Breve descrizione

Il sistema a bordo missile precedentemente implementato (SBM 2004) acquisiva i campioni di accelerazione con una risoluzione di 8 bit memorizzandoli in una EEPROM, interna al microcontrollore, da 512 byte.

Lo scopo di questo aggiornamento consiste nello sfruttare appieno le potenzialità sia del microcontrollore che della scheda realizzata per la SBM 2004, infatti il convertitore AD interno può lavorare anche con 10 bit quadruplicando quindi la risoluzione in tensione ed inoltre i campioni sono ora memorizzati in una EEPROM esterna (prevista ma non utilizzata nel progetto iniziale) da 64 Kbyte. Forti dell'esperienza precedente e delle prove eseguite sulle memorie seriali con protocollo I^2C, abbiamo potuto implementare il tutto via software lasciando l'hardware immutato; quello che si ottiene è una scheda con una capacità di 32K campioni su 10 bit e data proprio l'elevata capacità della memoria esterna abbiamo potuto abbassare il periodo di campionamento a circa 3,7 ms con una registrazione complessiva di ben due minuti, contro i 15 secondi circa acquisiti con un periodo di campionamento di circa 33 ms. La semplice modifica del software ci ha quindi portato ad un sistema molto più fedele sui dati acquisiti oltre che versatile, infatti queste memorie sono prodotte in tagli di diversa capacità e possiamo scegliere quello più adatto alle nostre esigenze con una piccolissima modifica software e con una spesa economica esigua.

Descrizione del software

Come scritto sopra, le uniche modifiche sostanziali si possono trovare solo nel software. In questa nuova versione si è cambiata anche un po' la logica del programma e abbiamo cercato di migliorarne la leggibilità rinominando registri e facendo ampio uso delle routine (in modo anche da separare quelle porzioni di programma che svolgono una funzione specifica che potrebbe esser utilizzata altrove, vedi ad esempio le routine di scrittura e lettura nella memoria).

Per non ripetere ciò che abbiamo già descritto nel manuale della SBM 2004 e nel manuale dei test sulle memorie I^2C, faremo luce solo sulle differenze e sui punti di maggior interesse.

Notiamo intanto le impostazioni del convertitore AD, per avere un risultato su 10 bit dobbiamo scrivere il campione convertito nei registri ADCH e ADCL con un allineamento a destra, ossia riempiendo tutto ADCL e inserendo i due bit più significativi in ADCH. L'operazione è svolta ovviamente dal convertitore, a patto di modificare una impostazione come segue:

```
; IMPOSTAZIONI DELL'ADC
; configurazione di ADMUX
; riferimento su AVCC 5V,allineamento dato a destra (10 bit),
; canale PC0 all'ADC

.EQU CH0=0x40
```

Seguono le impostazioni per la comunicazione con la memoria esterna che sono le stesse già viste nel manuale dei test.
Andiamo quindi alla rinominazione dei registri che saranno utilizzati prevalentemente per particolari funzioni; abbiamo deciso di semplificare la lettura assegnando un nome ad ogni registro come segue:

```
; definizione dei registri utlizzati per particolari funzioni:
; campione su due byte, byte di controllo della eeprom e indirizzo
; della cella su due byte

.DEF sample_l=R16
.DEF sample_h=R17
.DEF sample_acc_l=R22
.DEF sample_acc_h=R23
.DEF eeprom_address=R18
.DEF eeprom_row_l=R24
.DEF eeprom_row_h=R25

; definizione della soglia di tensione per l'inizio campionamento
; e del registro di start al ciclo di campionamento

.DEF soglia=R14
.DEF start_AD=R19

; definizione delle due soglie inferiore e superiore per determinare
; la fascia di non intervento della registrazione, si utilizza un unico
; registro a 16 bit

.DEF soglia_h=R15
.DEF soglia_l=R13
```

In particolare abbiamo due registri per i campioni e due per la soglia in quanto tratteremo dati su 10 bit, inoltre in sample_acc memorizzeremo il valore precedente di accelerazione ed in sample quello

attuale in analogia al vecchio programma. Abbiamo una sola soglia poiché la utilizzeremo prima per controllare quella superiore e poi per controllare quella inferiore, risulta così un risparmio di due registri creando inoltre una routine apposita per questi controlli.
Poiché la risoluzione è quadruplicata, la nuova fascia di non intervento andrà moltiplicata per quattro, passando da due ad otto:

```
; ogni LSB corrisponde a circa 0.065g di accelerazione, soglia impostata a circa
; g/8

LDI R16,8
MOV soglia,R16
```

