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 infrarosso

L. Bedini A. Ribolini

Nota Interna B4-21

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/FLIR 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
FIFO	First In First Out
IR	Infra Red

4. RIFERIMENTI

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

OFFICINE GALILEO ST/I-91-038
Requisiti del software VIRS-201 Modulo di Elaborazione - MES - DSP_FLIR

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-FLIR 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-FLIR puo' essere descritto come un unico sistema, costituito dai moduli MAIN, ISR_TIMER, ISR_RIGA 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 MAIN che, a sua volta, puo' essere interrotta dalle tre ISR inizializzate dalla stessa.
Le componenti ISR_FRAME ed ISR_RIGA sono attivate dagli interrupt

generati rispettivamente dai segnali di sincronismo di frame e di riga.
La componente ISR_TIMER interviene ogni 2 msecondi ed e' utilizzata come base tempi di riferimento per l'esecuzione delle varie attivita' periodiche.
Il processo MAIN espleta le proprie funzioni svolte sincronizzandosi con gli eventi generati da tali componenti.

5.3 Definizione funzionale di ciascuna componente

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

5.3.1 Componente MAIN

La componente MAIN viene attivata all'accensione o al reset della carta DSP-FLIR. Fanno parte di detta componente le procedure per la 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

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 MAIN si deve intendere sempre attiva in quanto puo' essere interrotta solamente da una delle altre tre componenti ISR_TIMER, ISR_FRAME o ISR_RIGA.

5.3.2 Componente ISR_TIMER

La ISR_TIMER e' attivata, ogni 2 msec, dall'interrupt generato dal timer interno al DSP56000; la routine svolge i seguenti compiti principali:
- riceve IRQB_TO da ISR_RIGA e gestisce il time-out per il controllo di presenza del sincronismo di riga;
- in caso di assenza di sincronismo di riga segnala errore;
- riceve IRQA_TO da ISR_FRAME e gestisce il time-out per il controllo di presenza del sincronismo di frame;
- in caso di assenza del sincronismo di frame segnala errore;
- genera CLOCK_10, che viene utilizzato dal MAIN per la temporizzazione delle varie attivita'
- legge ciclicamente ogni 10 msec, utilizzando per la temporizzazione CLOCK_10, il comando depositato dalla CO.S.C.A nella Dual Port relativo alla modalita' di funzionamento;
- aggiorna ogni 200 msec. il flag PERIODICA, utilizzato dal MAIN per attivare l'attivita' periodica;
- aggiorna ciclicamente il contatore COUNT_WD per il reset dell'watch-dog e l'incremento della cella DSP_RUN;
- controlla il contenuto della cella ERRORE; se diverso da zero segnala al MAIN modo_c = MALFUNCTION in modo che venga attivata la funzione "errore".

5.3.3 Componente ISR_FRAME

La ISR_FRAME e' attivata, ogni 34 msec., dal segnale di sincronismo di frame. Esegue le seguenti attivita' principali:
- segnala al MAIN, tramite IRQA_FLAG, la presenza del sincronismo di frame;
- segnala ad ISR_TIMER, tramite IRQA_TO, la presenza del sincronismo di frame.

5.3.4 Componente ISR_RIGA

La ISR_RIGA e' attivata, ogni 512 msec., dal segnale di sincronismo di riga. Esegue le seguenti attivita' principali:

- segnala al MAIN, tramite IRQB_FLAG, la presenza del sincronismo di riga;
- riceve dal MAIN, tramite MEDIA_FLAG, l'ndicazione se resettare o meno la memoria FIFO;
- invia il segnale di acknowledge per l'interrupt generato dal sincronismo di riga.

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 MAIN e esterno
- Interfaccia tra MAIN e ISR_TIMER
- Interfaccia tra MAIN e ISR_FRAME
- Interfaccia tra MAIN e ISR_RIGA
- Interfaccia tra ISR_TIMER e ISR_FRAME
- Interfaccia tra ISR_FRAME e esterno
- Interfaccia tra ISR_RIGA e ISR_TIMER
- Interfaccia tra ISR_RIGA ed esterno

5.4.1 Interfaccia tra MAIN e esterno

RAM_FIFO:
una word attraverso la quale la componente MAIN e' in grado di leggere i componenti della riga in fase di elaborazione nella RAM locale.

RAM_LOCAL:
array di memoria di 512 words utilizzato per il mantenimento e l'aggiornamento dei dati effettuato dall'elaborazione in corso sulla riga corrente.

RAM_DP:
array di memoria (dual-port memory) di 4096 words attraverso la quale la componente MAIN 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 MAIN 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.

CLOCK_10:
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.

PERIODICA:
una word che verra' posta ad uno ogni 200 msecondi; viene utilizzata dalla componente MAIN per attivare l'attivita' periodica

5.4.3 Interfaccia tra MAIN e ISR_FRAME

IRQA_TO:
una word che verra' settata a uno dalla ISR_FRAME in modo da indicare la corretta segnalazione da parte del segnale FRAME_SYNC.

5.4.4 Interfaccia tra ISR_TIMER e ISR_FRAME

IRQA_TO:
una word che verra' settata a uno dalla componente ISR_FRAME per

segnalare l'avvenuto riconoscimento della linea FRAME_SYNC (Interrupt IRQA del processore).

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.

