Livello di tensione 0-5V.
2. Tempo di risposta $< 1\,\mu s$.
3. Funzioni, Ingressi e Uscite (consultabile qui)

Interazione con l'utente

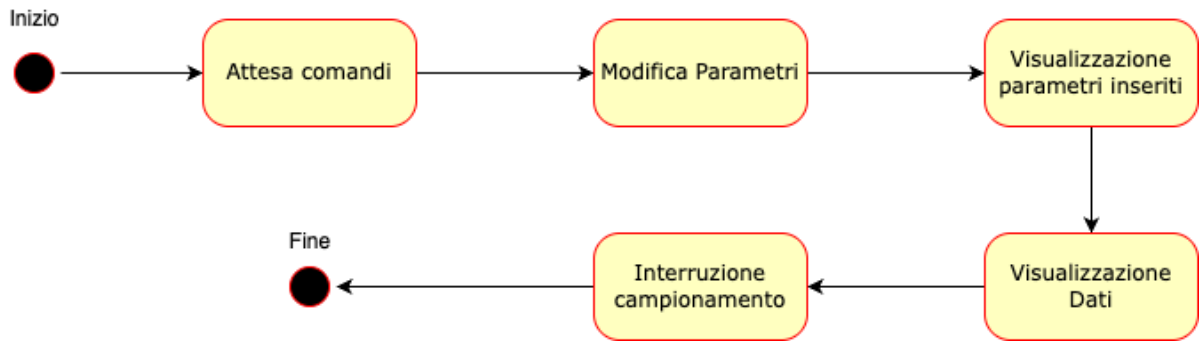


Figura 5: User Interaction State Chart

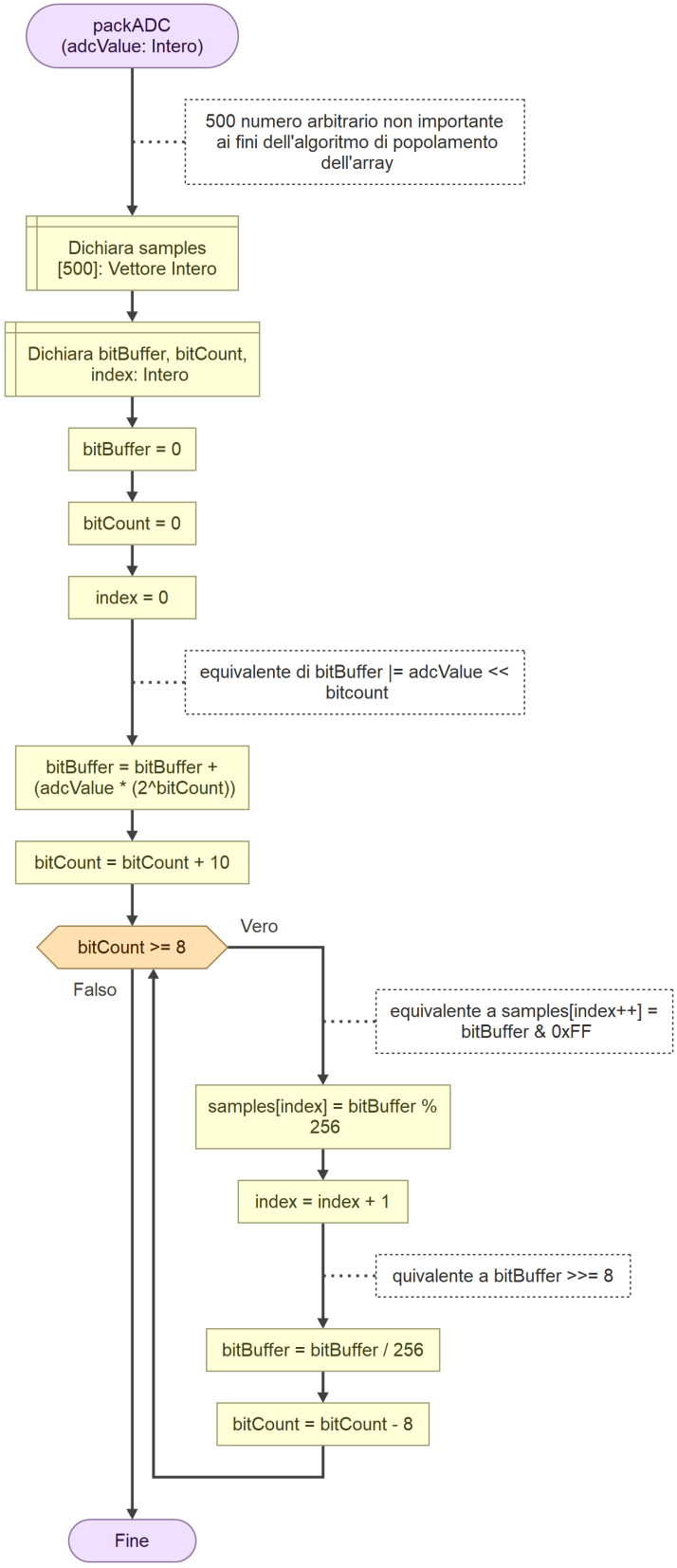
Componente fisico associato: Pulsanti o interfaccia seriale.
Specifiche tecniche utili per il progetto:

1. Pulsanti
   - Tempo di debounce necessario per evitare letture multiple, generalmente 10-50ms.
   - Devono essere collegati a pin digitali con resistori di pull-up.
2. Interfaccia Seriale
   - Baud Rate, cioè la velocità di comunicazione (9600).
   - Buffer di ricezione/trasmissione di dimensione limitata. Nell'ATMEGA 328p il buffer UART integrato ha una dimensione di 2 byte (1 byte per la trasmissione e 1 byte per la ricezione).
3. Funzioni, Ingressi e Uscite(consultabile qui)

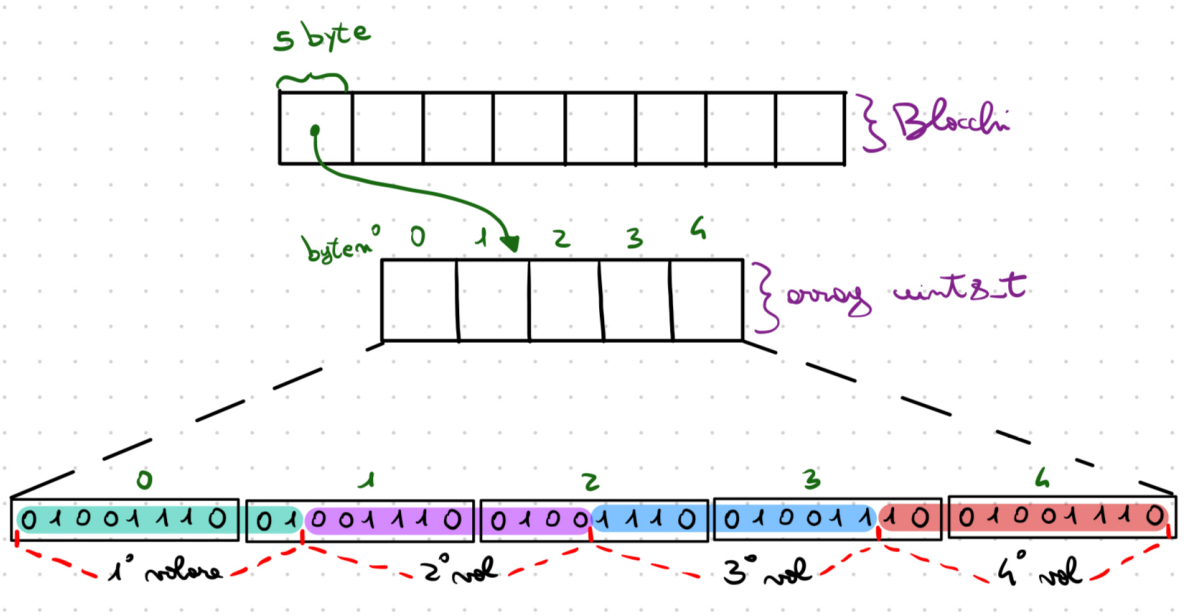
1.3.3 Algoritmi Computazionali

Algoritmi computazionali, descritti con flowchart, tramite l'utilizzo di FlowGorithm.

Popolamento dell'array uint8 in modo da compattare i dati uno di seguito all'altro evitando spreco di memoria.



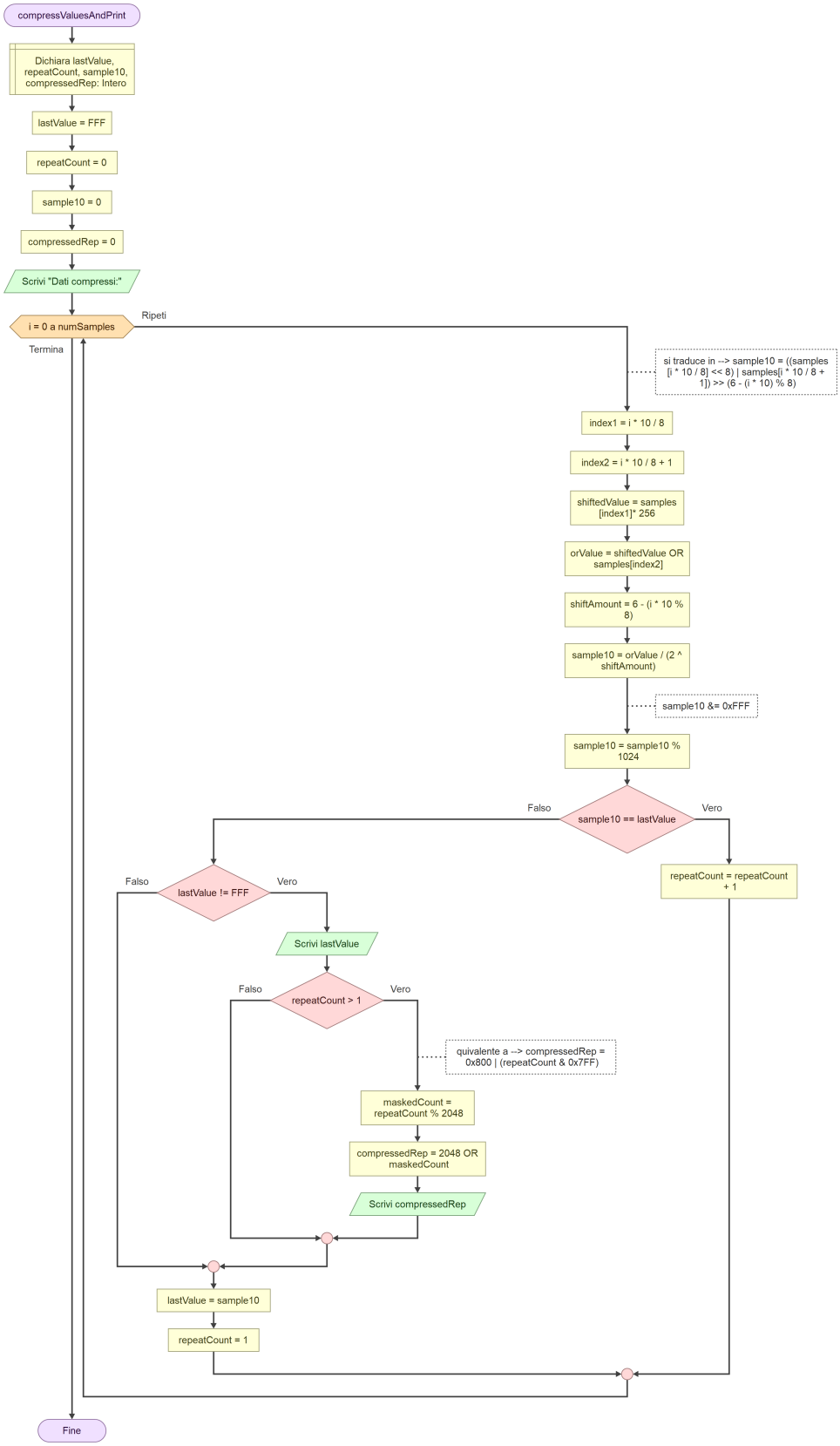
Spiegazione visuale della suddivisione dell'array.



