ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Napoli

INFN/TC–01/02
18 Gennaio 2001

MEDISOFT
Versione 2.2

Guida all'uso e file sorgente del programma per l'acquisizione di immagini con sensore Medipix

Ennio Bertolucci, Maurizio Conti, Giovanni Mettivier, Maria Cristina Montesi, Paolo Russo[1]

INFN Sezione di Napoli, Complesso Universitario di S. Angelo, Via Cinthia, 80126 Napoli

Published by ***SIS–Pubblicazioni***
Laboratori Nazionali di Frascati

1 Corrispondenza: Prof. Paolo Russo, INFN Sezione di Napoli, Complesso Universitario di S. Angelo, Via Cinthia, 80126 Napoli. Tel: 081676146, e-mail: Paolo.Russo@na.infn.it

Nell'ambito dell'esperimento MED-46 finanziato dalla V commissione nazionale dell'INFN per lo sviluppo di un Sistema di Imaging Mammografico Integrato e di un sistema per autoradiografia digitale, il Gruppo di Fisica Medica del Dipartimento di Scienze Fisiche dell'Università "Federico II" di Napoli e della Sezione INFN di Napoli ha sviluppato un sistema software, denominato "Medisoft", capace di gestire le funzionalità di un sistema di acquisizione di immagini radiografiche basato sul chip VLSI Medipix 1 disegnato dal CERN di Ginevra[*].

Il software (in linguaggio C) e l'interfaccia grafica per il sistema Medipix sono stati realizzati utilizzando l'ambiente "LabWindows/CVI" della National Instruments come piattaforma di sviluppo. L'interfaccia grafica, che ha lo scopo di rendere semplice all'utente finale l'uso del sistema Medipix e di consentire la visualizzazione e una prima elaborazione delle immagini acquisite, è organizzata in pannelli in cui bottoni virtuali e barre di menù consentono di implementare tutte le diverse funzioni relative al test del sistema ed all'acquisizione di immagini.

Il software è stato scritto in modo da rendere possibile la gestione, tramite personal computer IBM compatibile, di ogni tipo di impostazione che si renda necessaria per le acquisizioni; permette ad esempio di impostare le tensioni di lavoro del chip, di eseguire la calibrazione delle soglie di rivelazione della radiazione e quindi di costruire la maschera di ottimizzazione dell'array del sistema Medipix da impostare per l'acquisizione.

La seconda parte di cui è costituito il software serve alla lettura dei dati e alla traduzione in una immagine grafica della matrice di numeri proveniente dal chip.

Questo documento si compone in due parti:

- Breve manuale
- codice sorgente

e ha come finalità una buona descrizione del sistema hardware/software per la gestione del sistema di imaging Medipix 1.

Di seguito sono riassunte alcune fondamentali operazioni eseguibili per ottenere l'acquisizione di un'immagine:

- accensione dell'alimentazione del sistema Medipix e caricamento delle tensioni di lavoro (DAC bias) del chip,
- caricamento della maschera;

[*] Il sistema di rivelazione Medipix 1 è stato sviluppato dalla collaborazione Medipix 1 attiva presso il CERN di Ginevra.

- impostazione del tempo τ di acquisizione dell'immagine;
- impostazione della modalità di acquisizione Radiography (il sistema immagazzina dati per il tempo τ) o Autoradiography (il sistema acquisisce per un tempo τ immagini ripetute e le salva automaticamente su file).

Altre funzioni implementate dall'interfaccia grafica sono:

- visualizzazione grafica delle immagini acquisite, mediante visualizzazione in scala di colori del conteggio contenuto nei 64x64 pixel del rivelatore;
- elaborazione immagini tramite operazioni di limitazione dell'intervallo di conteggi e relativa impostazione del numero dei livelli di colore ritenuto ottimale per la visualizzazione;
- somma e sottrazione di file di immagini già acquisite;
- possibilità di definire aree rettangolari in cui si può valutare dei conteggi ed effettuare degli zoom;
- controllo micrometrico della movimentazione del campione.

Il software è pensato per avere una grande modularità in modo che differenti utenti possano integrarlo con caratteristiche particolari senza cambiare il nucleo del software originale. La versione attuale del codice Medisoft, così come descritta nel presente documento, è la 2.2 (Medisoft 2.2).

Il sistema Medisoft si compone di:

- file di libreria
- file sorgente
- file di valori di default
- file di LUT

Ci sono i file di libreria (function.h, main.h, medipix.h) nei quali sono definiti le variabili usate e le funzioni di basso livello per le operazioni di lettura e scrittura sui registri di memoria; file di sorgente (function.c, main.c, menu.c, pulse.c) dove ogni tipo di funzione e routine usate sono definite; file che contengono tutti i valori di default usati dal Medipix chip (default.msk, default.arc, default.pat) e una look-up table file per la corrispondenza tra il numero sequenza generato dal contatore pseudorandom di Medipix e una ordine sequenziale di numerazione.

PREREQUISITI

Conoscenza del funzionamento del chip Medipix 1

REQUISITI HARDWARE

Chip Medipix 1

Create VME

Scheda VMEboard

Scheda MOTHERboard

Scheda CHIPboard

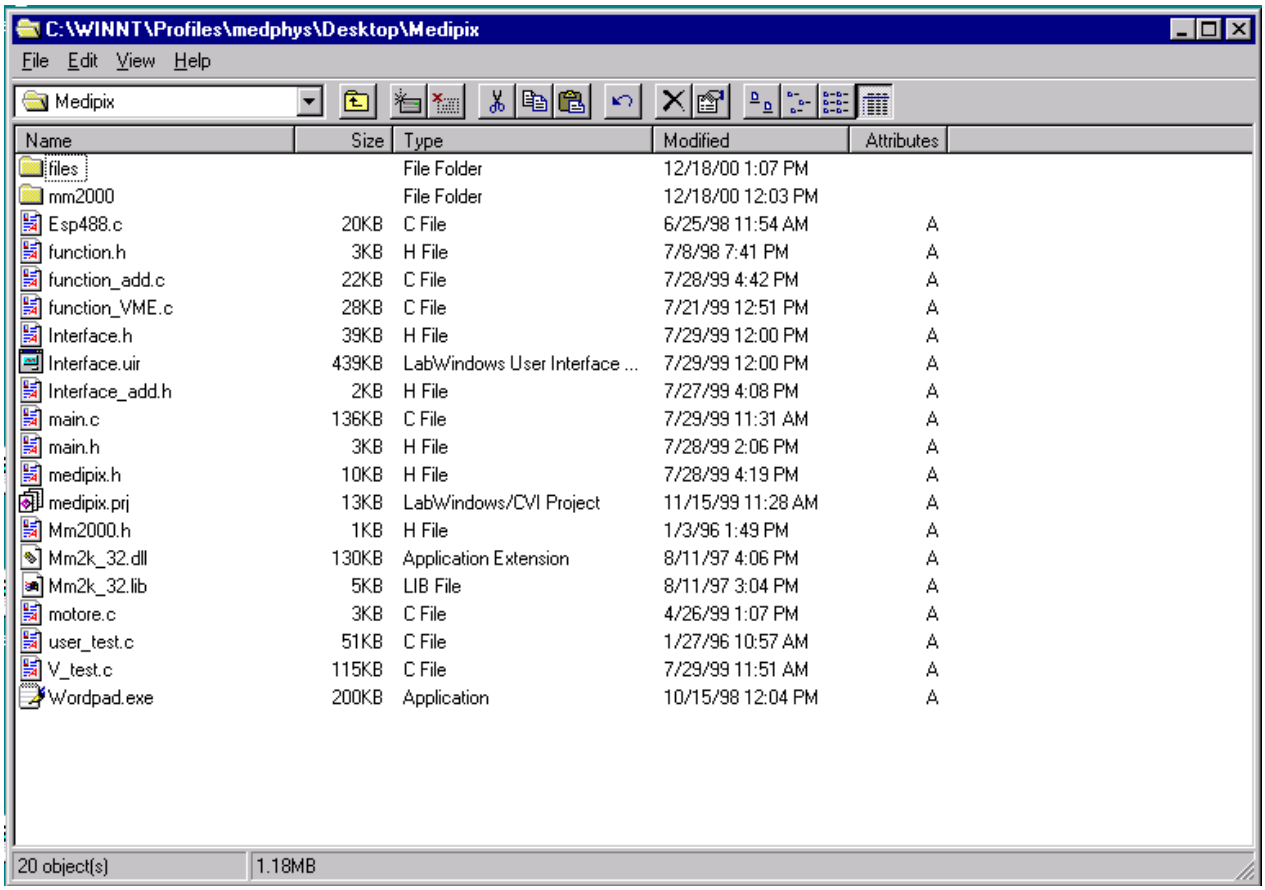
Kit d’interfaccia VXI PCI8015

REQUISITI SOFTWARE

Personal Computer IBM compatibile

Sistema Operativo Windows NT o 98

LABWindows/CVI (preferibile)


C:\WINNT\Profiles\medphys\Desktop\Medipix
File Edit View Help
Medipix
Name Size Type Modified Attributes
files File Folder 12/18/00 1:07 PM
mm2000 File Folder 12/18/00 12:03 PM
Esp488.c 20KB C File 6/25/98 11:54 AM A
function.h 3KB H File 7/8/98 7:41 PM A
function_add.c 22KB C File 7/28/99 4:42 PM A
function_VME.c 28KB C File 7/21/99 12:51 PM A
Interface.h 39KB H File 7/29/99 12:00 PM A
Interface.uir 439KB LabWindows User Interface ... 7/29/99 12:00 PM A
Interface_add.h 2KB H File 7/27/99 4:08 PM A
main.c 136KB C File 7/29/99 11:31 AM A
main.h 3KB H File 7/28/99 2:06 PM A
medipix.h 10KB H File 7/28/99 4:19 PM A
medipix.prj 13KB LabWindows/CVI Project 11/15/99 11:28 AM A
Mm2000.h 1KB H File 1/3/96 1:49 PM A
Mm2k_32.dll 130KB Application Extension 8/11/97 4:06 PM A
Mm2k_32.lib 5KB LIB File 8/11/97 3:04 PM A
motore.c 3KB C File 4/26/99 1:07 PM A
user_test.c 51KB C File 1/27/96 10:57 AM A
V_test.c 115KB C File 7/29/99 11:51 AM A
Wordpad.exe 200KB Application 10/15/98 12:04 PM A
20 object(s) 1.18MB

Descrizione dell’

Hardware di ReadOut

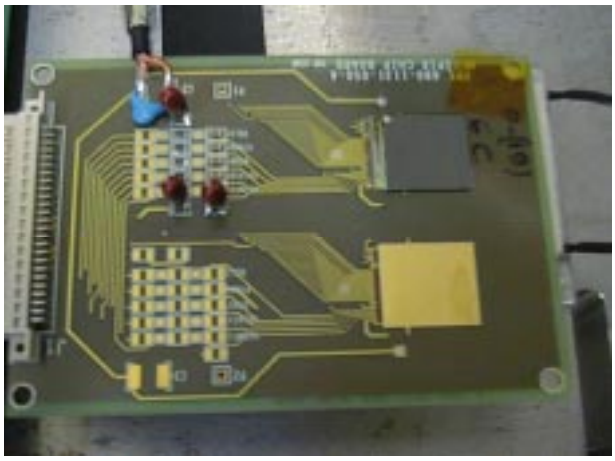
Il sistema di lettura elettronico digitale che fa da supporto al sistema di lettura Medipix, indicato come Medipix Readout System (MRS) è realizzato come indicato di seguito.

Esso è stato progettato dall'INFN e realizzato dalla LABEN S.p.A (Milano) e permette l'acquisizione dei dati e il controllo delle funzioni del chip.

Basato sullo standard VME è costituito da vari componenti:

- **VMEbus**

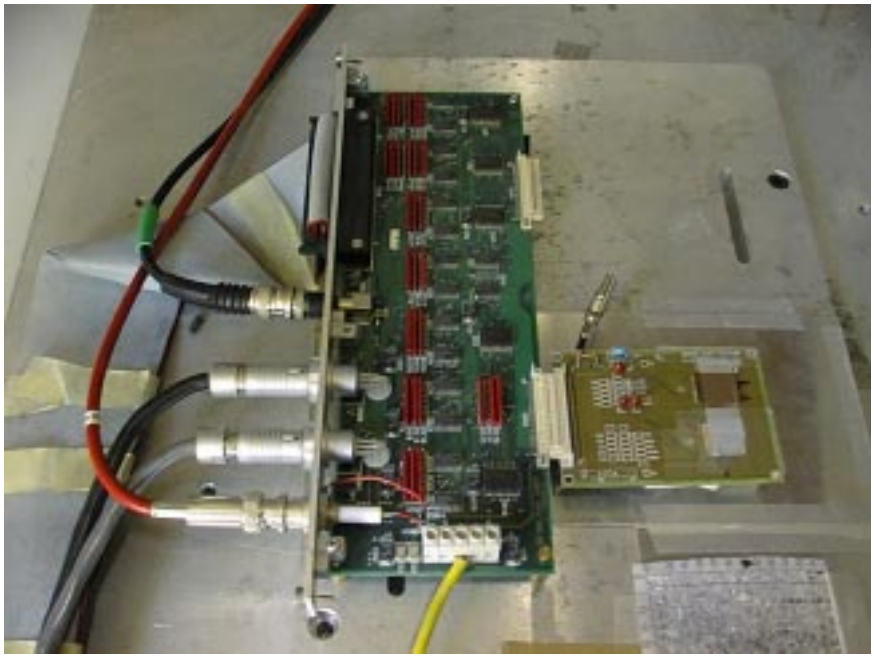
- **CHIPboard**: E' una scheda stampata stand-alone connessa alla MOTHERboard attraverso un connettore da 50 pin. Serve da alloggiamento del chip Medipix 1+ rivelatore al Silicio o GaAs (nel seguito detto "assembly"), incollato con una pasta conduttrice al piano di massa, ed è anche contattato alle piste di connessione attraverso delle microsaldature ad ultrasuoni. Può ospitare fino a due Medipix e contiene condensatori di by-pass e resistenze; i primi, servono a scaricare verso massa gli sbalzi di tensione, le seconde sono resistenze di polarizzazione tramite cui arrivano al chip alcuni valori di riferimento quali le tensioni analogiche di controllo del chip e quella per la polarizzazione del rivelatore (1nF, 1kV, COG dielettrico, 1812 SMD) (fig 2.3). Vi sono anche due condensatori di alta tensione sulle linee di alimentazione del rivelatore.



CHIPboard

- **la MOTHERboard**: E’ anch’essa una scheda stand-alone che fa da buffer per la trasmissione dei segnali. Supporta la CHIPboard e ospita l’elettronica digitale e analogica necessaria per la trasmissione dei segnali da e per la VMEboard, scheda progettata e posizionata per il controllo di Medipix.

  La MOTHERboard è anche predisposta, attraverso un connettore dedicato, per ricevere il segnale di alta tensione per la polarizzazione del rivelatore di Medipix, tensione che viene fornita da un generatore esterno; fornisce alla CHIPboard (ricevendoli dalla VMEboard) tutte le tensioni di alimentazione per il chip.



MOTHERboard

- **VMEboard**: La scheda VME gestisce lo scambio dei dati dal PC al chip Medipix 1 e viceversa. Essa è una scheda standard VME (dimensione B(6U, 160x233 mm), single slot VMEbus card, interfaccia slave A32:A24:D32:D16). La sua funzione è di generare i controlli della parte digitale del chip e le tensioni analogiche necessarie al funzionamento di Medipix tramite dei Convertitori Digitale-Analogico (DAC). Anche tutte le funzioni della parte analogica digitale vengono controllate attraverso il VME. (fig. 2.5)

VMEboard

La scheda genera inoltre il segnale di acquisizione per Medipix (un segnale che abilita i contatori a contare quando il suo livello è basso) e segnali di test/controllo per strumenti esterni attraverso cavi coassiali: ANIN (ANalog INput), TOUT (Trigger OUT) e GATE. Il segnale di GATE serve per abilitare una eventuale strumentazione esterna per emissione di radiazione (per esempio tubo a Raggi X); il segnale TOUT permette di far partire il segnale di test da un eventuale generatore di impulsi esterno (i cui parametri di segnale sono programmabili via scheda GPIB, e che si pone in stato di attesa di un trigger esterno), in cui l'ampiezza variabile possa simulare diversi segnali in carica provenienti dai rivelatori; il segnale ANIN è un segnale di test per le celle di Medipix di caratteristiche fisse da usare come semplice alternativa ad un impulsatore esterno per verificare che i contatori funzionino correttamente. Essa comunica con la MOTHERboard attraverso un flatcable e un cavo n poli AB0-1 circolare multicore.

- **Impulsatore**: Il generatore di impulsi (nel nostro caso HP 8130A) è usato per generare una sequenza di impulsi (normalmente 1000) con ampiezza, periodo, tempo di salita, completamente controllati dal PC per mezzo del GPIB controller presente nel VME crate. Il generatore d'impulsi è triggerato via software da uno standard TTL generato dalla VMEboard.

- Il **kit d’interfaccia** VXI-PCI8015 permette la comunicazione tramite il bus PCI, del computer con la MOTHERboard usando il bus d’alta velocità Multisystem eXtension Interface (MXI) alloggiata nel create VME.

L’ MRS distingue quattro fasi di funzionamento:

1- SETUP: in questa fase vengono fissate ed inviate le tensioni di alimentazione, le tensioni analogiche, il reset dei contatori e viene caricata la maschera
2- AQUISIZIONE: In questa fase Medipix acquisisce dati dal rivelatore
3- TEST: acquisisce dati da un generatore di impulsi interfacciato al sistema e comandato in remoto tramite una scheda GPIB
4- READOUT: lettura dei dati dai contatori del chip Medipix 1

Logica di controllo di Medipix

La logica di controllo è composta da una serie di registri e celle di memoria raggruppabili nelle seguenti categorie:

1. Registro di stato (Status Register)
2. Registri per impostare i parametri dei DAC (Analog Register)
3. Registro per l’impostazione dei parametri di acquisizione e impostazione dei DAC
4. Memoria RAM per la memorizzazione dei parametri di acquisizione
5. Registri FIFO per la memorizzazione dei dati acquisiti da Medipix

Descrizione dell’Interfaccia Grafica

MEDISOFT

Version 2.2 January 2000

by

University of Napoli, Federico II

INFN - Napoli

Quit

Accensione del MRS

Una volta che il sistema di read-out MRS sia stato montato e il software installato[1], si può procedere alla procedura di accensione del sistema.

1. Accendere le alimentazioni del **CREATE VME**
2. **Aprire la cartella Medphys**. Poi in sequenza mandare in esecuzione:
   - vxint.exe
   - resman.exe (Appare una schermata. Bisogna aspettare il messaggio ***"Reusorce manager Operations Compeited"*** e poi chiudere)

   A seconda se l'utilizzatore abbia un eseguibile
   - medisoft.exe

   oppure l'intero codice
   - medphys.prj

   In questo caso appare la schermata, dove sono riportati tutti i file contenuti nel software. Per far eseguire il programma bisogna cliccare ***Run*** e poi ***Run project***.

   A questo punto appare la schermata del pannello di controllo di Medipix

3. **Alimentare il rivelatore**
4. **Accendere l'impulsatore**

[1] Per software installato si intende che siamo state eseguite tutte le istruzione riportate nel file Readme.txt che segue ogni versione del software

Descrizione del Pannello Principale

Il pannello principale (fig.1) è composto da vari riquadri e bottoni. Ad ogni riquadro è associato una funzione di controllo del sistema Medipix. Di seguito è riportata una descrizione di questi vari riquadri.

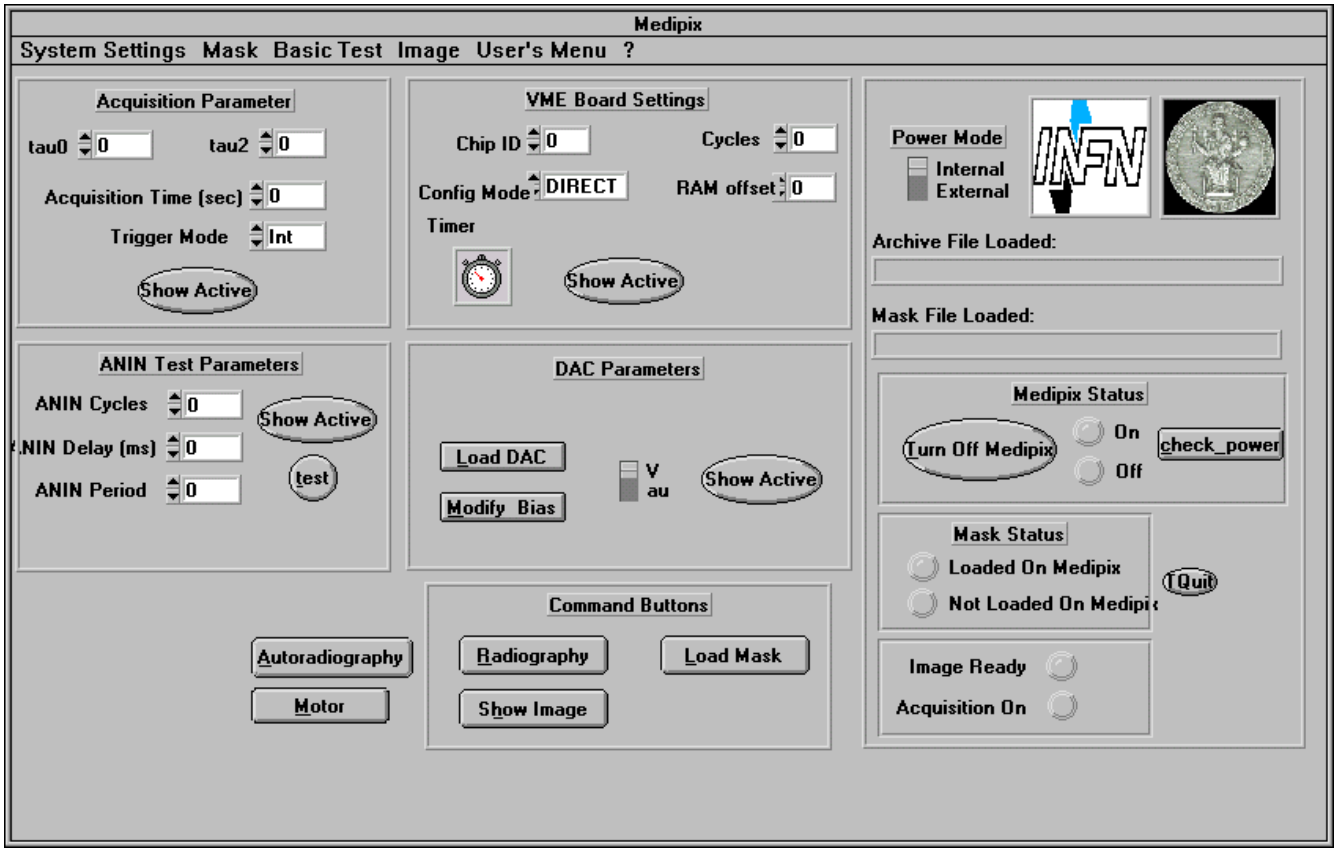


Fig. 1: Pannello Principale

Esso può dividersi in quattro parti principali:

1) la ***Barra dei Menù***:
2) L'area relativa all'***impostazione dei parametri***. In quest'area è possibile impostare i parametri del sistema attraverso i controlli numerici; attraverso i bottoni è invece possibile controllare i valori attivi (ossia relativi all'ultima acquisizione fatta). L'area è suddivisa in quattro sottosezioni ***(Acquisition Parameter, VME Board Setting, ANIN Test Parameters, DAC Parameters*)**.
3) L'area relativa alle ***informazioni di stato*** del sistema. In quest'area è possibile ottenere informazioni circa lo stato del sistema. In particolare, partendo dall'alto, si hanno informazioni

sui file di archivio e di maschera caricati, sullo stato di Medipix (acceso o spento), se la maschera è caricata o meno su Medipix, ed infine se è terminata l'acquisizione e se l'immagine è disponibile.

4) L'area relativa ai ***comandi*** da impartire al sistema. Mediante quest'area è possibile dare il via all'acquisizione di una immagine radiologica o effettuare radiografie ripetute

Descrizione della Barra dei Menù

System Settings

Chiama i sottomenù per l'impostazione dei parametri del sistema, il settaggio e il caricamento di tali parametri.

- ***Open System file****:* Questa istruzione ci permette di aprire un file di archivio (.arch) che contiene tutti i valori necessari per il funzionamento di un particolare sistema di rivelazione

- ***Save System file****:* Permette di salvare tutte le impostazioni correnti in un file di archivio

- ***Quit***: Permette l'uscita dal programma. ATTENZIONE: è necessario prima portare Medipix da *turn ON* a *turn OFF*.(Vedi pannello principale.)

Mask

Chiama il sottomenù per il caricamento e il salvataggio dei file di maschere, permette inoltre il caricamento delle maschere su Medipix e l'acquisizione dei dati.

- ***Open Mask File***: Permette l'apertura, per la visualizzazione, di un file di maschera (.msk). Questo è un file che contiene 5 bit di configurazione per ogni pixel

- ***Load & Upload Mask***: Permette di selezionare un file di maschera tra quelli contenuti nella directory FILES e di caricarlo in memoria. Una volta selezionato, il nome del file è visualizzato nella finestra *Mask File Loaded* presente sul pannello principale. Per caricare questo file sul chip bisogna cliccare il tasto *LOAD* presente sul pannello principale.

- ***Save Mask File***: Una volta realizzata la maschera, questo comando permette di salvarla

  nella directory Medipix\files

- ***Modify Mask**:* Permette di modificare, pixel per pixel, il contenuto di una delle seguenti matrici

- *Enable mask*: Abilita (1) o disabilita (0) il funzionamento di un pixel. Esistono due modalità per cambiare questa maschera. Infatti appena cliccato questo comando appare un pannello (fig. 2) che chiede quale tra le due modalità indicate è quella che si vuole adottare per procedere, se per singolo pixel o per gruppi di pixel.

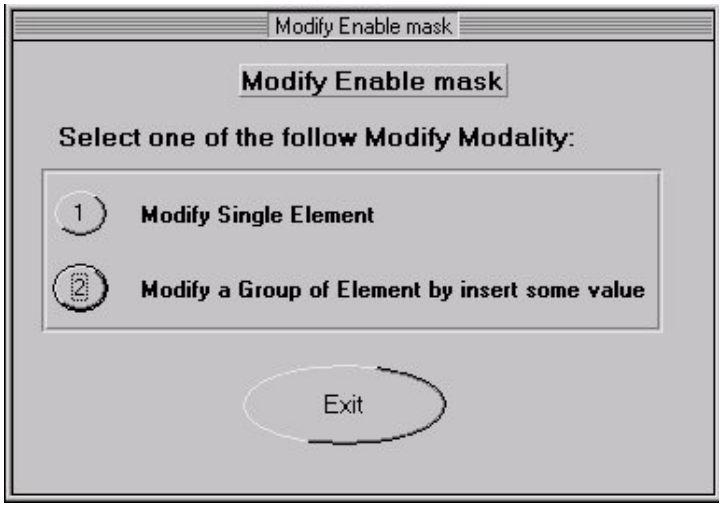


Fig. 2:

Premendo *Exit* è possibile ritornare al pannello principale senza effettuare nessuna scelta. Se si clicca 1 e cioè si effettua la scelta della variazione del singolo pixel appare il pannello (fig. 3) che richiede le coordinate (row, col) del pixel che si vuole modificare, il valore attuale del pixel e il nuovo valore assegnato. Per effettuare il cambiamento bisogna cliccare sul pulsante *Done*. *Cancel* permette di uscire senza effettuare nessun'operazione.

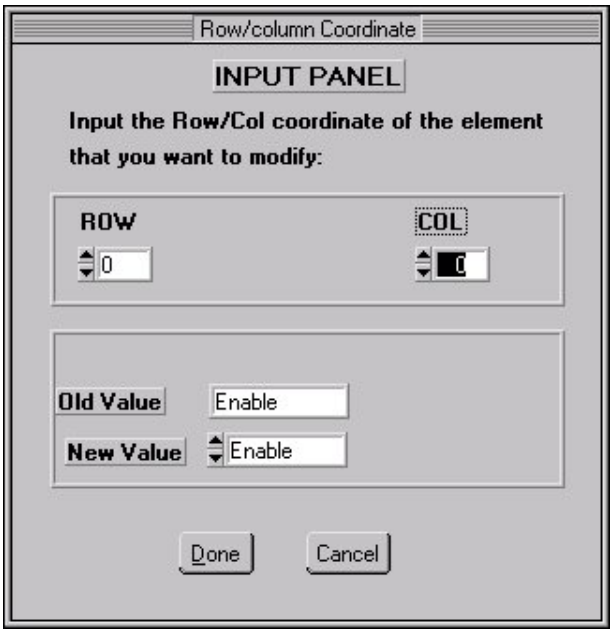


Fig: 3:

Se invece, si sceglie 2, cioè la modifica di un gruppo di pixel, appare il pannello (fig. 4) in cui bisogna riportare le coordinate necessarie per l'individuazione del gruppo di pixel (sottomatrice). Quando si clicca su Ok appare un nuovo pannello (fig. 5) che permette di scegliere il nuovo valore da settare per il gruppo di pixel.