5.4.6 Interfaccia tra MAIN ed ISR_TIMER

IRQB_FLAG:
e' una word che viene posta ad uno da ISR_RIGA in concordanza con la ricezione del sincronismo di riga e resettata dal MAIN.

MEDIA_FLAG:
e' una word che viene utilizzata dal MAIN per segnalare che il segnale di reset della memoria FIFO e' a carico del MAIN stesso. Quando assume il valore "0" il reset e' eseguito dalla ISR_RIGA.

5.4.7 Interfaccia tra ISR_RIGA e ISR_TIMER

IRQB_TO:
e' una word che viene posta ad uno da ISR_RIGA per segnalare la ricezione di un sincronismo di riga; viene resettata da ISR_TIMER.

5.4.8 Interfaccia tra ISR_RIGA ed esterno

RIGA_SYNC:
e' un segnale di interrupt generato ognii 512 msecondi dall'elettronica esterna presente nella componente sensore IR.

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>9), viene inviata una segnalazione di errore ed attivata la modalita' di funzionamento MALFUNCTION (n=10). 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; il valore "0" quando 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.
La corrispondenza tra i valori assunti da MODO_C e la funzione attivata e' dato dalla seguente tabella:

MODO_C	FUNZIONE
0	stand_by
1	setup
2	diagnostica
3	operativo
4	calibrazione
5	comandi registratore
6	bande_vis
7	gain_shutter
8	param_IR
9	ripristino
10	malfunction

L'attivita' corrispondente a MODO_C=10 viene attivata in caso si riveli un errore; le altre modalita' sono selezionate dalla CO.S.C.A.
CLOCK_FLAG viene posto ad "1" dalla ISR_TIMER ogni 10 msecondi e viene utilizzato come base dei tempi dal MAIN.
MODO_C viene aggiornato, ogni 10 msecondi, dalla ISR_TIMER.
IRQA_FLAG viene posto ad "1" dalla ISR_FRAME in corrispondenza del sincronismo di frame.
IRQB_FLAG viene posto ad "1" dalla ISR_RIGA in corrispondenza del sincronismo di riga.
MEDIA_FLAG viene posto ad "1" dal MAIN quando sono in esecuzione le attivita' 3, 4, 8 allo scopo di comunicare alla ISR_RIGA che il reset della FIFO e' eseguito dal MAIN stesso.
STATO_CAL viene posto da CO.S.C.A uguale a CAL_RUN per avviare l'attivita 4 (calibrazione) e posto uguale a CAL_END dalla componente MAIN per segnalare la fine delle attivita'.
DATVAL_IST viene posta TRUE dalla componente MAIN al termine dell'attivita' 8 (param_IR) e posto FALSE dalla CO.S.C.A dopo la lettura dell'istogramma depositato nella DPRAM.

5.5.1 Sincronizzazione con la componente ISR_FRAME.

Per la sincronizzazione con ISR_FRAME viene utilizzato il flag FLAG_FS. Tale flag e' settato dalla ISR_FRAME e resettato dalla componente MAIN.
Il meccanismo di sincronizzazione e' reso operativo solo per le funzioni 3, 4 e 8. In fase di inizializzazione (power-up o reset) si pone FLAG_FS=0.
Per le funzioni 3,4,8 e' necessario assicurare che in un intervallo di frame (circa 34 msecondi) siano completate tutte le attivita' previste, in particolare dovranno essere eseguite:

a) le operazioni di media su 30 righe le tte alternativamente

dalla memoria FIFO;

b) la normalizzazione su 10 bits della riga mediata e la sua scrittura nella DPRAM;

c) l'esecuzione della attivita' periodica che verra' eseguita ogni circa 200 msecondi, subito dopo l'esecuzione dell'attivita' prevista al punto b).

Per l'attivita' b) si prevede un tempo di esecuzione di circa 700 usec; per l'attivita' c) un tempo di esecuzione di circa 800 usecondi.

5.5.2 Sincronizzazione con la componente ISR_RIGA

Per la sincronizzazione con la ISR_RIGA viene utilizzato il flag IRQB_FLAG. Tale flag e' settato da ISR_RIGA e resettato dalla componente MAIN. Il meccanismo di sincronizzazione e' reso operativo solo per le funzioni 3, 4, 8.
Durante l'esecuzione di tali funzioni la componente MAIN legge dalla memoria FIFO, una riga si ed una riga no, per un totale di 30 righe. Ciascuna riga K (K=1 - 64) viene letta ed accumulata in un buffer di accumulo; tale operazione e' eseguita prima di inviare il reset della memoria FIFO e comunque deve essere terminata prima dell'inizio della scrittura successiva nella FIFO, da parte del sensore IR, della riga con indice K+2. La temporizzazione delle varie attivita' connesse alla lettura ed all'accumulo e' illustrato in fig. 3.

5.5.3 Gestione degli errori

E' prevista la rilevazione dei seguenti errori:

ERR_FRAME: mancanza del sincronismo di frame, rilevato dalla ISR_TIMER;

ERR_RIGA: mancanza del sincronismo di riga, rilevato dalla ISR_TIMER;

ERR_CODE codice relativo alla modalita' di funzionamento selezionato dalla CO.S.C.A sconosciuto, rilevato dalla ISR_TIMER;

ERR_DPRAM: errore rilevato nel test della DPRAM, rilevato dalla componente MAIN al power-up;

ERR_DATVAL: codice errato letto dalla cella DATVAL_IST della DPRAM, al termine della esecuzione dell'istogramma nella modalita' 8 (parametri IR), rilevato dalla componente MAIN.