- 1° valore va da byte 0 (0:7bit) a byte 1 (0:1bit)
- 2° valore va da byte 1 (2:7bit) a byte 2 (0:3bit)
- 3° valore va da byte 2 (4:7bit) a byte 3 (0:5bit)
- 4° valore va da byte 3 (6:7bit) a byte 4 (0:7bit)

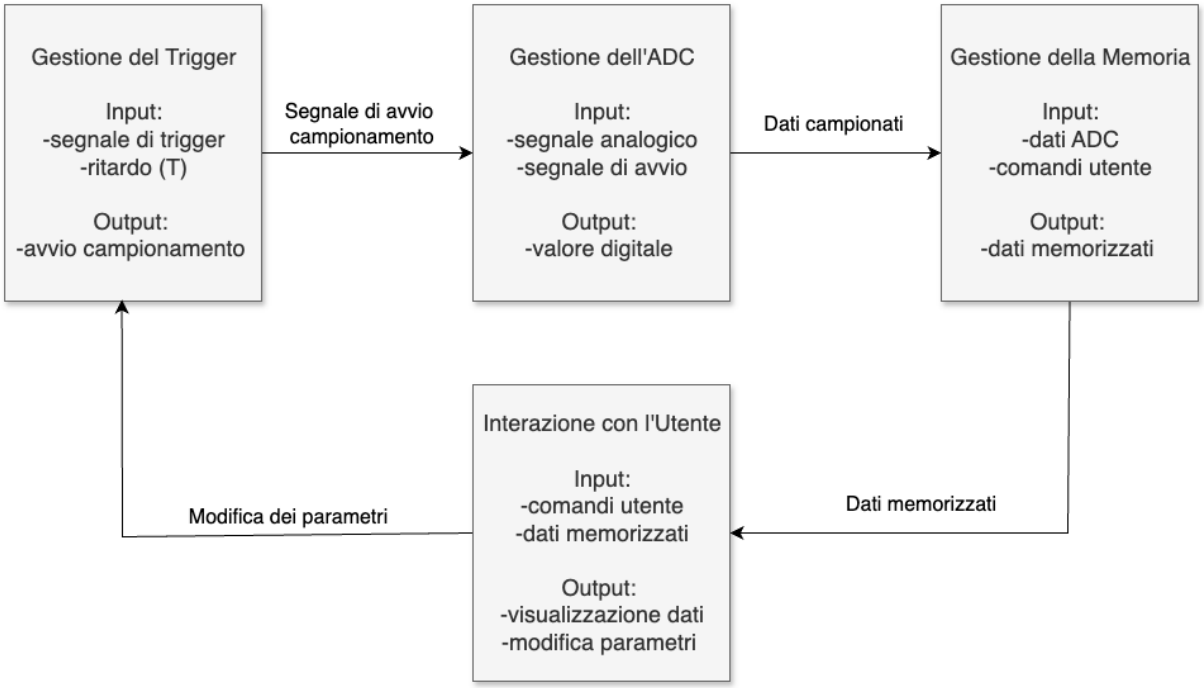
Si ripete poi ciclicamente per ogni blocco composto da 5 byte fino al riempimento dell'array.

Stampa dei valori compressi in base alla regola specificata all'interno del testo.