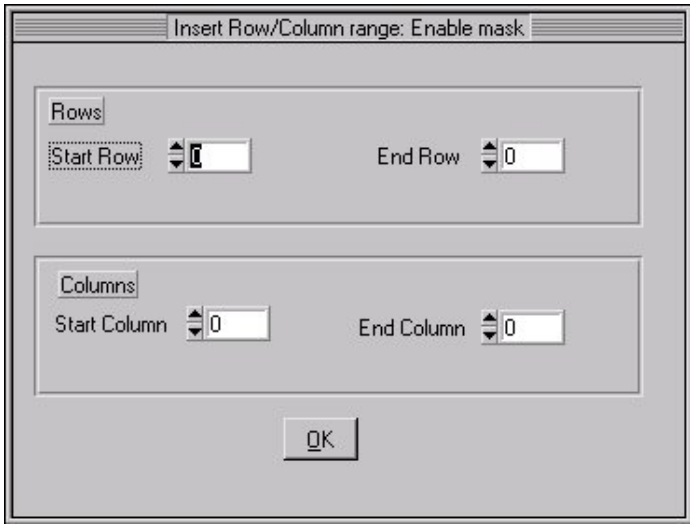


Fig. 4:

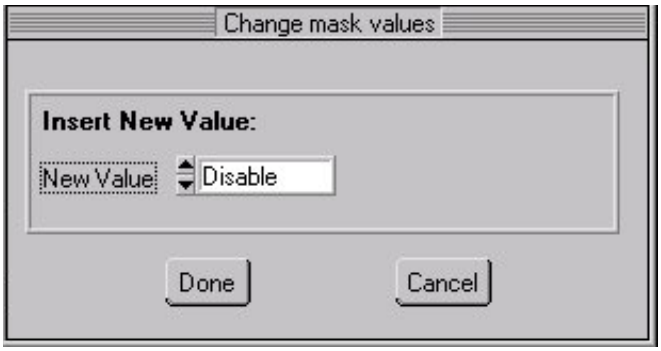


Fig. 5:

- *Test bit mask*: Permette di abilitare (1) o disabilitare (0) l'ingresso di test del preamplificatore. I passi effettuati sono quasi identici a quelli precedenti.

- *Threshold mask*:E' possibile inserire un valore compreso tra 0 e 7 che permette un aggiustamento della soglia di n volte Vtha, la soglia minima comune a tutti i pixel.

- ***Show Current Mask****:* Essa mostra una matrice i cui elementi sono i valori dei bit di configurazione presenti nelle maschere di enable, testbit e threshold.

- *Enable mask*: Cliccando su questo tasto appare un pannello come quello in fig.2 o come in fig. 4 in cui è possibile scegliere tra un singolo pixel o un gruppo di pixel. Cliccando su *Ok* appare il pannello di fig.6 che riporta il valore del bit di enable del gruppo di pixel selezionato.

Fig. 6:

- *Test bit mask*: Cliccando su questo tasto appare un pannello come quello in fig.2 o come in fig. 4 (singolo pixel o gruppo di pixel). Cliccando su *Ok* appare il pannello di fig. 6 che riporta il valore del bit di test del gruppo di pixel selezionato.

- *Threshold mask*: Cliccando su questo tasto appare un pannello come quello in fig. 2 o come in fig. 4 (singolo pixel o gruppo di pixel). Cliccando su *Ok* appare il pannello di fig. 6 che riporta il valore di threshold del gruppo di pixel selezionato.

- ***Display Matrixc:*** Questa funzione permette una visualizzazione grafica del contenuto delle matrici riportate di seguito.

- *Enable mask*: Scegliendo enable mask appare un pannello (fig. 7) riportante una matrice grafica di 64 x 64 pixel, i cui pixel possono assumere sono un valore tra due (0,1). A seconda del valore il pixel sarà di colore bianco o nero.

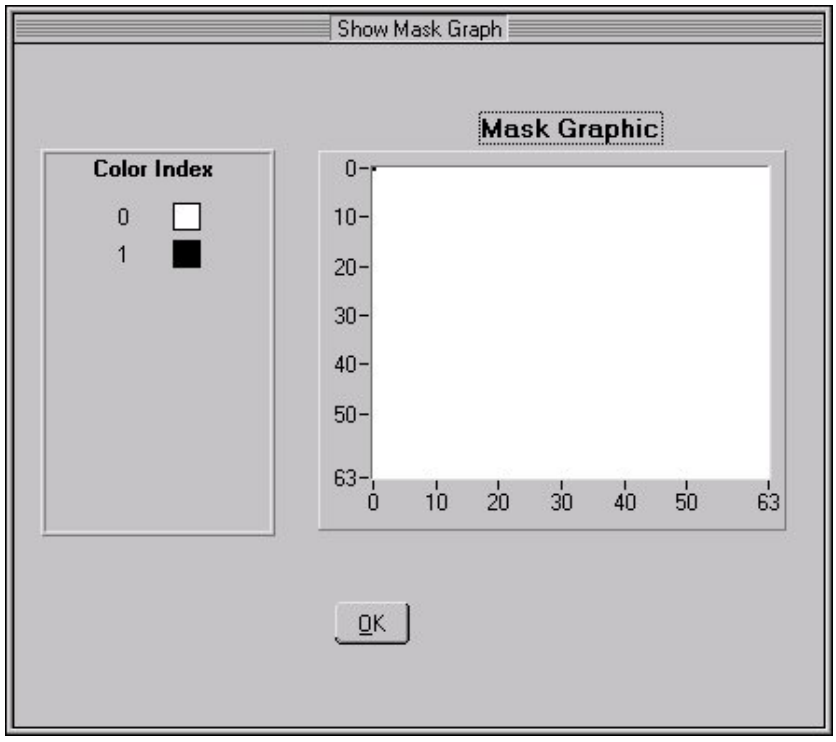


Fig: 7:

- *Test bit mask*: Scegliendo test bit mask appare un pannello (fig. 7) riportante una matrice grafica di 64 x 64 pixel, i cui pixel possono assumere sono un valore tra due (0,1). A seconda del valore il pixel risulterà di colore bianco o nero.

- *Threshold mask*:Scegliendo, invece, threshold mask appare un pannello come quello in fig. 7 ma questa volta i valori che può assumere il pixel sono 8 e quindi per la visualizzazione si usa una scala graduata di grigi. (Vedi fig. 8)

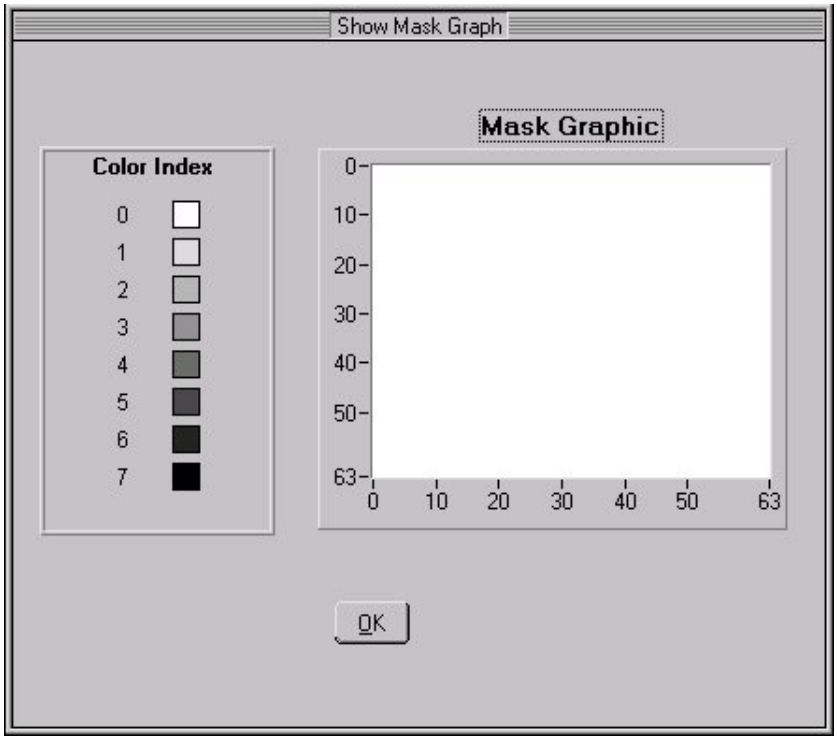


Fig. 8:

Basic Test

Menù che permette di eseguire le funzioni di test di base del sistema. Esso permette di accedere direttamente ai registri (descritti a pag.8) della scheda VME per testarli. I test possibili sono diversi e sono quelli riportati di seguito. Il riquadro al centro del pannello guida l'utilizzatore nei passi da effettuare.

- ***Register Access***: Premendo questo tasto appare il pannello riportato in fig. 9. Attraverso questo

  pannello è possibile modificare (accendere o spegnere un gruppo di pixel) o leggere/scrivere il contenuto di un qualsiasi registro.

  Bit: Questo tasto permette di testare il contenuto di un singolo bit nel registro

  Word: Questa tasto permette di testare il contenuto dell'intero registro

  Select register: Permette di selezionare il registro da testare. La scelta è tra:

  Tau 0-2 register
  Status register
  Analog register
  Chip select register:
  Cycles register
  Ram offset register

  select bit: Se si sceglie la modalità bit, in questo riquadro è possibile selezionare quale bit del registro indicato nel riquadro *Select register* si vuole testare.

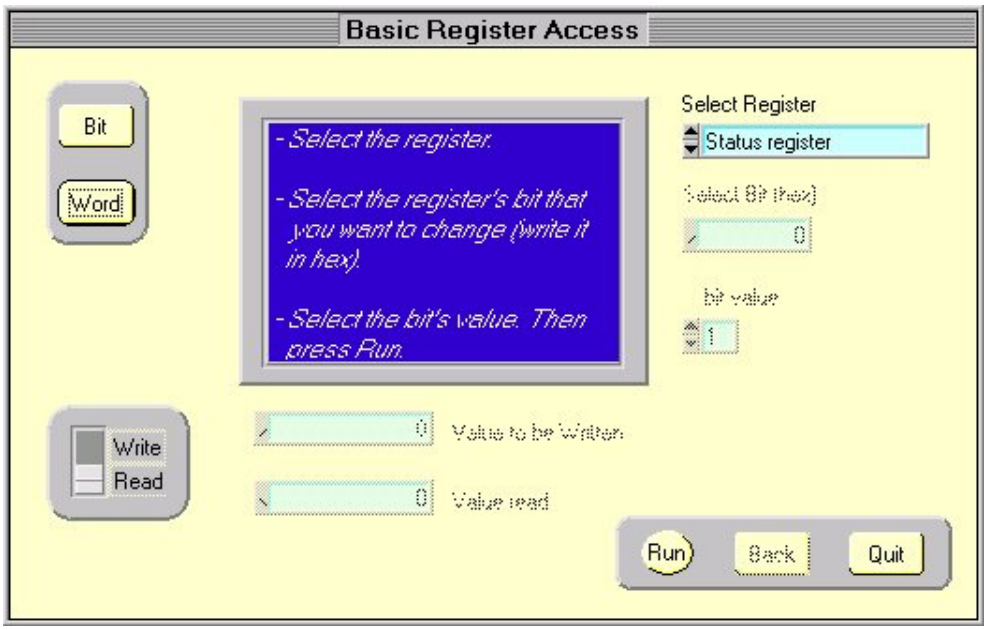


Fig. 9:

bit value: Permette di scegliere il valore da assegnare al bit selezionato

Tasto Write/Read: Questo selettore permette la scelta tra la modalità di test in scrittura o in lettura

Value to be Written: Inserire il valore che deve essere scritto nel registro

Value read: Mostra il valore letto nel registro, dopo aver fatto eseguire il programma.

Run: Cliccando su run si da avvio al test

Back: Permette di bloccare il test e di rimpostare i valori senza uscire dal pannello

Quit: Cliccando questo tasto si ritorna al pannello principale

- ***RAM or FIFO test***: Permette di eseguire un test sui registri RAM e FIFO (fig. 10). Esso consiste nell'inserire un indirizzo di partenza o un numero di dati da scrivere e andare a controllare se essi sono scritti correttamente nel registro.

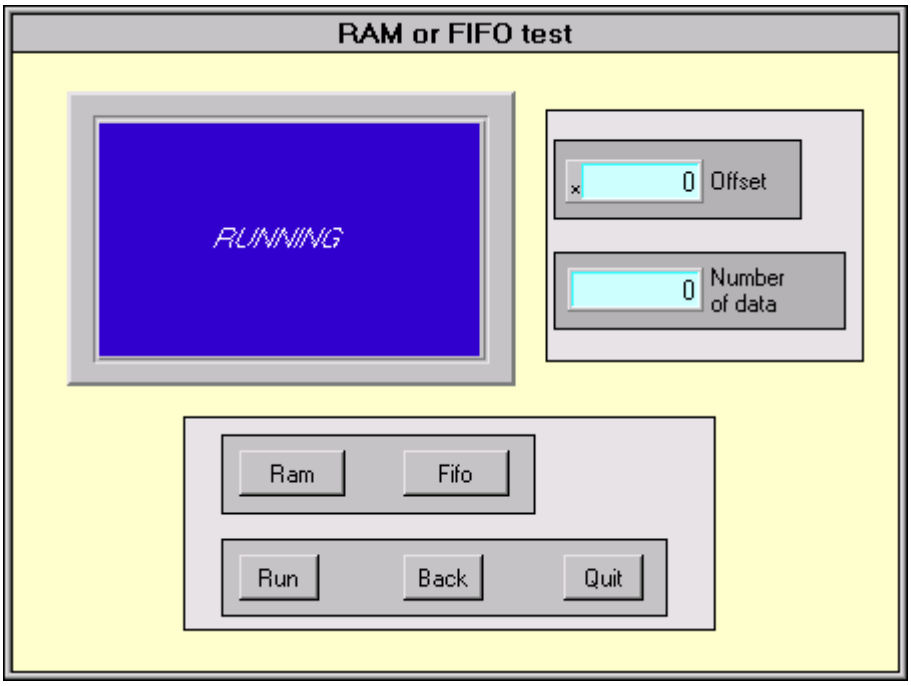


Fig. 10:

Offset: Inserire il numero del bit da cui incominciare a scrivere.

Number of data: Numero di dati da inserire.

Ram: Cliccando questo tasto si seleziona la Ram

Fifo: Cliccando questo tasto si selezione la Fifo

Run: Cliccando su Run si da avvio al test

Back: Permette di bloccare il test e di rimpostare i valori senza uscire dal pannello

Quit: Cliccando questo tasto si ritorna al pannello principale.

- ***Counters test***: Questo test permette di verificare il funzionamento dei contatori.

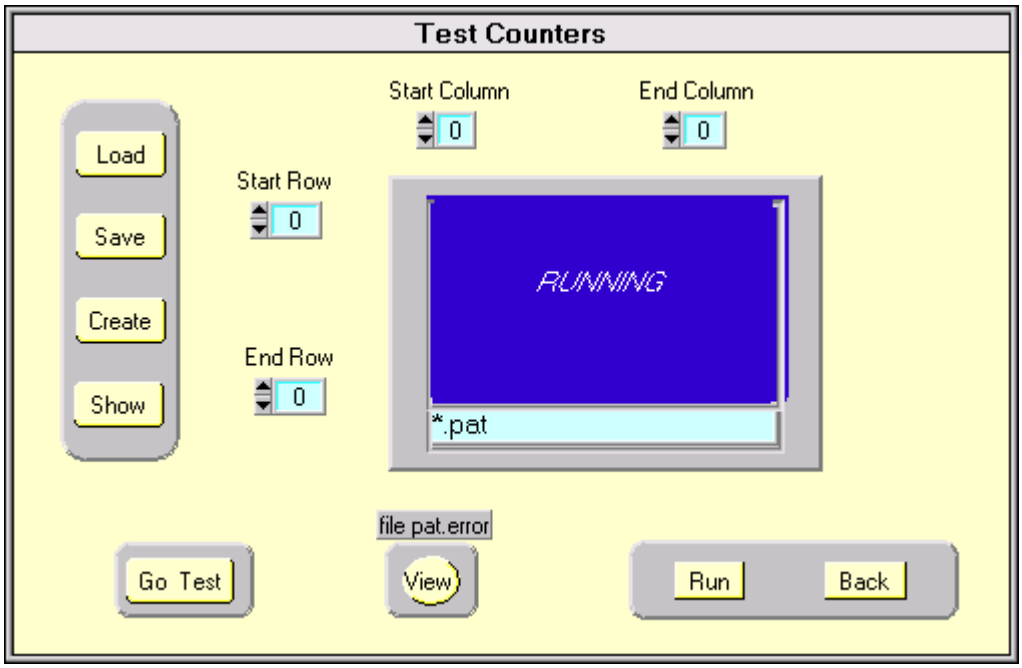


Fig. 11:

Load: Cliccare qui per inserire il nome del file di pattern (.pat) da caricare su Medipix.

Save: Cliccare qui per salvare il file di pattern creato.

Create: Cliccare qui per creare il pattern da caricare su Medipix.

Show: Con questo tasto è possibile visualizzare la matrice numerica caricata. Sono mostrati cinque numeri digitali, ogni numero in riferimento ad un contatore.

Go Test: Da avvio al test dei contatori.

Filepat.error (View): Premendo questo tasto è possibile visualizzare il risultato del test.

Run: Cliccando qui si carica il pattern su Medipix.

Back: Permette di bloccare il test e di rimpostare i valori senza uscire dal pannello

Quit: Cliccando questo tasto si ritorna al pannello principale.

- ***Mask Test***: Prima di caricare una maschera su Medipix è usuale fare un test sulla

maschera per essere sicuri che la matrice numerica mandata a Medipix corrisponda a qualla letta da Medipix stesso. Scegliendo masck test dal menu appare il pannello riportato in fig. 12.

Fig. 12:

Enable: Cliccando si visualizza la maschera di enable caricata su Medipix per il test.

TestBit: Cliccando si visualizza la maschera di testbit caricata su Medipix per il test.

Threshold: Cliccando si visualizza la maschera di threshold caricata su Medipix per il test.

File mask.error (View): Cliccando su questo tasto la matrice numerica caricata e qualla letta sono visualizzate.

OK:

Run: Cliccando questo tasto si da avvio al test. La maschera caricata e la maschera letta vengono comparate.

Back: Permette di bloccare il test e di rimpostare i valori senza uscire dal pannello

Quit: Cliccando questo tasto si ritorna al pannello principale.

- ***Noise & Enable test***: Cliccando su questo tasto appare un pannello come quello in fig.13 dove è possibile effettuare una scelta tra due diversi tipi di test: *turn off bad pixels, Best Vth Min.*

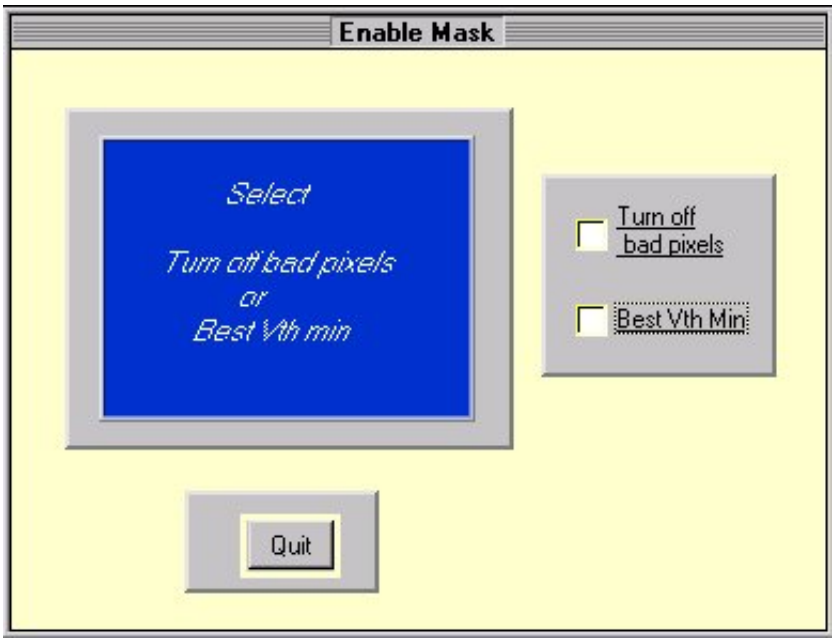


Fig. 13:

Scegliendo la modalità *Turn off* appare un pannello come quello in fig.14. In queso pannello è possibile inserire i valori delle tensioni di lavoro (Vbias, Vcomp, Vdl, Vtha), il tempo di acquisizione (in secondi) e il massimo numero di conteggi di rumore permesso. Cliccando su *Run* il sistema procede ad una acquisizione di rumore per il tempo indicato e alla fine di questa acquisizione spegne automaticamente tutti i pixel che hanno contato più di quanto indicato. Il sistema crea un file di maschera (enable) il cui nome è scelto, prima di cliccare *Run*, dall'utente nel riquadro *filename output*.

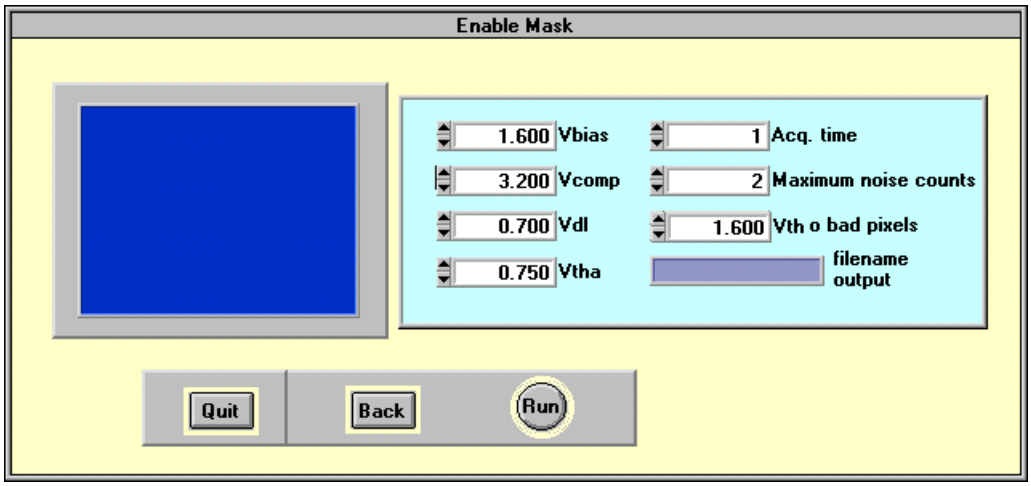


Fig. 14:

Scegliendo il modalità *Best Vth min*, il sistema richiederà le stesse informazioni ma come risultato fornirà il valore più basso possibile della tensione di soglia compatibile con le richieste effettuate.

- ***Threshold Calibration***: Cliccando su questo tasto appare una finestra (fig. 15) che chiede di effettuare una scelta tra *Vth Measurement* e *Calibration*.

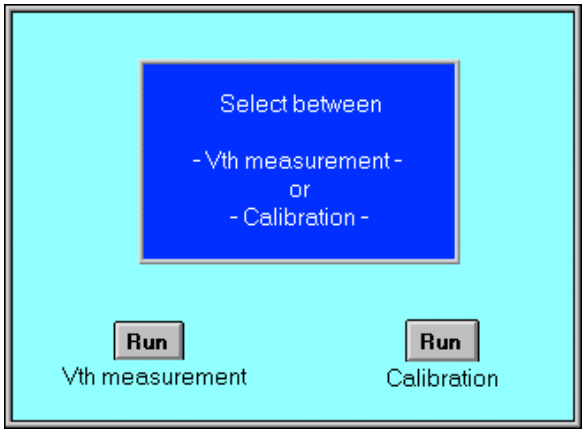


Fig. 15:

- *Vth Measurement*: Cliccando sul tasto *Run* di *Vth Measurement* diamo il via al processo del calcolo di Vth (la soglia minima) (fig. 16).
  La prima richiesta è il nome (senza estensione) del file che vogliamo creare, poi il nome della maschera da caricare con

estensione .msk (generalmente *default.msk*). Di seguito appare una finestra a pannelli a riempimento successivo.

Il primo pannello chiede:

Aquisition Time: Durata dell'acquisizione

Bad Pixel fraction: Frazione di pixel "rumorosi" richiesta

Maximum noise counts: Il numero di conteggi oltre il quale il pixel è considerato "rumoroso".

N. of different Vbias: Numero di valori di Vbias per cui effettuare il test.

N. of different Vcomp: Numero di valori di Vcomp per cui effettuare il test.

OK: Una volta inserite tutte le informazioni cliccare su OK per accedere al riquadro successivo.

Nella finestra centrale appaiono di seguito le rispettive domande:

Insert #1 Vbias:……..InvioValori per
Insert #1 Vcomp:……...Invio cui effettuare il test

A questo punto si attiva il tasto *Run* che da il via al procedimento di calcolo della Vth. Cliccando su *Run* si attiva il tasto *Visibile* che permette di seguire il procedimento, e conoscere il valore della Vthmin (*End of Vth min calculation*), il tasto *Hidden* e il tasto *Stop* per bloccare l'acquisizione.Il risultato di questa procedura è il valore di Vth che si deve poi utilizzare nel processo di calibrazione.

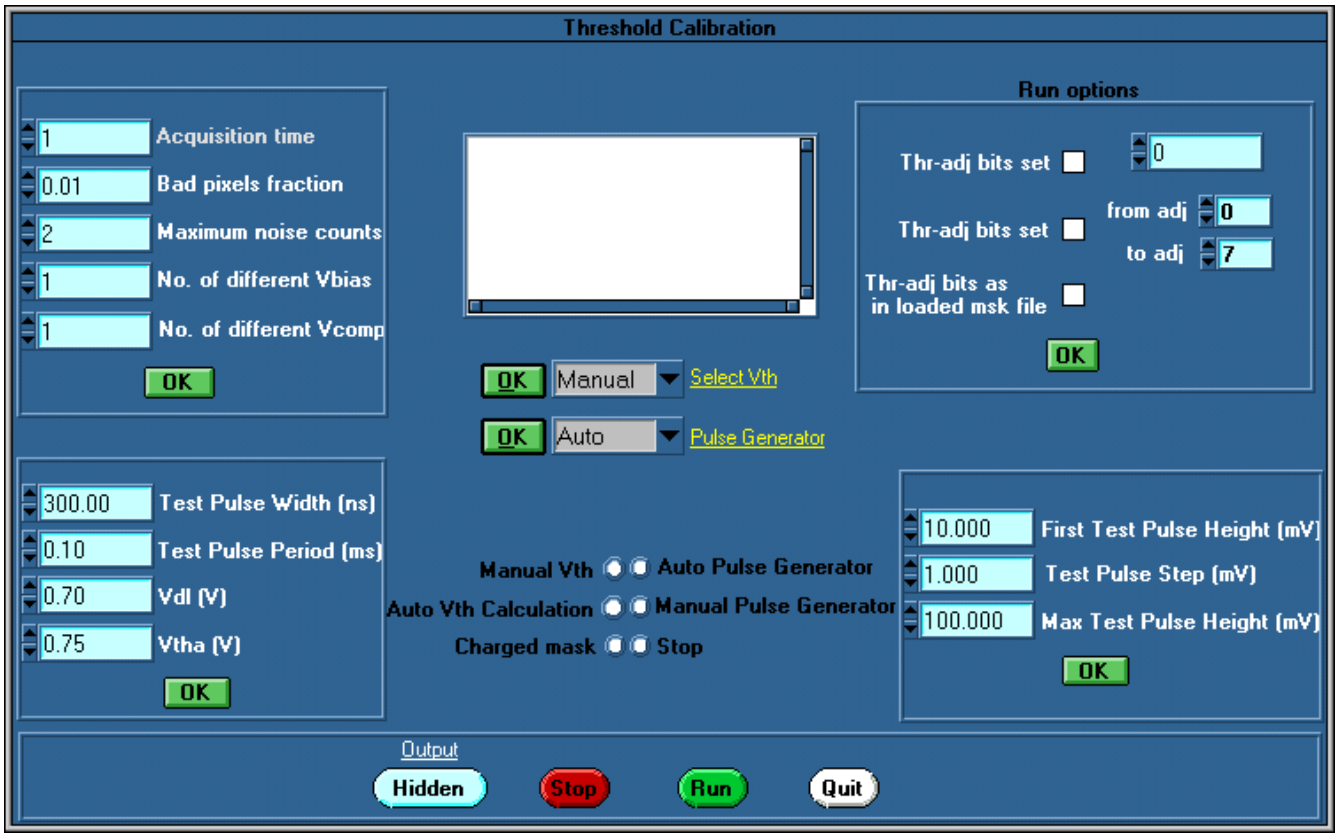


Fig. 16:

- *Calibration*: Cliccando sul tasto *Run* di *Calibration* diamo il via al processo di calibrazione del sistema. Esso consiste nell'inviare ad ogni pixel 1000 impulsi di periodo e durata fissa ma con ampiezza variabile e contare come varia il numero di pixel che contano 1000 al variare della soglia (da Vthmin a Vthmin+Vtha).
  Appare una finestra (fig.16) che chiede il nome del file da creare (senza estensione). La seconda finestra ci chiede il nome del file della maschera da caricare (con l'estensione .msk) (generalmente *default.msk*). Accediamo quindi alla finestra a riempimento successivo. La prima parte da riempire è la *Run Option* che fornisce tre diverse scelte:
  - *Thr-adj bits set*: Questa scelta permette di effettuare un solo processo in cui il valore della Vth viene corretto di

tante volte (1/7 di Vtha) quante indicate nel riquadro successivo.