Alla rilevazione dell'errore il software aggiorna la cella ERRORE della DPRAM e pone in MODO_C il codice (10) relativo alla modalita' MALFUNCTION. Il software resta quindi in attesa che venga azzerata la cella della DPRAM da parte della CO.S.C.A e che venga selezionata la modalita' stand-by.

5.6 Flusso dei dati

Durante l'esecuzione delle modalita' 3, 4 e 8 vengono letti dalla memoria FIFO i dati relativi alle righe generate dal sensore IR; le righe vengono lette alternativamente ed accumulate in un buffer; al termine dell'accumulo, eseguito su 30 righe, i dati del buffer vengono divisi per 8 in modo da avere una rappresentazione su 10 bit e depositati nella DPRAM, ove vengono letti dalla CO.S.C.A.
Nella modalita' 3, l'operazione sopraccitata viene eseguita ininterrottamente per ciascun frame fin tanto che non venga cambiata la modalita' di funzionamento o non si verifichi un errore.
Nella modalita' 4, l'operazione viene attivata dal comando CAL_RUN depositato nella cella STATO_CAL della DPRAM da parte della CO.S.C.A. L'operazione sopraccitata viene svolta un frame si ed uno no per 20 frame successivi. Al termine viene calcolata la riga media sui 10 frame analizzati e depositata nella DPRAM; il software deposita in STATO_CAL il valore CAL_END e resta in attesa di un nuovo comando CAL_RUN.
L'operazione ha termine quando viene cambiata, da CO.S.C.A, la modalita' di funzionamento o viene rilevato un errore. Il test sulla modalita' di funzionamento viene eseguito al termine delle elaborazioni previste per ogni frame.
Nella modalita' 8 si procede in maniera analoga a quanto previsto nella modalita' 4, solo il fatto che per ogni frame analizzato, la riga mediata viene utilizzata per l'aggiornamento dell'istogramma. Al termine l'istogramma calcolato sulle 10 righe mediate viene depositato nella DPRAM e la cella DATVAL_IST viene aggiornata con il valore TRUE.
Il software resta in attesa che la CO.S.C.A aggiorni DATVAL_IST con il valore FALSE prima di effettuare un altro istogramma.

6. PROGETTAZIONE DETTAGLIATA DEI COMPONENTI

6.1 Componente MAIN

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) controlla il funzionamento corretto della RAM y attraverso una serie di letture e scritture successive;
b) tramite il convertitore A/D, controlla la presenza del segnale di riferimento (5 Volt);
c) 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) attende il sincronismo di frame: IRQA_FLAG attivato da ISR_FRAME;
b) invia un reset della memoria FIFO;
c) attende due sincronismi di riga (IRQB_FLAG) in modo da assicurare che la memoria FIFO sia stata correttamente scritta;
d) legge i dati dalla memoria FIFO e li accumula in un buffer di lavoro;
e) controlla che sia gia' pervenuto un sincronismo di riga (IRQB_FLAG);
f) invia un reset alla memoria FIFO, attende un nuovo sincronismo di riga e ripete il tutto dal punto e) fino a quando non sono state accumulate 30 righe;
g) divide i dati della riga accumulata per 8 e li memorizza nella

DPRAM.

L'operazione e) deve essere eseguita in meno di 600 usecondi; a tale scopo il software relativo a tale operazione viene automaticamente caricato nella memoria programma interna al DSP. Anche l'operazione g) deve essere eseguita in 700 usecondi; pertanto anche il codice relativo a tale operazione e' caricato automaticamente nella memoria programma interna al DSP.
Poiche' il codice relativo alla presente modalita' verra' utilizzato anche per le modalita' 4 e 8, tale codice e' stato organizzato nella subroutine CALCOLA_MEDIA() che ha come parametro la modalita' di funzionamento selezionata.

MODALITA' CALIBRAZIONE

Il modulo esegue le seguenti attivita':

a) attende che la CO.S.C.A aggiorni la cella STATO_CAL con il valore CAL_RUN;
b) chiama la subroutine CALCOLA_MEDIA();
c) attende il sincronismo di frame (IRQA_FLAG);
d) accumula in un buffer la riga mediata su ciascun frame analizzato (un frame ogni due);
e) ripete dal punto b) fino al completamento delle 10 righe;
f) divide per 10 la riga finale accumulata e la deposita nella DPRAM.

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

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

MODALITA' PARAM_IR

Il modulo esegue le seguenti attivita':

a) chiama la subroutine CALCOLA_MEDIA();
b) attende il sincronismo di frame (IRQA_FLAG);

c) aggiorna l'istogramma relativo alla riga mediata su ciascun frame analizzato (un frame ogni due);
d) ripete dal punto b) fino al completamento delle 10 righe;
e) deposita l'istogramma calcolato e normalizzato su 5 bit nella DPRAM;
f) aggiorna la cella DATVAL_IST con il valore TRUE ed attende che la CO.S.C.A ponga in tale cella il valore FALSE.