1.3.4 Diagramma complessivo dei componenti

Si evidenziano le dipendenze funzionali (eventi, variabili, connessioni, ecc,) dei "moduli di elaborazione"



Descrizione delle dipendenze funzionali

Gestione del Trigger:

- Dipende dall'Interazione con l'Utente per ricevere i parametri (ritardo, livello di trigger).
- Comunica con la Gestione dell'ADC per avviare il campionamento.

Gestione dell'ADC:

- Dipende dalla Gestione del Trigger per avviare il campionamento.
- Comunica con la Gestione della Memoria per inviare i dati campionati.

Gestione della memoria:

- Dipende dalla Gestione dell'ADC per ricevere i dati.
- Comunica con l'Interazione con l'Utente per inviare i dati memorizzati.

Interazione con l'Utente:

- Dipende dalla Gestione della Memoria per ricevere i dati.
- Comunica con la Gestione del Trigger per modificare i parametri.

Eventi condivisi

Trigger rilevato:

- Generato dalla Gestione del Trigger.
- Avvia il campionamento nella Gestione dell'ADC.

Fine Campionamento:

- Generato dalla Gestione dell'ADC.
- Comunica alla Gestione della Memoria che i dati sono pronti per essere memorizzati.

Interruzione utente:

- Generato dall'Interazione con l'Utente.
- Comunica alla Gestione del Trigger e alla Gestione dell'ADC di interrompere il campionamento.

Variabili condivise

- **Frequenza di campionamento:** condivisa tra Gestione del Trigger e Gestione dell'ADC.
- **Livello di Trigger:** condiviso con la Gestione del Trigger.
- **Ritardo T:** condiviso con la Gestione del Trigger.
- **Dati campionati:** condivisi tra Gestione dell'ADC e Gestione della Memoria.
- **Dati memorizzati:** condivisi tra Gestione della Memoria e Interazione con l'Utente.

1.3.5 Conclusioni

Grazie all'analisi approfondita del progetto, condotta seguendo i consigli del professore, sono riuscito a comprendere con maggiore chiarezza ogni aspetto dello sviluppo, individuando cosa stessi facendo, quali scelte adottare e quale percorso seguire per raggiungere l'obiettivo. Questo approccio metodico mi ha permesso di dare ordine alle mie idee, evitando di procedere in modo confusionario e aiutandomi a trasformare concetti teorici in soluzioni concrete. L'esperienza maturata è stata fondamentale per acquisire una visione più strutturata e consapevole del lavoro sui microcontrollori, e sicuramente rappresenterà una base solida per progetti futuri.

1.4 Analisi delle Soluzioni Alternative

Nel corso dello studio delle informazioni necessarie per lo sviluppo del progetto, dell'analisi del testo e delle varie problematiche riscontrate, è stato sviluppato un nuovo codice con l'obiettivo di correggere i problemi presenti nella versione precedente.

Utilizzo del Registro GPIOR0 per le Flag

Nella nuova soluzione, non vengono più utilizzate flag booleane volatili per i vari trigger. Come spiegato nelle dispense, è stato adottato il registro `GPIOR0`, un registro a 8 bit di uso generale nel microcontrollore ATmega328P. Questo registro è accessibile con istruzioni a singolo ciclo macchina, rendendolo estremamente efficiente rispetto all'uso della RAM. `GPIOR0` è ideale per variabili temporanee in operazioni critiche come ISR (Interrupt Service Routine), poiché evita l'overhead dell'accesso alla SRAM, migliorando la velocità di esecuzione.

Ottimizzazione dell'ISR

Grazie all'uso del registro `GPIOR0`, è stato ottimizzato il campionamento dell'ADC. Nella prima versione del codice, il valore campionato veniva atteso all'interno della ISR del TIMER1. Questo comportava un allungamento del tempo di esecuzione e ritardi inutili.

Nella nuova soluzione, invece, viene sfruttata la ISR di fine campionamento dell'ADC per settare un bit del registro `GPIOR0`. Il loop principale rileva il set del flag e aggiunge il valore campionato all'interno dell'array senza spreco di tempo di elaborazione. Questa tecnica è in linea con quanto illustrato nella dispensa del corso, nella sezione relativa al campionamento di segnali analogici.

Struttura dell'Array di Campioni

Nella prima versione del codice, l'array di campioni era costituito da elementi a 16 bit, sprecando 6 bit per valore e riducendo il numero massimo di campioni memorizzabili.

La nuova versione utilizza invece un array di `uint8`, con valori a 8 bit. Grazie all'algoritmo descritto nel flowchart (descritto qui), i valori vengono "compattati" in blocchi consecutivi, eliminando sprechi di spazio in memoria.

Gestione Dinamica dei Parametri

Nel codice precedente, la modifica dei parametri avveniva solo attraverso la modifica manuale delle variabili nel codice sorgente. Nella nuova implementazione, invece, i parametri vengono gestiti dinamicamente tramite la porta seriale. L'utente può impostare valori come la frequenza di campionamento, il livello di trigger e il numero di campioni direttamente in fase di esecuzione, rendendo il sistema più flessibile e adattabile.

