Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

Radiometro spettrale ad immagine VIRS-201: specifiche del software di elaborazione dei dati nella banda visibile

L. Bedini A. Ribolini

Nota Interna B4-20

Giugno 1992

Lavoro eseguito nell'ambito di una collaborazione con Sie-Lab s.r.l. e Officine Galileo S.p.A.

I N D I C E

1. SCOPO

Scopo del presente documento e' definire le specifiche del software residente sulla carta DSP_VIS/NIR del sistema di rilevazione radiometrica VIRS201.

2. VALIDITA'

Questo documento ha validita' per il solo progetto MCS-VIRS201 e fa riferimento al relativo "piano di Qualita'" per quanto riguarda l'utilizzazione di tecniche e strumenti di software engineering per la definizione dell'architettura del progetto stesso.

3. DEFINIZIONI, ACRONIMI E ABBREVIAZIONI

ASCII	American Standard Code International Interchange
byte	dato intero senza segno a 8 bits
CASE	Computer Aided Software Engineering
CFD	Control Flow Diagram
CO.S.C.	COntrollore Sottosistemi Comunicanti
CPU	Central Processing Unit
DFD	Data Flow Diagram
doubleword	dato intero senza segno a 32 bits
DPRAM	Dual Port Random Access Memory
HW	HardWare
ISR	Interrupt Service Routine
MCS	Modulo Controllo Sistema
MES	Modulo Elaborazione Sistema
real	dato floating-point a 32 bits
RTE	Real Time Executive
STD	State Transition Diagram
SW	SoftWare
VIRS	Visible Infra-Red Scanner
word	dato intero senza segno a 16 bits

4. RIFERIMENTI

OFFICINE GALILEO ST91I0021
Requisiti per il software on-line del modulo di controllo ed

acquisizione

OFFICINE GALILEO ST/I-91-007/3
Requisiti del software VIRS-201 Modulo di Elaborazione - MES - DSP_CCD

5. PROGETTAZIONE DEL SISTEMA

5.1 Metodologia e ambiente di progettazione

L'ambiente di progettazione e' costituito da un PC IBM o compatibile con sistema operativo MS-DOS; su tale ambiente e' installata la fabbrica SW Motorola costituita da:

- Assemblatore Motorola ASM56000
- Linker Motorola LNK56000
- Simulatore Motorola SIM56000
- In-Circuit Emulator Motorola ADS56000

L'ambiente target e' costituito dalla scheda DSP-VIS/NIR basata sul processore Motorola DSP56001

Il SW e' stato strutturato in maniera top-down in componenti di livello sempre piu' basso.

Vengono dapprima individuate le componenti costituenti il software, ed evidenziate le loro interconessioni. Vengono quindi dettagliate, sia con schemi di flusso che con un linguaggio pseudo-C, le funzioni svolte da ciascuna componente.

5.2 Scomposizione del sistema nelle sue componenti

Con riferimento alla metodologia evidenziata nel paragrafo precedente e al documento di requisiti del software, il SW applicativo residente sulla carta DSP-VIS/NIR puo' essere descritto come un unico sistema, costituito dai moduli PRINCIPALE, ISR_TIMER ed ISR_FRAME, che scambia dati e informazioni con gli elementi esterni a cui deve essere collegato.
Lo schema riportato in figura 1, evidenzia tali interfacce esterne ed il relativo flusso di dati nel suo complesso.

(FIGURA 1)

Non avendo a disposizione un ambiente operativo di tipo multi-tasking real-time, le operazioni vengono svolte totalmente dalla componente PRINCIPALE che, a sua volta, puo' essere interrotta dalle due ISR inizializzate dalla stessa.
La componente ISR_FRAME e' essenziale in quanto deve gestire l'interrupt generato dal segnale di sincronismo frame_sync connesso come interrupt esterno IRQA.
La componente ISR_TIMER interviene ogni 10 msecondi ed e' utilizzata come base tempi di riferimento per eseguire il reset ciclico del watch-dog timer ogni 500 msecondi. Il processo PRINCIPALE inoltre espleta le proprie funzioni svolte sincronizzandosi con gli eventi generati da tale componente (lettura della memoria a doppia porta, scrittura nella stessa ecc.).
La componente ISR_TIMER e' anche utilizzata per la lettura della DPRAM (ogni 10 msecondi) ed e' la sola che in assenza di errori puo' inizializzare una nuova modalita' di funzionamento.

5.3 Definizione funzionale di ciascuna componente

Di seguito vengono descritte le funzioni svolte da ciascuna componente costituente il sistema.

5.3.1 Componente PRINCIPALE

La componente PRINCIPALE e' quella che viene attivata all'accensione o al reset della carta DSP-VIS/NIR. Fanno parte di detta componente le procedure atte alla gestione di tutte le modalita' di funzionamento previste nella specifica in riferimento.
Le modalita' svolte sono le seguenti:

0 = stand by
1 = setup
2 = diagnostica
3 = operativo
4 = calibrazione
5 = comandi registratore
6 = selezione bande VIS
7 = impostazione guadagno - shutter VIS
8 = impostazione parametri IR
9 = ripristino configurazione
10 = errore

Dopo la fase iniziale o power-up il modulo si pone in stand-by in attesa di ricevere dalla componente ISR_TIMER il comando relativo ad una nuova modalita' di funzionamento.
La componente PRINCIPALE si deve intendere sempre attiva in quanto puo' essere interrotta solamente da una delle altre due componenti ISR_TIMER o ISR_FRAME.

5.3.2 Componente ISR_TIMER

La ISR_TIMER e' attivata, ogni 10 msec, dall'interrupt generato dal timer interno al DSP56000; la routine svolge i seguenti compiti principali:

a) legge la cella di memoria della dual-port memory in cui la CO.S.C-A ha memorizzato la modalita' di funzionamento;
b) controlla, attraverso un time_out, la presenza del sincronismo di frame;
c) controlla il valore di una variabile atta a segnalare la presenza di errori nel modulo principale;

d) trasmette al modulo principale l'identificatore (modo_c) della modalita' di funzionamento; nel caso riveli la mancanza del sincronismo di frame o la presenza di un errore nel modulo principale(vedi punto c) trasmette la modalita' di funzionamento "MALFUNCTION";
e) invia, ogni 500 msec (e, al massimo, ogni 500 msecondi), il segnale di reset del watch dog timer;
f) incrementa ogni 500 msecondi il contenuto della cella DSP_RUN della dual-port memory.