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	ERG	EDP	TDP	COD	EFS

ERR = identifica la presenza di almeno un errore;
EFS = ERR_FRAME: mancanza del sincronismo di frame, rilevato dalla ISR_TIMER;
COD = ERR_CODE codice relativo alla modalita' di funzionamento selezionato dalla CO.S.C.A sconosciuto, rilevato dalla ISR_TIMER;
TDP = ERR_DPRAM: errore rilevato nel test della DPRAM, rilevato dalla componente MAIN al power-up;
EDP = ERR_DATVAL: codice errato letto dalla cella DATVAL_IST della DPRAM, al termine della esecuzione dell'istogramma nella modalita' 8 (parametri IR), rilevato dalla componente MAIN.
ERG = ERR_RIGA: mancanza del sincronismo di riga, rilevato dalla ISR_TIMER;

ATTIVITA' PERIODICA

Il software svolge periodicamente le seguenti attivita':

a) ogni 150 msecondi, su segnalazione del flag COUNT_WD aggiornato dalla ISR_TIMER, invia il reset allo watch-dog timer ed incrementa il contenuto della cella DSP_RUN;

b) ogni 200 msecondi, su segnalazione del flag PERIODICA aggiornato dalla ISR_TIMER esegue la attivita' ESEGUI_PERIODICA() che consiste in:
   -legge attraverso il convertitore A/D i segnali analogici in ingresso ai canali 1 e 2 del multiplexer e deposita i valori letti nelle celle THERM_INF e THERM_SUP della DPRAM;
   -legge dal registro HARD_REG2 il valore relativo al "dito freddo" ed aggiorna la cella DITO_FREDDO della DPRAM;
   -gestisce i comandi di apertura e di chiusura della elettrovalvola presenti in COMELEV, in particolare invia, tramite porta seriale sincrona, alla testa IR un pacchetto seriale costituito da 24 bit avente il seguente formato:

TINF.TSUP.XX

dove:

TINF e TSUP sono rispettivamente i valori inferiore e superiore relativi alla soglia di temperatura da impostare sulla testa IR; detti valori sono forniti dalla CO.S.C.A tramite due celle presenti nella DPRAM;

XX puo' valere, a seconda della cella COMELEV nella DPRAM e del contenuto della variabile ELETTROVALVOLA (stato precedente in cui si trova l'elettrovalvola):

03 se COMELEV = ON ed ELETTROVALVOLA = ON
00 se COMELEV = OFF ed ELETTROVALVOLA = ON
00 se COMELEV = OFF ed ELETTROVALVOLA = OFF

Nel caso sia COMELEV=ON ed ELETTROVALVOLA=OFF deve essere avviata la procedura seguente:
- viene inviato un pacchetto seriale con XX=07;
- viene attivata la procedura SPECIAL3() di attesa di 3 secondi;
- viene inviato un pacchetto eriale con XX=00.

L'attivita' ESEGUI_PERIODICA() descritta al punto b), di norma viene attivata ogni 200 msec. Per le modalita' di funzionamento 3, 4, 8, tale attivita' deve essere inibita durante l'esecuzione di CALCOLA_MEDIA(); viene eseguita, nel consenso del flag PERIODICA, settato ad "1" ogni 200 msecondi dalla ISR_TIMER, al

termine della subroutine CALCOLA_MEDIA(). Il tempo di esecuzione della ESEGUI_PERIODICA() e' risultato inferiore a 800 usecondi; in questo modo e' assicurata la corretta sincronizzazione con il sincronismo di frame.

ATTIVITA' SPECIALE

Tale attivita' ha inizio quando la CO.S.C.A deposita sulla Dual Port un comando di apertura dell'elettrovalvola e l'elettrovalvola risulta chiusa. In tale caso e' necessario attendere un tempo di circa 3 secondi; durante tale intervallo non viene svolta nessuna attivita'; verra' aggiornato DSP_RUN e inviato il reset al watch_dog timer a cura della ISR_TIMER. Al termine di questa attivita' viene attivata la modalita' STAND_BY.

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' 1,2,5,6,9 e' riportato in figura 4.

6.1.2.1 Flusso generale

Si ritiene sufficiente la descrizione riportata in 5.6.

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_FLIR. 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 Routines) devono sottostare.
Il programma principale, le tre Interrupt Service Routine e tutte le procedure, risiedono nella memoria programma a partire dall'indirizzo 0x0240, immediatamente dopo la pagina destinata agli interrupt vector (0x000 - 0x003F).

Al fine di rendere piu' veloce l'esecuzione, il codice relativo alla lettura della memoria FIFO, all'accumulo dei dati letti, alla loro normalizzazione e trasferimento alla DPRAM viene ricopiato automaticamente, in fase di POWER_UP, nella memoria programma interna al DSP (indirizzo $100).
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
 *  "     "      $000A (ORG   P:$000A)     JSR    isr_riga
 *                                         NOP
 *  "     "      $001C (ORG   P:$001C)     JSR    isr_timer
 *                                         NOP
 */
```

6.1.2.2.1 Sottocomponente POWER_UP

La prima cosa eseguita dalla procedura POWER_UP e' quella di scrivere nella memoria programma interna al DSP il codice relativo ai loop critici in velocita' della calcola_media() e identificato dalle label da_questo_punto: e a_questo_punto: . Per fare cio' il programma deve settare necessariamente il modo di funzionamento 2 altrimenti non vedrebbe la memoria interna al processore. E' compito della presente sottocomponente anche il test della memoria doppia porta (DPRAM) con eventuale settaggio della modalita' MALFUNCTION.