Il timer che mantiene la commutazione delle telecamere ha necessità di lavorare su due registri in quanto il periodo di scansione del programma è decuplicato e quindi è necessario un numero di scansioni maggiore per ottenere lo stesso tempo di circa 4 secondi. Per le operazioni che vedremo si necessità di un registro sempre nullo e di uno contenente sempre il valore 1:

```
; azzera un registro per utilizzarlo come zero fisso nell'incremento
; del timer telecamere

CLR R5
CLR R4

; prepara 1 nel registro utilizzato per l'incremento con riporto per lo
; stesso timer

INC R4
```

Fatto questo, possiamo andare a selezionare una delle due grandi funzionalità del programma, l’acquisizione e la lettura dei dati, decise dalla posizione del solito jumper su scheda. Andiamo ad analizzare la funzione di acquisizione:

Acquisiamo intanto un primo campione col quale calcoleremo la fascia di intervento alla prima scansione del programma:

```
scrivi: ; effettua un primo campionamento
        CALL AD_converti
```

Il registro start_AD consente di memorizzare lo stato del programma e seguire in tal modo un opportuna strada, ad esempio scriveremo 2 in start_AD quando la memorizzazione nella eeprom è terminata ed in tal modo dovremo saltare tutte le istruzioni di acquisizione e memorizzazione andando soltanto a controllare le telecamere

```
sample: ; a memorizzazione terminata blocca l'acquisizione ed il LED
        ; resta spento
        CPI start_AD,2
        BREQ salta
```

Se invece non abbiamo terminato la scrittura in memoria, o non l'abbiamo neppure cominciata, dovremo continuare la conversione; i dati precedentemente convertiti e memorizzati nei registri sample vengono copiati in sample_acc che diverrà l'accelerazione acquisita precedentemente

```
        ; accende il LED di attesa acquisizione
        SBI PORTC,PC3
        ; copia l'acquisizione attuale in due registri di appoggio
        MOV sample_acc_l,sample_l
        MOV sample_acc_h,sample_h
```

Passiamo poi a copiare in sample il nuovo valore della accelerazione, diciamo quello attuale

```
        ; conversione di un nuovo valore
        CALL AD_converti
```

Coi valori acquisiti in precedenza costruiamo la nuova fascia di non intervento e controlliamo se questa è superata dal valore attuale di accelerazione, tutto questo è svolto dalla routine compara

```
        ; compara i campioni con le soglie, se le superano
        ; avvia la registrazione (la routine compara scrive 1 in
        ; start_AD in tal caso )
        CALL compara
```

Se la fascia è stata superata la routine compara scrive 1 in start_AD e nel qual caso potremo iniziare la memorizzazione dei campioni acquisiti e stoccati in sample_acc, altrimenti potremo saltare questa fase e passare al controllo telecamere

```
CPI start_AD,1
BRNE salta
```

La scrittura in memoria avviene chiamando la routine dedicata eeprom_scrivi, successivamente dovremo incrementare la posizione degli indirizzi di 2 (il campione occupa due byte) e controllare se la memoria è stata completamente scritta

```
; scrive in memoria il valore campionato precedentemente
; in questo modo memorizza anche il valore all'istante zero,
; spegne il LED, acquisizione in corso con luminosità attenuata
CBI PORTC,PC3
; scrive in memoria
CALL eeprom_scrivi
; prepara la riga successiva
ADIW eeprom_row_h:eeprom_row_l,2
```

Se la memoria è da 64Kbyte come quella utilizzata quì, con l'ultimo incremento si ottiene un overflow su 16 bit che è segnalato nel flag di riporto del registro di sistema SREG, per cui basta andare a controllare tale flag (bit 0 di SREG)

```
; controlla se abbiamo riempito la memoria: la somma precedente
; dà un riporto
IN R10,SREG
SBRC R10,0
; a memoria piena termina l'acquisizione segnalandolo in start_AD
LDI start_AD,2
```

È interessante notare che per scrivere memorie a capienza inferiore basterà controllare quale bit si è settato nel registro eeprom_row_h, ad esempio per 32Kbyte basterà osservare il bit 7, per 16 Kbyte il bit 6 e così via dimezzando la capacità e scorrendo all'indietro di una posizione alla volta nel registro.