5.3.3 Componente ISR_FRAME

La ISR_FRAME e' attivata, ogni 34 msec., dal segnale di sincronismo di frame. La routine invia al modulo principale il segnale di sincronizzazione flag_fs necessario per lo svolgimento di quelle funzioni in cui e' richiesta la sincronizzazione tra l'attivita' di elaborazione, di acquisizione e di trasferimento dei dati.
Il segnale flag_fs e' inviato anche alla ISR_TIMER che lo utilizza per l'espletamento dell'attivita' b (vedi 5.3.2).

5.4 Struttura dati ed interfacce fra i componenti

Per quanto riguarda la struttura dei dati si rinvia alla progettazione dettagliata delle componenti costituenti il

sistema.
Di seguito si riporta una descrizione dettagliata delle interfacce tra le componenti e tra il sistema ed il mondo esterno. Sono previste le seguenti interfacce:

- Interfaccia tra PRINCIPALE e esterno
- Interfaccia tra PRINCIPALE e ISR_TIMER
- Interfaccia tra PRINCIPALE e ISR_FRAME
- Interfaccia tra ISR_TIMER e ISR_FRAME
- Interfaccia tra ISR_FRAME e esterno

5.4.1 Interfaccia tra PRINCIPALE e esterno

RAM_FRAME:
array di memoria di 16384 words che conterra' un frame di dati generati dalla testa CCD, ed acquisiti in sincronismo con la variabile FLAG_FS, aggiornata dalla componente ISR_FRAME.

RAM_DP:
array di memoria (dual-port memory) di 4096 words attraverso la quale la componente PRINCIPALE acquisisce le informazioni operative e restituisce i risultati elaborati.

ADD_LATCH:
un byte attraverso il quale si potra' scegliere la pagina della memoria frame visualizzabile attraverso RAM_FRAME.

HARD1_REG:
un byte attraverso il quale il software puo' inizializzare e/o monitorare alcune funzioni hardware della scheda.

OUT_REG:
un byte attraverso il quale il software effettua la scrittura del record a lunghezza fissa verso il registratore.

HARD2_REG:
un byte attraverso il quale il software potra' inizializzare e/o monitorare alcune funzioni hardware della scheda.

5.4.2 Interfaccia tra PRINCIPALE e ISR_TIMER

MODO_C:
una word che conterra' il numero della modalita' di funzionamento attivata dalla CO.S.C-A, o dalla modalita' "MALFUNCTION"; detta

variabile verra' modificata in istanti sincronizzati dalla ISR_TIMER.

ERRORE:
una word che conterra' l'identificatore dell'eventuale errore trovato durante l'esecuzione del programma, oppure il valore 0 nel caso in cui non si verifichi una condizione di errore.

FLAG_TO:
una word che verra' settata a zero quando utilizzata all'interno dell'intervallo base di 10 msecondi e verra' settata ad uno in concomitanza con lo scadere dell'intervallo di 10 millisecondi.

5.4.3 Interfaccia tra PRINCIPALE e ISR_FRAME

FLAG_FS:
una word che verra' settata a uno dalla ISR_FRAME in modo da indicare la corretta segnalazione da parte del segnale FRAME_SYNC. Quando utilizzata all'interno dell'intervallo compreso tra un segnale FRAME_SYNC e il successivo dalla componente PRINCIPALE, e' compito della stessa riazzerare detta word onde evitare la perdita della successiva segnalazione.

5.4.4 Interfaccia tra ISR_TIMER e ISR_FRAME

FLAG_FS:
una word che verra' settata a uno dalla componente ISR_FRAME per segnalare l'avvenuto riconoscimento della linea FRAME_SYNC (Interrupt IRQA del processore). Quando utilizzata all'interno dell'intervallo compreso tra un segnale FRAME_SYNC e il successivo dalla componente PRINCIPALE, e' compito della stessa riazzerare detta word onde evitare la perdita della successiva segnalazione.

COUNT_FS:
una word che verra' settata a zero ogni qualvolta arriva un evento dalla sottocomponente ISR_FRAME, e incrementata ad ogni evento della sottocomponente ISR_TIMER. Nel caso che detta variabile arrivi ad essere incrementata ad un valore di 4 (40 msec.) la ISR_TIMER attiva la condizione di errore "assenza frame_sync".

5.4.5 Interfaccia tra ISR_FRAME e esterno

FRAME_SYNC:
e' un segnale di interrupt generato ogni 34 millisecondi dall'elettronica esterna presente sulla componente CCD. Detto segnale e' connesso all'ingresso IRQA del 56000 opportunamente programmato per la gestione dell'interrupt esterno.

5.5 Flusso di controllo

Quando una nuova modalita' di funzionamento (nel seguito chiamata funzione) e' attivata si possono avere fondamentalmente due casi:

a) la funzione non utilizza i segnali di sincronismo generati dalla ISR_FRAME: l'attivita' del modulo relativa all'espletamento della funzione attivata viene eseguita prima che la ISR_TIMER possa selezionare una diversa funzione; il modulo esegue e si pone in attesa di un nuovo comando;

b) la funzione utilizza il segnale di sincronismo generato dalla ISR_FRAME: l'attivita' del modulo e' svolta in cicli successivi; ciascun ciclo si sincronizza con il segnale generato dalla ISR_FRAME. Il modulo esegue e, al termine di ciascun ciclo, controlla se e' stata selezionata una diversa funzione.