```
power_up()
{
   short index,ncode;
   long  *da, *a;

      setta_modalita'_2_su_OMR_56000;
/* calcola numero di istruzioni da caricare dentro al 56000 */
   ncode=a_questo_punto - da_questo_punto;
   da=(label)(da_questo_punto);    /* indirizzo sorgente */
   a = INTERNO;                    /* indirizzo destinazione */
   for(index=0; index<ncode; index++)
```

```
        *a++ = *da++;                /* trasferisce codice */

    *(HARD1_REG) = 0x30;      /*  setta a 1 il bit D4 reset FIFO
                                   e D5 reset interrupt */
    *(DSP_RUN) = 0;
    *(CONVERTER) = 0x00;
    clock_10m=0;
       setta_0_stati_di_wait_su_56000;
       setta_ad_1_il_bit7_di_OMR_56000;
    resetta_FIFO_a_zero();
    reset_IRQB();
    reset_watch_dog();
    error=0;
    errore=0;
    clock_10m=0;
    media_flag=0;
    for(index=0; index< 100; index++)
    {
       *(DUAL_PORT+index)=index;      /* scrive nella dual-port */
       if(*(DPRAM+index) != index)
       {
          error |= GEN_ERROR;
          error |= ERR_DPRAM;        /* errore su dual port ram */
          modo_c = MALFUNCTION;
          *(ERRORE) = error;
       }
    }
    *(ASSENZA_DATI)=TRUE;
    *(DITO_FREDDO)=NONREGIM;
    *(DATVAL_IST)=FALSE;
    *(ESITO_DIAG)=0;
    *(COMELEV)=OFF;
    *(THINF_IMP)=0x03f;
    *(THSUP_IMP)=0x07f;
    *(MOD_FUNCT)=0;
    *(STATO_CAL)=CAL_STBY;
    elettrovalvola=OFF;
    semaforo=OFF;
    conteggio3=0;
    modo_c=STAND_BY;
//  azzera tutti i contatori e i flag di utilizzo
    count_wd=0;
    count_periodica=0;
    irqa_to=0;
    dsp_run=0;
    setta_interrupt();    /* abilita interrupt da IRQA IRQB
                             setta timer interno a 2 msecondi */
    *(MR) &= 0x0fc;
```

```
    while (1)
    {
       while(clock_flag == 0);                    /* attesa 10 msec */
       clock_flag=0;
       (*function[modo_c])();
       if(modo_c != OPERATIVO && modo_c != PARAM_IR &&
                                 modo_c != CALIBRAZIONE)
       {
          if (periodica)
          {
             esegui_periodica();
             periodica=0;
          }
       }
    }
}
```

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 ;

   error=0;
   for(i=0; i<NFUNCT; i++)
      func_status[i]=-1;
   func_status[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 ;

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

6.1.2.2.4 Sottocomponente DIAGNOSTICA

La modalita' diagnostica effettua il test della memoria locale presente sulla scheda e se il valore letto dall'A/D converter e' o meno nelle tolleranze specificate. Nel caso che tutti detti 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;
   register short result;

   if(func_status[DIAGNOSTICA] != -1)
      return;
   func_status[DIAGNOSTICA]++;
   result = TRUE;
   pointer = LOCAL_MEM; /* pointer punta alla base della local */
   for(i=0; i<LOCAL_SIZE; i++)
   {
      *pointer = i;
      if(*pointer++ != i)
         result = FALSE;
   }
   legge_AD(VREF);                          legge valore VREF
   if(dato_AD < NINF || dato_AD > NSUP)     e' in tolleranza ?
      result = FALSE;
   if(result == TRUE)
      *(ESITO_DIAG) = DIAGNOS_OK;
   else
      *(ESITO_DIAG) = DIAGNOS_NO;
}
```

6.1.2.2.5 Sottocomponente OPERATIVO

La sottocomponente OPERATIVO utilizza prevalentemente la subroutine calcola_media() che e' quella che esegue il calcolo della riga mediata di ciascun frame e che pone la stessa nella DPRAM.

```
operativo()
{
   for(i=0; i<NFUNCT; i++)
      func_status[i]=-1;
   func_status[OPERATIVO]=0;
   while(modo_c == OPERATIVO)
      calcola_media(OPERATIVO);
}
```

6.1.2.2.6 Sottocomponente CALIBRAZIONE

La sottocomponente CALIBRAZIONE utilizza la subroutine calcola_media() per effettuare una media di 10 righe risultanti in 10 frame successivi. Il tempo totale di esecuzione della sottocomponente CALIBRAZIONE e' pari a 20 cicli di frame_sync in quanto un frame e' utilizzato completamente dalla calcola_media() e il successivo dall'elaborazione necessaria per l'aggiornamento della media globale delle 10 linee.

```
calibrazione()
{
   register short i;

   for(i=0; i<NFUNCT; i++)
      func_status[i] = -1;
   func_status[CALIBRAZIONE]=0;
   while (modo_c == CALIBRAZIONE) loop infinito fino a che modo_c
   {                              non diventi diverso da CALIBRAZIONE
      if(*(STATO_CAL) == CAL_RUN)
      {
         pointer = buffer2;              indirizzo di buffer2[]
         for(i=0; i<ARRAY_SIZE; i++)     azzera array di lavoro
            *pointer++ = 0;
         nframe = 0;
         while (nframe < 10)
         {
            calcola_media(CALIBRAZIONE);     calcola riga mediata
            IRQA_flag=0;
            while(IRQA_flag == 0 && modo_c == CALIBRAZIONE);
```

```
            IRQA_flag=0;
            if(modo_c != CALIBRAZIONE)
               return;
            from = CDATI_IR;              puntatore sorgente
            for(i=0; i<1024; i++)
               buffer2[i] += *from++;     somma dato
            nframe++;
            if(periodica)                 attivita' periodica
            {
               esegui_periodica();
               periodica=0;
            }
         }
         pointer = CDATI_IR;      puntatore all'array destinazione
         for(i=0; i<1024; i++)       loop esterno sulle 1024 somma
            *pointer++ = buffer2[i]/10;   divide per 10
         *(STATO_CAL) = CAL_END;     segnala avvenuta calibrazione
      }
      else
      {
         if(periodica)                    /* attivita' periodica */
         {
            esegui_periodica();
            periodica=0;
         }
      }
   }
}
```