Da misure effettuate nei test, si è notato che il tempo medio di scrittura nella eeprom per un campione è di circa 3.7 ms. Quando la memorizzazione è attiva il programma viene scorso praticamente con tale tempo e così il timer telecamere lo prenderà come periodo base per il conteggio. Per assicurarci una corretta commutazione delle telecamere anche nel caso di memorizzazione assente, dobbiamo inserire una pausa della stessa durata di 3.7 ms circa che verrà eseguita quando start_AD è diverso da 1 (che indica la memorizzazione assente)

```
salta:  ; simula temporalmente la scrittura di un campione se non è stata
        ; avviata l'acquisizione, in modo da garantire una commutazione
        ; corretta delle telecamere
        CPI start_AD,1
        BREQ timer
        CALL wait_4ms
```

Con un periodo base di circa 4 ms, la commutazione telecamere che mantenga il video laterale per circa 4 s si ottiene contando un migliaio di impulsi base; quindi potremo incrementare il registro R21-R20 fino a che non va alto l'undicesimo bit, ossia il bit numero 2 di R21. A conteggio avvenuto dovremo saltare l'istruzione di incremento che viene eseguita solo su R20, mentre R21 viene sommato a zero col riporto da R20 (in questo modo si realizza un incremento di un registro unico R21-R20 su 16 bit)

```
        ; conteggio temporale se è avvenuto il distacco,
        ; blocca il conteggio se il timer è giunto alla fine
timer:  SBRS R21,2
        ADD R20,R4
        ADC R21,R5
```

Se non c'è il segnale di distacco delle sezioni, potremo azzerare i contatori del timer in quanto lo stesso deve essere inattivo

```
        ; controlla se è avvenuto il distacco sezioni, in caso negativo azzera i
        ; contatori
        SBIC PIND,PD2
        CALL clear_reg
```

A questo punto dovremo effettuare alcune decisioni a seconda del valore del timer e del segnale di commutazione telecamere, questo ci consente di tornare sul video laterale se viene a mancare il segnale di distacco sezioni anche con il timer in conteggio e non ancora terminato.
Il programma torna poi alla fase di riconversione dei campioni seguendo il suo normale percorso ciclico

```
        ; se il timer ha terminato il conteggio passa alla telecamera di lato
        ; se c'è il segnale distacco sezioni ed il timer non ha terminato il
        ; conteggio, commuta sulla telecamera di fondo.
        SBRC R21,2
        JMP video_l
        SBIC PIND,PD2
        JMP video_l
```

Ad eseguire le successive due istruzioni, ossia a commutare sulla telecamera di fondo, si giunge quindi se abbiamo il segnale di distacco e se il timer non ha ancora terminato il conteggio

```
        SBI PORTD,PD5
        JMP sample
```

Se non abbiamo il segnale di distacco sezioni o se il timer ha terminato il conteggio si torna sulla telecamera laterale, con possibilità di rieseguire una commutazione semplicemente ricollegando e successivamente ricollegando le sezioni; questa funzionalità è molto importante in quanto dobbiamo tenere conto delle fasi di test del programma su rampa, che invia un segnale di distacco sezioni mentre il missile è ancora montato sulla rampa.

```
        ; in tutti gli altri casi lascia la telecamera laterale
video_l:CBI PORTD,PD5
        JMP sample
```

Come preannunciato, il programma presenta delle novità rispetto alla precedente versione anche nell'implementare certe funzionalità, sebbene sia rimasto del tutto invariato il suo scopo.

Per la fase di lettura l'unica novità è il trattamento dei dati a 16 bit, ma non vi sono grandi stravolgimenti della struttura:

```
leggi:  ; prepara indirizzi iniziali
        CLR R24
        CLR R25
        CLR R22
        CLR R23
        ; trasferimento al PC con handshake
read:   CALL PC_ricevi
        CALL eeprom_leggi
        ; invia il dato al PC
```

```
        MOV R10,sample_acc_h
        CALL PC_invia
        CALL PC_ricevi
        MOV R10,sample_acc_l
        CALL PC_invia
        ;prepara la riga successiva
        ADIW eeprom_row_h:eeprom_row_l,2
        ; controlla se abbiamo letto tutta la memoria
        IN R10,SREG
        SBRS R10,0
        JMP read
fine:   JMP fine
```

Non ci resta che da vedere le routine, in particolare per quelle relative alla memoria abbiamo "tagliato" alcune porzioni del programma di test in modo da isolare l'insieme di istruzioni atte alla scrittura ed alla lettura. Una funzione completamente nuova è quella di comparazione delle due soglie su 16 bit che è stato necessario aggiungere anche per non appesantire troppo il programma principale.