Conclusioni

Nonostante la nuova soluzione rispetti tutti i vincoli di progetto e risulti più efficiente e corretta, non è stato possibile portarla a termine con successo. Dopo diversi giorni di debug e analisi delle problematiche, non sono riuscito a ottenere una versione completamente funzionante. Tuttavia, la presento comunque, includendo il link al progetto su WokWi, come prova del tentativo di correzione e dell'analisi aggiuntiva condotta.

Pur non essendo completamente operativa, questa versione risulta teoricamente più coerente e migliorata rispetto alla precedente, e rappresenta un importante passo avanti nella comprensione e ottimizzazione del sistema. Il link della nuova versione del progetto può essere trovato nella sezione [consultabile qui].

2 Sintesi

Descrizione dei componente e del loro comportamento.

2.1 Sintesi componente "Potenziometro"

Figura 6: WokWi Potenziometro

Ruotando la manopola del potenziometro in Wokwi, si modifica la tensione in uscita (OUT). Questa variazione può essere letta da un pin analogico di un microcontrollore tramite un ADC. Il valore letto varia da 0V (GND) a VCC (5V). Nel nostro caso questo componente simula il segnale analogico in ingresso da un componente, con la possibilità di "generare" manualmente il segnale analogico.

Esempio di utilizzo

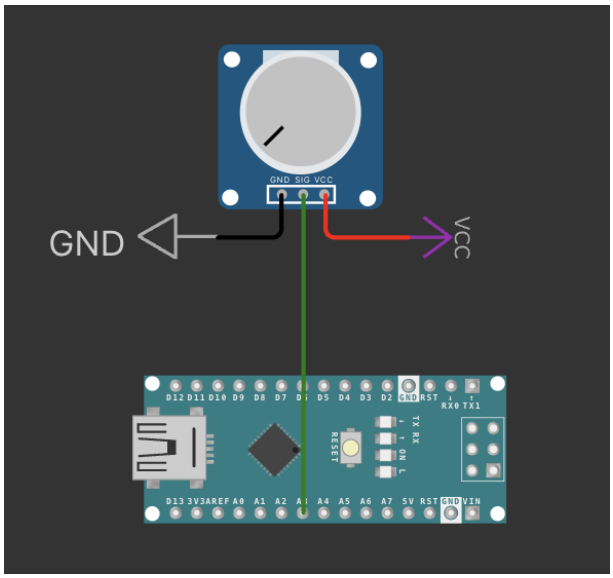


Figura 7: Circuito Potenziometro

Un semplice circuito mostra il collegamento del potenziometro ad Arduino. Come descritto in precedenza, i terminali esterni vanno collegati a GND e VCC, mentre il terminale centrale si connette a un ingresso analogico, in questo caso A3, per la lettura del segnale.

```ino
void setup() {
  // Configura il pin A3 come input
  DDRC &= ~(1 << PC3);  // Imposta il bit 3 del registro DDRC a 0 (input)

  // Configura il riferimento di tensione e il prescaler per l'ADC
  ADMUX = (1 << REFS0);  // Usa VCC come riferimento (5V)
  ADMUX |= (1 << MUX1) | (1 << MUX0);  // Seleziona il canale A3 (MUX1 e MUX0 impostati a 1)

  ADCSRA = (1 << ADEN);  // Abilita l'ADC
  ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);  // Imposta il prescaler a 128 (16MHz/128 = 125kHz)

  // Configura la seriale (Serial.begin(9600))
  Serial.begin(9600);
}

void loop() {
  // Avvia la conversione ADC
  ADCSRA |= (1 << ADSC);

  // Attendi la fine della conversione
  while (ADCSRA & (1 << ADSC));

  // Leggi il valore ADC (10 bit)
  uint16_t adc_value = ADC;

  // Stampa il valore sulla seriale utilizzando Serial.println()
  Serial.println(adc_value);

  // Attendi un po' prima di ripetere la lettura
  delay(100);
}
```

Figura 8: Codice Potenziometro

Un esempio di codice funzionante in cui, tramite i registri di sistema, configuriamo un pin di Arduino come ingresso e impostiamo i valori dei registri dell'ADC. Grazie all'analisi precedente, è stato possibile scegliere il prescaler corretto e, di conseguenza, la frequenza di aggiornamento.

Ecco un esempio di output per capire il funzionamento

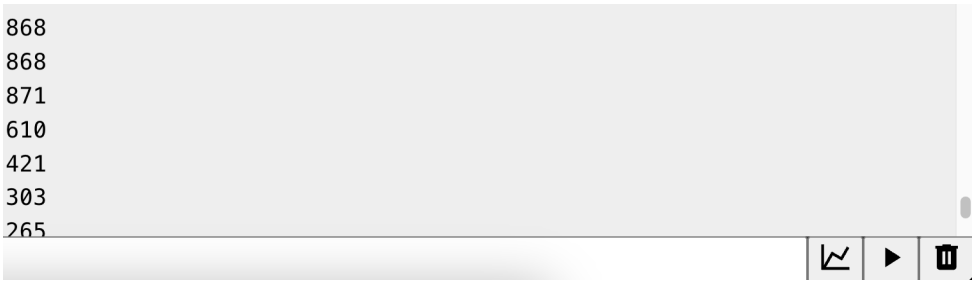


Figura 9: Stampa potenziometro

Ruotando manualmente il potenziometro durante l'esecuzione del codice, l'ADC interno del microcontrollore campiona e stampa i valori rilevati.