- *Thr-adj bits set*: Permette di effettuare tanti processi con tanti passi di adj quanti indicati dai due riquadri di partenza e di fine.
- *Thr-adj bits as in loaded msk file*: Questa opzione vale nel caso si abbia già a disposizione un file di maschera.

Fatta la scelta e settati i valori di adj compare *OK*. Cliccato appare *Select Vth* che permette la scelta tra:

- *Manual*: Inserimento della Vth a mano (Calcolata in precedenza)
- *Auto*: Ricerca autonomatica

Cliccato *OK* appare il settore in alto a sinistra che chiede di settare:

Aquisition Time: Durata dell'acquisizione

N. of different Vbias: Numero di Vbias per cui effettuare il test.

N. of different Vcomp: Numero di Vcomp per cui effettuare il test.

OK: Una volta inserite tutte le informazioni cliccare su OK per accedere al riquadro successivo.

Nella finestra centrale appaiono di seguito le rispettive domande:

Insert #1 Vbias:……..Invio
Insert #1 Vcomp:……...Invio
Insert #1 Vth Invio

Valori per cui effettuare il test

Nel riquadro in basso a sinistra, si chiede di impostare il valori per la larghezza dell'impulso, per il periodo e delle tensioni di Vdl e Vtha.

Di seguito viene la scelta tra l'utilizzo dell'impulsatore in modalità automatica (l'impulsatore è direttamente gestito dal software) o in modalità manuale.

Ancora nell'ultimo riquadro sono richiesti i valori in mV dell'ampiezza dell'impulso da cui partire e terminare e lo step da utilizzare.

A questo punto si attiva il tasto Run e il sistema è pronto a procedere nel suo test.

Cliccando su *Run* si attiva il tasto *Visibile* che permette di seguire il procedimento, il tasto *Hidden* (che invece nasconde il procedimento) e il tasto *Stop* per bloccare l'acquisizione.

- ***Threshold Analysis***: Ci permette di analizzare i file realizzati dal processo di calibrazione (fig. 17).

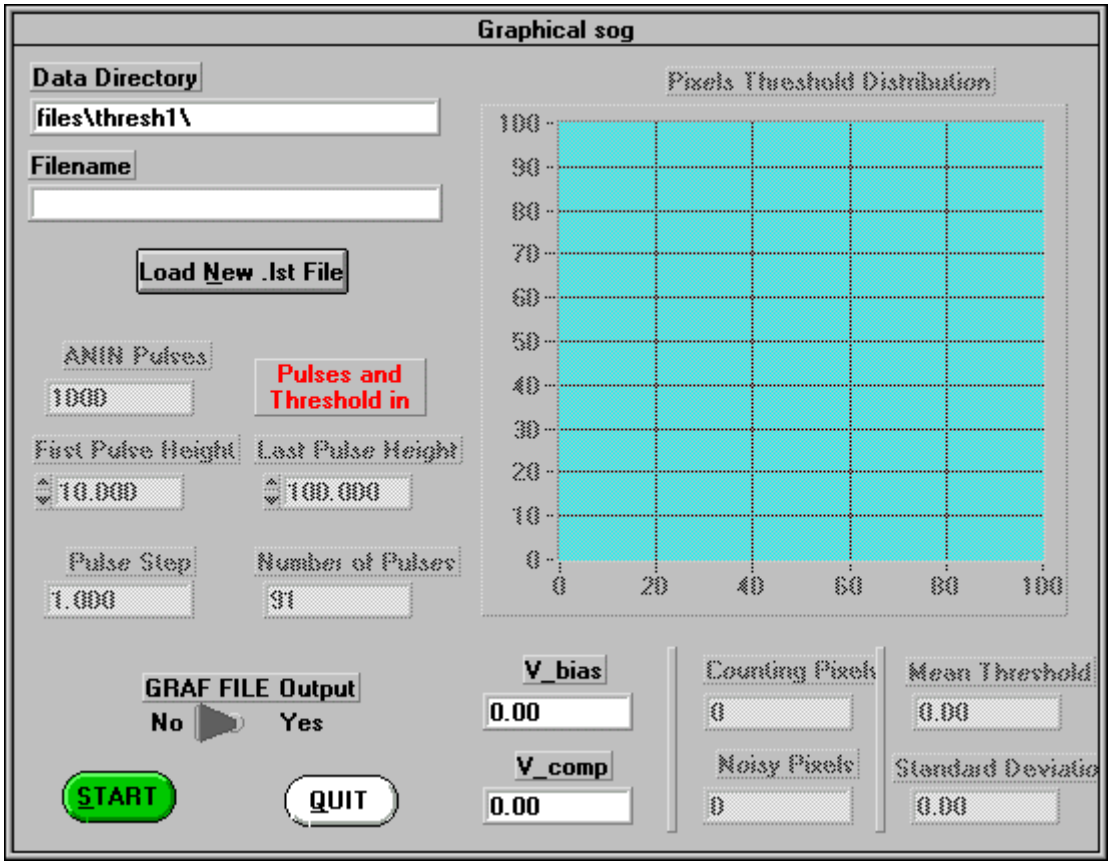


Fig.17:

Una volta scelto uno degli 8 file generati, ciccando su start appare un grafico che rappresenta il numero di pixel che contano 1000 rispetto al valore dell'ampiezza dell'impulso.L'area della gaussiana deve fornire 4096. Questa operazione deve essere effettuata per ognuno degli 8 file di calibrazione.

- ***Threshold Equilization***: Questo processo permette di realizzare la maschera di equalizzazione del chip. Appare una finestra di dialogo che chiede di seguito:
  I valori di Vcomp, Vbias, Vth (calcolato attraverso il processo di *noise & enable mask*)
  Il valore della distribuzione centrale, calcolata come la media aritmetica tra il massimo e il minimo valore delle soglie calcolate con il threshold analisis.
  Il nome del file di maschera di riferimento (.msk)
  Il nome del nuovo file di maschera (.msk)
  Il suo prefisso

Image:

- ***Show Image***: Vedi descrizione show image a pag.
- ***Add Subtract Files***: Permette di modificare il contenuto di un'immagine (fig. 18). In particolare, una nuova immagine (Target file) può essere creata dalla combinazione di altre due (target file = Factor I*File I ± Factor II*File II). L'operazione avviene pixel per pixel.

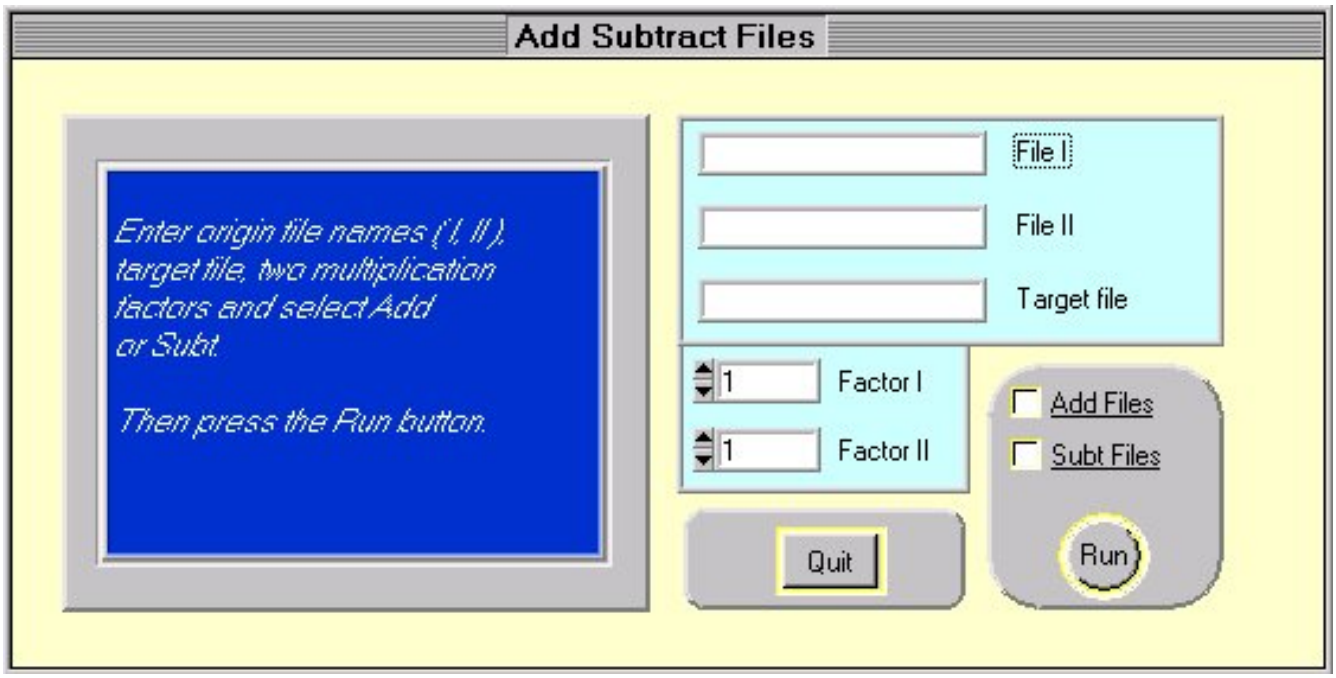


Fig. 18:

File I: Inserire il nome del file della prima immagine

File II: Inserire il nome del file della prima immagine

target file: Inserire il nome del file da creare

Factor I: Inserire il fattore moltiplicativo per la prima immagine

Factor II: Inserire il fattore moltiplicativo per la seconda immagine

Add/sub files: Permette di scegliere se sommare o sottrarre

Run: Cliccando su questo tasto si da avvio all'operazione.

Quit: Questo tasto permette di uscire dal pannello e ritornare al pannello Principale (fig.18).

- ***Pixel Response Calibration***: Permette di ottenere un file di dati normalizzati rispetto ad una acquisizione di fondo. Infatti questo calcola il rapporto tra il conteggio in ogni pixel delle due matrici. Se esiste un pixel della matrice di dati che ha un alto numero di conteggio, esso sarà mediato con quelli che gli sono accanto (fig. 18).

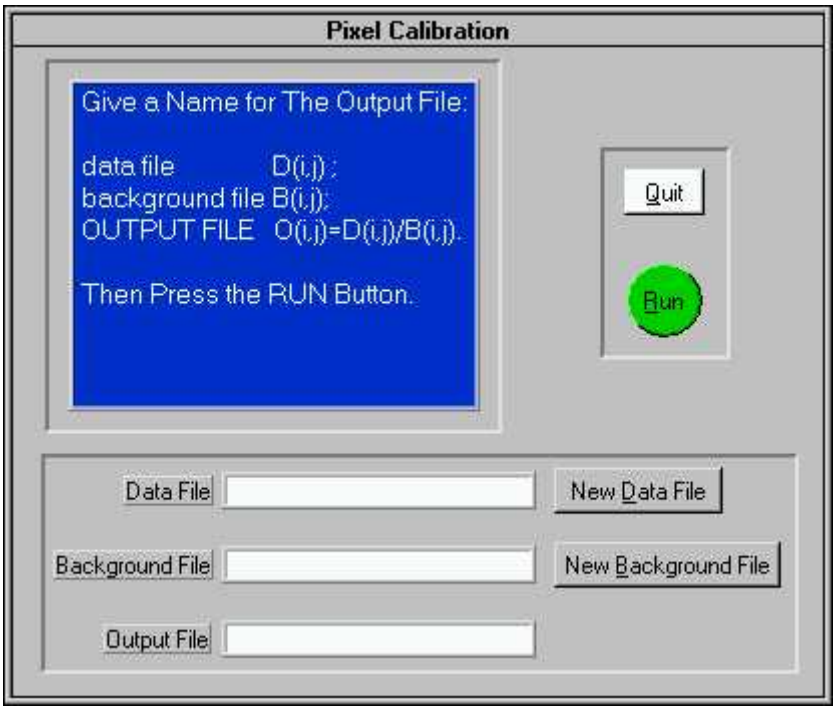


Fig.19:

Data file: Inserire il nome del file di dati da normalizzare. E' possibile sceglierne uno cliccando su *New Data File.*

Background File: Inserire il nome del file di dati da usare come peso. E' possibile sceglierne uno cliccando su *New Background File.*

Output File: Inserire il nome del file di dati generato dalla calibrabrazione.

New Data File: Cliccando su questo tasto si ha la possibilità di ricercare il file di dati da inserire in *Data File.*

New Background File: Cliccando su questo tasto si ha la possibilità di ricercare il file di dati di background da inserire in *Background File.*

Run: Cliccando su questo tasto si da avvio all'operazione.

Quit: Cliccando su questo tasto si esce dal pannello e si ritorna al pannello principale.

- *Low Passs Filter*: Quando si clicca su questo tasto appare una finestra che chiede il

nome del file immagine su cui applicare il filtro. Il nuovo file creato in Medipix\files i cui pixel sono rimpiazzati dalla media dei pixel intorno.

User's Menù:

- *User's basic test*: Permette l'inserimento di un programma in C per qualsiasi operazione, in modo da implementare una qualsiasi funzione in Medisoft.

? : *About Medisoft*: Informazioni su Medisoft. Versione, data rilascio,….

Descrizione dell'area relativa all'impostazione dei parametri

Acquisition Parameter: Questi sono i parametri che gestiscono i tempi e le modalità di tutte le acquisizioni sia che esse siano dei test o no.

- *Tau0*: Se nel sistema è prevista la presenza di qualche strumentazione, come un tubo a raggi X, la cui accensione può essere gestita dal sistema stesso (*gate*) questo tempo gestisce il ritardo tra l'apertura di questa strumentazione e l'inizio dell'acquisizione. Il valore di default è 0
- *Tau2*: Questo gestisce il ritardo tra le fine dell'acquisizione e la chiusura del gate. Il valore di default è 0
- *Acquisition Time*: Questo è il tempo della durata della singola acquisizione in modalità radiography. Attenzione per l'autoradiography. Il valore di default è 0
- *Trigger mode*: La scelta è tra tre possibili modalità; ANIN: se si vuole utilizzare test di impulsi analogici triggerati via software; INT: se si vuole utilizzare un gate interno generato via software; EXT: se si utilizza un trigger esterno. Il valore di default è INT

- *Show active*: Cliccando su questo tasto appare una finestra (fig.20) che mostra i parametri relativi a questa finestra attivi nell'ultima acquisizione.

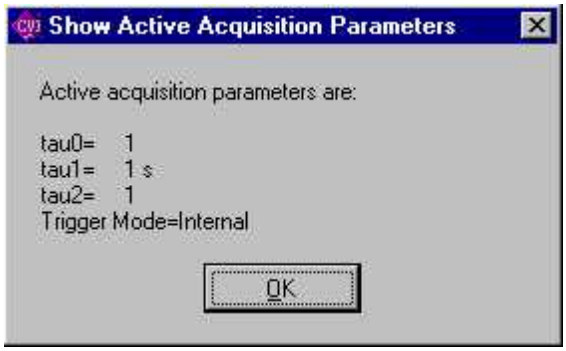


Fig. 20:

VME board Setting:

- *Chip ID*: In questa casella viene indicato il numero del chipselect attivo. Il valore di default è 0
- *Cycles*: Questo è il numero di cicli necessario per caricare/scaricare i bit di configurazione o di dati dal e verso il chip.
- *Config Mode*: Indica il modo con cui avviene la scrittura dei bit di configurazione sul chip: in modo diretto (DIRECT) o attraverso la VME RAM (INDIRECT).
- *RAM offset*: In modalità indiretta deve qui essere indicato l'indirizzo della VME RAM
- *Show active*: Cliccando su questo tasto appare una finestra (fig. 21) che mostra i parametri relativi a questa finestra attivi nell'ultima acquisizione.

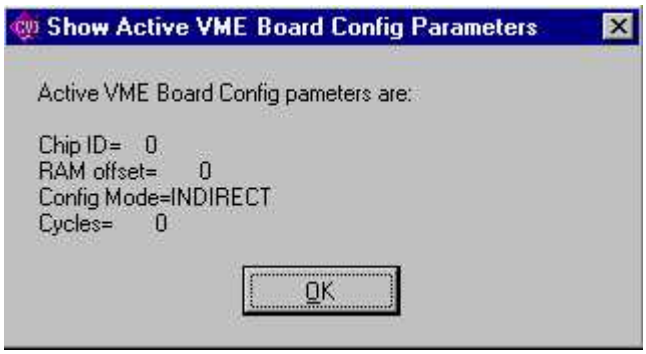


Fig. 21:

ANIN Test Parameter:

- *ANIN Cycles*: Questo indica il numero di impulsi inviati a ogni pixel in caso di ANIN test
- *ANIN Delay*: Indica il ritardo tra l'apertura del gate e l'impulso di test
- *ANIN Period*: Indica il periodo del segnale di test. In unità arbitrarie
- *Show active*: Cliccando su questo tasto appare una finestra (fig. 22) che mostra i parametri relativi a questa finestra attivi nell'ultima acquisizione.

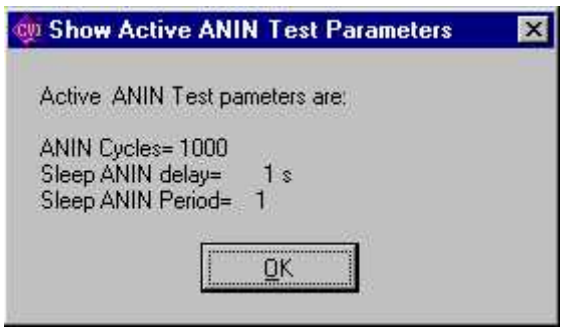


Fig. 22:

- *Test*: Cliccando questo tasto si da inizio al test ANIN con i parametri mostrati nel riquadro. Durante il test il led acquisition on è acceso. Il generatore di impulsi esterno non è programmato remotamente.

DAC Parameter:

- *Load DAC*: Cliccando su questo si rendono operativi i valori dei DAC dopo che questi sono stati modificati in *Modify Bias*. Se questo non viene fatto i valori restano sempre quelli settati in precedenza.

- *Modify Bias*: Cliccando su tale tasto appare una finestra (fig. 23) in cui possono essere modificati i valori dei parametri DAC per entrambi i tipi di Chip (0,1). Essi sono indicati in Volt.

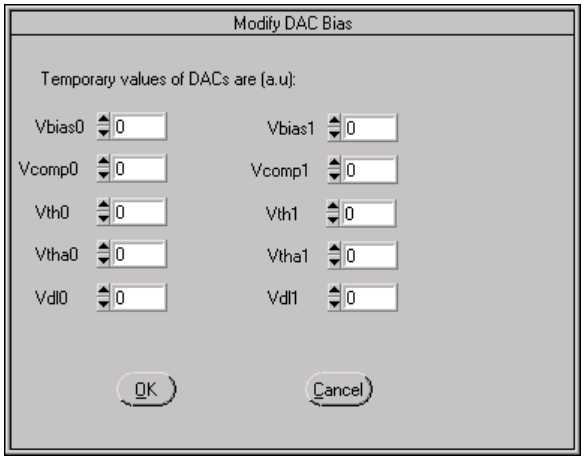


Fig. 23:

- *Tasto V/au*: Questo selettore serve per selezionare le unità di misura dei DAC
- *Show active*: Cliccando su questo tasto appare una finestra (fig. 24) che mostra i parametri relativi a questa finestra attivi nell'ultima acquisizione relativi al Chip indicato in *Chip ID*.

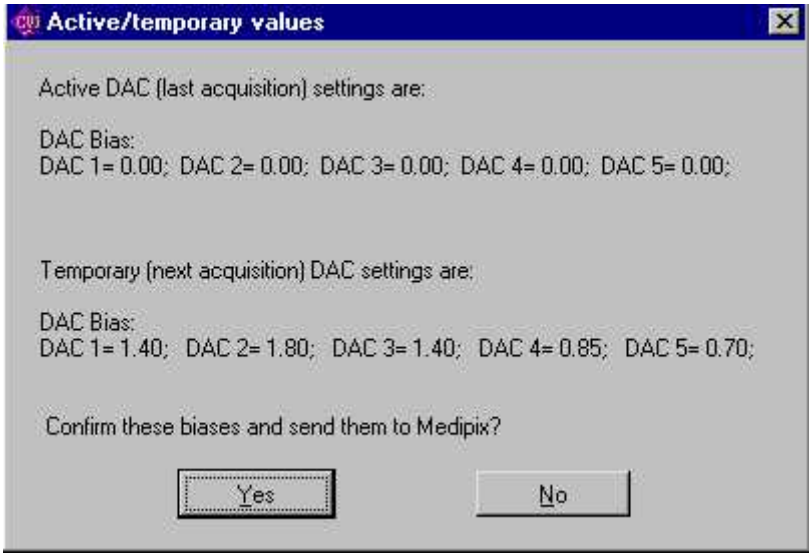


Fig. 24:

Descrizione dell'area relativa ai comandi da impartire al sistema

Command Bottons:

- *Radiography*: Cliccando questo tasto si da inizio ad una acquisizione con i parametri impostati nei precedenti riquadri. Si accende il led *Acquisition On*. Quando l'acquisizione termina ed è pronta l'immagine si accende il led *Image ready*.
- *Show image*: Quando l'immagine è pronta (*Imagin ready* acceso) cliccando su tale tasto appare una finestra (fig. 25) in cui un riquadro mostra l'immagine.

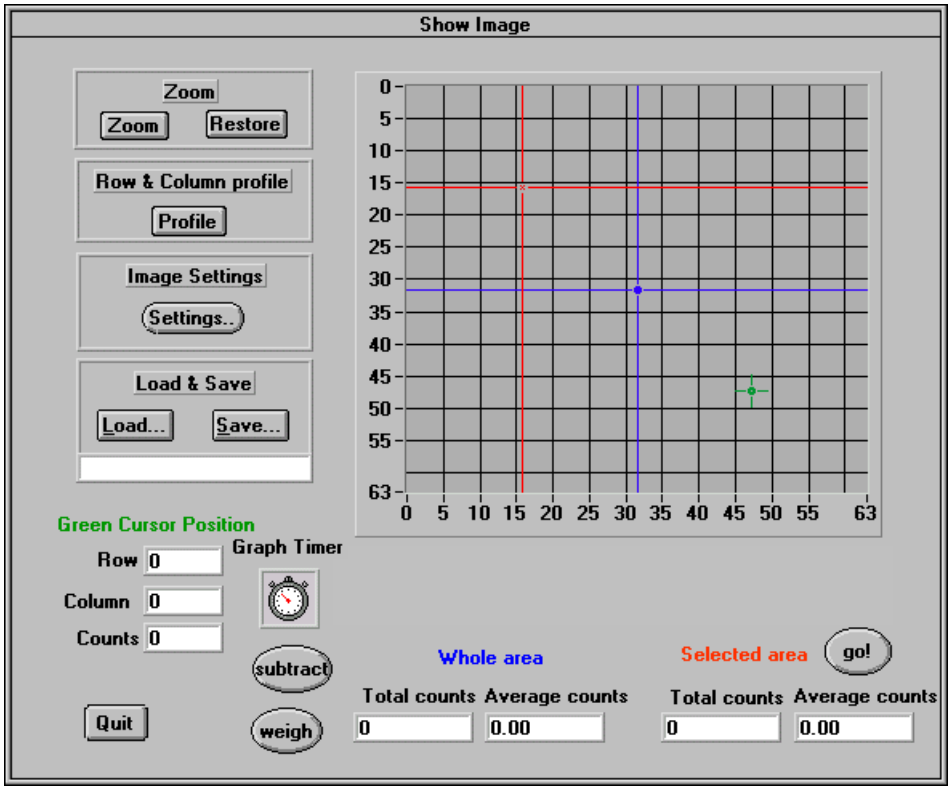


Fig. 25:

Quando non è acceso *Imagin Ready* cliccando su tale tasto viene mostrata l'ultima acquisizione effettuta. In tale riquadro vi sono tre cursori:

Verde: è legato al *Green Cursor Position* che ci identifica il pixel (*Row/Colum*) e il numero di conteggi in tale pixel (*Counts*).

Blu e *Rosso*: Entrambi servono per selezionare un'area su cui voglio effettuare un conteggio.

Altri riquadri sono:

Zoom: Fa uno zoom della zona delimitata dai cursori blu e rosso.

Restore: Riporta l'immagine alla dimensione normale.

Row & Colum Profile: *Profile*: Premendo tale bottone si passa la pannello di visualizzaizone del profilo.

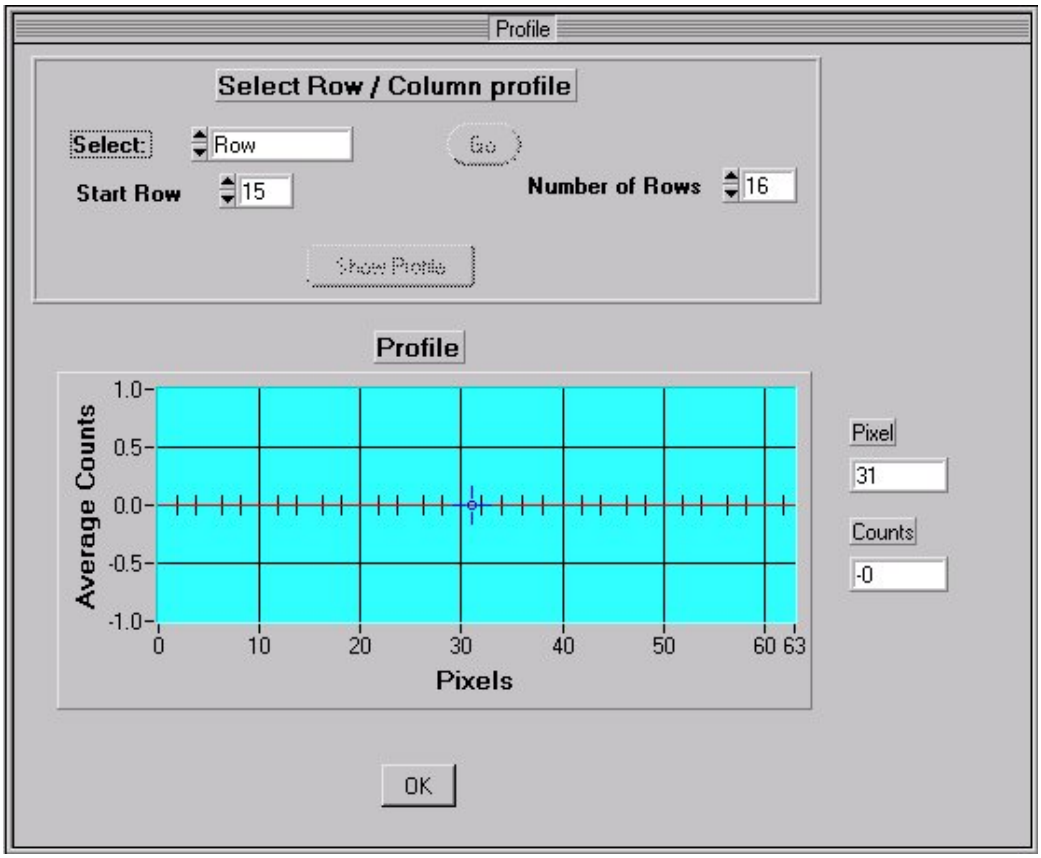


Fig. 26:

Il profilo è un grafico riportante i valori di conteggio su di una determinata riga o colonna. Questa funzione può essere utile al fine di

verificare un guasto di uno o più pixel su di una riga o una colonna (valore di conteggio molto più alto rispetto alla media). Nel caso il profilo è eseguito su più righe o colonne viene visualizzata la media dei valori di colonna o di riga.

Image settings: *Settings*: Premendo tale bottone si passa al pannello (fig. 27) per l'impostazione dei parametri dell'immagine. In questo pannello è possibile impostare:

Fig. 27.

Set Height: Il colore corrispondente al valore del conteggio più basso e quello corrispondente al valore di conteggio più alto.

Resolution: Il numero di livelli di colore compresi nell'intervallo specificato.

La possibilità di effettuare uno smoothing tra i colori di pixel adiacenti

La possibilità di effettuare uno smoothing tra i differenti conteggi di pixel adiacenti

Min e Max: I valori di conteggio minimo e massimo al fine di ridurre l'intervallo di valori da visualizzare e quindi di aumentare il numero di punti visualizzati.