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' GAIN_SHUTTER, nella presente versione del software, non effetta nessuna operazione particolare.

```
gain_shutter()
{
   register short i;

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

6.1.2.2.10 Sottocomponente PARAM_IR

La sottocomponente PARAM_IR utilizza la subroutine calcola_media() per effettuare un istogramma su 10 righe risultanti in 10 frame successivi. Il tempo totale di esecuzione della sottocomponente PARAM_IR e' pari a 20 cicli di frame_sync in quanto un frame e' utilizzato completamente dalla calcola_media() e il successivo dall'elaborazione necessaria per l'aggiornamento dell'istogramma.

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

   primo_frame = 0;
   nframe = 0;
   *(DATVAL_IST) = 0;      azzera la cella dati validi istogramma
   while (modo_c == PARAM_IR)
   {
      if(primo_frame = 0)              se il primo frame
      {
         pointer = buffer2;            indirizzo di buffer2[]
         for(i=0; i<ARRAY_SIZE; i++)   azzera array di lavoro
            *pointer++ = 0;
         primo_frame = 1;
      }
      while (nframe < 10)
      {
         calcola_media(PARAM_IR);      calcola la riga mediata
         IRQA_flag=0;
         while(IRQA_flag == 0 && modo_c == PARAM_IR);
         IRQA_flag=0;
         if(modo_c != PARAM_IR)
            return;
         from = CDATI_IR;              puntatore sorgente
         for(i=0; i<1024; i++)
         {
            dato = *from++;            legge dato
            if(dato > 1023)            testa overflow
               buffer2[1023]++;        incrementa cella 1023a
            else
               buffer2[dato]++;        incrementa cella indiciata
         }
```

```
            nframe++;
            if(periodica)                    attivita' periodica
            {
               esegui_periodica();
               periodica=0;
            }
         }
         pointer = buffer2;  puntatore all'array sorgente (inc.)
         to = ISTOGRAMMA;    puntatore alla destinazione dati DPRAM
         val_max = 1;        conterra' il valore massimo
         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 loc.
            *to++ = accumula;          successive
            if(accumula > val_max)
               val_max = accumula;
         }
         pointer = ISTOGRAMMA; puntatore ai 64 dati istogramma DPRAM
         if(*(DATVAL_IST) != FALSE)
         {
            error |= GEN_ERROR;
            error |= ERR_DATVAL;   errore su cella DATI VALIDI ISTOG
            modo_c = MALFUNCTION;
            *(ERRORE) = error;
         }
         else                        /*      *(DATVAL_IST == FALSE)  */
         {
            for(i=0; i<64; i++)      /*      aggiorna la dual port */
            {
               dato = *pointer;
               *pointer++ = dato*32/val_max;
            }
            *(DATVAL_IST) = TRUE;
            while(*(DATVAL_IST) == TRUE && modo_c == PARAM_IR)
            {
               if(periodica)                 /* attivita' periodica */
               {
                  esegui_periodica();
                  periodica=0
               }
            }
            primo_frame = 0;
         }
      }
   }
```

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 e' l'unica attivata dal programma interno e non attivata dalla CO.S.C-A. Essa trasferisce il contenuto della variabile error nella cella ERRORE predisposta nella DPRAM.

```
errore()
{
   if(func_status[MALFUNCTION] != -1)
      return;
   for(i=0; i<NFUNCT; i++)
      func_status[MALFUNCTION]=-1;
   func_status[MALFUNCTION]++;
   *(ERRORE) = error;
}
```

6.1.2.2.13 Sottocomponente SPECIAL3

E' la procedura che gestisce l'attesa di tre secondi da parte del processore quando viene rilevata la richiesta di accensione dell'elettrovalvola da parte della CO.S:C.A e, quest'ultima, risulta spenta.

```
special3()                          /* attesa 3 secondi */
```

```
{
   short index;
   register short i;

   for(i=0; i<NFUNCT; i++)
      func_status[SPECIAL_3]=-1;
   func_status[SPECIAL_3]++;
   while(conteggio3 <= N3SECONDI);          /* N3SECONDI = 1500 */
   semaforo=OFF;
   conteggio=0;
   elettrovalvola=ON;
   modo_c=STAND_BY;
}
```