2.2 Sintesi componente "Bottone"

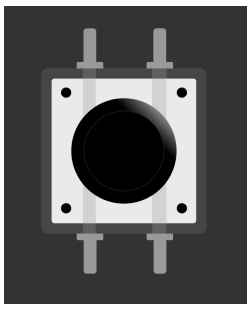


Figura 10: Immagine WokWi Bottone

Il bottone in Wokwi è un interruttore digitale con due stati: premuto o rilasciato. Ha due pin principali: uno collegato a VCC o GND, l'altro a un pin digitale del microcontrollore. Quando premuto, il circuito si chiude e il segnale passa da LOW (0V) a HIGH (VCC). Si può usare una pull-down (resistenza a GND) o una pull-up interna per evitare segnali instabili. Nel nostro caso si è scelto l'utilizzo della resistenza di pull-up interna di arduino.

Esempio di utilizzo

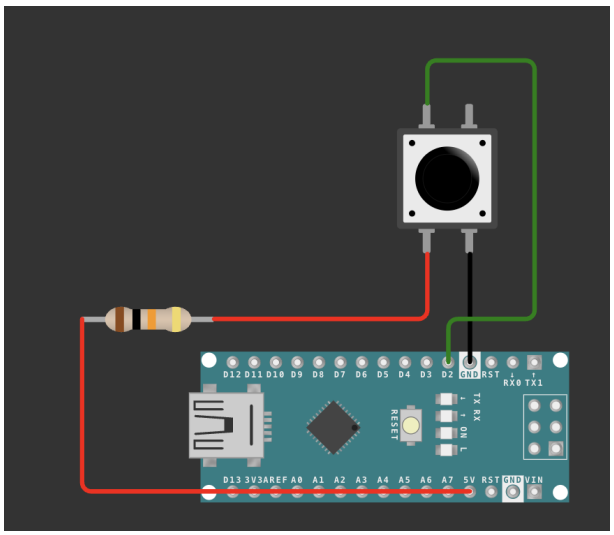


Figura 11: Circuito Bottone

Un semplice circuito mostra il collegamento del bottone. Come descritto in precedenza, i due capi vanno a GND e VCC, mentre uno dei piedini opposti è collegato al pin digitale D2 di Arduino, scelto appositamente per sfruttare gli interrupt.

```
#define MFLAG GPIOR0 // Gestione flag binarie
#define FLAG1 0 // flag per GPIOR0

void setup() {
    // Configura il pin D2 (PD2) come input con pull-up attivato
    DDRD &= ~(1 << PD2);  // Imposta PD2 come input
    PORTD |= (1 << PD2);  // Attiva la resistenza di pull-up interna

    // Configura l'interrupt esterno su INT0 (PD2) per catturare il fronte di discesa
    EICRA |= (1 << ISC01);  // Interrupt su fronte di discesa
    EICRA &= ~(1 << ISC00);
    EIMSK |= (1 << INT0);    // Abilita l'interrupt su INT0

    Serial.begin(9600);
    sei();  // Abilita gli interrupt globali
}

ISR(INT0_vect) {
    if (!bit_is_set(MFLAG, FLAG1)) {
        sbi(MFLAG, FLAG1);
        Serial.println("Bottone Premuto");
    }
}

void loop() {
    if (bit_is_set(MFLAG, FLAG1) && (PIND & (1 << PD2))) {
        cbi(MFLAG, FLAG1); // Resetta la flag solo quando il pulsante è rilasciato
    }
}
```

Figura 12: Codice Bottone

Un esempio di codice funzionante che utilizza i registri di sistema per configurare un pin di Arduino come ingresso, attivare l'interrupt alla pressione del bottone e abilitare la resistenza di pull-up interna. Inoltre, il codice imposta i vari registri necessari, come mostrato nel codice. Per ottimizzare lo spazio, è stato scelto di utilizzare il registro GPIOR0 a 8 bit, come spiegato nelle dispense del corso, evitando flag aggiuntive e sfruttando quelle già presenti e ottimizzate, permettendo inoltre di verificarne il funzionamento.

Esempio di output

Figura 13: Stampa bottone

2.3 Sintesi componente "Led"

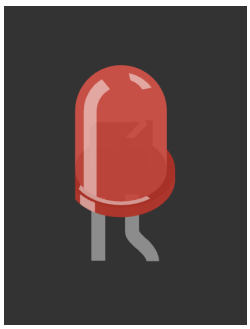


Figura 14: Immagine WokWi Led

Il LED è un diodo che emette luce quando attraversato da corrente. Ha due terminali: Anodo (+) e Catodo (-). Si collega l'anodo a un pin digitale e il catodo a GND. Il LED si accende quando il pin digitale è HIGH (VCC) e si spegne con LOW (GND). Nel nostro caso viene utilizzato per segnalare la fine del campionamento.