Load & Save: *Load*: Permette di andare a rivedere un'immagine salvata precedentemente.

Save: Permette di salvare un'immagine nella directory D:\Medipix\files. Una volta salvato appare il nome nel riquadro sottostante.

Subtract: Da la possibilità di sottrarre ad un'immagine un'altra.

Weigh: Da la possibilità di andare a pesare un'acquisizione con un'immagine ottenuta attraverso un irraggiamento uniforme. Questo ci permette di uniformare il comportamento dei vari pixel.

Go: Cliccando su tale tasto nel riquadro *Select area* appare il numero di conteggi integrale dell'area selezionata dal cursore rosso e blu.

Whole Area: Nel riquadro sottostante è mostrata il numero di conteggi totale sull'intera area

Select Area: Nel riquadro sottostante è mostrata il numero di conteggi totale dell'area selezionata dal cursore rosso e blu. Per mostrare il conteggio bisogna cliccare *Go*.

Quit: Per uscire e ritornate al pannello principale

- *Load Mask:* Cliccando su questo tasto si carica il file di maschera indicato nel riquadro *Mask file loaded* e il led On di *Mask Status* si accende.

Turn On/Off Medipix: Tasto di accensione di Medipix. Quando è scritto *Turn On* significa che il sistema è spento. Questo lo si può vedere anche dal led di *Medipix Status*. Quando si preme *Turn On* appare una schermata che informa sui parametri *DAC di Bias* e *temporary* attivi. Questa chiede di confermare la scelta. Serve per sapere quali sono i parametri da cui si parte.

Lo *status di Medipix* accende il led On.

Per spegnere il sistema basta cliccare su *turn off* e il led off di *Medipix Status* si accende.

Autoradiography: Essa è una modalità di acquisizione multipla. Premendo su questo tasto appare una finestra (fig. 28) con caselle a riempimento successivo. La prima casella chiede il tempo di acquisizione di ogni singolo slide (*Acquisition Time*) e il tempo di acquisizione totale (tempo che deve durare l'intera acquisizione) (*Total Time*). Premendo *OK* appare un ulteriore finestra con un bottone che chiede la modalità di visualizzazione (acquisizione) *slide/integral* (fig. 29). Premendo OK appare una finestra che chiede il nome con cui si intende salvare tali file (esso viene salvato nella directory D:/medipix/files) . Scritto il nome del file (senza estensione siccome i file generati ne sono molti) e premendo *OK* si torna alla finestra di partenza dove sono mostrati il numero di slide che saranno effettuati (*Number of slide*), il nome del file (*Image file*) e viene attivato il tasto *Start Acquisition*. Premendo tale tasto parte l'acquisizione (viene mostrato il procedere del tempo e del numero di slide) mostrando alla fine di ogni slide l'immagine acquisita. A seconda se si è scelti *slide* od *Integral* l'immagine mostrata è una singola acquisizione o è la somma di tutte le acquisizioni precedenti. Si accende il tasto di *Stop Acquisition* e di *Resume Acquisition* per fermare o far ripartire l'acquisizione. Durante l'acquisizione si accende il led *Acquision On*. Finita l'acquisizione per uscire premere *Exit* ritornando al pannello principale.

Il tasto show image è identico a quello già visto a pag.29.

La barra time mostra il trascorrere del tempo dell'acquisizione.

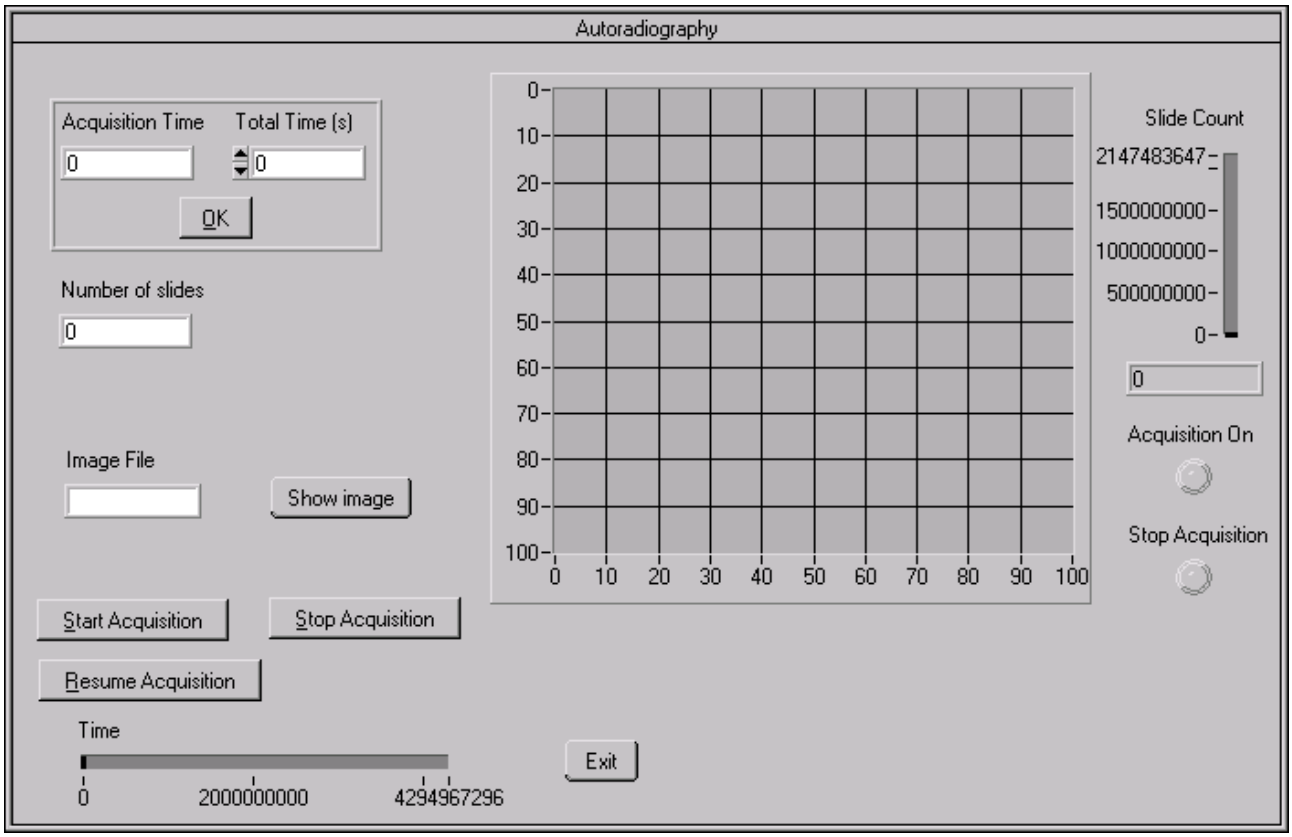

Autoradiography
Acquisition Time
Total Time (s)
0
0
OK
Number of slides
0
Image File
Show image
Start Acquisition
Stop Acquisition
Resume Acquisition
Time
0
2000000000
4294967296
Exit
0
10
20
30
40
50
60
70
80
90
100
Slide Count
2147483647
1500000000
1000000000
500000000
0
0
Acquisition On
Stop Acquisition

Fig. 28:

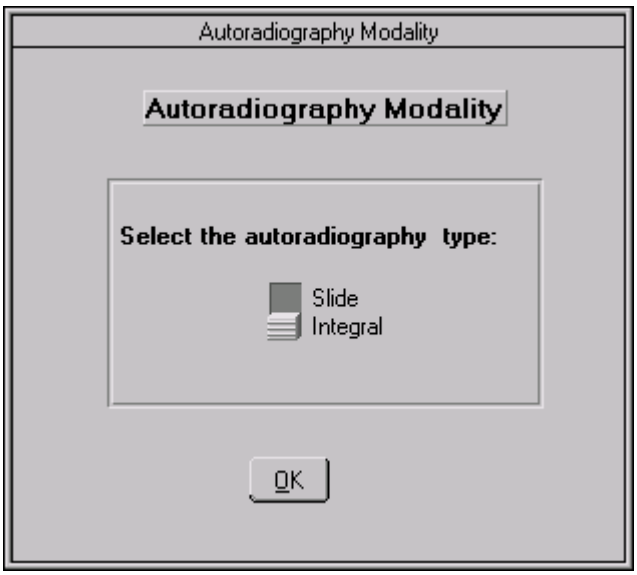

Autoradiography Modality
Autoradiography Modality
Select the autoradiography type:
Slide
Integral
OK

Fig. 29:

Motor: Questo tasto permette di controllare, quando presente, un meccanismo di movimentazione micrometrica. Quando si preme tale pulsante appare una finestra (fig. 30) con:

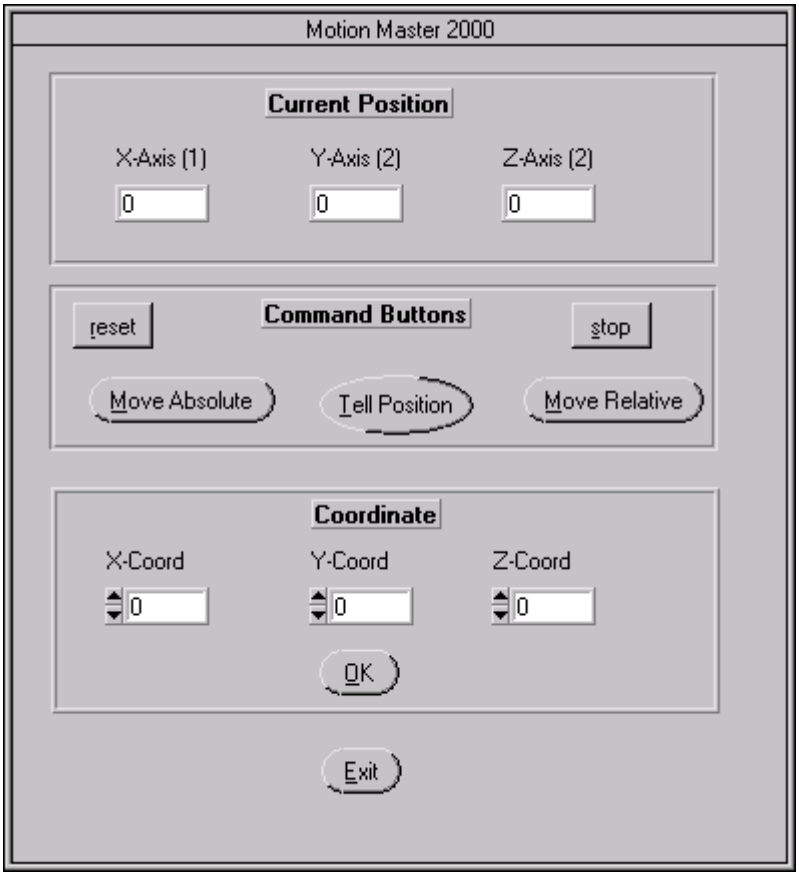


Fig. 30:

- *Current Position*: Mostra la posizione (x,y,z) corrente, in micron, del portacampione. Questa è mostrata ogni volta che si preme *Tell Position.*
- *Command Buttons*:
  - *Reset*: Porta il campione in una posizione fissata di partenza che è esterno all'aria di rivelazione. Tale posizione è (5000,30000,20000).
  - *Stop*: Permette in qualsiasi istante di bloccare il posizionamento del porta campione
  - *Tell position*: Cliccando su tale tasto appare la posizione occupata dal

portacampione in quell'istante

- *Move Absolute* e *Move relative*: Cliccando appare una finestra che permette di modificare le componenti x,y,z della posizione del portacampione o in modalità assoluta (*Absolute position*) o in modalità relativa (*Relative displacement*)
- *Exit*: Permette di uscire da questo pannello e di ritornare al pannello principale

NOTA: La rotazione del portacampione avviene manualmente ed esiste una posizione di riferimento che indica quando il portacampione è perpendicolare al rivelatore. Tale posizione è 330/60

Descrizione dell'area relativa alle informazioni di stato del sistema

Power Mode:

- *Selettore Internal/External:* Permette di scegliere tra l'alimentazione interna, i valori di tensione arrivano direttamente dalla VME board, o esternamente, i valori arrivano da alimentatori esterni attraverso la motherboard.
- *Archive file loaded*: Un file di archivio contiene tutti i valori specifici per tutti i parametri. Qui è mostrato quello caricato. (Default.arh)
- *Mask file loaded*: Qui è indicato il file di maschera. (Default.msk)

Medipix Status: Contiene due led che mostrano lo stato del sistema: ON per acceso (verde) OFF per spento (rosso)

- *Check power*: Cliccando si fa un test sull'elettronica per vedere se funziona. In caso di non funzionamento appare una finestra di errore.

Mask status: Contiene due led che danno informazioni sul fatto che sia o meno stata caricata una maschera sul sistema. Prima di partire con un'acquisizione e importante controllare questo riquadro.

Image ready: A seconda se questo led è acceso o spento ci informa se è pronta o no una immagine.

Acquisition On: A seconda se questo è acceso o no ci dice se il sistema è in fase di acquisizione o no.

ELENCO PUBBLICAZIONI

1. L.Abate, E.Bertolucci, M.Conti, G.Mettivier, M.C.Montesi, P.Russo, *"GaAs pixel arrays for beta imaging in medicine and biology"*. Presentato al 1^{sT} international Workshop on radiation imaging Detectors, Sundsvall, Svezia, Giugno 1999 e accettato per la pubblicazione su NIMA

2. L.Abate, E.Bertolucci, M.Conti, G.Mettivier, M.C.Montesi, P.Russo, *"Noise and interpixel dead space studies of GaAs pixellated detectors"*, presentato al 11th International Workshop on room temperature semiconductor X- and Gamma – ray detectors and associated electronics, Vienna, Austria, Ottobre 11-15, 1999 e accettato per la pubblicazione su NIMA

3. E.Bertolucci, M.Conti, A. Di Cosmo, C. Di Cristo, G.Mettivier, M.C. Montesi, P.Russo, *"Quantitative dynamic imaging of biological processes with solid state radiation detectors"*, presentato al Nuclear Scientific Symposium 1999, Seathle, USA e accettato per la pubblicazione su IEEE Trans. Nucl. Scien.

Medisoft 2.2
Code

Esp488.c

```
#include "function.h"
˝


˝
// Medisoft user note:
˝
// IBad, VME-GPIB Board VMEaddress, is defined in function_VME.c
˝
// using the macro definition set in medipix.h (Gpib_ad)
˝


˝
/* We changed (here and in pulser.c) the names of
ibcmd       ->    Ibcmd
ibwrt ->    Ibwrt
ibonl ->    Ibonl
ibsic ->    Ibsic
ibsre ->    Ibsre
ibrd  ->    Ibrd
ibwait  ->  Ibwait
ibrpp   ->  Ibrpp
ibgts   ->  Ibgts
ibcac   ->  Ibcac
ibpad   ->  Ibpad
ibtmo   ->  Ibtmo
in order to use at the same time also the GPIB NI libraries for GPIB-PCI board
*/

/*
 * Engineering Software Package GPIB-1014P functions
 * Rev B
 * (c) Copyright 1990, National Instruments
 * All rights reserved.
 *
 * The GPIB functions implemented here are a straightforward
 * set of independent functions that can be used to directly
 * manipulate an interface board.  They are at the level of
 * example code provided in the hardware reference manual but
 * structured to be a subset of the complete integrated driver
 * package.  Sufficient functionality is implemented so that
 * an application program can control and coordinate data
 * transfers among a suite of instruments on the GPIB.  The
 * primary purpose of these functions is to demonstrate the
 * use of the GPIB interface board in an ATE environment so
 * that an engineer or programmer is able to customize the
 * GPIB functions for a dedicated stand-alone or ROM-based
 * system, rather than implement the software from scratch.
 * Whenever an operating system environment is available the
 * recommended approach is to use a specific binary driver or
 * handler package containing the full GPIB capability rather
 * than use this engineering software package.
 *
```

```
 * The function calls support synchronous, non-interrupt,
 * non-DMA I/O, for a single interface board which is always
 * the system controller and the controller in charge.  All
 * functions return the subset of the standard GPIB status
 * bit vector consisting of END,SRQI,CIC,ATN,LACS and TACS status bits.
 * The result of the last call is also available in the global
 * variable, ibsta.  If it was an I/O operation the actual
 * count transferred is available in the global variable,
 * ibcnt.  Before any other call is made the ibonl function
 * must be called to initialize the GPIB interface.  Prior to
 * calling ibrd or ibwrt the appropriate devices, including
 * the interface board, must be addressed by calling ibcmd
 * with the proper addressing commands.
 */

#define IB      ((struct ibregs *) IBad)      /* short I/O address of */
                                          /* interface board      */

//extern unsigned ibsta;                /* status bits returned by last call */
//extern unsigned ibcnt;              /* actual byte count of last I/O transfer
*/

unsigned long int timeout = GPIB_TIMEOUT;    /* number of loops to count before
timing out */
                                                  /* default fffff */
char stat1, stat2;          /* software copies of isr1 and isr2; These bits */
                            /* are cleared when read, so each function must */
                            /* update a software copy of them for use in */
                            /* updating ibsta.  If the updated() function */
                            /* actually read these registers, it would */
                            /* clear some bits which might be needed by the */
                            /* next function to be executed. */


/* Hardware register definitions */

struct ibregs {             /*  write-only hardware registers       */
      char    x0,     cdor;   /*  +1 byte out register                */
      char    x2,     imr1;   /*  +3 interrupt mask register 1        */
      char    x4,     imr2;   /*  +5 interrupt mask register 2        */
      char    x6,     spmr;   /*  +7 serial poll mode register        */
      char    x8,     admr;   /*  +9 address mode register            */
      char    xA,     auxmr;  /*  +B auxiliary mode register          */
      char    xC,     adr;    /*  +D address register 0/1             */
      char    xE,     eosr;   /*  +F end of string register           */
};

/* Read-only register mnemonics corresponding to write-only registers */

#define dir     cdor
#define isr1    imr1
#define isr2    imr2
#define spsr    spmr
#define adsr    admr
#define cptr    auxmr
#define adr0    adr
#define adr1    eosr
```

```
/* Control masks for hidden registers (auxmr) */

#define ICR     0040
#define PPR     0140
#define AUXRA   0200
#define AUXRB   0240
#define AUXRE   0300

/* Hardware register bit definitions */
/*      Name            Bit(s)          register       */

#define HR_DI           (1<<0)          /* isr1         */
#define HR_DO           (1<<1)          /* isr1         */
#define HR_END          (1<<4)          /* isr1         */
#define HR_CO           (1<<3)          /* isr2         */
#define HR_SRQI         (1<<6)          /* isr2         */
#define HR_DMAI         (1<<4)          /* imr2         */
#define HR_DMAO         (1<<5)          /* imr2         */
#define HR_ADM0         (1<<0)          /* admr         */
#define HR_TRM0         (1<<4)          /* admr         */
#define HR_TRM1         (1<<5)          /* admr         */
#define HR_DL           (1<<5)          /* adr          */
#define HR_DT           (1<<6)          /* adr          */
#define HR_ARS          (1<<7)          /* adr          */
#define HR_PPU          (1<<4)          /* ppr          */

/* 7210 Auxiliary Commands */

#define AUX_PON         000     /* Immediate Execute pon        */
#define AUX_CR          002     /* Chip Reset                   */
#define AUX_SEOI        006     /* Send EOI                     */
#define AUX_TCA         021     /* Take Control Asynchronously  */
#define AUX_GTS         020     /* Go To Standby                */
#define AUX_EPP         035     /* Execute Parallel Poll        */
#define AUX_SIFC        036     /* Set IFC                      */
#define AUX_CIFC        026     /* Clear IFC                    */
#define AUX_SREN        037     /* Set REN                      */
#define AUX_CREN        027     /* Clear REN                    */


/* I/O macros; The In and Out functions have been defined as these two */
/* macros which pass the I/O address and data width to two functions */
/* called input and output.  This allows you to define your own method */
/* of accessing the registers, since some special steps may need to be */
/* taken if the GPIB-1014 is in another chassis linked by an extender */
/* or some other unusual setup. */

#define In(x)           input((long)(&(IB->x)), 1)
#define Out(x,a)        output((long)(&(IB->x)), (long)a, 1)


input(long address, int width)      /* reads a register on the GPIB-1014; */
                                        /* This input function is for a cpu */
                                      /* with the same addressing and byte */
                                    /* ordering scheme as the VMEbus. If */
   {                                /* you have an Intel processor or a */
```

```
   switch(width)                    /* processor which is incapable of */
      {                             /* 32-bit accesses, you may need to */
      case 1:                       /* write your own input routine. */
         return *(char *)address;
      case 2:
         return *(int *)address;
      case 4:
         return *(long *)address;
      }
   return(0);
   }


output(long address, long value, int width) /* writes a register on the GPIB-
1014 */
                                    /* This output function is for a cpu */
                                    /* with the same addressing and byte */
                                    /* ordering scheme as the VMEbus. If */
   {                                /* you have an Intel processor or a */
   switch(width)                    /* processor which is incapable of */
      {                             /* 32-bit accesses, you may need to */
      case 1:                       /* write your own output routine. */
         *(char *)address = value;
         break;
      case 2:
         *(int *)address = value;
         break;
      case 4:
         *(long *)address = value;
         break;
      }
   return(0);
   }


unsigned int updated(unsigned int status)      /* updates the global variable
ibsta */

/* the meanings of the bits in ibsta are:

   Bit   Mnemonic    Condition flagged

   13    ERR         The operation was unable to complete successfully
   12    ---
   11    END         A read operation was halted by EOI
   10    SRQI        SRQ was asserted at least once since the last operation
   9     ---
   8     ---
   7     ---
   6     ---
   5     CIC         The GPIB-1014 is controller in charge
   4     ATN         ATN is asserted
   3     TACS        The GPIB-1014 is addressed to talk
   2     LACS        The GPIB-1014 is addressed to listen
   1     ---
   0     ---
```

```
*/

   {
   ibsta &= 0x8000;                              /* clear all but ERR bit */
   ibsta |= ( (stat1 & 0x04) ? 0x8000:0x00 );    /* ERR */
   ibsta |= ( (In(adsr) & 0x02) ? 0x08:0x00 );   /* TA */
   ibsta |= ( (In(adsr) & 0x04) ? 0x04:0x00 );   /* LA */
   ibsta |= ( (In(adsr) & 0x40) ? 0x00:0x10 );   /* ATN */
   ibsta |= ( (In(adsr) & 0x80) ? 0x20:0x00 );   /* CIC */
   ibsta |= ( (stat2 & 0x40) ? 0x1000:0x00 );    /* SRQI */
   ibsta |= ( (stat1 & 0x10) ? 0x2000:0x00 );    /* END */
   return ibsta;
   }


/*
 * IBRD
 * Read up to cnt bytes of data from the GPIB into buf.  In
 * addition to I/O complete, the read operation terminates
 * on detection of EOI.  Prior to beginning the read the
 * interface is placed in the controller standby state.  No
 * handshake holdoffs are used so care must be exercised
 * when taking control (i.e., asserting ATN) following ibrd.
 * Prior to calling ibrd, the intended devices as well as
 * the interface board itself must be addressed by calling
 * ibcmd.
 */
Ibrd(char *buf,int cnt)
{
   unsigned long int i;                       /* time counter */

      Out(auxmr, AUX_GTS);                       /* if CAC, go to standby */
      ibcnt = 0;                                 /* reset count */
   ibsta = 0;                                 /* clear ibsta */
      stat1 = 0;                                 /* clear our isr1 copy */
   i = timeout;
   printf("\n  Waiting for %d bytes...\n", cnt);
      while((ibcnt<cnt) && !(stat1 & HR_END) && i)    /* get bytes until count
filled,
                                              EOI received, or timeout occurs */
      {
            while(i && !(stat1 = In(isr1)))          /* wait for Data In or
timeout */
         i--;                                  /* decrement time counter */
      if (i)
               buf[ibcnt++]= In(dir);           /* store byte and increment count
*/
      else                                     /* handle timeout */
         {
         printf("\n  Error - timed out.\n");
         ibsta |= 0x8000;
         }
      }
   stat2 = In(isr2);                /* update isr2 copy */
      return updated(ibsta); }

/*
```

```
 * IBWRT
 * Write cnt bytes of data from buf to the GPIB.  The write
 * operation terminates only on I/O complete.  By default,
 * EOI is always sent along with the last byte.  Prior to
 * beginning the write the interface is placed in the
 * controller standby state.  Prior to calling ibwrt, the
 * intended devices as well as the interface board itself
 * must be addressed by calling ibcmd.
 */
Ibwrt(char *buf, int cnt) {
   unsigned long int i;              /* timeout counter */

      Out(auxmr, AUX_GTS);              /* if CAC, go to standby */
      ibcnt= 0;                         /* reset count */
   ibsta = 0;                         /* clear ibsta */
   i = timeout;
      while(i && ibcnt<cnt){
            if(ibcnt==cnt-1)                /* send EOI with last byte */
                  Out(auxmr, AUX_SEOI);
            Out(cdor, buf[ibcnt++]);                        /* output byte and
increment count */
            while(i && !((stat1 = In(isr1)) & HR_DO))    /* wait for Data Out */
         --i;
            }
   if (!i)
      {
      printf("\n  Error - timed out.\n");
      ibsta |= 0x8000;
      }
   stat2 = In(isr2);                   /* update isr2 copy */
      return updated(ibsta); }

/*
 * IBCMD
 * Write cnt command bytes from buf to the GPIB.  The
 * command operation terminates only on I/O complete.  Prior
 * to beginning the command the interface is placed in the
 * controller active state.  Before calling ibcmd for the
 * first time, ibsic must be called to initialize the GPIB
 * and enable the interface to leave the controller idle
 * state.
 */
Ibcmd(char *buf,int cnt)
{
      Out(auxmr, AUX_TCA);                      /* if standby, go to CAC */
      ibcnt = 0;                                /* clear ibcnt */
   ibsta = 0;                                 /* clear ibsta */
      stat2 = 0;
      while(ibcnt<cnt){
            stat2 &= ~HR_CO;           /* clear saved copy of Command Out */
            Out(cdor, buf[ibcnt++]);                      /* output command and
increment count */
            while(((stat2 |= In(isr2)) & HR_CO)==0) ; /* wait for Command Out;
"or" to preserve SRQI */
            }
   stat1 = In(isr1);                          /* update isr1 copy */
      return updated(ibsta); }
```

```
/*
 * IBWAIT
 * Check or wait for a GPIB event to occur.  The mask argument
 * is a bit vector corresponding to the status bit vector.  It
 * has a bit set for each condition which can terminate the wait
 * (only SRQI in this implementation).  If the mask is 0 then no
 * condition is waited for and the current status is simply
 * returned.  Note that since the hardware SRQI bit is volatile
 * it will only be reported once for each occurrence of SRQ.  If
 * another function has returned the SRQI status bit, then a
 * call to ibwait with a mask containing SRQI will never return.
 */
Ibwait(int mask){
   ibcnt = 0;                              /* no count for this function */
   ibsta = 0;                              /* clear ibsta */
   do
      {
      stat1 = In(isr1);                    /* update isr1 copy */
      stat2 = In(isr2);                    /* update isr2 copy */
      }
      while(!(updated(ibsta) & mask) && mask);  /* quit if pattern matched or
mask = 0 */
      return ibsta; }


/*
 * IBRPP
 * Conduct a parallel poll and return the byte in buf.  Prior
 * to conducting the poll the interface is placed in the
 * controller active state.
 */
Ibrpp(char *buf)
{
   ibcnt = 0;                              /* no count for this function */
   ibsta = 0;                              /* clear ibsta */
      Out(auxmr, AUX_TCA);                    /* if standby, go to CAC */
      Out(auxmr, AUX_EPP);                    /* enable parallel poll */
   while (!((stat2 = In(isr2)) & HR_CO)); /* wait for the CO bit to be set */
      *buf= In(cptr);                         /* store the response byte */
   stat1 = In(isr1);                       /* update isr1 copy */
      return updated(ibsta); }

/*
 * IBONL
 * Initialize the interface hardware.  If v is non-zero then
 * the GPIB chip is enabled online.  If v is zero then it is
 * left disabled and offline.  Ibonl must be called before
 * any other function.
 */
Ibonl(int v){

   ibcnt = 0;              /* no count for this function */
   ibsta = 0;              /* clear ibsta */
      Out(auxmr, AUX_CR);     /* reset chip */
      stat1 = In(isr1) && 0;  /* clear status registers by reading */
      stat2 = In(isr2) && 0;
```

```
      Out(imr1, 0);           /* disable all interrupts */
      Out(imr2, 0);
      Out(spmr, 0);                           /* reset serial poll response */
      Out(adr, 0);                            /* set GPIB address; MTA= 00, MLA=
00 */
      Out(adr, HR_ARS | HR_DT | HR_DL);       /* disable secondary addressing */
      Out(admr, HR_TRM1 | HR_TRM0 | HR_ADM0); /* enable dual primary addressing
mode */
      Out(eosr, 0);
      Out(auxmr, AUX_CIFC);           /* by default enable system controller */
      Out(auxmr, ICR | 8);            /* set internal counter register N= 8 */
      Out(auxmr, PPR | HR_PPU);       /* parallel poll unconfigure */
      Out(auxmr, AUXRA | 0);          /* clear auxmr hidden registers */
      Out(auxmr, AUXRB | 0);
      Out(auxmr, AUXRE | 0);
      if(v) Out(auxmr, AUX_PON);      /* release pon state to bring online */
      return updated(ibsta); }


/*
 * IBSIC
 * Send IFC for at least 100 microseconds.  Ibsic must be
 * called prior to the first call to ibcmd in order to
 * initialize the bus and enable the interface to leave
 * the controller idle state.
 */
Ibsic(){
      int i;

   ibcnt = 0;                 /* no count for this function */
   ibsta = 0;                 /* clear ibsta */
      Out(auxmr, AUX_SIFC);      /* assert IFC */
      for(i= 0; i<100; i++) ;    /* busy wait >=100us */
      Out(auxmr, AUX_CIFC);      /* clear IFC */
   stat1 = In(isr1);          /* update isr1 copy */
   stat2 = In(isr2);          /* update isr2 copy */
      return updated(ibsta); }

/*
 * IBSRE
 * Send REN true if v is non-zero or false if v is zero.
 */
Ibsre(int v){
   ibcnt = 0;                              /* no count for this function */
   ibsta = 0;                              /* clear ibsta */
      Out(auxmr, (v? AUX_SREN : AUX_CREN));   /* set or clear REN */
   stat1 = In(isr1);                       /* update isr1 copy */
   stat2 = In(isr2);                       /* update isr2 copy */
      return updated(ibsta); }

/*
 * IBGTS
 * Go to the controller standby state from the controller
 * active state, i.e., turn ATN off.
 */
Ibgts(){
   ibcnt = 0;                 /* no count for this function */
```

```
   ibsta = 0;                  /* clear ibsta */
      Out(auxmr, AUX_GTS);        /* go to standby */
   stat1 = In(isr1);           /* update isr1 copy */
   stat2 = In(isr2);           /* update isr2 copy */
      return updated(ibsta); }

/*
 * IBCAC
 * Return to the controller active state from the
 * controller standby state, i.e., turn ATN on.  Note
 * that in order to enter the controller active state
 * from the controller idle state, ibsic must be called.
 */
Ibcac(){
   ibcnt = 0;                  /* no count for this function */
   ibsta = 0;                  /* clear ibsta */
      Out(auxmr, AUX_TCA);        /* assert ATN (asynchronously) */
   stat1 = In(isr1);           /* update isr1 copy */
   stat2 = In(isr2);           /* update isr2 copy */
      return updated(ibsta); }


/*
 * IBPAD
 * change the GPIB address of the interface board.
 * The address must be 0 through 31. ibonl resets the address to 0.
 */
Ibpad(int v)
   {
   ibcnt = 0;                                  /* no count for this function */
   ibsta = 0;                                        /* clear ibsta */
   printf("\n  Previous value was %d.\n", In(adr0));
   Out(adr, v);                                      /* set the new address */
   stat1 = In(isr1);                                 /* update isr1 copy */
   stat2 = In(isr2);                                 /* update isr2 copy */
   return updated(ibsta);
   }

/*
 * IBTMO
 * change the timeout value.  The number given is the number of
 * times (hex) that ibrd and ibwrt check for the operation to be finished
 * before timing out.  The actual amount of time this takes will
 * depend on the speed of your system.
 */
Ibtmo(unsigned long int v)
   {
   ibcnt = 0;                                  /* no count for this function */
   ibsta = 0;                                              /* clear ibsta */
   printf("\n  Previous timeout value: %lx,  ", timeout);
   timeout = v;
   printf("New value: %lx\n", timeout);
   stat1 = In(isr1);                           /* update isr1 copy */
   stat2 = In(isr2);                           /* update isr2 copy */
   return updated(ibsta);
   }
```