6.1.2.2.14 Sottocomponente ESEGUI_PERIODICA

Con la temporizzazione del flag "periodica", aggiornato dalla sottocomponente ISR_TIMER, deve essere svolta ogni 200 msecondi circa, l'attivita' periodica; in particolare deve essere inviato un dato seriale costituito da un pacchetto di 24 bit in maniera sincrona.
La temporizzazione per l'invio dei singoli bit e' ottenuta via software. Durante l'invio del dato seriale (la cui durata e' di 240 usecondi) non devono avvenire interruzioni; si controllera' pertanto che l'invio del dato inizi dopo che e' arrivato un sincronismo di riga (IRQB); che durante l'invio non possa arrivare una interruzione IRQA e si provvedera' a disabilitare l'interrupt da parte del timer interno.
L'attivita' periodica dovra' essere inoltre sincronizzata con le altre attivita'; in particolare se e' in corso un'attivita' che prevede la lettura continua delle righe, l'attivita' periodica dovra' essere servita al termine dell'elaborazione in corso sulle righe ma prima del sincronismo di frame successivo.

```
esegui_periodica()
{
   short ditofreddo;
   short termal1, thermal2;
   short electv;

   legge_AD(LEVEL_TH1);
   thermal1 = dato_AD;          /* dato ad 8 bit */
   legge_AD(LEVEL_TH2);
   thermal2 = dato_AD;          /* dato ad 8 bit */
   *(THERM_INF) = thermal1;
   *(THERN_SUP) = thermal2;
   ditofreddo = *(HARD2_REG) & 0x40;
```

```
    if(ditofreddo == 1)
       *(DITO_FREDDO) = TRUE;
    else
       *(DITO_FREDDO) = FALSE;
    if(*(COMELEV) == OFF)
    {
       if(elettrovalvola == ON)
          electv=0;
       else
          electv=0;
    }
    else
    {
       if(elettrovalvola == OFF)
       {
          electv=0x07;
          invio(electv);
          conteggio3=0;
          semaforo=ON;
          special3();
       }
       else
          electv=0x03;                /* continua a stare acceso */
    }
    invio(electv);
}
```

6.1.2.3 Elenco subroutine

Nel presente paragrafo vengono elencate le procedure a basso livello utilizzate dalle componenti il programma.

6.1.2.3.1 Procedura INVIO.

```
void invio(electv)
short electv;
{
   unsigned char sthermal1,sthermal2;
   long (24bit) dato;

   sthermal1 = *(THINF_IMP);
   sthermal2 = *(THSUP_IMP);
   dato = sthermal1 * 65536 + sthermal2 * 256;
   dato |= electv;      /* setta i 3 LSB */
   *(MR) |= 0x03;       /* disabilita tutti gli interrupt */
   serial_out(dato);    /* trasmissione verso porta seriale */
```

```
    *(MR) &= 0x0fc;      /* riabilita gli interrupt */
}
```

6.1.2.3.2 Procedura RESET_IRQB.

```
Reset_IRQB()
{
    *(HARD1_REG) &= 0xDF;     /* azzera il bit d5 di Hardware 1 */
    *(HARD1_REG) |= 0x20;
}
```

6.1.2.3.3 Procedura RESET_WATCH_DOG.

```
reset_watch_dog()
{
    *(HARD1_REG) |= 0x40;          setta a 1 il solo bit D6
    *(HARD1_REG) &= 0xBF;          setta a 0 il solo bit D6
    *(DSP_RUN)++;          incrementa cella DSP_RUN della dual-port
}
```

6.1.2.3.4 Procedura LEGGE_AD.

```
legge_AD(short select)
{
    register unsigned char select;
    register short i,mask;

    *(CONVERTER) = select;
    for(i=0; i<1000; i+++);    attende 100uS (Fc = 100nS)
    *(CONVERTER) |= 0x03;      setta bit D0 e D1 (SOC e ALE)
    *(CONVERTER) = 0;          mette a zero bit D0 e D1 (SOC e ALE)
    mask = *(HARD2_REG);
    while((mask & 0x80) == 0) attesa bit fine conversione
       mask = *(HARD2_REG);
    *(CONVERTER) |= 0x04;      setta il bit OED
    dato_AD = *(CONVERTER);    legge dato convertito;
    *(CONVERTER) = 0;
}
```

6.1.2.3.5 Procedura RESETTA_FIFO_A_ZERO.

```
resetta_FIFO_a_zero()
{
   *(HARD1_REG) &= 0xEF;     /* reset il bit D4 del hardware 1 */
   *(HARD1_REG) |= 0x10;
}
```

6.1.2.3.6 Procedura SERIAL_OUT.

```
serial_out(dato)
long(24 bit)   dato;
{
   short index,jdex;

   maschera=0x$000001;                  /* sempre a 24 bit */
   *(HARD1_REG) |= 0x04;                /* alza il bit DAV */
   for(index=0; index<24; index++)
   {
      if((maschera & dato) == 0)
         *(HARD1_REG) &= 0x0FD;    /* azzera linea seriale DATO */
      else
         *(HARD1_REG) |= 0x002;    /* setta linea seriale DATO */
 /* aspetta 1 microsecondo prima di settare il clock          */
      for(jdex=0; jdex<10; jdex++);
      *(HARD1_REG) |= 0x01;             /* setta linea di clock */
      for(jdex=0; jdex<50; jdex++);     /* attesa 5 microsecondi */
      *(HARD1_REG) &= 0xFE;             /* azzera linea di clock */
      for(jdex=0; jdex<40; jdex++);     /* attesa 4 microsecondi */
      maschera = maschera << 1;         /* shift 1 place left */
   }
   *(HARD1_REG) &= 0xFB;                /* riabbassa la linea DAV */
}
```