E' prevista l'utilizzazione di un vettore func_status[] di variabili atte a riassumere lo stato dell'attivita' in corso per ciascuna delle funzioni. Nel caso non venga riconosciuta una modalita' di funzionamento prevista (n>10), viene inviata una segnalazione di errore. Ad ogni funzione (individuata nel pseudo-codice con function[n], n=0-10), e' associata una variabile di stato func_status[n]. Tale variabile assume il valore -1, quando la funzione non e' attivata. La modalita' di funzionamento selezionata da ISR_TIMER e' memorizzata nella variabile modo_c.
Lo schema di flusso delle varie modalita' di funzionamento e' riportato in figura 2.

Le variabili di stato func_status[] assumono i seguenti valori:

a) per le funzioni con indice n diverso da 3 e da 7:

func_status[n]=-1	quando le funzioni non sono state attivate
func_status[n]=0	quando le funzioni sono in esecuzione o sono state eseguite

b) per la funzione con indice n=3:

func_status[n]=-1 quando la funzione non e' stata attivata

func_status[n]=e	e >= 0, quando la funzione e' stata attivata; la quantita' e esprime il numero dei frame trasferiti dalla testa VIS-NIR

c) per la funzione con indice n =7:

func_status[n]=-1	quando la funzione non e' stata attivata
func_status[n]=e	0<=e<=10 quando la funzione e; in corso;la quantita' e esprime il numero dei frame trasferiti dalla testa VIS_NIR
func_status[n]=10	quando la funzione e' stata eseguita

Per ogni scheda DSP e' definita una variabile PRIMO_FRAME che segnala la presenza di un frame valido sulla ram X della scheda DSP.
Tale variabile e' aggiornata dalle funzioni con indice 3 o 7. Assume il valore 1 quando e' presente un frame valido.

5.5.1 Sincronizzazione con la componente ISR_FRAME.

Per la sincronizzazione con ISR_FRAME viene utilizzato un flag flag_fs. Tale flag e' settato dalla ISR_FRAME e resettato dai moduli principali, operanti in ciascuna scheda, che lo utilizzano.
Il meccanismo di sincronizzazione e' reso operativo solo per le funzioni 3 e 7. In fase di inizializzazione (power-up o reset) si pone flag_fs=0.
Al fine di assicurare l'uscita dal ciclo di attesa su flag_fs, anche quando manchi il segnale di sincronismo di frame, viene utilizzato il time out gestito dalla ISR_TIMER. Allo scadere del time out prefissato,e cioe' nel caso manchi il sincronismo di frame, ISR_TIMER forza modo_c=10 (MALFUNCTION) e setta la variabile error al valore ERR_CODE quindi pone flag_fs=1 per forzare l'uscita dalla fase di attesa.
5.5.2 Gestione degli errori

Si hanno fondamentalmente due tipi di errori:

a) errori rilevati dal modulo principale, quali, ad esempio, modo_c > 9 o numero di frame trasferiti dalla RAM X maggiore di NMAX;
b) mancanza del sincronismo di frame, rilevata attraverso il time out gestito da ISR_TIMER.

Nel caso a il modulo principale utilizza la variabile error per segnalare la presenza e il tipo di errore rilevato; ISR_TIMER controlla la variabile error e forza la modalita' di funzionamento MALFUNCTION.
Nel caso b l'errore viene direttamente rilevato da ISR_TIMER che

aggiorna la variabile error e forza la modalita' di funzionamento MALFUNCTION.
La valore della variabile error e' trasferito nella cella ERRORE della dual port in modo da rendere l'informazione disponibile a CO.S.C-A. Il modulo principale, dopo aver segnalato l'errore, resta in attesa di un segnale di acknowledge dalla CO.S.C-A e della funzione STAND_BY.

5.6 Flusso dei dati

La modalita' operativa prevede un trasferimento di dati dalla RAM_FRAME a dalla RAM_DP alla memoria locale e da questa al registro OUTREG di interfaccia con la componente registratore. La modalita' GAIN_SHUTTER prevede un trasferimento da RAM_FRAME e memoria locale e da questa a RAM_DP. Per una descrizione dettagliata delle modalita' di trasferimento si invia a 6.1.1, a 6.1.2.2.5 e 6.1.2.2.9.

6. PROGETTAZIONE DETTAGLIATA DEI COMPONENTI

6.1 Componente PRINCIPALE

6.1.1 Descrizione funzionale

Nel presente paragrafo vengono descritte in dettaglio le funzione svolte dal modulo principale.

MODALITA' STAND_BY

In questa modalita' le variabili di stato associate a ciascuna funzione vengono reinizializzate al valore -1 e la variabile error resettata a 0.

MODALITA' SETUP

Non viene eseguita nessuna attivita'. Il controllo viene restituito immediatamente al modulo principale.

MODALITA' DIAGNOSTICA

Il modulo esegue la seguente attivita':

a) verifica che in coda al campo dati della testa VIS-NIR, sulla RAM Y del DSP56000, siano presenti 64 byte con contenuto 2EH;
b) controlla che nella Dual Port memory, all'indirizzo Y:8C80 sia

memorizzato il valore FALSE (esito diagnostica);
c) acquisisce (solo il DSP 0) un frame campione dalla testa CCD e ne controlla la costituzione come segue:
le righe pari 0,2,4,6,... 18 devono contenere 512 valori = 0
le righe dispari 1,3,5,7,...19 devono contenere valori <> 0
d) comunica alla CO.S.C-A, tramite la Dual Port il risultato della diagnostica ('O' = risultato OK, 'K' = esito negativo).

MODALITA' OPERATIVA

Il modulo esegue le seguenti attivita':

a) controlla, tramite la Dual Port, quale scheda DSP e' in fase di acquisizione dati dalla testa VIS-NIR. Per tale controllo utilizza le informazioni memorizzate dalla CO.S.C-A sulla Dual Port relative all'identificazione del DSP e al DSP selezionato in scrittura dati dalla testa VIS-NIR;
b) se la scheda DSP e' in fase di acquisizione dati dalla testa VIS-NIR, il modulo controlla lo stato del registratore. Se il registratore e' attivo il modulo gestisce il trasferimento dei dati memorizzati nella memoria Y e nella DPRAM verso l'interfaccia con il registratore;
c) se la scheda DSP non e' in fase di acquisizione dati dalla testa VIS-NIR, il modulo trasferisce i dati presenti nella RAM X del DSP alla RAM Y, inserendo al termina di ciascuna riga di 512 dati, un campo checksum di 8 dati (2 byte per dato).