function.h

```
// Here there are declared a lot of external variables

#include "medipix.h"
#include <nivxi.h>
#include <ansi_c.h>
#include <formatio.h>
#include <utility.h>
#include <gpib.h>

/* external variables : register addresses*/
extern unsigned int *status_reg;
extern unsigned int *analog_reg;
extern unsigned int *chip_sel_reg;
extern unsigned int *cycles_reg;
extern unsigned int *ram_offset_reg;
extern unsigned int *tau0_reg;
extern unsigned int *tau1_reg;
extern unsigned int *tau2_reg;
extern unsigned int *dac0_reg;
extern unsigned int *dac1_reg;
extern unsigned int *config_reg;
extern unsigned int *data_reg;
extern unsigned int *config_ram;
extern unsigned int *data_fifo;
extern unsigned int *IBad; /* gpib */

//extern unsigned int ibsta, ibcnt;

/* external "file" variables */
extern char out_data_file[FILENAMELEN];
extern FILE *fpin;
extern FILE *fpout;
extern char Directory_Name[DIRNAMELEN];

/* external data values */
extern int data_matrix[64][64];
//extern char config_matrix[64][64];
extern int config_matrix[64][64];
extern int pattern_matrix[64][64];


extern unsigned short cold_start;
extern char mask_status;
extern char *selected_file;
extern int charged_values[33];

extern int switch_on;
extern int acquisition_on;

/* external variables : temporary configuration values */
extern unsigned int power_mode_tmp;
extern unsigned int tau0_tmp;
```

```
extern unsigned int tau1_tmp;
extern unsigned int tau2_tmp;
extern unsigned int anin_cycles_tmp;
extern unsigned int sleep_anin_period_tmp;
extern unsigned int sleep_anin_delay_tmp;
extern unsigned int sleep_dac_tmp;
extern unsigned int trig_mode_tmp;
extern unsigned int dac_bias_tmp[10];
extern unsigned int chip_sel_tmp;
extern unsigned int cycles_tmp;
extern unsigned int config_mode_tmp;
extern unsigned int ram_offset_tmp;

/* external variables : active configuration values */
extern unsigned int power_mode_act;
extern unsigned int tau0_act;
extern unsigned int tau1_act;
extern unsigned int tau2_act;
extern unsigned int anin_cycles_act;
extern unsigned int sleep_anin_period_act;
extern unsigned int sleep_anin_delay_act;
extern unsigned int sleep_dac_act;
extern unsigned int trig_mode_act;
extern unsigned int dac_bias_act[10];
extern unsigned int chip_sel_act;
extern unsigned int cycles_act;
extern unsigned int config_mode_act;
extern unsigned int ram_offset_act;
```

Function_add.c

```
#include <userint.h>
#include "Interface.h"
#include "function.h"
#include "main.h"

void load_mask(void)
{
  int err = 0;

  if (MISSING_MEDIPIX == 0) {
    load_config_bits();
    mask_status = 1;
   }
   else if (MISSING_HARDWARE == 1) {   // Usata per la simulzione
    mask_status = 1;
    config_mode_act = config_mode_tmp;
  }
}


/****************************************************************************
****************************************************************************
****************************************************************************/

int skipflin(FILE *);
int skipnflin(FILE *, int);


int read_file_mask(char *filename, int m1[64][64], int m2[64][64],
                   int m3[64][64])
{
  int i, j;
  char c;

  fpin = fopen (filename, "r");

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++) {
        m1[i][j] = 0;
        m2[i][j] = 0;
        m3[i][j] = 0;
      }

  skipnflin(fpin, 2);

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
        fscanf(fpin, "%d", &m1[i][j]);       /* gets the enable_bit_mask*/

  skipnflin(fpin, 3);

  for(i = 0; i < 64; i++)                        /* gets the test_bit_mask*/
```

```
      for(j = 0; j < 64; j++)
      fscanf(fpin, "%d", &m2[i][j]);

  skipnflin(fpin, 3);

  for(i = 0; i < 64; i++)                    /* gets the threshold_bit_mask*/
      for(j = 0; j < 64; j++)
        fscanf(fpin, "%d", &m3[i][j]);

  fclose(fpin);
  return(0);
}


void save_mask(char *filename)
{
  int i, j, k;
  char buff[30], filepath[FILENAMELEN];

/**** Questa funzione serve a salvare un file di tipo "msk"
      a partire dalla attuale maschera in ram (config_matrix)****/


  fpout = fopen (filename, "w");
  fprintf(fpout, "* enable mask *\n");
  fprintf(fpout, "\n");
  for(i = 0; i < 64; i++) {
      for(j = 0; j < 64; j++)
        fprintf(fpout, "%d ", (config_matrix[i][j] & 1));
                                        // write the enable_bit_mask
      fprintf(fpout, "\n");
  }

  fprintf(fpout, "\n");
  fprintf(fpout, "* test mask *\n");
  fprintf(fpout, "\n");
  for(i = 0; i < 64; i++) {
      for(j = 0; j < 64; j++)
        fprintf(fpout, "%d ", (config_matrix[i][j] & 2) >> 1);
                                                // write the test_bit_mask
      fprintf(fpout, "\n");
  }

  fprintf(fpout, "\n");
  fprintf(fpout, "* threshold mask *\n");
  fprintf(fpout, "\n");
  for(i = 0; i < 64; i++) {
    for(j = 0; j < 64; j++) {
        k = (config_matrix[i][j] & 4) +
              ((config_matrix[i][j] & 8) >> 2) +
              ((config_matrix[i][j] & 16) >> 4);
        fprintf(fpout, "%d ", k);
                                       // write the threshold_bit_mask
      }
    fprintf(fpout,"\n");
  }
  fclose(fpout);
```

```
}


void change_mask_values(int m[64][64], char *whichmask, int new_value, int L1,
                        int L2, int C1, int C2)
{
  int i,j;

  for(i = L1; i <= L2; i++)          // changes only selected _bit_mask
      for(j = C1; j <= C2; j++)
        m[i][j] = new_value;
}


void modify_mask(char *whichmask, int new_value, int L1, int L2, int C1, int C2)
{
  int threshold_mask[64][64], testbit_mask[64][64], enable_mask[64][64];
  int choice, k, i, j;
  char buff[80];

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
        enable_mask[i][j] = config_matrix[i][j] & 1;
                  // expand the enable_bit_mask starting from "config_matrix"

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
        testbit_mask[i][j] = (config_matrix[i][j] & 2) >> 1;
                  // expand the testbit_bit_mask starting from "config_matrix"

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++) {
        k = (config_matrix[i][j] & 4) +
            ((config_matrix[i][j] & 8) >> 2) +
              ((config_matrix[i][j] & 16) >> 4);
        threshold_mask[i][j] = k;
                  // expand the threshold_bit_mask starting from "config_matrix"
      }

  if (strcmp(whichmask, "enable_mask") == 0)
      change_mask_values(enable_mask, "enable_mask", new_value, L1, L2, C1, C2);

  if (strcmp(whichmask, "testbit_mask") == 0)
      change_mask_values(testbit_mask,  "testbit_mask",  new_value,  L1,  L2,  C1,
C2);

  if (strcmp(whichmask, "threshold_mask") == 0)
      change_mask_values(threshold_mask, "threshold_mask", new_value,
                         L1, L2, C1, C2);

// here config_matrix is reconstructed
  for(i = 0; i < 64; i++)
    for(j = 0; j < 64; j++)
        config_matrix[i][j] = ((threshold_mask[i][j] & 1) << 4) +
                                    ((threshold_mask[i][j] & 2) << 2) +
                                          (threshold_mask[i][j] & 4) +
                                          (testbit_mask[i][j] << 1) +
```

```
                                            enable_mask[i][j];
}


void load_file_arc(char *filename)
{
  char buff[50];
  int i;

  fpin = fopen(filename, "r");
  fscanf(fpin, "%d", &power_mode_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &tau0_tmp);
  fgets(buff,40,fpin);
  fscanf(fpin, "%d", &tau1_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &tau2_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &anin_cycles_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &sleep_anin_period_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &sleep_anin_delay_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &trig_mode_tmp);
  fgets(buff, 40, fpin);
  for (i = 0; i <= 9; i++){
    fscanf(fpin, "%d", &dac_bias_tmp[i]);
    fgets(buff, 40, fpin);
  }
  fscanf(fpin, "%d", &chip_sel_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &cycles_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%d", &config_mode_tmp);
  fgets(buff, 40, fpin);
  fscanf(fpin, "%x", &ram_offset_tmp);
}


void save_file(char *filename)
{
  char buff[30], filepath[FILENAMELEN];
  int i;

// questa funzione serve a salvare un file tipo ACT in uno tipo ARC

  fpout = fopen (filename, "w");

  fprintf(fpout, "%d\t /* power_mode_act */ \n", power_mode_act);
  fprintf(fpout, "%d\t /* tau0_act */ \n", tau0_act);
  fprintf(fpout, "%d\t /* tau1_act */ \n", tau1_act);
  fprintf(fpout, "%d\t /* tau2_act */ \n", tau2_act);
  fprintf(fpout, "%d\t /* anin_cycles_act */ \n", anin_cycles_act);
  fprintf(fpout, "%d\t /* sleep_anin_period_act */ \n", sleep_anin_period_act);
  fprintf(fpout, "%d\t /* sleep_anin_delay_act */\n", sleep_anin_delay_act);
  fprintf(fpout, "%d\t /* trig_mode_act */ \n", trig_mode_act);
```

```
  for (i = 0; i <= 9; i++)
    fprintf(fpout, "%d\t /* dac_bias_act[%d] */ \n", dac_bias_act[0], i);
  fprintf(fpout, "%d\t /* chip_sel_act */ \n", chip_sel_act);
  fprintf(fpout, "%d\t /* cycles_act */ \n", cycles_act);
  fprintf(fpout, "%d\t /* config_mode_act */ \n", config_mode_act);
  fprintf(fpout, "%x\t /* ram_offset_act */ \n", ram_offset_act);

  fclose(fpout);
}


/*  ***************  OPEN FILE TO READ  ****************/
int open_read_file(char *filename)
{
/* this function must return 0 if the file to read exists
      Must return 1 if the file does not exists */

  int answer=0;
  char filein[FILENAMELEN], filepath[FILENAMELEN];

  if (filename[0] == 'L' && filename[1] == 'U' && filename[2] == 'T')
      strcpy(filein, filename);
   else {
      sprintf(filepath, PATHF);
      strcpy(filein, filepath);
      strcat(filein, filename);
  }

  do {
    if ((fpin = fopen (filein, "r")) == NULL) {
        static char message[300];

        sprintf(message, "CANNOT OPEN FILE: %s\n"
                  "Would you try again changing filename?", filename);
        answer = ConfirmPopup("Error !!!", message);
     }
     else
      answer = 0;
  } while (answer);

  return(0);
}


/*  ***************  OPEN FILE TO WRITE  *****************/
int open_write_file(char *filename, int k)
{
/* this function must return 0 if operation of write on a file
   is a safe operation (or the user decides to overwrite an existing file).
   Must return 1 if the file already exists */

  int answer, i = 0, j = 0, exist;
  char fileout[FILENAMELEN], filepath[FILENAMELEN], trash[FILENAMELEN];

  sprintf(filepath, PATHF);
  while ((fileout[i] = filepath[i]) != '\0')
    i++;
```

```
  while ((fileout[i + j] = filename[j]) != '\0')
    j++;
  sprintf(fileout, "%s", fileout);
  i=0;
  j=0;

  if(k == 0) { // to write on a file without control on his previous existance.
    fpout = fopen(fileout, "w");
    return(0);
   }
   else     {      // to write on a file with control on his previous existance.
      exist = GetFileInfo(fileout, &exist);
      if (exist != NULL) {
        static char message[300];

        sprintf(message, "%s ALREADY EXISTS!!"
                "\n\nDo you really want to OVERWRITE it?", filename);
        answer = ConfirmPopup("Error !!!", message);
        if (answer) {
          fpout = fopen(fileout, "w");
          return (0);
         }
         else {
          return(1);
        }
       }
       else {
        fpout = fopen(fileout, "w");
        return (0);
      }
  }
}


/*************************************************************************
************** funzioni (gualtiero) richiamate in V_test.c ***************
*************************************************************************/


/*************** basic access register ***********************************
*************************************************************************/


unsigned int * registro(int reg)
{
  switch(reg) {
    case 1:
      return(status_reg = (unsigned int *)((int)status_reg + STATUS));
    case 2:
      return(analog_reg = (unsigned int *)((int)status_reg + ANALOG_REG));
    case 3:
      return(chip_sel_reg = (unsigned int *)((int)status_reg + CHIP_SEL));
    case 4:
      return(cycles_reg = (unsigned int *)((int)status_reg + CYCLE_NUM));
    case 5:
      return(ram_offset_reg = (unsigned int *)((int)status_reg + RAM_START));
    case 6:
```

```
      return(tau0_reg = (unsigned int *)((int)status_reg + TIMER_TAU0));
    case 7:
      return(tau1_reg = (unsigned int *)((int)status_reg + TIMER_TAU1));
    case 8:
      return(tau2_reg = (unsigned int *)((int)status_reg + TIMER_TAU2));
  }
 return(0);
}


void start_B_A(void)
{
  SetCtrlAttribute(basic_r_a, BASIC_R_A_MSG1, ATTR_VISIBLE, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_MSG2, ATTR_VISIBLE, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_MSG3, ATTR_VISIBLE, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_MSG4, ATTR_VISIBLE, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_MSG5, ATTR_VISIBLE, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_MSG6, ATTR_VISIBLE, 0);

  SetCtrlAttribute(basic_r_a, BASIC_R_A_BIT, ATTR_DIMMED, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_WORD, ATTR_DIMMED, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_BACK, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_VALUE_WRITTEN, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_VALUE_READ, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_WRITE_READ, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_DECORATION_5, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_REGISTER, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_TURN_ON_OFF, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_MASK, ATTR_DIMMED, 1);

  SetCtrlAttribute(basic_r_a, BASIC_R_A_RUN, ATTR_VISIBLE, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_RUN2, ATTR_VISIBLE, 0);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_WRITE_READ, ATTR_DIMMED, 1);
  SetCtrlAttribute(basic_r_a, BASIC_R_A_QUIT, ATTR_VISIBLE,1);
}


/*************** RAM or FIFO test ****************************************
*************************************************************************/

void erase_message (void)
{
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT1, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT2, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT3, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT4, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT5, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT6, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT7, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT7_2, ATTR_VISIBLE, 0);
}


void start_R_F(void)
{
  SetCtrlVal(ram_fifo, RAM_FIFO_READ, 0);
  SetCtrlVal(ram_fifo, RAM_FIFO_OFFSET, 0);
```

```
  SetCtrlAttribute(ram_fifo, RAM_FIFO_READ, ATTR_DIMMED, 1);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_OFFSET, ATTR_DIMMED, 1);

  SetCtrlAttribute(ram_fifo, RAM_FIFO_RAM, ATTR_DIMMED, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_FIFO, ATTR_DIMMED, 0);

  SetCtrlAttribute(ram_fifo, RAM_FIFO_RUN_RAM, ATTR_VISIBLE, 0);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RUN_FIFO, ATTR_VISIBLE, 0);

  SetCtrlAttribute(ram_fifo, RAM_FIFO_QUIT, ATTR_VISIBLE, 1);
  SetCtrlAttribute(ram_fifo, RAM_FIFO_QUIT_RF, ATTR_VISIBLE, 0);

  SetCtrlAttribute(ram_fifo, RAM_FIFO_BACK, ATTR_DIMMED, 1);
  erase_message();
  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT5, ATTR_VISIBLE, 1);
}


/***************** TEST COUNTERS ********************************************
*******************************************************************************
***/

void start(void)
{
  SetCtrlAttribute(Test_Counters, COUNTERS_DECORATION_5, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_VIEW, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_CREATE, ATTR_DIMMED, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_LOAD, ATTR_DIMMED, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_SAVE, ATTR_DIMMED, 1);
  SetCtrlAttribute(Test_Counters, COUNTERS_GO_TEST, ATTR_DIMMED, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_SHOW, ATTR_DIMMED, 0);

  SetCtrlAttribute(Test_Counters, COUNTERS_QUIT, ATTR_VISIBLE, 1);
  SetCtrlAttribute(Test_Counters, COUNTERS_BACK, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_START_COLUMN, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_END_COLUMN, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_START_ROW, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_END_ROW, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_VALUE, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_FILE_NAME, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_RUN_CREATE, ATTR_DIMMED, 1);
  SetCtrlAttribute(Test_Counters, COUNTERS_RUN_SAVE, ATTR_DIMMED, 1);
  SetCtrlAttribute(Test_Counters, COUNTERS_RUN_LOAD, ATTR_DIMMED, 1);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG10, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG1, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG2, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG3, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG4, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG5, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG6, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG9, ATTR_VISIBLE, 0);

  SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG7, ATTR_VISIBLE, 1);

  SetCtrlAttribute(Test_Counters, COUNTERS_DECORATION_2, ATTR_VISIBLE, 1);
```

```
  SetCtrlAttribute(Test_Counters, COUNTERS_DECORATION_3, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Counters, COUNTERS_DECORATION_4, ATTR_VISIBLE, 1);
}


void gdisplay_matrix_values(int m[64][64], char *whichmask)
{
  char buff1[1024];
  char buff[2065];
  int  i, j;

  sprintf(buff, "           %s                          \n\n", whichmask);

  for(i = 0; i < 64; i++)
      for(j = 0; j < 4; j++) {
      sprintf(buff1, "%.5d ", m[i][j]);
        if (j == 3)
          strcat(buff1, "\n");
        strcat(buff, buff1);
      }

  MessagePopup ("Show Counters Pattern", buff);
}


void save_files(void)
{
  int i1, j1;

  open_write_file("pat.error", SAVE_WITHOUT_CONTROL);

  fprintf(fpout, "/******** pattern matrix *******/\n");

  for(i1 = 0; i1 < 64; i1++) {
      for(j1 = 0; j1 < 64; j1++)
        fprintf(fpout, "%d ", pattern_matrix[i1][j1]);
    fprintf(fpout, "\n");
  }

  fprintf(fpout, "\n/******** data matrix *******/\n");

  for(i1 = 0; i1 < 64; i1++) {
      for(j1 = 0; j1 < 64; j1++)
        fprintf(fpout, "%d ", data_matrix[i1][j1]);
      fprintf(fpout,"\n");
  }

  fclose(fpout);
}


int Open_Read_File(char *filename)
{
  char filein[FILENAMELEN], filepath[FILENAMELEN];

  sprintf(filepath, PATHF);
  strcpy(filein, filepath);
```

```
  strcat(filein, filename);

  if ((fpin = fopen(filein, "r")) == NULL)
    return(1);

  return(0);
}


/***************************** TEST MASK ***********************************
*******************************************************************************
***/


int Load_Mask(void)
{
  int err = 0;

  if (MISSING_MEDIPIX == 0) {
      load_config_bits();
      mask_status = 1;
  }

// Usata per la simulazione
  if (MISSING_HARDWARE == 1) {
      mask_status = 1;
      config_mode_act = config_mode_tmp;
  }

  return(1);
}


void get_array(int M[64][64])
{
  int i, j;

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
        fscanf(fpin, "%d", &M[i][j]);
}


/*****************************************************************************
************** funzione per la visualizzazione dei messaggi di errore **********
*****************************************************************************/

void error(int code)
{

  switch (code) {
    case 1:
      MessagePopup("Error", "File does not exist");
      break;
    case 2:
      MessagePopup("Error", "Medipix is off!");
      break;
```

```
    case 3:
      MessagePopup("Warning", "Select Add or Subt files and then press Run.");
      break;
    case 4:
      MessagePopup("Warning", "Input filename .msk");
      break;
  }
}


int Read_File_Mask(char *filename, int m1[64][64], int m2[64][64],
                   int m3[64][64])
{
  int i, j;
  char c;

  if ((fpin = fopen(filename, "r")) != NULL) {
      for(i = 0; i < 64; i++)
        for(j = 0; j < 64; j++) {
            m1[i][j] = 0;
            m2[i][j] = 0;
            m3[i][j] = 0;
        }

      skipnflin(fpin, 2);
      get_array(m1);

      skipnflin(fpin, 3);
    get_array(m2);

      skipnflin(fpin, 3);
    get_array(m3);

      fclose(fpin);
      return(0);
   }
   else
    error(1);

  return(1);
}


int skipflin (FILE *fpin)
{
  char c;

  while (c = fgetc(fpin) != '\n');
  return 0;
}


int skipnflin (FILE *fpin, int n)
{
  int i;

  for (i = 0; i < n; i++)
```

```
    skipflin(fpin);
  return 0;
}
```