6.1.2.3.7 Procedura CALCOLA_MEDIA.

```
calcola_media(short modalita)
{
   short index,linea;
   short dato,overflow;

   for(index=0; index<1024; index++)     /* azzera buffer d'uso*/
      buffer[index]=0;
/*  se  modo_c e' uguale a operativo esegue il loop  in  continuo
```

```
    altrimenti il loop viene eseguito una volta solamente e
    ritorna al chiamante */
   IRQA_flag=0;
   while(IRQA_flag == 0 && modo_c == modalita);
   IRQA_flag=0;
   if(modo_c != modalita)
      return;
   IRQB_flag=0;
   while(IRQB_flag == 0 && modo_c == modalita);
   IRQB_flag=0;
   if(modo_c != modalita)
      return;
   media_flag=1;        /* gestione da programma del reset FIFO */
   linea=0;
   while(linea < NLINEE)   /* le linee valide devono essere 30 */
   {
      while(IRQB_flag == 0&& modo_c == modalita);
      IRQB_flag=0;
      if(modo_c != modalita)
      {
         media_flag=0;
         return;
      }
      goto interno:

/*
==========================================================================
==== */

da_questo_punto:
      linea++;                    /* incrementa contatore linea */
      for(index=0; index<1024; index++)
      {
         dato=*(FIFO_DATI);           /* legge dato dalla FIFO */
         dato=overflow | (dato & 0x0080FF);  /* setta dato */
         buffer[index] += dato;              /* somma dato */
      }
      while(IRQB_flag == 0 && modo_c == modalita);
      IRQB_flag=0;
      if(modo_c != modalita)
      {
         media_flag=0;
         return;
      }
      resetta_FIFO_a_zero();   /* assicura l'azzeramento dei
                  puntatori nella FIFO prima della riscrittura */
   }
   media_flag=0;      /* abilita clear FIFO da parte isr_riga() */
```

```
   for(index=0; index<1024; index++)     /* normalizza a 10 bit */
   {
      dato=buffer[index];
      if(dato > 0x01fe0)
         overflow=0x008000;
      else
         overflow=0;
      dato &= 0x01FFF;
      dato /= 8;
      *(CDATI_IR+index) = dato | overflow;      /* add overflow */
   }
   goto return_out:
a_questo_punto:

/*
==========================================================================
==== */

return_out:
   if(periodica)                          /* attivita' periodica */
   {
      esegui_periodica();
      periodica=0;
   }
}
```

6.2 Componente ISR_TIMER

6.2.1 Descrizione funzionale

La ISR_TIMER() viene attivata ogni 2 msecondi 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. 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 150 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()
{
```

```
if(IRQB_to == 0)
{
   error |= GEN_ERROR;
   error |= ERR_RIGA; /* errore mancanza sicronismo di riga */
   modo_c = MALFUNCTION;
   *(ERRORE) = error;
   *(ASSENZA_DATI) = TRUE;  /* setta variabile assenza dati */
}
IRQB_to=0;
clock_10m++;
if(clock_10m >= NUMCK10)
{
   clock_flag=1;                  /* settato ogni 10 msecondi */
   clock_10m=0;
   modo = *(MOD_FUNCT);
   if(modo < STAND_BY || modo  > MALFUNCTION)
   {
      error |= GEN_ERROR;
      error |= ERR_CODE;
      *(ERRORE) = error;
      modo_c = MALFUNCTION;
   }
   if(*(ERRORE))
      modo_c=MALFUNCTION;
   else
      modo_c=modo;
}
IRQA_to++;
if(IRQA_to >= NUMFSTO)
{
   error |= GEN_ERROR;
   error |= ERR_FRAME;
   *(ERRORE) = error;
   modo_c = MALFUNCTION;
}
count_WD++;
if(count_WD == NTO_WD)
{
   count_WD=0;
   reset_watch_dog();
}
if(semaforo == ON)
   conteggio3++;
count_periodica++
if(count_periodica == START_PER)
{
   periodica=1;
   count_periodica=0;
```

```
    }
    return_from_interrupt;
}
```

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 IRQA_flag di avvenuto evento (detto flag deve essere riazzerato dalla procedura che ne fa' uso) e azzera il flag IRQA_to necessario alla ISR_TIMER per il conteggio del time_out di controllo sulla presenza del segnale frame_sync stesso.

```
isr_frame()
{
    IRQA_flag=1;
    IRQA_to=0;
    return_from_interrupt;
}
```

6.4 Componente ISR_RIGA

6.4.1 Descrizione funzionale

La ISR_RIGA viene attivata dal segnale di sincronizzazione esterno sync_riga (512 usec). Setta il flag IRQB_flag di avvenuto evento (detto flag deve essere riazzerato dalla procedura che ne fa' uso) e azzera il flag IRQB_to necessario alla ISR_TIMER per il conteggio del time_out di controllo sulla presenza dati in arrivo sulla FIFO.

```
isr_riga()
{
    IRQB_flag=1;
    IRQB_to=1;
    Reset_IRQB();
    if(media_flag == 0)
        resetta_FIFO_a_zero();
}
```