Il modulo esegue ciclicamente le attivita' sopra citate, sincronizzandosi con i segnali generati da ISR_FRAME. Nel primo ciclo il trasferimento dati al registratore non viene effettuato perche' non sono ancora presenti dati da trasferire. Tale attivita' ha termine quando la CO.S.C-A forza una diversa modalita' di funzionamento o quando viene rilevata una condizione di errore.

MODALITA' CALIBRAZIONE

La presente modalita' viene eseguita solo dalla scheda DSP 0. E' compito di detta modalita' acquisire 20 frame successivi, alternativamente con le altre schede DSP, eseguire una decimazione per 4 sulle prime 10 righe x 512 campioni.
La matrice risultante viene mediata per 20 frame successivi e il risultato scritto nel campo dati IR presente nella dual port.

MODALITA' COM_REGISTRA

Non viene eseguita nessuna attivita'. Il controllo viene restituito immediatamente al modulo principale.

MODALITA' SELEZIONA BANDE VIS

Non viene eseguita nessuna attivita'. Il controllo viene restituito immediatamente al modulo principale.

MODALITA' GAIN_SHUTTER

Il modulo residente sulle schede DSP con numero di identificazione diverso da 0 operano in stand by; il modulo residente sulla scheda DSP0 esegue le seguenti attivita':

a) azzera un blocco di memoria Y costituito da 1024 dati (2 byte per dato) in cui verra' memorizzato una prima versione dell'istogramma;
b) azzera la cella della Dual Port relativa ai "dati validi istogramma";
c) controlla, tramite Dual Port, se la scheda DSP0 e' in fase di acquisizione dati dalla testa VIS-NIR;
d) se la scheda DSP0 e' in fase di acquisizione si pone in fase di attesa;
e) se la scheda DSP0 non e' in fase di acquisizione dati, ed ha acquisito un frame, gestisce il trasferimento dei dati dalla memoria X e aggiorna i dati istogramma nel blocco di 1024 dati predisposto nella memoria Y.

Il modulo esegue le attivita' c, d, e, in modo ciclico, sincronizzandosi con i segnali trasmessi dalla ISR_FRAME. L'attivita' ha termine o perche' la CO.S.C-A ha forzato una nuova modalita', o perche' sono stati completati 10 cicli o perche' e' stata rilevata la mancanza del sincronismo di frame.

Se sono stati completati 10 cicli il modulo esegue:

f) compatta l'istogramma calcolato su 1024 punti in 64 punti, sommando i dati di 16 celle contigue;
g) normalizza i dati dell'istogramma cosi' calcolato in modo da avere come valore massimo 32;
h) trasferisce i dati istogramma sulla Dual Port e segnala alla CO.S.C-A che i dati di istogramma sono validi scrivendo il valore 0FFH nella cella "dati validi istogramma" della Dual Port.

MODALITA' PARAM_IR

Non viene eseguita nessuna attivita'. Il controllo viene restituito immediatamente al modulo principale.

MODALITA' RIPRISTINO

Non viene eseguita nessuna attivita'. Il controllo viene restituito immediatamente al modulo principale.

MODALITA' ERRORE (O MALFUNCTION)

Il modulo segnala alla CO.S.C-A la presenza di un malfunzionamento settando la cella ERRORE della Dual Port. Il tipo di malfunzionamento viene comunicato settando il bit 15 della parola ERRORE in DPRAM e settando il bit corrispondente al tipo di errore trovato. Tale funzione rimane in attesa finche' la cella ERRORE della Dual Port non venga azzerata dalla CO.S.C-A a conferma del riconoscimento dello stato di errore.
Il significato dei singoli bit e' il seguente:

D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
ERR	X	X	X	X	X	X	X	X	X	X	X	ENM	COD	EFS	

ERR = identifica la presenza di almeno un errore;
ENM = se settato indica che il conteggio dei frame validi per la modalita' OPERATIVO ha superato il valore 224 - 1
COD = se settato indica che e' stato ricevuto un codice relativo ad una modalita' di funzionamento non prevista;
EFS = se settato evidenzia l'assenza del segnale di sincronismo frame_sync.

6.1.2 Flusso di controllo

Di seguito si riportano i flussi di controllo relativi alle varie modalita' di funzionamento.
Il flusso di controllo per la modalita' STAND_BY e' riportato nella figura 3.

Il flusso di controllo per le modalita' 1,2,4,5,6,8,9 e' riportato in figura 4.

Il flusso di controllo per la modalita' OPERATIVO (n=3) e' riportato nella figura 5.

Sono previste due possibili cause di errore:

a) il numero di frame trasferiti e' maggiore di NMAX;
b) la ISR_TIMER rileva la mancanza del sincronismo di frame.

Nel caso b la ISR_TIMER forza flag_fs=1 in modo da consentire l'uscita dal ciclo di attesa; segnala l'errore settando la variabile "error" in maniera appropriata; resta quindi in attesa che la ISR_TIMER forzi la modalita' di funzionamento ERRORE.

Il flusso di controllo della modalita' 7 (GAIN_SHUTTER) e' riportato in figura 6.

Il flusso di controllo della modalita' 10 e' riportato nella figura 7.

6.1.2.1 Flusso generale

Si ritiene sufficiente la descrizione riportata in 5.5.