Function_VME.c

```
#include "function.h"

void set_default(char mask_file[512])
{
  int i, j, threshold_mask[64][64], testbit_mask[64][64], enable_mask[64][64];

  mask_status = 0;
// At start the mask is not loaded on Medipix...
// Inizialmente la maschera non è caricata su Medipix
  acquisition_on = 0;
// ...nor there's been done any acquisition.
// Inizialmente non è stata fatta nessuna cquisizione

/*************************************************************************
***************** Carica la maschera dal file default.msk ******************
*************************************************************************/
  read_file_mask(mask_file, enable_mask, testbit_mask, threshold_mask);

/* here config_matrix is build starting from enable_mask,
   testbit_mask, threshold_mask */

  for(i = 0; i < 64; i++)
    for(j = 0; j < 64; j++)
        config_matrix[i][j] = ((threshold_mask[i][j] & 1) << 4) +
                              ((threshold_mask[i][j] & 2) << 2) +
                              ((threshold_mask[i][j] & 4)) +
                              (testbit_mask[i][j] << 1) +
                              enable_mask[i][j];
/*************************************************************************
****************** Fine caricamento Maschera di Default ********************
*************************************************************************/

// Get Medipix Status
// Ottiene lo stato di medipix
  if (MISSING_MEDIPIX == 0)
    switch_on = check_power();
  if (MISSING_HARDWARE == 1)
    switch_on = 0;             // medipix off in simulation

/*************************************************************************
******************    Fine impostazione dei valori     *******************
*************************************************************************/

  read_act();  // Legge i valori attivi dalla scheda VME
}

void read_act(void)
{
  int i;

  if (MISSING_HARDWARE == 1) {
     /* ************************************************** */
```

```
     /* **** Valori di prova delle variabili attive         ** */
     /* ************************************************** */
    power_mode_act = 0;
    tau0_act = 60000;
    tau1_act = 20000;
    tau2_act = 5000;
    chip_sel_act = 0;
    cycles_act = 256;
    ram_offset_act = 0;
    dac_bias_act[0] = 100; /*Vbias*/
    dac_bias_act[1] = 100; /*Vcomp*/
    dac_bias_act[2] = 100; /* Vth */
    dac_bias_act[3] = 100; /* Vtha */
    dac_bias_act[4] = 100; /* Vdl */
    dac_bias_act[5] = 100;
    dac_bias_act[6] = 100;
    dac_bias_act[7] = 100;
    dac_bias_act[8] = 100;
    dac_bias_act[9] = 500;

     /* external variables: software options and variables */
    sleep_anin_delay_act = 10000;
    anin_cycles_act = 5;
    trig_mode_act = 1;
    config_mode_act = 0; /* 1 = direct mode for configuration bits */
     /************************************************************
     ************** Fine impostazione valori di prova ***************
     ************************************************************/
  }

  if (MISSING_HARDWARE == 0) {
      power_mode_act = power_mode_tmp;
      read_data(tau0_reg, &tau0_act, 1, "s");
      read_data(tau1_reg, &tau1_act, 1, "s");
      read_data(tau2_reg, &tau2_act, 1, "s");
      read_data(chip_sel_reg, &chip_sel_act, 1, "s");
      read_data(cycles_reg, &cycles_act, 1, "s");
      read_data(ram_offset_reg, &ram_offset_act, 1, "s");

      for(i = 0; i < 5; i++) {
        read_data(dac0_reg, dac_bias_act, 5, "m");
        read_data(dac1_reg, dac_bias_act+5, 5, "m");
    }

       /* external variables: software options and variables */
    sleep_anin_delay_act = sleep_anin_delay_tmp;
    sleep_anin_period_act = sleep_anin_period_tmp;
    anin_cycles_act = anin_cycles_tmp;
    trig_mode_act = trig_mode_tmp;
    config_mode_act = config_mode_tmp; // 1 = direct mode for configuration bits
  }

}


void make_address()
{
```

```
  short int ret, ret_gpib;
  unsigned int win, win_gpib, wine, winb;

  status_reg = (unsigned int *)(VMEbase | STATUS);
  analog_reg = (unsigned int *)(VMEbase | ANALOG_REG);
  chip_sel_reg = (unsigned int *)(VMEbase | CHIP_SEL);
  cycles_reg = (unsigned int *)(VMEbase | CYCLE_NUM);
  ram_offset_reg = (unsigned int *)(VMEbase | RAM_START);
  tau0_reg = (unsigned int *)(VMEbase | TIMER_TAU0);
  tau1_reg = (unsigned int *)(VMEbase | TIMER_TAU1);
  tau2_reg = (unsigned int *)(VMEbase | TIMER_TAU2);
  dac0_reg = (unsigned int *)(VMEbase | DAC_VALUE0);
  dac1_reg = (unsigned int *)(VMEbase | DAC_VALUE1);
  config_reg = (unsigned int *)(VMEbase | CONFIG_REG);
  data_reg = (unsigned int *)(VMEbase | DATA_REG);
  config_ram = (unsigned int *)(VMEbase | CONFIG_RAM);
  data_fifo = (unsigned int *)(VMEbase | DATA_FIFO);
  if (GPIBFLAG == 1)
    IBad = (unsigned int *)(Gpib_ad);

/*  re-addressing for PC+VXI NI hardware/software   */
  if (PC_VXI == 1) {
    ret = MAKE_ADDRESS_SIZE(0x100000);
    status_reg = MAKE_ADDRESS(VMEadd32, (VMEbase | STATUS), 1000, &win, &ret);
    analog_reg = (unsigned int *)((int)status_reg + ANALOG_REG);
    chip_sel_reg = (unsigned int *)((int)status_reg + CHIP_SEL);
    cycles_reg = (unsigned int *)((int)status_reg + CYCLE_NUM);
    ram_offset_reg = (unsigned int *)((int)status_reg + RAM_START);
    tau0_reg = (unsigned int *)((int)status_reg + TIMER_TAU0);
    tau1_reg = (unsigned int *)((int)status_reg + TIMER_TAU1);
    tau2_reg = (unsigned int *)((int)status_reg + TIMER_TAU2);
    dac0_reg = (unsigned int *)((int)status_reg + DAC_VALUE0);
    dac1_reg = (unsigned int *)((int)status_reg + DAC_VALUE1);
    config_reg = (unsigned int *)((int)status_reg + CONFIG_REG);
    data_reg = (unsigned int *)((int)status_reg + DATA_REG);;
    config_ram = (unsigned int *)((int)status_reg + CONFIG_RAM);
    data_fifo = (unsigned int *)((int)status_reg + DATA_FIFO);

    ret_gpib = MAKE_ADDRESS_SIZE(0x10000);
    if (GPIBFLAG == 1)
      IBad = MAKE_ADDRESS(VMEadd16, Gpib_ad, 1000, &win_gpib, &ret_gpib);

/*ret=GetWindowRange(win, &winb, &wine);
ret=ret;

ret=GetWindowRange(win_gpib, &winb, &wine);
ret=ret;     */
  }
}


unsigned int * make_address_DUMMY(unsigned int par, unsigned int addr,
                                  int time, unsigned int *wind, short int *ret)
{
  return((unsigned int *)addr);
}
```

```
int make_address_DUMMY_size(unsigned int par)
{
  return(0);
}


int modify_register(TYPE *Register, unsigned int on_off, unsigned int mask,
                    char *W_R)
{
/*
Register = register address,
on_off = value to write: 0xffffffff for on, 0x0 for off,
mask = mask which selects the bit to modify
*W_R = "W" to modify the register, "R" to read a bit
in the register.
This function returns new_bit; in "W" mode we do not use it,
in "R" mode new_bit=0 means that the bit we were checking is 0,
new_bit different from 0 means that the bit we were checking is 1.
*/
  unsigned int new_value, old_value, new_bit;

  new_value = *Register;
  if(*W_R == 'W') {
      old_value = new_value & ~mask;
      new_bit = (mask & on_off);
      *Register = (new_bit | old_value);
  }
  if (*W_R == 'R')
    new_bit = (new_value & mask);

  return(new_bit);
}


void write_data(TYPE *Register, unsigned int *data_pointer, int number_data,
                char *addr_flag)
/*
Register = register address,
data_pointer = pointer to the first datum,
number_data = number of data to write,
*addr_flag = "m" is multiple address, as in a RAM, other
      is single address, as in a FIFO.
If Register is pointer to unsigned integer, Register++
points to next register in VME A32
*/
{
  int i;

  for (i = 0; i < number_data; i++) {
      *Register = *data_pointer;
      data_pointer++;
      if(*addr_flag == 'm')
        Register++;
//    sleep_soft(100);
  }
}
```

```
void read_data(TYPE *Register, unsigned int *data_pointer, int number_data,
               char *addr_flag)
/*
Register = register address,
data_pointer = pointer to the first datum,
number_data = number of data to read,
*addr_flag = "m" is multiple address, as in a RAM, other
      is single address, as in a FIFO.
If Register is pointer to unsigned integer, Register++
points to next register in VME A32.
*/
{
  int i;

  for (i = 0; i < number_data; i++) {
      *data_pointer = *Register;
      data_pointer++;
      if (*addr_flag == 'm')
        Register++;
//    sleep_soft(100);
  }
}


/********************** INIT_BOARD ******************/

int init_board(void)
{

  modify_register(status_reg, turn_on, bit_resetVME, "W");  // reset VMEboard
  modify_register(status_reg, turn_on, bit_resetFIFO, "W"); // reset FIFO
  modify_register(status_reg, turn_off, bit_all, "W");      // reset status VME

/* modify_register(analog_reg,turn_on,bit_DAC0_res,"W");     reset DAC0 */
/* modify_register(analog_reg,turn_on,bit_DAC1_res,"W");     reset DAC1 */
/* modify_register(analog_reg,turn_off,bit_all,"W");                     reset
Vdd,Vdda */

  return(0);
}


/********************** RESET_AND_SETUP ******************/

int reset_and_setup(void)
{
  int err = 0;

  modify_register(status_reg, turn_on, bit_resetVME, "W");
/*reset VMEboard (except DACs and FIFOs)  */
  err += setup_chip();
  err += setup_cycles();
  err += setup_RAM_start();
  err += setup_tau();
```

```
  return(err);
}


int setup_tau(void)
{
/****** configure time_register **********/

  write_data(tau0_reg, &tau0_tmp, 1, "s");
  write_data(tau1_reg, &tau1_tmp, 1, "s");
  write_data(tau2_reg, &tau2_tmp, 1, "s");

  read_data(tau0_reg, &tau0_act, 1, "s");
  read_data(tau1_reg, &tau1_act, 1, "s");
  read_data(tau2_reg, &tau2_act, 1, "s");

  if ((tau0_act != tau0_tmp) ||
      (tau1_act != tau1_tmp) ||
      (tau2_act != tau2_tmp)) {
      Print_error(1);   // Error 1: VME write error on timer setup
      return(8);
   }
   else
      return(0);
}


/********** SETUP CHIP SELECTION *****************/

int setup_chip(void)
{

  write_data(chip_sel_reg, &chip_sel_tmp, 1, "s");
  read_data(chip_sel_reg, &chip_sel_act, 1, "s");
  if (chip_sel_tmp != chip_sel_act) {
      Print_error(2); // Error 2: VME write error on chip_sel setup
      return(1);
   }
   else
      return(0);
}


/********** SETUP RAM START ADDRESS ****************/

int setup_RAM_start(void)
{
  unsigned int ram_add_tmp, ram_add_act;

  write_data(ram_offset_reg, &ram_offset_tmp, 1, "s");
  read_data(ram_offset_reg, &ram_offset_act, 1, "s");
  if (ram_offset_tmp != ram_offset_act) {
      Print_error(3);   // Error 3: VME write error on RAM start address setup
      return(4);
   }
   else
      return(0);
```

```
}


/********** SETUP CYCLES OF R/W ******************/

int setup_cycles(void)
{
  write_data(cycles_reg, &cycles_tmp, 1, "s");
  read_data(cycles_reg, &cycles_act, 1, "s");
  if (cycles_tmp != cycles_act) {
      Print_error(4); //Error 4: VME write error on cycle_number setup
      return(2);
   }
   else
      return(0);
}


/*********** SETUP COUNTERS' ZERO ***********/

int setup_counters(unsigned int i)
{
  unsigned int a;

  write_data(data_reg, &i, 1, "s");
  read_data(data_reg, &a, 1, "s");
  if (i != a ) {
      Print_error(5);  // Error 5: VME write error on setup_counters
      return(16);
   }
   else
      return(0);
}


/*  ***************  RESET COUNTERS  ***************      */

void reset_counters(unsigned int num)
{
  int i, a, err;

  setup_counters(num);

/********** write "zero" on medipix ******/
  modify_register(status_reg, turn_on, bit_zero_wr, "W");

/********** read loop ***********************/
/*  start polling to verify that 'zero' is written on medipix  */

//   PROPOSED CODE INSTEAD OF THE LOOP (c) 2000, Maiorino Marino
//while ((a=modify_register(status_reg,0,bit_zero_wr,"R"))!=0);

  for(i = 0; ; i++)
      if((a = modify_register(status_reg, 0, bit_zero_wr, "R")) == 0)
        break;
}
```

```
/*  ***************  START ACQUISITION  *****************/

void start_acq(int radiography)
{
  int i, j, a, err;
  float sleep1;

  trig_mode_act = trig_mode_tmp;
  anin_cycles_act = anin_cycles_tmp;
  sleep_anin_period_act = sleep_anin_period_tmp;
  sleep_anin_delay_act = sleep_anin_delay_tmp;
  err = reset_and_setup();

  if (err)
    Print_error(11);

  reset_counters(ZERO);

/*  reset Medipix */
  modify_register(status_reg, turn_on, bit_resetb, "W");

/* acquisition with timer (internal trigger) */
  if (trig_mode_act == 0) {
    modify_register(status_reg, turn_on, bit_start, "W");

/*  start polling waiting for end of acquisition */
/* nel caso di autoradiografia il polling non deve essere fatto */
    if (radiography == 0)
// OTHER LOOP TO BE REWRITTEN BY THE PROPOSED CODE
      for(i = 0; ; i++)
          if((a=modify_register(status_reg,0,bit_start,"R"))==0)
          break;
  }

/* acquisition with timer (external trigger) */
  if(trig_mode_act == 1) {
/* wait until bit_start is set via external trigger */
    for(i = 0; ; i++)
        if((a = modify_register(status_reg, 0, bit_start, "R")) == 0x10)
      break;

/*  start polling waiting for end of acquisition */
    for(j = 0; ; j++)
        if((a = modify_register(status_reg, 0, bit_start, "R")) == 0)
          break;
  }

/* acquisition with anin that works as pulse generator and
   shutter signal goes off after the last pulse */
  if(trig_mode_act == 2) {
      sleep1 = (float)sleep_anin_delay_act / 1000.;
      modify_register(status_reg, turn_on, bit_shutter, "W");
//             sleep_soft(sleep_anin_delay_act);
      Delay(sleep1);

    for(i = 0; i < anin_cycles_act; i++) {
```

```
        modify_register(status_reg, turn_on, bit_ANIN, "W");
        sleep_soft(sleep_anin_period_act);
      }

      Delay(sleep1);
//             sleep_soft(sleep_anin_delay_act);
      modify_register(status_reg, turn_off, bit_shutter, "W");
  }

}


/*  ***************  DOWNLOAD DATA  ****************/

void download_data(void)
{
  int k, i, j, ii, a, kk, x = 0;
  int control, err = 0;
  unsigned int count_number[2048];

  err = reset_and_setup();
  if (err)
    Print_error(11);
   else {
/****** enable the data transfer from medipix counters to VME FIFO ********/
      modify_register(status_reg, turn_on, bit_data_rd, "W");

/********* read loop *****************/

/* start polling waiting for data download end     */
      for(ii = 0; ; ii++)
        if((a = modify_register(status_reg, 0, bit_data_rd, "R")) == 0)
            break;

/******** read from VME FIFO ***********/
    read_data(data_fifo, count_number, 2048, "s");

/* DATA RE-ORGANIZATION  */
    for(k = 0; k < 4; k++)
        for(i = 63; i >= 0; i--)
            for(kk = 0; kk < 8; kk++) {
            data_matrix[i][kk * 4 + k] = (int) (0x00007fff & count_number[x]);
              data_matrix[i][(8 + kk) * 4 + k] = (int) ((0x7fff0000 &
                                                    count_number[x]) >> 16);
              x++;
            }
  }
}


/*************** TURN_ON_DAC **********g**********/

int turn_on_dac(void)
{
  int i, erdac = 0;

  if (MISSING_HARDWARE == 0) {
```

```
    for(i = 0; i < 5; i++) {
        write_data(dac0_reg, dac_bias_tmp, 5, "m");
        write_data(dac1_reg, dac_bias_tmp + 5, 5, "m");
      }
      for(i = 0; i < 5; i++) {
        read_data(dac0_reg, dac_bias_act, 5, "m");
        read_data(dac1_reg, dac_bias_act + 5, 5, "m");
      }
      for(i = 0; i < 10; i++) {
        if (dac_bias_tmp[i] != dac_bias_act[i]) {
          static int i;
          static char message[60];

            Print_error(8);  // Error 8: VME write error on dac channel
            erdac++;
        }
      }
      if(erdac != 0)
        return(erdac);

      modify_register(analog_reg, turn_on, bit_DAC0_en, "W");
      modify_register(analog_reg, turn_on, bit_DAC1_en, "W");
  }

// Simulazione
  if (MISSING_HARDWARE == 1) {
      power_mode_act = power_mode_tmp;
      for (i = 0; i < 10; i++)
      dac_bias_act[i] = dac_bias_tmp[i];
  }

  return(0);
}


/*************** TURN_ON_DAC ******************/
/*
int turn_on_dac(void)
{
      int i,a,erdac=0;

      if (MISSING_HARDWARE==0) {
         modify_register(analog_reg,turn_on,bit_Vdd_en,"W");
         modify_register(analog_reg,turn_on,bit_Vdda_en,"W");

//       a=modify_register(analog_reg,0,bit_Vdd_en,"R");
//       a=modify_register(analog_reg,0,bit_Vdda_en,"R");

         if((a=modify_register(analog_reg,0,bit_Vdd_en,"R"))==0)
         {
            Print_error(6);  //Error 6: Vdd is not on
            return(1);
         }

         if((a=modify_register(analog_reg,0,bit_Vdda_en,"R"))==0)
         {
            Print_error(7);  //Error 7: Vdda is not on
```

```
            return(1);
         }

         for(i=0;i<5;i++)
         {
             write_data(dac0_reg,dac_bias_tmp,5,"m");
             write_data(dac1_reg,dac_bias_tmp+5,5,"m");
         }
         for(i=0;i<5;i++)
         {
            read_data(dac0_reg,dac_bias_act,5,"m");
          read_data(dac1_reg,dac_bias_act+5,5,"m");
         }
         for(i=0;i<10;i++)
         {
            if (dac_bias_tmp[i] != dac_bias_act[i] )
            {
           static int i;
           static char message[60];

             Print_error(8);  // Error 8: VME write error on dac channel
             erdac++;
            }
         }
         if(erdac != 0) return(erdac);

         modify_register(analog_reg,turn_on,bit_DAC0_en,"W");
         modify_register(analog_reg,turn_on,bit_DAC1_en,"W");

      }

      // Simulazione
      if (MISSING_HARDWARE==1) {
            power_mode_act=power_mode_tmp;
            for (i=0; i<10; i++)
                  {
                  dac_bias_act[i]=dac_bias_tmp[i];
                  }
      }

      return(0);
}

*/


/*************** TURN_ON_POWER ******************/

int turn_on_power(void)
{
  int i,a;

  if (MISSING_HARDWARE == 0) {
    modify_register(analog_reg, turn_on, bit_Vdd_en, "W");
    modify_register(analog_reg, turn_on, bit_Vdda_en, "W");

      if((a = modify_register(analog_reg, 0, bit_Vdd_en, "R")) == 0) {
```

```
        Print_error(6);  //Error 6: Vdd is not on
        return(1);
      }
      if((a = modify_register(analog_reg, 0, bit_Vdda_en, "R")) == 0) {
        Print_error(7);  //Error 7: Vdda is not on
        return(1);
    }
  }

// Simulazione
  if (MISSING_HARDWARE == 1) {
      power_mode_act = power_mode_tmp;
      for (i = 0; i < 10; i++)
        dac_bias_act[i] = dac_bias_tmp[i];
  }

  return(0);
}


/*************** TURN_OFF_POWER ******************/

int turn_off_power(void)
{
  int a;

  modify_register(analog_reg, turn_off, bit_Vdda_en, "W");
  modify_register(analog_reg, turn_off, bit_Vdd_en, "W");

  if((a = modify_register(analog_reg, 0, bit_Vdda_en, "R")) != 0) {
      Print_error(9);   // Error 9: Vdda is not off
      return(1);
  }
  if((a = modify_register(analog_reg, 0, bit_Vdd_en, "R")) != 0) {
      Print_error(10);   // Error 10: Vdd is not off
      return(1);
  }

  return(0);
}

/*************** TURN_OFF_DAC ******************/

int turn_off_dac(void)
{

  modify_register(analog_reg, turn_on, bit_DAC0_res, "W");
  modify_register(analog_reg, turn_on, bit_DAC1_res, "W");
  return(0);
}


/*************** TURN_OFF_DAC ******************/
/*
int turn_off_dac(void)
{
  int i,a,erdac=0;
```

```
  unsigned int zero0=0;

      modify_register(analog_reg,turn_on,bit_DAC0_res,"W");
      modify_register(analog_reg,turn_on,bit_DAC1_res,"W");

      modify_register(analog_reg,turn_off,bit_Vdda_en,"W");
      modify_register(analog_reg,turn_off,bit_Vdd_en,"W");

      if((a=modify_register(analog_reg,0,bit_Vdda_en,"R"))!=0)
      {
            Print_error(9);   // Error 9: Vdda is not off
            return(1);
      }

      if((a=modify_register(analog_reg,0,bit_Vdd_en,"R"))!=0)
      {
            Print_error(10);   // Error 10: Vdd is not off
            return(1);
      }

      return(0);
}
*/


/*  ***************  LOAD CONFIGURATION BITS  *****************/

void load_config_bits(void)
{
  int i, ii, a, j, k, kk, kk4 ,x = 0, err = 0;
  unsigned int config_number[1024], cycles_dummy;

  config_mode_act = config_mode_tmp;

/* re-organization of configuration data ready to be sent to Medipix */

// La RAM è composta da parole di 32 bit (4 byte)
// Ogni numero della matrice di maschere è memorizzato in un byte
  for(k = 0; k < 4; k++)
      for(i = 0; i < 64; i++)
        for(kk = 0; kk < 4; kk++) {
          kk4 = kk * 4 + k;
            config_number[x]=       config_matrix[i][kk4] +
                                256 * config_matrix[i][kk4 + 16] +
                              65536 * config_matrix[i][kk4 + 32] +
                           16777216 * config_matrix[i][kk4 + 48];
            x++;
        }

/****************************/
/****** direct mode **********/
/****************************/

/***** write on FPGA *****/

  if (config_mode_act == 1) {
      cycles_dummy = cycles_tmp;
```

```
      cycles_tmp = 2;
      for(i = 0; i < 128; i++) {
        err += reset_and_setup();
        write_data(config_reg, &config_number[i * 8], 8, "m");
        modify_register(status_reg, turn_on, bit_direct_config_wr, "W");

/*  start polling to verify that configuration bits have
      been written on medipix  */
// PROPOSED MODIFICATION (c) 2000 Marino Maiorino
// while((a=modify_register(status_reg,0, bit_config_wr,"R"))!=0)
      for(ii = 0; ; ii++)
            if((a = modify_register(status_reg, 0, bit_config_wr, "R")) == 0)
              break;
    }
    cycles_tmp = cycles_dummy;
  }

/***************************/
/****** indirect mode *******/
/***************************/
  if(config_mode_act == 0) {
      err += reset_and_setup();
      write_data(config_ram + ram_offset_act, config_number, 1024, "m");
      if (BAD_RAM == 1)
        write_data(config_ram + ram_offset_act, &config_number[0], 1, "s");
      modify_register(status_reg, turn_on, bit_config_wr, "W");

/*  start polling to verify that configuration bits
      have been written on medipix  */
      for(ii = 0; ; ii++)
      if((a = modify_register(status_reg, 0, bit_config_wr, "R")) == 0)
        break;
  }
}


/********** SLEEP FUNCTION *******/

void sleep_soft(int N)
{
  int i;

  for (i = 0; i <= N; i++);
}


/************** LUT conversion **************/

void LUT_conversion(void)
{
  int A[32768];
  int matrix2[64][64];
  int i, j;
  char filepath[FILENAMELEN];

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
```

```
      matrix2[i][j] = 0;

  A[0] = 696969;  /* unrealistic number to be read if counters output
                       pseudorandom number 0 (impossible) */

  open_read_file("LUT_real.lut");
  for (i = 1; i <= 32767; i++)
      fscanf(fpin, "%d", &A[i]);
  fclose(fpin);

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
      matrix2[i][j] = A[data_matrix[i][j]];

/* FILE NOT CONVERTED IS WRITTEN *****/

/*          sprintf(filepath,"%sraw.dat",PATHF);
            fpout=fopen(filepath,"w");
            for (i=0;i<64;i++)
            {
                  for (j=0;j<64;j++)
                        fprintf(fpout,"%x ",data_matrix[i][j]);
                  fprintf(fpout,"\n");
            }
            fclose(fpout);
*/

  for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
      data_matrix[i][j] = matrix2[i][j];

}


int check_power(void)
{
  int power = 0;  // Suppongo che medipix sia spento
  int i, a = 0, cycles_dummy, err;

  switch (power_mode_tmp) {
      case 0:                                    // External
        Print_error(13);      // Warning external power supply
        power = 1;
        break;
      case 1:                                              // Internal
        switch_on = modify_register(analog_reg, 0, bit_all, "R");
        if ((switch_on & 0x3) == 3)
          power = 1;
              // Controlla i bit Vdd e Vdda per vedere se Medipix è acceso
      break;
  }

  return(power);
}


/*
```

```
int check_power(void)
{
      int power=0;  // Suppongo che medipix sia spento
      int i,a,cycles_dummy,err;

      switch (power_mode_tmp) {
            case 0:  // External
                  cycles_dummy=cycles_tmp;
                  cycles_tmp=256;
                  err=reset_and_setup();

                  if (err==0)
                  {
                   modify_register(status_reg ,turn_on,bit_config_rd,"W");

                   for (i=0; i<1000; i++)
                       a=modify_register(status_reg, 0, bit_config_rd, "R");

                   if (a) power=0;
                   else power=1;
                  }
                  else {
                        Print_error(11);  // Error 11: Acquisition Aborted
                  }

                  cycles_tmp=cycles_dummy;
                  break;
            case 1:  // Internal
                  switch_on=modify_register(analog_reg,0,bit_all,"R");
              if((switch_on & 0x3) == 3) power=1;
              // Controlla i bit Vdd e Vdda per vedere se Medipix è acceso
                  break;
      }

      return(power);
}

*/


void read_dac(void)
{
  int i;

  for(i = 0; i < 5; i++) {
      read_data(dac0_reg, dac_bias_act, 5, "m");
      read_data(dac1_reg, dac_bias_act + 5, 5, "m");
  }
}
```