Cominciamo dalla routine di scrittura nella memoria, a causa del numero piuttosto limitato di registri utilizzabili direttamente dal programma (senza cioè stivare i dati nella memoria SRAM accessibile con opportune istruzioni), abbiamo necessità di salvare temporaneamente il registro R16 nella pila poiché sarà utilizzato dalla routine per appoggiarci alcuni dati, mentre la sua funzione all'esterno è quella di memorizzare il byte basso dei campioni attuali

```
; routine per la scrittura nella memoria

eeprom_scrivi:; salva momentaneamente R16 per utilizzarlo come appoggio
                PUSH R16
```

A questo punto comincia la comunicazione con la memoria secondo il noto protocollo visto nel manuale del programma test:

```
                ; invia lo start e il codice di scrittura
avvio:          CALL E2PROM_start
                LDI eeprom_address,eeprom_address_w
                OUT TWDR,eeprom_address
                CALL E2PROM_invia
                ; controlla che la memoria sia pronta per la scrittura
```

```
IN R16,TWSR
ANDI R16,0xF8
CPI R16,0x18
BRNE avvio
```

Spediamo poi gli indirizzi del campione in memoria

```
; invio indirizzo su due byte
OUT TWDR,eeprom_row_h
CALL E2PROM_invia
OUT TWDR,eeprom_row_l
CALL E2PROM_invia
```

Caricati gli indirizzi e informata la memoria che vogliamo accedere in scrittura spediamo i due byte del campione acquisito, come noto si carica sempre il dato convertito nel ciclo di scansione precedente, in modo da memorizzare anche il valore appena prima dell'avvio alla registrazione (istante zero)

```
; invia 2 byte dati e stop trasmissione
OUT TWDR,sample_acc_h
CALL E2PROM_invia
OUT TWDR,sample_acc_l
CALL E2PROM_invia
CALL E2PROM_stop
```

Terminata la registrazione, riprende dalla pila il valore originale di R16 ed esce

```
; riprende R16
POP R16
RET
```

Questa routine utilizza delle altre procedure che sono le stesse viste nel programma di test, ossia l'invio dello start, del dato e dello stop.

Passiamo quindi alla routine di lettura, abbiamo al solito una prima comunicazione con la memoria in modalità scrittura per caricare gli indirizzi, dopodichè si passa in lettura per ricevere i due byte del campione registrato precedentemente:

```
; routine per la lettura dalla memoria

eeprom_leggi:; trasmissione dell'indirizzo alla memoria
             CALL E2PROM_start
             LDI eeprom_address,eeprom_address_w
             OUT TWDR,eeprom_address
             CALL E2PROM_invia
             ; controlla se la memoria non è occupata
             IN R16,TWSR
             ANDI R16,0xF8
             CPI R16,0x18
             BRNE eeprom_leggi
             ; trasmissione dei due byte di indirizzo riga
             OUT TWDR,eeprom_row_h
             CALL E2PROM_invia
             OUT TWDR,eeprom_row_l
             CALL E2PROM_invia
             ; lettura dei dati
             CALL E2PROM_start
             LDI eeprom_address,eeprom_address_r
             OUT TWDR,eeprom_address
             CALL E2PROM_invia
             CALL E2PROM_ricevi
             RET
```

Questa procedura utilizza una routine importante, la EEPROM_ricevi, che consente di leggere due byte consecutivi dalla memoria, chiudendo la comunicazione con lo stop ed inserendo i due byte letti nei registri sample_acc che verranno poi trasmessi al PC

```
E2PROM_ricevi: LDI R16,eeprom_ack
               OUT TWCR,R16
wait_e2_ack:   IN R16,TWCR
               SBRS R16,TWINT
               JMP wait_e2_ack
               IN sample_acc_h,TWDR
               LDI R16,eeprom_nack
               OUT TWCR,R16
wait_e2_nack:  IN R16,TWCR
               SBRS R16,TWINT
               JMP wait_e2_nack
               IN sample_acc_l,TWDR
```

```
                CALL E2PROM_stop
                RET
```