6.1.2.2 Descrizione sottocomponenti

Nel presente paragrafo vengono riportate le descrizioni dettagliate di tutte le sottocomponenti costituenti il software residente sulla scheda DSP_CCD. Ogni procedura viene descritta in linguaggio pseudo-C in modo da rendere piu' agevole lo sviluppo delle routine in codice assembler. Dove necessario vengono ampliamente descritti i vincoli hardware a cui alcune procedure (Reset, Power-Up, Interrupt Service Routine) devono sottostare.
Il programma principale, le due Interrupt Service Routine e tutte le procedure, risiedono nella memoria programma a partire dall'indirizzo 0x0040, immediatamente dopo la pagina destinata agli interrupt vector (0x000 - 0x003F). Nei vettori di interrupt, utilizzati dal modulo software, devono essere caricati i rispettivi riferimenti alle procedure di gestione:

```
/* CARICAMENTO VETTORI DI INTERRUPT:
 * ================================
 * istruzioni da caricare della memoria programma a livello assembler
 *
 * all'indirizzo $0000 (ORG   P:$0000)     JMP    power_up
 *                                         NOP
 *  "      "       $0008 (ORG   P:$0008)     JSR    ISR_FRAME
 *                                         NOP
 *  "      "       $001C (ORG   P:$001C)     JSR    ISR_TIMER
 *                                         NOP
 * dall'indirizzo ORG   P:$0040 puo' partire il codice
 */
```

6.1.2.2.1 Sottocomponente POWER_UP

La procedura power_up() e' la prima ad essere eseguita sia all'accensione della scheda DSP che all'eventuale reset da parte del watch-dog. E' compito della presente routine l'inizializzazione dei registri hardware (out_registratore, A/D converter ecc.), della memoria dual port e l'abilitazione dei due interrupt utilizzati dal programma.

```
power_up()                    codice di partenza (power_up o reset)
{
   register short *pointer;
   register short i;

   OMR |= 0x80;                setta stati di wait in input al 56000
   *(HARD1_REG) = 0x80;        setta a 1 il bit D7 (DATO VALIDO)
                               setta a 0 il bit D3 (T.S.)
                               setta a 0 il bit D6 (RESET WATCH DOG)
   *(ADD_LATCH) = 0x00;        azzera address latch
   error = 0;
   reset_watch_dog();          reset hardware watch-dog
   *(DSP_IDENT) = 0xFF;
   ident = *(DSP_IDENT);       legge identificatore DSP
   while (ident < 0 || ident > 7)   codice non previsto ?
      ident = *(DSP_IDENT);         rilegge dual port
   pointer = RIEMPITIVO;
   for(i=0; i<NW_RIEMP; i++)     inizializzazione campo riempitivo
      *pointer++ = 0x02E2E;
   *(ESITO_DIAG) = 0;            inizializzazione dual-port
   *(DATVAL_IST) = FALSE;
   *(MOD_FUNCT)  = STAND_BY;
   *(STAT_REG)   = REGIS_NO;
   *(DSP_SELECT) = 0xFF;
   flag_to = 0;                  inizializza flag e contatori
```

```
    flag_fs = 0;
    count_wd = 0;
    count_fs = 0;
    setta_interrupt();    setta timer interno (SCI) per to = 10mS
                          e abilita interrupt da ISRQA
    *(MR) &= 0xFC         abilita interrupts nel MR (Mode Register)
    while (1)             loop infinito di attivazione funzioni
    {
       if(modo_c < STAND_BY || modo_c > MALFUNCTION)
       {
          error |= GEN_ERROR;      setta errore codice illegale
          error |= ERR_CODE;
          modo_c = MALFUNCTION;
       }
       (*function[modo_c])();     chiama la funzione associata
    }
}
```

6.1.2.2.2 Sottocomponente STAND_BY

La modalita' di funzionamento stand_by inizializza a -1 tutte le variabili di stato delle singole funzioni, incrementa il proprio flag di status e resetta la variabile di errore (unica condizione per resettare la condizione di errore).

```
stand_by()
{
   register short i;

   error = 0;
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[STAND_BY]++;      segnala attivazione stand_by
}
```

6.1.2.2.3 Sottocomponente SETUP

La modalita' di setup, nella presente versione del software, non effetta nessuna operazione particolare.

```
setup()
{
   register short i;

   if(func_status[SETUP] != -1)
      return
   for(i=0; i<NFUNCT; i++)
```

```
        func_status[i] = -1;
    func_status[SETUP]++;
}
```

6.1.2.2.4 Sottocomponente DIAGNOSTICA

La modalita' diagnostica effettua un controllo su quanto scritto nella locazione ESITO DIAGNOSTICA della Dual Port, sulla presenza nel campo riempitivo di 64 byte del valore 0x2E e, nel caso di DSP 0, effettua l'acquisizione di un frame dalla testa CCD e ne controlla la costituzione se conforme a quanto specificato.
Il frame utilizzato per la diagnostica deve avere le righe pari, con indice 0,2,4,...18, costituite da 512 valori uguali a zero, mentre le righe dispari, con indice 1,3,5,...19, costituite da 512 valori diversi da zero.
Nel caso che tutti e tre i controlli diano esito positivo restituisce nell'apposita cella della dual port il valore DIAGNOS_OK altrimenti restituisce il valore DIAGNOS_NO.

```
diagnostica()
{
    register short *pointer;
    register short i,ident;
    register short result;
    short    dato,abilita,start_elab;

    if(func_status[DIAGNOSTICA] != -1)
        return
    for(i=0; i<NFUNCT; i++)
        func_status[i] = -1;
    func_status[DIAGNOSTICA]++;
    result = TRUE;
    if(*(ESITO_DIAG) == TRUE)      test valore esito diagnostica
        result = FALSE;
    ident = *(DSP_IDENT);          legge identificatore
    if(ident == 0)                 se siamo il DSP 0
    {
        *(ADD_LATCH)  =  0x00;     seleziona pagina zero
        abilita=0xFF;
        while(abilita != 0)        attesa RAM_LOAD=0 scrittura
        {
            flag_fs = 0;           attesa frame sync
            while(flag_fs == 0 && modo_c == DIAGNOSTICA);
            flag_fs = 0;
            if(modo_c != DIAGNOSTICA)
                return;
            abilita = *(HARD2_REG) & 0x04;
        }          /* il frame successivo e' sicuramente valido */
        flag_fs = 0;                   attesa frame sync
```

```
      while(flag_fs == 0 && modo_c == diagnostica);
      flag_fs = 0;
      if(modo_c != DIAGNOSTICA)
         return;
      pointer = DATA_PAGE;              puntatore buffer sorgente
      for(i=0; i<N_LINEE/2; i++)        loop sulle linee
      {
         for(j=0; j<LINE_SIZE; j++)     linee di 512 dati
         {
            if(*pointer++ != 0)     le righe 0,2,4,..18 sono a zero
               result=FALSE;
         }
         for(j=0; j<LINE_SIZE; j++)
         {
            if(*pointer++ == 0)     le righe 1,3,5,..19 sono <> zero
               resul=FALSE;
         }
      }
   }
   pointer = RIEMPITIVO;            indirizzo campo riempitivo
   for(i=0; i<NW_RIEMP; i++)        controllo campo dati riempitivo
   {
      if(*pointer++ != 0x2E2E)
         result = FALSE;
   }
   if(reult == TRUE)
      *(ESITO_DIAG) = DIAGNOS_OK;
   else
      *(ESITO_DIAG) = DIAGNOS_NO;
}
```