Interface.h

```
/*****************************************************************************/
/* LabWindows/CVI User Interface Resource (UIR) Include File                 */
/* Copyright (c) National Instruments 2000. All Rights Reserved.             */
/*                                                                           */
/* WARNING: Do not add to, delete from, or otherwise modify the contents  */
/*          of this include file.                                            */
/*****************************************************************************/

#include <userint.h>

#ifdef __cplusplus
    extern "C" {
#endif

     /* Panels and Controls: */

#define  ADD_SUBT                        1
#define  ADD_SUBT_FILENAME1              2
#define  ADD_SUBT_FILENAME2              3
#define  ADD_SUBT_FILENAME3              4
#define  ADD_SUBT_FACTOR2                5
#define  ADD_SUBT_FACTOR1                6
#define  ADD_SUBT_RUN                    7     /* callback function: Add_Subt */
#define  ADD_SUBT_CHECKSUBT         8     /* callback function: dimmed_checkSubt
*/
#define   ADD_SUBT_CHECKADD                   9          /* callback function:
dimmed_checkAdd */
#define   ADD_SUBT_QUIT                        10        /* callback function:
GeneralQuit */
#define  ADD_SUBT_TEXTBOX                11
#define  ADD_SUBT_DECORATION_15          12
#define  ADD_SUBT_DECORATION_16          13
#define  ADD_SUBT_DECORATION_14          14
#define  ADD_SUBT_DECORATION_2           15
#define  ADD_SUBT_DECORATION1            16
#define  ADD_SUBT_DECORATION4            17
#define  ADD_SUBT_DECORATION5            18
#define  ADD_SUBT_DECORATION6            19
#define  ADD_SUBT_DECORATION3            20
#define  ADD_SUBT_DECORATION_3           21

#define  ANALISI                         2
#define  ANALISI_PULSER_MIN              2
#define  ANALISI_PULSER_MAX              3
#define  ANALISI_PULSER_STEP             4
#define   ANALISI_START                        5          /* callback function:
start_analisi */
#define  ANALISI_QUIT                    6       /* callback function: quit */
#define   ANALISI_COMMANDBUTTON                7          /* callback function:
load_thr_name */
#define  ANALISI_PULSES_ANIN             8
#define  ANALISI_FILENAME                9
```

```
#define  ANALISI_NUMBER                 10
#define  ANALISI_DATA_DIR               11
#define  ANALISI_GRAPH                  12
#define  ANALISI_DEV                    13
#define  ANALISI_MEAN                   14
#define  ANALISI_SUM                    15
#define  ANALISI_GRAF_FLAG              16
#define  ANALISI_NOISE                  17
#define  ANALISI_V_bias                 18
#define  ANALISI_V_comp                 19
#define  ANALISI_TEXTMSG                20
#define  ANALISI_DECORATION             21
#define  ANALISI_DECORATION_2           22

#define  AUTORAD                        3
#define  AUTORAD_IMAGE                  2
#define  AUTORAD_EXIT                   3
#define  AUTORAD_COLOR_BAR              4
#define  AUTORAD_RESUME_ACQ             5
#define  AUTORAD_STOP_ACQ               6
#define  AUTORAD_START_ACQ              7
#define  AUTORAD_ACQUISITION_TIME       8
#define  AUTORAD_SLIDE                  9
#define  AUTORAD_TOTAL_TIME             10
#define  AUTORAD_FILENAME               11
#define  AUTORAD_SHOW_IMAGE             12
#define  AUTORAD_OK                     13
#define  AUTORAD_SLIDE_COUNT            14
#define  AUTORAD_ACQUISITION_ON         15
#define  AUTORAD_STOP                   16
#define  AUTORAD_TIME                   17
#define  AUTORAD_DECORATION             18

#define  AUTOR_MOD                      4
#define  AUTOR_MOD_BINARYSWITCH         2
#define  AUTOR_MOD_OK                   3
#define  AUTOR_MOD_DECORATION           4
#define  AUTOR_MOD_TEXTMSG_2            5
#define  AUTOR_MOD_TEXTMSG              6

#define  BASIC_R_A                      5
#define  BASIC_R_A_TURN_ON_OFF          2
#define  BASIC_R_A_REGISTER             3
#define  BASIC_R_A_WRITE_READ           4       /* callback function: ctrlWrite
*/
#define  BASIC_R_A_VALUE_WRITTEN        5
#define  BASIC_R_A_VALUE_READ           6
#define  BASIC_R_A_MASK                 7
#define   BASIC_R_A_RUN                        8         /* callback function:
run_only_bit */
#define  BASIC_R_A_RUN2                  9       /* callback function: run_word
*/
#define   BASIC_R_A_WORD                       10        /* callback function:
ModifyWord */
#define  BASIC_R_A_BIT                  11      /* callback function: ModifyBit
*/
#define  BASIC_R_A_BACK                 12
```

```
#define   BASIC_R_A_QUIT                          13        /* callback function:
GeneralQuit */
#define  BASIC_R_A_DECORATION            14
#define  BASIC_R_A_DECORATION_4          15
#define  BASIC_R_A_DECORATION_5          16
#define  BASIC_R_A_DECORATION_6          17
#define  BASIC_R_A_DECORATION_8          18
#define  BASIC_R_A_MSG2                  19
#define  BASIC_R_A_MSG4                  20
#define  BASIC_R_A_MSG3                  21
#define  BASIC_R_A_MSG5                  22
#define  BASIC_R_A_MSG6                  23
#define  BASIC_R_A_DECORATION_2          24
#define  BASIC_R_A_MSG1                  25

#define  COUNTERS                        6
#define  COUNTERS_FILE_NAME              2
#define   COUNTERS_GO_TEST                3       /* callback function: Go_Test
*/
#define  COUNTERS_LOAD                   4       /* callback function: Load */
#define  COUNTERS_SAVE                   5       /* callback function: Save */
#define  COUNTERS_CREATE                 6       /* callback function: Create */
#define  COUNTERS_SHOW                   7       /* callback function: Show */
#define  COUNTERS_END_COLUMN             8
#define  COUNTERS_START_ROW              9
#define  COUNTERS_END_ROW                10
#define  COUNTERS_START_COLUMN           11
#define  COUNTERS_VALUE                  12
#define  COUNTERS_RUN_CREATE             13
#define  COUNTERS_RUN_LOAD               14
#define  COUNTERS_RUN_SAVE               15
#define  COUNTERS_BACK                   16
#define   COUNTERS_QUIT                           17        /* callback function:
GeneralQuit */
#define  COUNTERS_VIEW                   18      /* callback function: view */
#define  COUNTERS_DECORATION2            19
#define  COUNTERS_DECORATION3            20
#define  COUNTERS_DECORATION             21
#define  COUNTERS_DECORATION_2           22
#define  COUNTERS_DECORATION_3           23
#define  COUNTERS_DECORATION_4           24
#define  COUNTERS_TEXTMSG1               25
#define  COUNTERS_TEXTMSG3               26
#define  COUNTERS_TEXTMSG2               27
#define  COUNTERS_TEXTMSG4               28
#define  COUNTERS_TEXTMSG5               29
#define  COUNTERS_TEXTMSG7               30
#define  COUNTERS_TEXTMSG9               31
#define  COUNTERS_TEXTMSG6               32
#define  COUNTERS_DECORATION_5           33
#define  COUNTERS_TEXTMSG                34
#define  COUNTERS_TEXTMSG10              35

#define  ENABLE                          7
#define  ENABLE_NUMERIC1                 2
#define  ENABLE_CHECK_BEST_VTH           3       /* callback function: Best_Vthr
*/
```

```
#define  ENABLE_CHECK_VTH               4       /* callback function: Vthr */
#define  ENABLE_NUMERIC2                5
#define  ENABLE_NUMERIC3                6
#define  ENABLE_NUMERIC7_2              7
#define  ENABLE_NUMERIC4                8
#define  ENABLE_NUMERIC5                9
#define  ENABLE_NUMERIC7                10
#define  ENABLE_NUMERIC6                11
#define  ENABLE_FILEMASK                12
#define   ENABLE_RUN                     13        /* callback function:
Run_E_Mask */
#define   ENABLE_BACK                    14        /* callback function:
Back_E_Mask */
#define   ENABLE_BUTTONQUIT              15        /* callback function:
Enable_Quit */
#define  ENABLE_TEXTBOX                 16
#define  ENABLE_DECORATION1             17
#define  ENABLE_DECORATION2             18
#define  ENABLE_DECORATION4             19
#define  ENABLE_DECORATION7             20
#define  ENABLE_DECORATION8             21
#define  ENABLE_DECORATION9             22
#define  ENABLE_DECORATION3             23
#define  ENABLE_DECORATION10            24
#define  ENABLE_DECORATION5             25

#define  INPUT                          8
#define  INPUT_NEW_VAL_EN               2
#define  INPUT_NEW_VAL_T                3
#define  INPUT_OLD_EN                   4
#define  INPUT_OLD                      5
#define  INPUT_OK                       6
#define  INPUT_DONE                     7
#define  INPUT_CANCEL                   8
#define  INPUT_COL                      9
#define  INPUT_ROW                      10
#define  INPUT_NEW_VAL_TH               11
#define  INPUT_OLD_TH                   12
#define  INPUT_TEXTMSG_3                13
#define  INPUT_TEXTMSG                  14
#define  INPUT_TEXTMSG_2                15
#define  INPUT_DECORATION               16
#define  INPUT_DEC                      17

#define  INPUTRANGE                     9
#define  INPUTRANGE_STARTROW            2
#define  INPUTRANGE_ENDROW              3
#define  INPUTRANGE_STARTCOLUMN         4
#define  INPUTRANGE_ENDCOLUMN           5
#define  INPUTRANGE_OK                  6
#define  INPUTRANGE_DECORATION          7
#define  INPUTRANGE_DECORATION_2        8
#define  INPUTRANGE_TEXTMSG_2           9
#define  INPUTRANGE_TEXTMSG_3           10

#define  INSERT                         10
#define  INSERT_NEW_VAL_EN              2
```

```
#define  INSERT_NEW_VAL_T                3
#define  INSERT_DONE                     4
#define  INSERT_CANCEL                   5
#define  INSERT_NEW_VAL_TH               6
#define  INSERT_TEXTMSG                  7
#define  INSERT_DECORATION               8

#define  MASK                            11
#define   MASK_QUIT_2                           2         /* callback function:
GeneralQuit */
#define  MASK_QUIT                       3
#define  MASK_GO_TEST                     4       /* callback function: go_test
*/
#define  MASK_THRESHOLD                  5       /* callback function: Threshold
*/
#define  MASK_TESTBIT                     6       /* callback function: Testbit
*/
#define  MASK_ENABLE                     7       /* callback function: Enable */
#define  MASK_FILENAME                   8
#define  MASK_RUN                        9
#define  MASK_BACK2                      10
#define  MASK_VIEW                       11      /* callback function: view */
#define  MASK_DECORATION_2               12
#define  MASK_DECORATION_3               13
#define  MASK_DECORATION                 14
#define  MASK_DECORATION_4               15
#define  MASK_DECORATION_6               16
#define  MASK_TEXTMSG2                   17
#define  MASK_MSG1                       18
#define  MASK_MSG3                       19
#define  MASK_DECORATION_5               20
#define  MASK_TEXTMSG                    21
#define  MASK_MSG4                       22
#define  MASK_MSG2                       23

#define  MOD_BIAS_1                      12
#define  MOD_BIAS_1_DAC0                 2
#define  MOD_BIAS_1_DAC1                 3
#define  MOD_BIAS_1_DAC2                 4
#define  MOD_BIAS_1_DAC3                 5
#define  MOD_BIAS_1_DAC4                 6
#define  MOD_BIAS_1_DAC5                 7
#define  MOD_BIAS_1_DAC6                 8
#define  MOD_BIAS_1_DAC7                 9
#define  MOD_BIAS_1_DAC8                 10
#define  MOD_BIAS_1_DAC9                 11
#define   MOD_BIAS_1_OKDACBIAS1                 12         /* callback function:
Ok_Dac_Bias1 */
#define   MOD_BIAS_1_CANCELDACBIAS2            13         /* callback function:
CancelDACBias1 */
#define  MOD_BIAS_1_BIASMSG              14

#define  MOD_BIAS_2                      13
#define  MOD_BIAS_2_DAC0                 2
#define  MOD_BIAS_2_DAC1                 3
#define  MOD_BIAS_2_DAC2                 4
#define  MOD_BIAS_2_DAC3                 5
```

```
#define  MOD_BIAS_2_DAC4                6
#define  MOD_BIAS_2_DAC5                7
#define  MOD_BIAS_2_DAC6                8
#define  MOD_BIAS_2_DAC7                9
#define  MOD_BIAS_2_DAC8                10
#define  MOD_BIAS_2_DAC9                11
#define   MOD_BIAS_2_OKDACBIAS2                12         /*  callback  function:
Ok_Dac_Bias2 */
#define   MOD_BIAS_2_CANCELDACBIAS2           13         /*  callback  function:
CancelDACBias2 */
#define  MOD_BIAS_2_BIASMSG             14

#define  MOTOR                          14
#define  MOTOR_ZAXIS                    2
#define  MOTOR_YAXIS                    3
#define  MOTOR_XAXIS                    4
#define  MOTOR_MOVEABS                  5
#define  MOTOR_MOVEREL                  6
#define  MOTOR_EXIT                     7
#define  MOTOR_ZCOORD                   8
#define  MOTOR_YCOORD                   9
#define  MOTOR_XCOORD                   10
#define  MOTOR_OK                       11
#define  MOTOR_TELL_P                   12
#define  MOTOR_RESET_M                  13
#define  MOTOR_STOP                     14
#define  MOTOR_TEXTMSG                  15
#define  MOTOR_TEXTMSG_2                16
#define  MOTOR_DECORATION_3             17
#define  MOTOR_DECORATION               18
#define  MOTOR_DECORATION_2             19
#define  MOTOR_MOVE_TYPE                20

#define  M_MODALITY                     15
#define  M_MODALITY_MOD_GROUP           2
#define  M_MODALITY_MOD_EXIT            3
#define  M_MODALITY_MOD_SINGLE          4
#define  M_MODALITY_TEXTMSG             5
#define  M_MODALITY_TEXTMSG_2           6
#define  M_MODALITY_DECORATION          7
#define  M_MODALITY_TEXTMSG_4           8
#define  M_MODALITY_MODIFY_TXT          9

#define  PANEL                          16
#define  PANEL_TAU0                     2
#define  PANEL_TAU2                     3
#define  PANEL_TAU1                     4
#define  PANEL_TRIG_MODE                5
#define   PANEL_SHOW_ACQ_PARAMS                6          /*  callback  function:
Show_active_acq_params */
#define  PANEL_CHIP_ID                  7
#define  PANEL_CYCLES                   8
#define  PANEL_CONFIG_MODE              9
#define  PANEL_RAM_OFFSET               10
#define   PANEL_SHOW_ACT_SETTINGS             11         /*  callback  function:
Show_act_settings */
#define  PANEL_ANIN_CYCLES              12
```

```
#define  PANEL_SLEEP_ANIN_DELAY          13
#define  PANEL_SLEEP_ANIN_FREQ           14
#define   PANEL_SHOW_ANIN_PARAMS              15         /*  callback  function:
show_act_anin */
#define  PANEL_V_AU_SWITCH               16
#define   PANEL_MOD_DAC_BIAS                  17         /*  callback  function:
modify_dac_bias */
#define   PANEL_SHOW_DAC_PARAMETERS          18         /*  callback  function:
show_act_DAC_params */
#define   PANEL_TURN_OFF                      19         /*  callback  function:
turn_off_medipix */
#define   PANEL_LOAD_MASK                     20         /*  callback  function:
load_mask_on_Medipix */
#define  PANEL_NEW_ACQUISITION_2          21       /* callback function: Set_DAC
*/
#define   PANEL_START_ACQ                     22         /*  callback  function:
Start_acquisition */
#define   PANEL_SHOW_IMAGE                    23         /*  callback  function:
Show_image */
#define  PANEL_FILENAME                  24
#define  PANEL_MASKFILE                  25
#define  PANEL_STATE_ON                  26
#define  PANEL_STATE_OFF                 27
#define  PANEL_LOAD                      28
#define  PANEL_NOT_LOAD                  29
#define   PANEL_TURN_ON                       30         /*  callback  function:
turn_on_medipix */
#define  PANEL_IMAGE_READY               31
#define  PANEL_END_ACQUISITION           32
#define   PANEL_AUTORADIOGRAPHY              33         /*  callback  function:
Autoradiography */
#define   PANEL_CHECK_POWER                   34         /*  callback  function:
update_power */
#define  PANEL_MOTOR                     35       /* callback function: Motor */
#define  PANEL_ANIN_TEST                 36       /* callback function: test_ANIN
*/
#define  PANEL_TOTAL_QUIT                37       /* callback function: Quit2 */
#define  PANEL_POWER_MODE                38
#define  PANEL_TEXTMSG_6                 39
#define  PANEL_TEXTMSG_9                 40
#define  PANEL_TEXTMSG_7                 41
#define  PANEL_DECORATION_5              42
#define  PANEL_DECORATION_6              43
#define  PANEL_DECORATION_7              44
#define  PANEL_DECORATION_8              45
#define  PANEL_LOGO_2                    46
#define  PANEL_TEXTMSG_5                 47
#define  PANEL_DECORATION_3              48
#define  PANEL_TEXTMSG_10                49
#define  PANEL_TEXTMSG                   50
#define  PANEL_DECORATION_2              51
#define  PANEL_DECORATION                52
#define  PANEL_DECORATION_4              53
#define   PANEL_TIMER                         54         /*  callback  function:
Main_timer */
#define  PANEL_DECORATION_9              55
#define  PANEL_LOGO                      56
```

```
#define  PANEL_PICTURE_2                57
#define  PANEL_PICTURE                  58
#define  PANEL_TEXTMSG_3                59
#define  PANEL_TEXTMSG_8                60
#define  PANEL_TEXTMSG_4                61

#define  PIXELS_CAL                     17
#define  PIXELS_CAL_STRING3             2
#define  PIXELS_CAL_RUN                 3       /* callback function: Run */
#define   PIXELS_CAL_COMMANDBUTTON            4           /*  callback  function:
New_Data_File */
#define   PIXELS_CAL_COMMANDBUTTON_2         5           /*  callback  function:
New_Back_File */
#define   PIXELS_CAL_QUIT                      6          /*  callback  function:
GeneralQuit */
#define  PIXELS_CAL_STRING1             7
#define  PIXELS_CAL_STRING2             8
#define  PIXELS_CAL_TEXTBOX             9
#define  PIXELS_CAL_DECORATION          10
#define  PIXELS_CAL_DECORATION_2        11
#define  PIXELS_CAL_DECORATION_3        12

#define  PROFILE                        18
#define  PROFILE_SELECT                 2
#define  PROFILE_NUM_COL                3
#define  PROFILE_NUM_ROW                4
#define  PROFILE_START_COL              5
#define  PROFILE_START_ROW              6
#define  PROFILE_SHOW                   7
#define  PROFILE_OK                     8
#define  PROFILE_DONE                   9
#define  PROFILE_GRAPH                  10
#define  PROFILE_COUNT                  11
#define  PROFILE_PIXEL                  12
#define  PROFILE_TEXTMSG                13
#define  PROFILE_DECORATION             14
#define  PROFILE_TIMER                   15 /* callback function: Profile_timer
*/

#define  PSELECT                        19
#define  PSELECT_CALIBRATION            2 /* callback function: Run_Calibration
*/
#define  PSELECT_VTH                    3 /* callback function: Run_Voltage_Aut
*/
#define  PSELECT_DECORATION             4
#define  PSELECT_TEXTMSG2               5
#define  PSELECT_TEXTMSG3               6
#define  PSELECT_TEXTMSG1               7

#define  RAM_FIFO                       20
#define  RAM_FIFO_RAM                   2       /* callback function: ram */
#define  RAM_FIFO_FIFO                  3       /* callback function: fifo */
#define  RAM_FIFO_QUIT                  4  /* callback function: GeneralQuit */
#define  RAM_FIFO_QUIT_RF               5       /* callback function: QuitRF */
#define  RAM_FIFO_OFFSET                6
#define  RAM_FIFO_READ                  7
#define  RAM_FIFO_BACK                  8
```

```
#define  RAM_FIFO_RUN_FIFO              9
#define  RAM_FIFO_RUN_RAM               10
#define  RAM_FIFO_DECORATION_5          11
#define  RAM_FIFO_DECORATION_4          12
#define  RAM_FIFO_RESULT1               13
#define  RAM_FIFO_RESULT2               14
#define  RAM_FIFO_RESULT3               15
#define  RAM_FIFO_RESULT4               16
#define  RAM_FIFO_RESULT7               17
#define  RAM_FIFO_DECORATION_6          18
#define  RAM_FIFO_DECORATION_7          19
#define  RAM_FIFO_DECORATION            20
#define  RAM_FIFO_DECORATION_8          21
#define  RAM_FIFO_DECORATION_3          22
#define  RAM_FIFO_DECORATION_9          23
#define  RAM_FIFO_RESULT7_2             24
#define  RAM_FIFO_RESULT6               25
#define  RAM_FIFO_RESULT5               26

#define  SET_COLORS                     21
#define  SET_COLORS_HIGH                2
#define  SET_COLORS_LOW                 3
#define  SET_COLORS_NUM                 4
#define  SET_COLORS_INTERP_PIX          5
#define  SET_COLORS_INTERP_COL          6
#define  SET_COLORS_PLOT                7
#define  SET_COLORS_CANCEL              8
#define  SET_COLORS_MAX_VAL             9
#define  SET_COLORS_MIN_VAL             10
#define  SET_COLORS_TEXTMSG             11
#define  SET_COLORS_TEXTMSG_2           12
#define  SET_COLORS_DECORATION          13
#define  SET_COLORS_DECORATION_2        14
#define  SET_COLORS_DECORATION_3        15
#define  SET_COLORS_TEXTMSG_3           16

#define  SHOW                           22
#define  SHOW_PROFILE                   2      /* callback function: Profile */
#define  SHOW_RESTORE                   3      /* callback function: Restore */
#define  SHOW_ZOOM                      4    /* callback function: Zoom_in */
#define  SHOW_COLORS                    5  /* callback function: Colors */
#define  SHOW_IMAGE                     6
#define  SHOW_QUIT                      7 /* callback function: Quit_Show_Image
*/
#define  SHOW_COLOR_BAR                 8
#define  SHOW_LOAD                      9   /* callback function: Load_image */
#define  SHOW_SAVE                      10  /* callback function: Save_image */
#define  SHOW_Y_COORD                   11
#define  SHOW_COUNTS                    12
#define  SHOW_X_COORD                   13
#define  SHOW_LOCAL_COUNTS_2            14
#define  SHOW_LOCAL_MEDIA               15
#define  SHOW_MEDIA                     16
#define  SHOW_INTEGRAL                  17
#define  SHOW_WEIGHING                   18      /* callback function: weighing
*/
```

```
#define  SHOW_LOCAL_COUNTS                19 /* callback function: local_counts
*/
#define  SHOW_SUBTRACT                    20    /* callback function: subtract */
#define  SHOW_FILE_READ                   21
#define  SHOW_TEXTMSG                     22
#define  SHOW_DECORATION                  23
#define  SHOW_TEXTMSG_2                   24
#define  SHOW_DECORATION_2                25
#define  SHOW_TEXTMSG_3                   26
#define  SHOW_DECORATION_3                27
#define  SHOW_TEXTMSG_4                   28
#define  SHOW_DECORATION_4                29
#define   SHOW_TIMER                            30        /* callback function:
Update_Cursor_Pos */
#define  SHOW_TEXTMSG_5                   31
#define  SHOW_TEXTMSG_6                   32
#define  SHOW_TEXTMSG_7                   33

#define  SHOWIMAGE2                       23
#define  SHOWIMAGE2_CALL_IMAGE            2   /* callback function: Show_image */
#define  SHOWIMAGE2_TEXTMSG_IM            3

#define  SHOW_MASK                        24
#define  SHOW_MASK_GRAPH                  2
#define  SHOW_MASK_OK                     3  /* callback function: Quit_MaskGraph
*/
#define  SHOW_MASK_COLOR_INDEX            4
#define  SHOW_MASK_DECORATION             5

#define  VERSION                          25
#define  VERSION_QUIT                     2
#define  VERSION_MESSAGGIO                3
#define  VERSION_PICTURE                  4
#define  VERSION_MESSAGGIO3               5
#define  VERSION_MESSAGGIO3_2             6
#define  VERSION_PICTURE_2                7
#define  VERSION_MESSAGGIO2               8

#define  VTH_CAL                          26
#define  VTH_CAL_CHECKNUM5                2   /* callback function: En_Thr_adj */
#define  VTH_CAL_CHECKNUM6                3  /* callback function: En_rep_acq */
#define    VTH_CAL_CHECKLOADED_MASK_FILE     4            /*  callback  function:
En_Loaded_msk */
#define  VTH_CAL_NUM5                     5
#define  VTH_CAL_NUM6_1                   6
#define  VTH_CAL_NUM6_2                   7
#define  VTH_CAL_RING1                    8
#define  VTH_CAL_BUTTON4                  9
#define  VTH_CAL_NUM4_2                   10
#define  VTH_CAL_NUM1                     11
#define  VTH_CAL_NUM2                     12
#define  VTH_CAL_NUM3                     13
#define  VTH_CAL_NUM4                     14
#define  VTH_CAL_BUTTON1                  15
#define  VTH_CAL_TEXTBOX1                 16
#define  VTH_CAL_NUM7                     17
#define  VTH_CAL_NUM8                     18
```

```
#define  VTH_CAL_NUM10                  19
#define  VTH_CAL_NUM11                  20
#define  VTH_CAL_BUTTON2                21
#define  VTH_CAL_RING2                  22
#define  VTH_CAL_BUTTON5                23
#define  VTH_CAL_NUM12                  24
#define  VTH_CAL_NUM13                  25
#define  VTH_CAL_NUM14                  26
#define  VTH_CAL_BUTTON3                27
#define  VTH_CAL_RUN                    28      /* callback function: Start */
#define  VTH_CAL_LED1                   29
#define  VTH_CAL_LED6                   30
#define  VTH_CAL_LED5                   31
#define  VTH_CAL_LED4                   32
#define  VTH_CAL_LED3                   33
#define  VTH_CAL_LED2                   34
#define  VTH_CAL_STOP                   35      /* callback function: Stop */
#define  VTH_CAL_QUIT                   36      /* callback function: Exit */
#define   VTH_CAL_OUTPUT                       37        /* callback function:
Show_StandardInputOutput */
#define   VTH_CAL_GO                           38        /* callback function:
Sel_Voltage */
#define  VTH_CAL_DECORATION             39
#define  VTH_CAL_RUNOPTIONS             40
#define  VTH_CAL_DECORATION_2           41
#define  VTH_CAL_DECORATION_6           42
#define  VTH_CAL_DECORATION_5           43
#define  VTH_CAL_DECORATION_3           44


     /* Menu Bars, Menus, and Menu Items: */

#define  MENU                           1
#define  MENU_SYSTEMSE                  2
#define  MENU_SYSTEMSE_OPEN             3    /* callback function: Open_file */
#define  MENU_SYSTEMSE_SAVE             4   /* callback function: Save_file */
#define  MENU_SYSTEMSE_SEPARATOR        5
#define  MENU_SYSTEMSE_QUIT             6       /* callback function: Quit */
#define  MENU_MASK                      7
#define   MENU_MASK_OPENMASKFILE              8          /* callback function:
Open_mask_file */
#define   MENU_MASK_LOAD_UPLOAD               9          /* callback function:
Load_Upload */
#define   MENU_MASK_SAVEMASKFILE              10         /* callback function:
Save_mask_file */
#define  MENU_MASK_MODIFYMASK           11
#define  MENU_MASK_MODIFYMASK_SUBMENU   12
#define   MENU_MASK_MODIFYMASK_EN_MASK       13          /* callback function:
Modify_Enable_mask */
#define   MENU_MASK_MODIFYMASK_TEST_MASK    14          /* callback function:
Modify_Testbit_Mask */
#define   MENU_MASK_MODIFYMASK_TH_MASK       15          /* callback function:
Modify_Threshold_Mask */
#define  MENU_MASK_SHOWMASK             16
#define  MENU_MASK_SHOWMASK_SUBMENU     17
#define   MENU_MASK_SHOWMASK_ENABLEMASK     18          /* callback function:
Show_enable_mask */
```

```
#define    MENU_MASK_SHOWMASK_TESTBITMASK    19          /*  callback  function:
Show_testbit_mask */
#define    MENU_MASK_SHOWMASK_THRESOLDMASK  20          /*  callback  function:
Show_threshold_mask */
#define  MENU_MASK_DISPLAYM              21
#define  MENU_MASK_DISPLAYM_SUBMENU      22
#define    MENU_MASK_DISPLAYM_ENABLEGRAPH    23          /*  callback  function:
Enable_graph */
#define    MENU_MASK_DISPLAYM_TESTBITGRAPH  24          /*  callback  function:
Testbit_graph */
#define    MENU_MASK_DISPLAYM_THRESOLDGRAPH  25         /*  callback  function:
Threshold_graph */
#define  MENU_BASIC                      26
#define   MENU_BASIC_BUSACCESS                27        /*  callback  function:
Bus_access */
#define   MENU_BASIC_RAM_FIFO                 28        /*  callback  function:
Ram_Fifo_test */
#define   MENU_BASIC_COUNTER_TEST             29        /*  callback  function:
Counter_Test */
#define  MENU_BASIC_MASK_TEST            30   /* callback function: Mask_Test */
#define   MENU_BASIC_ENABLE_MASK              31        /*  callback  function:
Enable_Mask */
#define  MENU_BASIC_VTH_CAL              32  /* callback function: vth_cal */
#define  MENU_BASIC_analysis             33   /* callback function: Analisi */
#define    MENU_BASIC_Equalization                34    /*  callback  function:
Thres_Equalization */
#define  MENU_IMAGE                      35
#define   MENU_IMAGE_SHOWIMMAGINE             36        /*  callback  function:
Showimage2 */
#define  MENU_IMAGE_ADD_SUBT_FILES       37    /* callback function: add_subt */
#define  MENU_IMAGE_PIXELS_CAL           38      /* callback function: Calib */
#define  MENU_IMAGE_LOWPASSFILTER        39   /* callback function: LP_filtro */
#define  MENU_USER                       40
#define   MENU_USER_ALPHA_TEST                41        /*  callback  function:
BasicAccess */
#define  MENU_INFO_VERSION               42
#define  MENU_INFO_VERSION_MEDISOFT      43    /* callback function: Version */


     /* Callback Prototypes: */

void CVICALLBACK add_subt(int menubar, int menuItem, void *callbackData, int
panel);
void CVICALLBACK Analisi(int menubar, int menuItem, void *callbackData, int
panel);
int    CVICALLBACK Autoradiography(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK Back_E_Mask(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK BasicAccess(int menubar, int menuItem, void *callbackData, int
panel);
int    CVICALLBACK Best_Vthr(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Bus_access(int menubar, int menuItem, void *callbackData, int
panel);
void CVICALLBACK Calib(int menubar, int menuItem, void *callbackData, int
panel);
```

```
int    CVICALLBACK CancelDACBias1(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK CancelDACBias2(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK Colors(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
void CVICALLBACK Counter_Test(int menubar, int menuItem, void *callbackData, int
panel);
int  CVICALLBACK Create(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int     CVICALLBACK ctrlWrite(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK dimmed_checkAdd(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK dimmed_checkSubt(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK Enable(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
void CVICALLBACK Enable_graph(int menubar, int menuItem, void *callbackData, int
panel);
void CVICALLBACK Enable_Mask(int menubar, int menuItem, void *callbackData, int
panel);
int     CVICALLBACK Enable_Quit(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK En_Loaded_msk(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int     CVICALLBACK En_rep_acq(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int     CVICALLBACK En_Thr_adj(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK Exit(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK fifo(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int     CVICALLBACK GeneralQuit(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK Go_Test(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK Load(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int     CVICALLBACK Load_image(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int   CVICALLBACK load_mask_on_Medipix(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK load_thr_name(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Load_Upload(int menubar, int menuItem, void *callbackData, int
panel);
int     CVICALLBACK local_counts(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK LP_filtro(int menubar, int menuItem, void *callbackData, int
panel);
int     CVICALLBACK Main_timer(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Mask_Test(int menubar, int menuItem, void *callbackData, int
panel);
```

```
int     CVICALLBACK   ModifyBit(int   panel,   int   control,   int   event,   void
*callbackData, int eventData1, int eventData2);
int     CVICALLBACK   ModifyWord(int   panel,   int   control,   int   event,   void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  modify_dac_bias(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
void   CVICALLBACK   Modify_Enable_mask(int   menubar,   int   menuItem,   void
*callbackData, int panel);
void   CVICALLBACK   Modify_Testbit_Mask(int   menubar,   int   menuItem,   void
*callbackData, int panel);
void   CVICALLBACK   Modify_Threshold_Mask(int   menubar,   int   menuItem,   void
*callbackData, int panel);
int   CVICALLBACK Motor(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int    CVICALLBACK  New_Back_File(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  New_Data_File(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  Ok_Dac_Bias1(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  Ok_Dac_Bias2(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Open_file(int menubar, int menuItem, void *callbackData, int
panel);
void CVICALLBACK Open_mask_file(int menubar, int menuItem, void *callbackData,
int panel);
int   CVICALLBACK Profile(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int    CVICALLBACK  Profile_timer(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Quit(int menubar, int menuItem, void *callbackData, int panel);
int   CVICALLBACK Quit2(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int   CVICALLBACK QuitRF(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int    CVICALLBACK  Quit_MaskGraph(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  Quit_Show_Image(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int   CVICALLBACK ram(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
void CVICALLBACK Ram_Fifo_test(int menubar, int menuItem, void *callbackData,
int panel);
int   CVICALLBACK Restore(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int   CVICALLBACK Run(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int    CVICALLBACK  Run_Calibration(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int     CVICALLBACK   Run_E_Mask(int   panel,   int   control,   int   event,   void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  run_only_bit(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  Run_Voltage_Aut(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int   CVICALLBACK run_word(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
```

```
int  CVICALLBACK Save(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
void CVICALLBACK Save_file(int menubar, int menuItem, void *callbackData, int
panel);
int     CVICALLBACK  Save_image(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Save_mask_file(int menubar, int menuItem, void *callbackData,
int panel);
int     CVICALLBACK  Sel_Voltage(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK Set_DAC(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK Show(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
void CVICALLBACK Showimage2(int menubar, int menuItem, void *callbackData, int
panel);
int  CVICALLBACK Show_active_acq_params(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  show_act_anin(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int   CVICALLBACK show_act_DAC_params(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int   CVICALLBACK Show_act_settings(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Show_enable_mask(int menubar, int menuItem, void *callbackData,
int panel);
int     CVICALLBACK  Show_image(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int   CVICALLBACK Show_StandardInputOutput(int panel, int control, int event,
void *callbackData, int eventData1, int eventData2);
void   CVICALLBACK   Show_testbit_mask(int   menubar,   int   menuItem,   void
*callbackData, int panel);
void   CVICALLBACK   Show_threshold_mask(int   menubar,   int   menuItem,   void
*callbackData, int panel);
int  CVICALLBACK Start(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int   CVICALLBACK Start_acquisition(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK  start_analisi(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int  CVICALLBACK Stop(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK subtract(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK Testbit(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
void CVICALLBACK Testbit_graph(int menubar, int menuItem, void *callbackData,
int panel);
int     CVICALLBACK  test_ANIN(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
int     CVICALLBACK  Threshold(int  panel,  int  control,  int  event,  void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Threshold_graph(int menubar, int menuItem, void *callbackData,
int panel);
void   CVICALLBACK   Thres_Equalization(int   menubar,   int   menuItem,   void
*callbackData, int panel);
```

```
int    CVICALLBACK turn_off_medipix(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK turn_on_medipix(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK Update_Cursor_Pos(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
int    CVICALLBACK update_power(int panel, int control, int event, void
*callbackData, int eventData1, int eventData2);
void CVICALLBACK Version(int menubar, int menuItem, void *callbackData, int
panel);
int  CVICALLBACK view(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK Vthr(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
void CVICALLBACK vth_cal(int menubar, int menuItem, void *callbackData, int
panel);
int  CVICALLBACK weighing(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);
int  CVICALLBACK Zoom_in(int panel, int control, int event, void *callbackData,
int eventData1, int eventData2);


#ifdef __cplusplus
    }
#endif
```