I dati convertiti su 10 bit necessitano di una opportuna routine che estrae i due byte dai registri del convertitore integrato, ADCH e ADCL. La modalità è la seguente: il byte meno significativo viene scritto in ADCL e i due bit rimanenti prendono posizione sui due bit meno significativi di ADCH. Per sicurezza azzeriamo tutti i bit di ADCH che non sono interessati alla conversione, mediante una ANDI con un'opportuna maschera che azzera tutti i bit successivi al secondo meno significativo:

```
; routine per la conversione AD

AD_converti: LDI R16,AD_en
             OUT ADCSR,R16
wait_AD:     SBIC ADCSR,ADSC
             JMP wait_AD
             IN sample_l,ADCL
             IN sample_h,ADCH
             ANDI sample_h,0x03
             RET
```

Non ci resta che esaminare la routine compara, le sue funzioni sono ormai note e cioè creare le soglie superiori ed inferiori dal campione precedentemente acquisito e compararle col campione attuale, se quest'ultimo esce dal range di non intervento segnaliamo in start_AD l'inizio della registrazione in memoria. Per non sprecare troppi registri abbiamo fatto uso dei due soglia_h e soglia_l con i quali calcoleremo una soglia alla volta:
Iniziamo col creare la soglia superiore, al campione precedente sommiamo la soglia pari ad 8 per generare la semifascia superiore di non intervento:

```
; routine per la comparazione dei campioni con le soglie
; calcolate sui valori precedentemente acquisiti

compara: MOV soglia_h,sample_acc_h
         MOV soglia_l,sample_acc_l
         ADD soglia_l,soglia
         ADC soglia_h,R5
```

Il microcontrollore non consente di effettuare direttamente una comparazione su 16 bit, per cui dobbiamo inserire una serie di istruzioni che realizzino tale funzione, se il campione attuale supera

la soglia superiore dobbiamo uscire dalla routine con start_AD = 1, altrimenti eseguiremo il confronto sulla soglia inferiore. Il confronto su 16 bit tiene conto di quanto segue per decidere se sample ha superato la soglia:

Se il byte più significativo di sample è superiore a quello della soglia

Se il byte più significativo di sample è uguale quello della soglia ma il byte meno significativo di sample è maggiore di quello della soglia

```
; se il byte più significativo è inferiore alla soglia
; vai subito al controllo inferiore
sopra:   CP sample_h,soglia_h
         BRLO sotto
; se il byte più significativo è maggiore segnala subito
; la registrazione
         CP sample_h,soglia_h
         BRNE rec
; col byte più significativo uguale e quello meno
; significativo maggiore o uguale segnala la registrazione
         CP sample_l,soglia_l
         BRSH rec
```

Se non siamo sopra la soglia superiore confrontiamoci con quella inferiore:

```
; sicuramente ora siamo sotto la soglia superiore
sotto:   MOV soglia_h,sample_acc_h
         MOV soglia_l,sample_acc_l
         SUB soglia_l,soglia
         SBC soglia_h,R5
```

Adesso la regola per vedere se abbiamo superato (ossia se siamo scesi sotto) la soglia inferiore è la seguente:

Se il byte più significativo di sample è inferiore a quello della soglia

Se il byte più significativo di sample è uguale a quello della soglia ma il byte meno significativo di sample è inferiore a quello della soglia

```
; se il byte più significativo è inferiore segnala
; l'acquisizione
         CP sample_h,soglia_h
         BRLO rec
; se il byte più significativo è maggiore esci poichè
```

```
; il campione è nel gap di sicurezza
         CP sample_h,soglia_h
         BRNE fine_k
; col byte più significativo uguale e quello meno
; significativo maggiore o uguale esci
         CP sample_l,soglia_l
         BRSH fine_k
rec:     LDI start_AD,1
fine_k:  RET
```

Si noti che la routine compara modifica start_AD al massimo scrivendoci 1, altrimenti lo lascia invariato e perciò può essere eseguita anche durante la registrazione senza temere un eventuale blocco dovuto al cambiamento di start_AD. L'unico cambiamento possibile di start_AD a registrazione avviata lo compie la parte di programma che si occupa della registrazione stessa, scrivendovi 2 a memoria esaurita.

Il nuovo programma per SBM necessita di un opportuno programma su PC col quale andremo ad estrarre i campioni memorizzati su scheda, rispetto alla precedente versione questo programma non è cambiato di molto, apparte ricevere due byte per lo stesso campione. Per ricostruire un numero a 16 bit da due byte distinti basta eseguire una semplice operazione, ossia moltiplicare il byte più significativo per 256 e poi sommare il byte meno significativo, il valore così ottenuto viene scritto nel solito file che poi sarà trattato dal software di costruzione dei grafici.