6.1.2.2.5 Sottocomponente OPERATIVO

La modalita' di funzionamento operativa necessita del continuo sincronismo con il segnale frame_sync. La variabile frame_fs = 1 viene utilizzata a questo scopo in quanto indica l'avvenuto riconoscimento del segnale frame_sync. La locazione dell'array func_status associata alla presente funzione, contiene il numero di frame validi trasferiti dalla memoria frame (X) nella memoria locale (Y) e quindi, in congiunzione con il contenuto della variabile DSP_sel, fornisce l'informazione del numero di record trasferiti al registratore.La presente modalita' e' terminata quando la variabile modo_c, settata dalla ISR_TIMER, ha contenuto diverso dal valore OPERATIVO. In caso di mancanza del segnale di sincronismo frame_sync, la ISR_TIMER restituisce il flag flag_fs settato a 1 in modo che la presente procedura possa ritornare regolarmente ma la procedura chiamante trovera' la variabile modo_c settata come modalita' errore per cui il modulo software entrera' nella fase di malfunzionamento.

```
operativo()
{
   register short i,j;
   register short *from, *to;
   register unsigned accumula;
   short dato,start_elab;
   short primo_frame;
   short DSP_sel, stato_reg;

   if(func_status[OPERATIVO] != -1)
      return
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[OPERATIVO]++;
   primo_frame = 0;
   start_elab=0;
   flag_fs = 0;
   while (modo_c == OPERATIVO)
   {
      while (flag_fs == 0 && modo_c == OPERATIVO);
      if(modo_c != OPERATIVO)
         return;
      flag_fs = 0;
      if(error != 0)
         return;
      ident = *(DSP_IDENT);              legge identificatore
      stato_reg = *(STAT_REG);           legge stato registratore
      while (stato_reg != REGIS_OK && stato_reg != REGIS_NO)
         stato_reg = *(STAT_REG);
      DSP_sel = *(DSP_SELECT);
      while (DSP_sel < 0 || DSP_sel > 7)
         stato_reg = *(STAT_REG);
      *(STAT_REG)    = 0xFF;
      *(DSP_SELECT) = 0xFF;
      *(ADD_LATCH) = 0x00;               azzera address latch
      if(DSP_sel != ident)
      {
       if(start_elab == 0)
       {
         func_status[OPERATIVO]++;
         if(func_status[OPERATIVO] == NMAX)
         {
             error |= GEN_ERROR;
             error |= ERR_NMAX;
             *(ERRORE) = error;
         }
         to   = BASE_LOCAL;                 puntatore destinazione
         from = DATA_PAGE;                  puntatore sorgente
         for(i=0; i<NLINEE; i++)            loop sulle 20 linee
```

```
            {
               accumula = 0;
               for(j=0; j<LINE_SIZE; j++)     linea di 512 dati
               {
                  dato = *from++;          legge dato
                  if(i == 0)
                     accumula = (dato & 0xFFF);
                  else
                     accumula ^= (dato & 0x0FFF);  OR esclusivo
                  dato  /= 4;              rende il valore su 10 bit
                  if(dato > 0x03FF;
                     dato = 0x8000;        setta overflow
                  *to++ = dato;            scrive dato in memoria
               }
               for(j=0; j<CHECK_SIZE; j++)
                  *to++ = accumula;              scrive checksum;
            }
            start_elab=1;
          }
        }
        else
        {
           start_elab=0;
           if(primo_frame != 0)
           {
              if(stat_reg == REGIS_OK)
              {
                 *(HARD1_REG) |= 0x08;                setta bit T.S.
                 *(HARD1_REG) |= 0x80;                dato valido
                 out_block(PREAMBOLO, PRE_SIZE);      preambolo
                 out_block(HEADER_REC, HEA_SIZE);     header record
                 for(i=0; i<DESC_SIZE; i++)           descr record
                    out_word(0x2E2E);
                 for(i=0; i<N_LINEE; i++)             campo dati VIS
                 {
                    out_block(SOTTOC_VIS+i*N_SUBC,      N_SUBC);
                    out_block(BASE_LOCAL+i*DATA_CHECK, DATA_CHECK);
                 }
                 out_block(RIEMPITIVO, NW_RIEMP);
                 out_block(CDATI_IR,   CIR_SIZE);     campo dati IR
                 out_block(POSTAMBOLO, POS_SIZE);     postambolo
                 *(HARD1_REG) &= 0xF7;                azzera bit T.S.
                 *(HARD1_REG) &= 0x7F;          azzera bit dato valido
              }
           }
           else
              primo_frame = 1;
        }
     }
}
```