Esempio di utilizzo

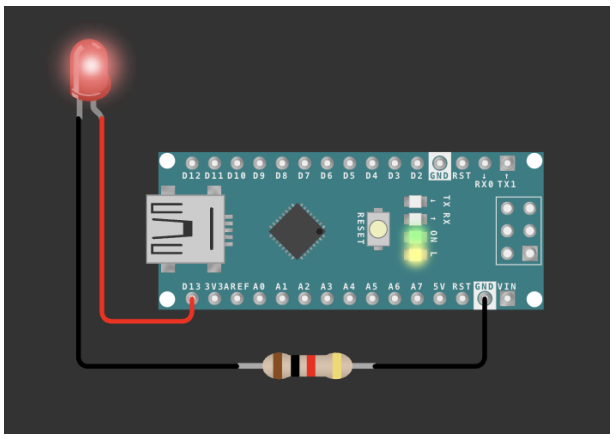


Figura 15: Circuito Led

Un semplice circuito mostra il collegamento del bottone. Il catodo, riconoscibile dal piedino dritto, è collegato a una resistenza da 1 k e poi al GND di Arduino, mentre l'anodo, con il piedino leggermente piegato, è collegato ai 5V.

```arduino
#define LED_PIN PB5  // Il LED su Arduino Uno è sul pin 13 (PB5)

void setup() {
    // Configura il pin del LED come output
    DDRB |= (1 << LED_PIN);

    // Disabilita gli interrupt globali durante la configurazione
    cli();

    // Configura il Timer1 in modalità CTC
    TCCR1A = 0;  // Modalità normale, senza PWM
    TCCR1B = (1 << WGM12) | (1 << CS12) | (1 << CS10); // CTC mode, prescaler 1024

    // Calcolo del valore OCR1A per ottenere una frequenza desiderata
    // Formula: OCR1A = (F_CPU / (2 * prescaler * frequenza_desiderata)) - 1
    // Supponiamo di voler lampeggiare a 2 Hz
    OCR1A = (16000000 / (2 * 1024 * 2)) - 1; // Per 2 Hz

    // Abilita l'interrupt di confronto su OCR1A
    TIMSK1 |= (1 << OCIE1A);

    // Riabilita gli interrupt globali
    sei();
}

// Interrupt Service Routine del TIMER1
ISR(TIMER1_COMPA_vect) {
    PORTB ^= (1 << LED_PIN); // Toggle del LED
}

void loop() {
    // Il codice principale non fa nulla, tutto è gestito dall'ISR
}
```

Figura 16: Codice Led

Un esempio di codice funzionante che, impostando manualmente i registri del microcontrollore, fa lampeggiare il LED a una certa frequenza. Questo viene realizzato utilizzando la Interrupt Service Routine (ISR) del TIMER1, con il confronto del valore nel registro di confronto OCR1A, per ottenere la frequenza desiderata. Il valore nel registro è stato calcolato nel seguente modo:

$$OCR1A = \frac{F_{\text{CPU}}}{2 \times \text{Prescaler} \times f_{\text{desiderata}}} - 1$$

Dove:

- $F_{\text{CPU}} = 16.000.000$ Hz (frequenza del clock del microcontrollore),
- Prescaler $= 1024$ (valore scelto per il Timer1),
- $f_{\text{desiderata}} = 2$ Hz (frequenza del lampeggio).

La frequenza effettiva del Timer1 dopo la divisione del prescaler è:

$$F_{\text{timer}} = \frac{F_{\text{CPU}}}{\text{Prescaler}} = \frac{16.000.000}{1024} = 15625 \text{ Hz}$$

Il Timer1 in modalità CTC genera un interrupt ogni:

$$T_{\text{timer}} = \frac{OCR1A + 1}{F_{\text{timer}}}$$

Imponendo che l'interrupt avvenga ogni 0.5 secondi (metà periodo per ottenere 2 Hz):

$$0.5 = \frac{OCR1A + 1}{15625}$$

Risolviamo per $OCR1A$:

$$OCR1A + 1 = 15625 \times 0.5$$

$$OCR1A = 7812$$

Quindi, per ottenere un lampeggio a 2 Hz, impostiamo:

$$OCR1A = 7812$$

Il calcolo del valore del registro e tutta l'analisi necessaria per il suo funzionamento derivano dalla precedente analisi descritta nella sezione (consultabile qui) **"Ritardo dell'avvio di campionamento"**, in cui vengono definite le formule per calcolare i valori in base alla frequenza desiderata e all'impostazione del prescaler più adatta.

3 Sistema Complessivo

3.1 Manuale utilizzo v1

Per utilizzare il programma, segui questi passaggi:

Configurazione iniziale Prima di avviare l'esecuzione del programma, inserisci i valori desiderati nelle apposite variabili nel codice.

```
35  // ######## Modificare queste variabili per modificare i dati in input ########
36  uint16_t num_samples = 200; // Numero di campioni voluti
37  uint16_t sampling_freq = 200; // Frequenza in Hz (>0)
38  float trigger_voltage = 2.5; // Trigger in Volt --> da 0V a 5.0V
39  uint16_t timeout_ms = 1000; // Ritardo nell'avvio del campionamento in ms
40  uint8_t trigger_source = 0; // Sorgente di trigger (0 = bottone, 1 = interna, 2 = esterna)
41  // ##########################################################################
```

Figura 17: Variabili Codice

Una volta avviato, l'utente può interagire con due potenziometri per simulare un segnale analogico in ingresso:

- **External Trigger Source:** Regola il livello del trigger esterno.
- **Analog Signal Source:** Controlla il segnale da campionare.