6.1.2.2.6 Sottocomponente CALIBRAZIONE

La presente modalita' viene eseguita in maniera completamente diversa a seconda che si tratti del DSP 0 o di altro DSP. Nel caso di DSP 0 detta modalita' effettua una acquisizione di 10 frame in sequenza, tra quelli memorizzati sulla scheda DSP 0. Per ogni matrice di dati, costituente il frame stesso, vengono prese le prime 10 righe x 512 campioni sulle quali viene effettuata una decimazione per 4 in modo da ottenere una matrice risultante di 5 righe x 256 campioni. Ogni matrice risultante viene accumulata per 10 frame successivi presi in maniera alternata rispetto alle altre scede DSP presenti nel sistema.
Una volta accumulate le 10 matrici risultanti, viene memorizzato nel campo dati IR, presente nella dual port, la matrice media risultante.
Per le schede diverse dal DSP 0 non viene effettuata nessuna attivita'.

```
calibrazione()
{
   register short i,j;
   register short *pointer;
   register short nframe;
   register short accumula;
   short *indice1, *indice2, *to;
   short DSP_sel, dato, ident, start_elab;
   unsigned char abilita;

   if(func_status[CALIBRAZIONE] != -1)
      return
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[CALIBRAZIONE]++;
   ident = *(DSP_IDENT);                    legge identificatore;
   if(ident != 0)                           se non siamo il DSP 0
      return;
   flag_fs = 0;
   *(ADD_LATCH) = 0x00;      seleziona pagina 0 memoria frame
   while(modo_c == CALIBRAZIONE)
   {
      while(*(STATO_CAL != CAL_RUN && modo_c == CALIBRAZIONE);
      pointer = buffer;                      azzera buffer di lavoro
      for(i=0; i<BUFFER_SIZE; i++)
         *pointer++ = 0;
      if(modo_c != CALIBRAZIONE)
         return;
      nframe = 0;
```

```
        start_elab = 0;
        while(nframe < 10)
        {
            abilita = 0;
            while(abilita == 0)
            {
                flag_fs = 0;                attesa frame sync
                while(flag_fs == 0 && modo_c == CALIBRAZIONE);
                flag_fs = 0;
                if(modo_c != CALIBRAZIONE)
                    return;
                abilita = *(HARD2_REG) & 0x04;    test su RAM_LOAD
                if(abilita == 0)
                    start_elab = 1;
            }
            if(start_elab == 1)
            {
                func_status[CALIBRAZIONE]++;  conteggia frame validi
                indice1=DATA_PAGE;            primo elemento Ia linea
                indice2=DATA_PAGE+NC_LINEA;   primo elemento IIa linea
                to = buffer;                  indirizzo destinazione
                for(i=0; i<NUMLINEE; i++)
                {
                    for(j=0; j<NUMCAMP; j++)
                    {
                        accumula = *indice1++ + *indice2++;
                        accumula = accumula + *indice1++ + *indice2++;
                        dato = accumula/4;
                        if(dato > 0x0FFF)
                            dato = 0x20000;
                        *to++ += dato;
                    }
                    indice1 += LINE_SIZE;  riallinea indici
                    indice2 += LINE_SIZE;
                }
                nframe++;
                start_elab=0;
            }
        }
        pointer = buffer                puntatore al sorgente
        to = CDATI_IR;                  puntatore destinazione
        for(i=0; i<BUFFER_SIZE; i++)
        {
            dato = *pointer++/40;       valore medio su 10 bit
            if(dato   >  0x3FF)         valore massimo senza overflow
                dato = 0x8000;          setta overflow e azzera dato;
            *to++ = dato;               memorizza dato
        }
        *(STATO_CAL) = CAL_END;
    }
}
```

6.1.2.2.7 Sottocomponente COM_REGISTRA

La modalita' com_registra, nella presente versione del software, non effetta nessuna operazione particolare.

```
com_registra()
{
   register short i;

   if(func_status[COM_REGISTRA] != -1)
      return
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[COM_REGISTRA]++;
}
```

6.1.2.2.8 Sottocomponente BANDE_VIS

La modalita' bande_VIS, nella presente versione del software, non effetta nessuna operazione particolare.

```
bande_VIS()
{
   register short i;

   if(func_status[BANDE_VIS] != -1)
      return
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[BANDE_VIS]++;
}
```

6.1.2.2.9 Sottocomponente GAIN_SHUTTER

La modalita' di funzionamento gain_shutter necessita del continuo sincronismo con il segnale frame_sync. La variabile frame_fs = 1 viene utilizzata a questo scopo in quanto indica l'avvenuto riconoscimento del segnale frame_sync. La locazione dell'array func_status associata alla presente funzione, contiene il numero di frame validi utilizzati per l'operazione di media necessaria