Modalità di avvio del campionamento Il campionamento può essere avviato in base alla modalità di trigger selezionata. Per impostarla, scrivi uno dei seguenti valori in una variabile del codice:

- **0 - Bottone:** Il campionamento viene avviato manualmente premendo un pulsante dedicato.

- **1 - Trigger interno:** Il campionamento si avvia automaticamente quando il segnale supera la soglia di trigger impostata.
- **2 - Ingresso analogico riservato:** Il campionamento viene avviato al verificarsi di un segnale proveniente da un ingresso analogico dedicato.

Comportamento del sistema Una volta raggiunta la soglia di trigger:

- Se è impostato un ritardo (delay) in millisecondi, il sistema attenderà il tempo specificato prima di avviare il campionamento.
- Se non è impostato alcun ritardo, il campionamento inizierà immediatamente.

Elaborazione e Output Dopo il campionamento:

- I dati raccolti vengono compressi tramite un apposito algoritmo.
- I risultati vengono visualizzati a schermo.
- Un LED si accende per segnalare il completamento del processo.

Interruzione del campionamento Se desideri interrompere manualmente il campionamento prima della sua conclusione automatica, puoi premere un pulsante dedicato. Il sistema interromperà immediatamente il campionamento e stamperà a schermo i dati raccolti fino a quel momento.

Questa procedura permette all'utente di gestire il processo di acquisizione dei dati in modo flessibile ed efficiente, adattandolo alle proprie esigenze operative.

3.1.1 Link progetto v1

`https://wokwi.com/projects/423404471296505857`

3.2 Manuale utilizzo v2

Versione non terminata che rispecchia maggiormente le specifiche richieste.

Per utilizzare il programma, segui questi passaggi:

Configurazione iniziale Una volta avviato, il programma chiederà all'utente di inserire i parametri necessari per il campionamento, tra cui:

- **Frequenza di campionamento**
- **Sorgente di Trigger**
- **Voltaggio di trigger**
- **Delay**
- **Numero di campioni da registrare**

Dopo aver impostato questi parametri, il programma avvierà il campionamento in base al tipo di trigger selezionato, rispettando il delay specificato.

Simulazione del segnale analogico Sono presenti due potenziometri che simulano un segnale analogico in ingresso:

- **External Trigger Source:** Muovendolo, si modifica il valore letto dall'ADC per il trigger da ingresso analogico riservato.
- **Analog Signal Source:** Muovendolo, si modifica il valore letto dall'ADC per il segnale da campionare.

Semplicemente regolando questi potenziometri, vengono generati valori che l'ADC legge e campiona.

Gestione del campionamento Oltre al trigger automatico, sono disponibili due bottoni per gestire il campionamento manualmente:

- **Bottone per la fine del campionamento anticipata:** Può essere premuto in qualsiasi momento per terminare immediatamente il campionamento e visualizzare i valori raccolti fino a quel momento.
- **Bottone per l'avvio del campionamento:** Funziona come segnale di trigger di avvio manuale.

Questa struttura consente all'utente di controllare in modo flessibile il processo di acquisizione dati, combinando metodi automatici e manuali a seconda delle esigenze operative.

3.2.1 Link progetto v2

`https://wokwi.com/projects/422772694849557505`

3.3 Considerazioni finali

Il progetto rappresenta il frutto di un intenso lavoro di studio, analisi e approfondimento delle numerose specifiche tecniche necessarie per il suo sviluppo. Durante questo percorso, ho avuto modo di confrontarmi con sfide complesse che mi hanno spinto a esplorare in profondità il funzionamento dei registri di Arduino e la loro gestione a basso livello. Sebbene al momento non sia stata raggiunta una versione perfettamente aderente a tutte le specifiche richieste, l'esperienza maturata si è rivelata estremamente formativa e ha ampliato in maniera significativa le mie competenze nel campo della programmazione dei microcontrollori.

Questo progetto ha rappresentato per me una straordinaria opportunità di crescita, permettendomi di consolidare e mettere in pratica i fondamenti teorici acquisiti nel corso di Architettura degli Elaboratori. Grazie alle spiegazioni contenute nelle dispense fornite dal professore, ho potuto comprendere e applicare concetti chiave che, altrimenti, sarebbero rimasti puramente teorici. In particolare, ho imparato a controllare e ottimizzare l'uso dei registri, a gestire in modo efficiente le risorse hardware disponibili e a tradurre specifiche astratte in un'implementazione concreta e funzionale.

L'approccio seguito nel corso mi ha permesso di sviluppare un metodo di lavoro rigoroso e analitico, che sicuramente mi sarà di grande utilità anche in progetti futuri. La chiarezza e la profondità delle lezioni, unite alla qualità del materiale didattico fornito, hanno rappresentato un supporto essenziale nel mio percorso di apprendimento, permettendomi di affrontare le difficoltà con una solida base teorica e una visione più consapevole delle problematiche legate alla programmazione dei microcontrollori.

In definitiva, al di là del risultato finale, considero questa esperienza di grande valore, poiché mi ha fornito competenze fondamentali che vanno ben oltre il singolo progetto, ponendo le basi per uno sviluppo professionale più maturo e consapevole nel settore.