al calcolo dell'istogramma.
La presente modalita' e' terminata quando la variabile modo_c, settata dalla ISR_TIMER, ha contenuto diverso dal valore OPERATIVO. In caso di mancanza del segnale di sincronismo frame_sync, la ISR_TIMER restituisce il flag flag_fs settato a 1 in modo che la presente procedura possa ritornare regolarmente ma la procedura chiamante trovera' la variabile modo_c settata come modalita' errore per cui il modulo software entrera' nella fase di malfunzionamento.

```
gain_shutter()
{
   register short i, j;
   register short *pointer, *from, *to;
   register short nframe;          conteggio 10 frame successivi
   register short accumula;i
   short val_max,abilita;
   short primo_frame, dato;
   short ident, DSP_sel;

   if(func_status[GAIN_SHUTTER] != -1)
      return;
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[GAIN_SHUTTER]++;
   primo_frame = 0;
   flag_fs = 0;
   nframe = 0;
   if(ident != 0)                        se non siamo il DSP0
      return;
   *(ADD_LATCH) = 0x00;   seleziona pagina 0 sulla memoria frame
   while (modo_c == GAIN_SHUTTER)
   {
      while (flag_fs == 0);              attesa evento frame sync
      flag_fs = 0;
      if(error != 0)
         return;
      if(primo_frame = 0)                se il primo frame
      {
         pointer = ARRAY_ISTO;
         for(i=0; i<ARRAY_SIZE; i++)     azzera array di lavoro
            *pointer++ = 0;
         primo_frame = 1;
         *(DATVAL_IST) = 0;       azzera dati validi istogramma
      }
      while (nframe < 10)
      {
         abilita = *(HARD2_REG) & 0x04; legge line RAM_LOAD
         if(abilita != 0)   elabora dati solo se la scrittura in
         {                  corso avviene su un altro processore
            func_status[GAIN_SHUTTER]++;  conteggia frame validi
```

```
            from = DATA_PAGE;                puntatore sorgente
            for(i=0; i<TOTAL_DATA; i++)      loop su 10240 dati
            {
               dato = *from++;               legge dato
               dato /= 4;          shift a destra di due posizioni
               if(dato > 1023)               testa overflow
                  *(ARRAY_ISTO+1023)++;      incrementa cella 1023a
               else
                  *(ARRAY_ISTO+dato)++;      incrementa cella indice
            }
            nframe++;
         }
         while(flag_fs == 0);              attesa evento frame sync
         flag_fs=0;
         if(error != 0)
            return;
         if(modo_c != GAIN_SHUTTER)     test sulla mod. funz.
            return;
      }
      pointer = ARRAY_ISTO;            puntatore all'array sorgente
      to = ISTOGRAMMA;               puntatore alla destinazione dati
      val_max = 0;
      for(i=0; i<64; i++)           loop esterno su 64 risultati ISTO
      {
         accumula = 0;
         for(j=0; j<16; j++)
            accumula += *pointer++; somma i gruppi di 16 locazioni
         *to++ = accumula;
         if(accumula > val_max)
            val_max = accumula;
      }
      pointer = ISTOGRAMMA;        puntatore ai 64 dati istogramma
      for(i=0; i<64; i++)          normalizzazione;
      {
         dato = *pointer;
         *pointer++ = dato*32/val_max;
      }
      *(DATVAL_IST) = TRUE;
            /* attesa della lettura dati da parte della CO.S.C-A
               e   del   relativo   azzeramento   della   cella.
               L'operazione   anche   se   viene   cambiata   la
               modalita' di funzionamento */
      while(*(DATVAL_IST) == TRUE || modo_c == GAIN_SHUTTER);
      primo_frame = 0;
   }
}
```

6.1.2.2.10 Sottocomponente PARAM_IR

La modalita' param_IR, nella presente versione del software, non effetta nessuna operazione particolare.

```
param_IR()
{
   register short i;

   if(func_status[PARAM_IR] != -1)
      return
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[PARAM_IR]++;
}
```

6.1.2.2.11 Sottocomponente RIPRISTINO

La modalita' ripristino, nella presente versione del software, non effetta nessuna operazione particolare.

```
ripristino()
{
   register short i;

   if(func_status[RIPRISTINO] != -1)
      return
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[RIPRISTINO]++;
}
```

6.1.2.2.12 Sottocomponente ERRORE

La modalita' errore, l'unica attivata dal programma interno e non attivata dalla CO.S.C-A, trasferisce il contenuto della variabile error nella cella ERRORE predisposta nella Dual Port RAM. La CO.S.C-A azzerando il contenuto della cella ERRORE ed invia la modalita' di funzionamento stand_by.

```
errore()
{
   register short i;

   if(func_status[MALFUNCTION] != -1)
      return
   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
```

```
    func_status[MALFUNCTION]++;
    *(ERRORE) = error;          setta locazione di errore nella DPRAM
}
```

6.1.2.3 Elenco subroutine

Quelle segnalate piu' quelle di servizio (vedi)

6.2 Componente ISR_TIMER

6.2.1 Descrizione funzionale

La ISR_TIMER() viene attivata ogni 10msecondi dal SCI_timer interno al DSP56000 programmato per tale periodo. Legge il contenuto della cella modalita' di funzionamento e lo trasferisce nella variabile modo_c, setta il flag di avvenuto time_out e controlla la presenza del segnale di sincronizzazione frame_sync tramite un time_out locale di 40 msecondi. In caso di asssenza di tale segnale attiva la modalita' di malfunzionamento e carica la cella Dual Port ERRORE con l'indicazione di assenza frame_sync. Gestisce, ogni 500 msecondi, il reset del watch dog esterno in grado, in mancanza di tale reset, di reinizializzare il modulo software dalla procedura di power_up().

```
ISR_TIMER()   ;interrupt service routine for timeout
{
   register short modo;

   flag_to = 1;                    setta flag evento
   modo = *(MOD_FUNCT);            legge modalita' di funzionamento
   if(*(ERRORE))
      modo_c = MALFUNCTION;  forza la modalita' errore
   else
   {
      modo_c = modo;       altrimenti inizializza modo_c in accordo
      errore = 0;          azzera flag di errore
   }
   count_fs++;             conteggia eventi per timeout frame_sync
   if(count_fs == NTO_FS)
   {
      error |= GEN_ERROR;          gestione errore per diagnostica
      error |= ERR_FRAME;         indica errore assenza frame_sync
      modo_c = MALFUNCTION;       setta modalita' errore
      flag_fs = 1;                forza la presenza del frame_sync
   }
```

```
    count_wd++;
    if(count_wd == NTO_WD)
    {
        count_wd = 0;
        reset_watch_dog();
    }
    return_from_interrupt;        RTI
}
```

6.3 Componente ISR_FRAME

6.3.1 Descrizione funzionale

La ISR_FRAME viene attivata dal segnale di sincronizzazione esterno frame_sync (34 msec). Setta il flag flag_fs di avvenuto evento (detto flag deve essere riazzerato dalla procedura che ne fa' uso) e azzera il contatore count_fs necessario alla ISR_TIMER per il conteggio del time_out di controllo sulla presenza del segnale frame_sync stesso.

```
ISR_FRAME()    ;interrupt service routine for IRQA
{
    flag_fs = 1;              setta flag evento
    count_fs = 0;             riazzera contatore timeout frame_sync
    return_from_interrupt;    RTI
}
```