Main.c

```
#include <utility.h>
#include <analysis.h>
#include <formatio.h>
#include <cvirte.h>          /*  Needed  if  linking  in  external  compiler;
harmless otherwise */
#include <userint.h>
#include "Interface.h"
#include "main.h"


/*****************************************************************************
******     Prototipi delle funzioni grafiche     ****************************
*****************************************************************************/
void init_graph_interface(char arc_file_default[FILENAMELEN],
                          char mask_file_default[FILENAMELEN]);
                          // Imposta i valori delle variabili temporanee lette
                          // dai file di archivio e di maschere nell'interfaccia
grafica
void set_arc_params(char arc_file[FILENAMELEN]);      //   Impostazione   valori
delle variabili temporanee
void set_mask_params(char mask_file[FILENAMELEN]);  // Impostazione valori del
vettore di maschere
void Show_matrix(char *mask_name, int *start_row, int *end_row, int *start_col,
                 int *end_col);
              // Visualizza la sottomatrice di maschere di dimensioni
                                    // [end_row-start_row][end_col-start_col]
int GetParams(int m[64][64], ColorMapEntry colors[], int *, int *, int *, int
*);
void GetGraphMaskParams(int m[64][64], ColorMapEntry colors[], int *hiColor, int
numColors);
void plot_image(int matrix[64][64]);
void set_image_values(void);
void Start_acq_autoradiography (void);
void plot_slide(int matrix[64][64]);
int    GetSlideParams (int m[64][64], ColorMapEntry colors[], int *, int *, int
*, int *);

/*****************************************************************************
****    Fine prototipi delle funzioni grafiche     ************************
*****************************************************************************/

/*****************************************************************************
******** prototipi funzioni aggiuntive ***************************************
*****************************************************************************/

void copiastr(char*);
void Close(void);

/*****************************************************************************/

int panelHandle;
static int menuBarHandle, modifyDACBias1, modifyDACBias2,
```

```
          modifyModality, inputCoord, inputRange, insertNewVal,
          modifyMask, showMaskGraph, showImage, showAutorad, autoradType,
          setColors, profile, motor;
//int  basic_r_a, ram_fifo, Test_Counters, Test_Mask, E_Mask, Add_Subt_Files,
//     Pixels_Cal, Vth_Cal, PSelect;


int main (int argc, char *argv[])
{
  short int ret;
  char arc_file_default[FILENAMELEN];       // Nome del file default di archivio
  char mask_file_default[FILENAMELEN];           //  Nome  del  file  default  di
maschere
/*  SetCtrlAttribute (PANEL, PANEL_TEXTMSG_2, ATTR_VISIBLE, 0);   */
/*****************************************************************************
***************      LOADING PANELS              ****************************
************ Caricamento dei pannelli utilizzati  ****************************
*****************************************************************************/

  if (InitCVIRTE (0, argv, 0) == 0) // Needed if linking in external
                                    //   compiler; harmless otherwise
      return -1;      /* out of memory */
  if ((panelHandle = LoadPanel (0, "Interface.uir", PANEL)) < 0)
      return -1;     // Caricamento pannello principale
//  SetCtrlAttribute (panelHandle, PANEL_TEXTMSG_2, ATTR_VISIBLE, 0);

  if ((menuBarHandle = LoadMenuBar (panelHandle, "Interface.uir", MENU)) < 0)
      return -1;     // Caricamento barra dei menu
  if ((modifyDACBias1 = LoadPanel (panelHandle, "Interface.uir", MOD_BIAS_1)) <
0)
      return -1;     // Caricamento pannello per la modifca
                     // delle tensioni di polarizzazione (espresse in a.u.)
  if ((modifyDACBias2 = LoadPanel (panelHandle, "Interface.uir", MOD_BIAS_2)) <
0)
      return -1;     // Caricamento pannello per la modifca
                     // delle tensioni di polarizzazione (espresse in volt)
  if ((modifyModality = LoadPanel (panelHandle, "Interface.uir", M_MODALITY)) <
0)
      return -1;     // Caricamento pannello per la selezione della modalità
                     // di modifica della maschera (singolo elemento o multipla)
  if ((inputCoord = LoadPanel (panelHandle, "Interface.uir", INPUT)) < 0)
      return -1;     // Caricamento pannello per l'inserimento dell'elemento
                           // della matrice delle maschere da visualizzare o
modificare
                     // (modalità singola)
  if ((inputRange = LoadPanel (panelHandle, "Interface.uir", INPUTRANGE)) < 0)
      return -1;     // Caricamento pannello per l'inserimento dell'intervallo
                          // di righe/colonne della matrice delle maschere da
visualizzare
                     // o modificare (modalità multipla)
  if ((insertNewVal = LoadPanel (panelHandle, "Interface.uir", INSERT)) < 0)
      return -1;     // Caricamento pannello per l'inserimento del nuovo valore
                     // dell'elemento della matrice delle maschere
  if ((showMaskGraph = LoadPanel (panelHandle, "Interface.uir", SHOW_MASK)) < 0)
      return -1;     // Caricamento pannello per la visualizzazione del grafico
                     // dei valori delle maschere
  if ((showImage = LoadPanel (panelHandle, "Interface.uir", SHOW)) < 0)
```

```
      return -1;          //   Caricamento   pannello   per   la   visualizzazione
dell'immagine

  if ((showAutorad = LoadPanel (panelHandle, "Interface.uir", AUTORAD)) < 0)
      return -1;          //   Caricamento   pannello   per   la   visualizzazione
dell'autoradiografia
  if ((autoradType = LoadPanel (panelHandle, "Interface.uir", AUTOR_MOD)) < 0)
      return -1;     // Caricamento pannello per la l'impostazione del tipo di
                     // autoradiografia (singola o integrale)

  if ((setColors = LoadPanel (panelHandle, "Interface.uir", SET_COLORS)) < 0)
      return -1;     // Caricamento pannello per l'impostazione dei colori
                     // dell'immagine e dei valori di minimo e massimo
  if ((profile = LoadPanel (panelHandle, "Interface.uir", PROFILE)) < 0)
      return -1;     // Caricamento pannello per la visualizzazione del profilo
  if ((motor = LoadPanel (panelHandle, "Interface.uir", MOTOR)) < 0)
      return -1;     // Caricamento pannello per la visualizzazione del profilo

  basic_r_a = LoadPanel (panelHandle, "interface.uir", BASIC_R_A);
  ram_fifo = LoadPanel (panelHandle, "interface.uir", RAM_FIFO);
  Test_Counters = LoadPanel (panelHandle, "interface.uir", COUNTERS);
  Test_Mask=LoadPanel(panelHandle,"interface.uir",MASK);
  E_Mask=LoadPanel(panelHandle,"interface.uir",ENABLE);
  Add_Subt_Files=LoadPanel(panelHandle,"interface.uir",ADD_SUBT);
  Pixels_Cal=LoadPanel(panelHandle,"interface.uir",PIXELS_CAL);
  Vth_Cal=LoadPanel(panelHandle,"interface.uir",VTH_CAL);
  PSelect = LoadPanel (Vth_Cal, "interface.uir", PSELECT);
  analisi = LoadPanel (panelHandle, "interface.uir", ANALISI);
  versione=LoadPanel(panelHandle,"interface.uir",VERSION);
  showimage2=LoadPanel(panelHandle,"interface.uir",SHOWIMAGE2);
/*****************************************************************************
********  Fine caricamento dei pannelli utilizzati  ***************************
*****************************************************************************/

  SetCtrlAttribute(panelHandle, PANEL_TOTAL_QUIT, ATTR_VISIBLE, 0);
/* Questa istruzione serve per nasconere il pulsante, sull'interfaccia
   principale, relativo alla chiusura di Medipix tramite la consueta
   icona X in alto a destra della finestra Windows.*/
  GetProjectDir(Directory_Name);
  strcat(Directory_Name, "\\files");

  DisplayPanel(panelHandle);

/* ************************************************** */
/* ********* do not remove these 3 lines *************** */
/* ************************************************** */
  if (MISSING_MEDIPIX==0 && MISSING_HARDWARE ==0) {
    ret=init_VME;
    make_address();
    init_board();
  }
/* ************************************************** */
/* ************************************************** */
/* ************************************************** */

/* ************************************************** */
/* *********         user code area           *********** */
```

```
/* ************************************************** */

  sprintf(arc_file_default, "%sdefault.arch",PATHF2);
// default .arch filename
// Nome del file di archivio di default
  sprintf(mask_file_default, "%sdefault.msk",PATHF2);
// default .msk filename
// Nome del file di maschere di default

  load_file_arc(arc_file_default);

  init_graph_interface(arc_file_default,mask_file_default);
// Imposta i valori delle variabili lette nell'interfaccia grafica

  set_default(mask_file_default);

/* ************************************************** */
/* *********         end of user code area    *********** */
/* ************************************************** */

  RunUserInterface ();

/* ************************************************** */
/* ************ do not remove this line  *************** */
/* ************************************************** */
  if (MISSING_MEDIPIX==0)
    ret=close_VME;
/* ************************************************** */
/* ************************************************** */
/* ************************************************** */
  return 0;

}


/*****************************************************************************
*****************************************************************************
**************************    Barra dei menu  ********************************
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
*************************  Menu System Settings  ****************************
*****************************************************************************/
void CVICALLBACK Open_file (int menuBar, int menuItem, void *callbackData,
                            int panel)
{
  char filename[300];     // Path selezionato dall'utente
  int selection;          // Indica se il file selezionato è presente

//   selection = FileSelectPopup ("\\cvi401\\Roberto\\medipix\\files", "*.arch",
"*.arch",
  selection = FileSelectPopup (Directory_Name, "*.arch", "*.arch",
                                                  "Open   Temporary   File",
VAL_LOAD_BUTTON,
                                           0, 0, 1, 0, filename);
  if (selection==1) {
```

```
    load_file_arc(filename);
    set_arc_params(filename);
  }
}


void CVICALLBACK Save_file (int menuBar, int menuItem, void *callbackData,
                            int panel)
{
  char filename[300];     // Path selezionato dall'utente
  int selection;          // Indica se il file selezionato è presente

  selection = FileSelectPopup (Directory_Name, "*.arch", "*.arch",
                               "Save Active File in", VAL_SAVE_BUTTON,
                               0, 0, 1, 0, filename);

  if (selection!=0)        // Non vi sono errori
    save_file(filename);

// N.B.: Se il file selezionato esiste già il sistema mostrerà
//           una finestra dove chiede all'utente se desidera sovrascrivere
//           il file esistente
}


void CVICALLBACK Quit (int menuBar, int menuItem, void *callbackData,
                       int panel)
{
  QuitUserInterface (0);
}
/*****************************************************************************
******************** Fine Menu System Settings  ****************************
*****************************************************************************/


/*****************************************************************************
*****************************  Menu Mask  ************************************
*****************************************************************************/
void CVICALLBACK Open_mask_file (int menuBar, int menuItem, void *callbackData,
                                 int panel)
{
  char filename[512];     // Path selezionato dall'utente
  int selection;          // Indica se il file selezionato è presente
  int i,j;
  int enable_mask[64][64];
  int testbit_mask[64][64];
  int threshold_mask[64][64];

  selection = FileSelectPopup (Directory_Name, "*.msk", "*.msk",
                               "Open Mask File", VAL_LOAD_BUTTON,
                               0, 0, 1, 0, filename);
  if (selection==1) {
    read_file_mask(filename, enable_mask, testbit_mask, threshold_mask);
    set_mask_params(filename);
  }

   /* here config_matrix is build starting from enable_mask,
```

```
            testbit_mask,threshold_mask */

  for(i=0;i<64;i++)
    for(j=0;j<64;j++)
        config_matrix[i][j]= ((threshold_mask[i][j]&1)<<4)+
                             ((threshold_mask[i][j]&2)<<2)+
                             ((threshold_mask[i][j]&4))+
                              (testbit_mask[i][j]<<1)+
                               enable_mask[i][j];

}


void CVICALLBACK Save_mask_file (int menuBar, int menuItem, void *callbackData,
                                 int panel)
{  char filename[300];     // Path selezionato dall'utente
   int selection;          // Indica se il file selezionato è presente
   int i,j;
   unsigned int enable_mask[64][64];
   unsigned int testbit_mask[64][64];
   unsigned int threshold_mask[64][64];

   selection = FileSelectPopup (Directory_Name, "*.msk", "*.msk", "Open Mask
File",
                                VAL_SAVE_BUTTON, 0, 0, 1, 0,
filename);

   if (selection!=0) save_mask(filename);   // Non vi sono errori

// N.B.: Se il file selezionato esiste già il sistema mostrerà
//           una finestra dove chiede all'utente se desidera sovrascrivere
//           il file esistente
}

void CVICALLBACK Modify_Threshold_Mask (int menuBar, int menuItem, void
*callbackData,
            int panel)
{  // Disabilita il pannello ed il menu principale
   SetPanelAttribute (panelHandle, ATTR_DIMMED, 1);
   SetMenuBarAttribute (menuBarHandle, MENU_SYSTEMSE, ATTR_DIMMED, 1);
   SetMenuBarAttribute (menuBarHandle, MENU_MASK, ATTR_DIMMED, 1);
   // Fine disabilitazione

   modify_mask_values("Threshold mask");

   // Abilita il pannello ed il menu principale
   SetPanelAttribute (panelHandle, ATTR_DIMMED, 0);
   SetMenuBarAttribute (menuBarHandle, MENU_SYSTEMSE, ATTR_DIMMED, 0);
   SetMenuBarAttribute (menuBarHandle, MENU_MASK, ATTR_DIMMED, 0);
   // Fine abilitazione
}

void CVICALLBACK Modify_Enable_mask (int menuBar, int menuItem, void
*callbackData,
            int panel)
{  // Disabilita il pannello ed il menu principale
   SetPanelAttribute (panelHandle, ATTR_DIMMED, 1);
```

```
   SetMenuBarAttribute (menuBarHandle, MENU_SYSTEMSE, ATTR_DIMMED, 1);
   SetMenuBarAttribute (menuBarHandle, MENU_MASK, ATTR_DIMMED, 1);
   // Fine disabilitazione

   modify_mask_values("Enable mask");

   // Abilita il pannello ed il menu principale
   SetPanelAttribute (panelHandle, ATTR_DIMMED, 0);
   SetMenuBarAttribute (menuBarHandle, MENU_SYSTEMSE, ATTR_DIMMED, 0);
   SetMenuBarAttribute (menuBarHandle, MENU_MASK, ATTR_DIMMED, 0);
   // Fine abilitazione
}

void   CVICALLBACK   Modify_Testbit_Mask   (int   menuBar,   int   menuItem,   void
*callbackData,
            int panel)
{  // Disabilita il pannello ed il menu principale
   SetPanelAttribute (panelHandle, ATTR_DIMMED, 1);
   SetMenuBarAttribute (menuBarHandle, MENU_SYSTEMSE, ATTR_DIMMED, 1);
   SetMenuBarAttribute (menuBarHandle, MENU_MASK, ATTR_DIMMED, 1);
   // Fine disabilitazione

   modify_mask_values("Testbit mask");

   // Abilita il pannello ed il menu principale
   SetPanelAttribute (panelHandle, ATTR_DIMMED, 0);
   SetMenuBarAttribute (menuBarHandle, MENU_SYSTEMSE, ATTR_DIMMED, 0);
   SetMenuBarAttribute (menuBarHandle, MENU_MASK, ATTR_DIMMED, 0);
   // Fine abilitazione
}


void   CVICALLBACK   Show_threshold_mask   (int   menuBar,   int   menuItem,   void
*callbackData,
            int panel)
{
      int threshold_mask[64][64];
      int i,j,k;

      /* expand the threshold_bit_mask
         starting from "config_matrix"*/
      for(i=0;i<64;i++)
      {
            for(j=0;j<64;j++)
            {
                  k=(config_matrix[i][j]&4)+((config_matrix[i][j]&8)>>2)+
                ((config_matrix[i][j]&16)>>4);

                              threshold_mask[i][j]=k;

                              /* expand the threshold_bit_mask
                              starting from "config_matrix"*/
            }
      }
      /* end expansion */

      display_matrix_values(threshold_mask,"threshold mask");
```

```
}

void   CVICALLBACK   Show_testbit_mask   (int   menuBar,   int   menuItem,   void
*callbackData,
            int panel)
{
      int testbit_mask[64][64];
      int i,j,k;

      for(i=0;i<64;i++)
      {
            for(j=0;j<64;j++)
            {
                  testbit_mask[i][j]=(config_matrix[i][j]&2)>>1;
                  /* expand the testbit_bit_mask
                   starting from "config_matrix"*/
            }
      }


      display_matrix_values(testbit_mask,"testbit mask");
}

void   CVICALLBACK   Show_enable_mask   (int   menuBar,   int   menuItem,   void
*callbackData,
            int panel)
{
      int enable_mask[64][64];
      int i,j;

    for(i=0;i<64;i++)
      {
            for(j=0;j<64;j++)
            {
                  enable_mask[i][j]=config_matrix[i][j]&1;
                  /* expand the enable_bit_mask
                  starting from "config_matrix"*/
            }
      }


      display_matrix_values(enable_mask,"enable mask");
}

void display_matrix_values(int m[64][64],char *whichmask)
{
      int C1,C2,L1,L2;
      char buff1[20];
      char buff[70000];
      int  i,j;

      int bad_range=0;
      const max_row=25;
      const max_col=50;

      do {
                  Input_range(whichmask, &L1, &L2, &C1, &C2);
```

```
            if (((L2-L1) > max_row) || ((C2-C1) > max_col)){
                        MessagePopup ("Error !!!",
                              "You can show maximum 25 Row and 50 Column");
                         bad_range=1;
            } else bad_range=0;
      } while (bad_range);



      sprintf(buff,"Mask displayed: %s\n\n"
                                    "Range displayed:\n\n"
                                    "Start row=%2d;   End row=%2d;\n"
                                    "Start col=%2d;   End col=%2d;\n\n",
                                    whichmask,L1,L2,C1,C2);

      for(i=L1;i<=L2;i++)             /* shows only selected _bit_mask*/
      {
            for(j=C1;j<=C2;j++)
            {

                  sprintf(buff1,"%d ",m[i][j]);
                  if (j==C2) strcat(buff1,"\n");
                  strcat(buff,buff1);
            }

      }
      MessagePopup ("Show Mask", buff);


}

void modify_mask_values(char *whichmask)
{
   char Title[40];

   int selection;              // Vera se l'utente preme un bottone
   int mod_Modality;           // Pannello modalità di modifica selezionato
   int button;                       // Bottone premuto
   int button_mod_single;           // Bottone modify single (1)
   int button_mod_group;            // Bottone modify group  (2)
   int button_exit=0;          // Bottone exit

   int start_row,end_row; // Prima e ultima riga
   int start_col,end_col; // Prima e ultima colonna


   DisplayPanel (modifyModality);   // Visulizza il pannello delle modalità
                                    // di modifca di una maschera

   // Specializza la finestra Modify Modality a seconda della maschera
   sprintf(Title,"Modify %s",whichmask);
   SetCtrlVal (modifyModality, M_MODALITY_MODIFY_TXT, Title);
   SetPanelAttribute (modifyModality, ATTR_TITLE, Title);
   SetCtrlAttribute (inputCoord, M_MODALITY_MODIFY_TXT, ATTR_TEXT_JUSTIFY,
                              VAL_CENTER_JUSTIFIED);
   // Fine
```

```
   do {
         selection = GetUserEvent (1, &mod_Modality, &button);

/*****************************************************************************
*********   Inizio bottone Modify single premuto         **********************
*****************************************************************************/
         button_mod_single=selection && (mod_Modality==modifyModality) &&
                           (button==M_MODALITY_MOD_SINGLE);
         if (button_mod_single) {
              static int ok;      // Bottone OK
              static int select;    // Vero se l'utente preme il bottone
              static int select_ok; // Vero se l'utente preme il bottone

              DisplayPanel(inputCoord);
               // Nasconde la parte dei controlli relativa all'impostazione del
nuovo valore
              SetCtrlAttribute (inputCoord, INPUT_OK, ATTR_VISIBLE, 1);
              SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_T, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_EN, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_OLD_TH, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_TH, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_OLD, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_OLD_EN, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_DONE, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_CANCEL, ATTR_VISIBLE, 0);
              SetCtrlAttribute (inputCoord, INPUT_DEC, ATTR_VISIBLE, 0);
              // Fine

              do {       // Inizio attesa che l'utente prema il tasto OK
                        select = GetUserEvent (1, &inputCoord, &ok);

                          select_ok=select && (ok==INPUT_OK);
                          // Tasto OK premuto
                          if (select_ok){
                            static int row,col;     // Riga e colonna selezionata
                            static int val;             // Valore letto
                            static int button;          // Bottone premuto
                              static int select; // Vero se l'utente preme un
bottone
                            static int select_done=0;   // Vera se l'utente preme
il tasto Done
                            static int select_cancel=0; // Vera se l'utente preme
il tasto Cancel

                            // Visualizza la parte dei controlli relativa
                            // all'impostazione del nuovo valore e
                            // nasconde il tasto OK
                            SetCtrlAttribute (inputCoord, INPUT_OK, ATTR_VISIBLE,
0);
                          SetCtrlAttribute (inputCoord, INPUT_DONE, ATTR_VISIBLE,
1);
                        SetCtrlAttribute (inputCoord, INPUT_CANCEL, ATTR_VISIBLE,
1);
                           SetCtrlAttribute (inputCoord, INPUT_DEC, ATTR_VISIBLE,
1);
                        // Fine
```

```
                          // Lettura dei valori di riga e colonna immessi
dall'utente
                        GetCtrlVal (inputCoord, INPUT_ROW, &row);
                        GetCtrlVal (inputCoord, INPUT_COL, &col);
                        // Fine lettura

                        // Lettura del valore memorizzato in config_matrix
                        if (strcmp (whichmask, "Enable mask")==0) {
                           val=config_matrix[row][col]&1;                //
estrae Enable mask
                               SetCtrlAttribute  (inputCoord,  INPUT_OLD_EN,
ATTR_VISIBLE, 1);
                               SetCtrlAttribute  (inputCoord,  INPUT_OLD_TH,
ATTR_VISIBLE, 0);
                                SetCtrlAttribute   (inputCoord,   INPUT_OLD,
ATTR_VISIBLE, 0);
                             SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_EN,
ATTR_VISIBLE, 1);
                              SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_T,
ATTR_VISIBLE, 0);
                             SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_TH,
ATTR_VISIBLE, 0);
                              SetCtrlVal (inputCoord, INPUT_OLD_EN, val); //
Scrive il valore letto
                           SetCtrlVal (inputCoord, INPUT_NEW_VAL_EN, val); //
nei controlli Old Value

               // e New Value
                        }

                        if (strcmp (whichmask, "Testbit mask")==0) {
                           val=(config_matrix[row][col]&2)>>1;      // estrae
Testbit mask
                               SetCtrlAttribute  (inputCoord,  INPUT_OLD_EN,
ATTR_VISIBLE, 0);
                               SetCtrlAttribute  (inputCoord,  INPUT_OLD_TH,
ATTR_VISIBLE, 0);
                                SetCtrlAttribute   (inputCoord,   INPUT_OLD,
ATTR_VISIBLE, 1);
                             SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_EN,
ATTR_VISIBLE, 0);
                              SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_T,
ATTR_VISIBLE, 1);
                             SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_TH,
ATTR_VISIBLE, 0);
                              SetCtrlVal  (inputCoord,  INPUT_OLD,  val);  //
Scrive il valore letto
                           SetCtrlVal (inputCoord, INPUT_NEW_VAL_T, val); //
nei controlli Old Value

               // e New Value
                        }

                        if (strcmp (whichmask, "Threshold mask")==0) {

val=(config_matrix[row][col]&4)+((config_matrix[row][col]&8)>>2)+
```

```
                                ((config_matrix[row][col]&16)>>4);   // estrae
Threshold mask
                                SetCtrlAttribute  (inputCoord,  INPUT_OLD_EN,
ATTR_VISIBLE, 0);
                                SetCtrlAttribute  (inputCoord,  INPUT_OLD_TH,
ATTR_VISIBLE, 1);
                                  SetCtrlAttribute   (inputCoord,   INPUT_OLD,
ATTR_VISIBLE, 0);
                               SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_EN,
ATTR_VISIBLE, 0);
                                SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_T,
ATTR_VISIBLE, 0);
                               SetCtrlAttribute (inputCoord, INPUT_NEW_VAL_TH,
ATTR_VISIBLE, 1);
                                SetCtrlVal (inputCoord, INPUT_OLD_TH, val); //
Scrive il valore letto
                             SetCtrlVal (inputCoord, INPUT_NEW_VAL_TH, val); //
nei controlli Old Value
                  // e New Value
                           }
                           // Fine lettura


                            do { // Inizio attesa che l'utente prema il tasto
Done o Cancel
                                select = GetUserEvent (1, &inputCoord, &button);

                                select_done=select && (button==INPUT_DONE);
                                // L'utente ha selezionato il tasto Done
                                if (select_done) {
                                      static int enable_mask_bit=30; // Binario
11110:
                                                                        // permette
la modifica della
                                                                           // parte
relativa ai bit Enable
                                                                           // nella
matrice config_matrix

                            static int test_mask_bit=29;   // Binario 11101:
                                                                        // permette
la modifica della
                                                                           // parte
relativa ai bit Test
                                                                           // nella
matrice config_matrix

                                   static int threshold_mask_bit=3; // Binario
00011:
                                                                        // permette
la modifica della
                                                                           // parte
relativa ai bit Threshold
                                                                           // nella
matrice config_matrix
```

```
                                    if (strcmp (whichmask, "Enable mask")==0) {

                                       GetCtrlVal (inputCoord, INPUT_NEW_VAL_EN,
&val);

                                    // Scrive il nuovo valore in config_matrix

config_matrix[row][col]=(config_matrix[row][col] &
                                                        enable_mask_bit) | val;
                                    // Fine scrittura
                                    HidePanel(inputCoord);
                                    }


                                    if (strcmp (whichmask, "Testbit mask")==0) {

                                        GetCtrlVal (inputCoord, INPUT_NEW_VAL_T,
&val);

                                    val=val<<1;
                                    // Scrive il nuovo valore in config_matrix

config_matrix[row][col]=(config_matrix[row][col] &
                                                        test_mask_bit) | (val<<1);
                                    // Fine scrittura
                                    HidePanel(inputCoord);
                                    }

                                    if (strcmp (whichmask, "Threshold mask")==0)
{
                                       GetCtrlVal (inputCoord, INPUT_NEW_VAL_TH,
&val);
                                    val=val<<2;
                                    // Scrive il nuovo valore in config_matrix


config_matrix[row][col]=(config_matrix[row][col] &
                                                             threshold_mask_bit)
|
                                                                  ( (val&1)<<4 +
(val&2)<<2+
                                                                (val&4) );
                                    // Fine scrittura
                                    HidePanel(inputCoord);
                                    }
                                }
                                // Fine tasto Done

                                select_cancel=select && (button==INPUT_CANCEL);
                                // L'utente ha selezionato il tasto Cancel

                                if (select_cancel) {
                                    HidePanel(inputCoord);
                                }
                                // Fine tasto Cancel
```

```
                          } while(!select_done && !select_cancel); // Fine loop
di attesa tasti
                                                                   // Done o
Cancel

                       }// Fine tasto OK premuto
                  }  while(!select_ok);  // Fine loop di attesa tasto OK

          }
/*****************************************************************************
***************   Fine bottone Modify single premuto     **********************
*****************************************************************************/



/*****************************************************************************
*************    Inizio bottone Modify group premuto     ********************
*****************************************************************************/
         button_mod_group=selection && (mod_Modality==modifyModality) &&
                          (button==M_MODALITY_MOD_GROUP);
         if (button_mod_group) {
              // Tasti premuti dall'utente
              static int status;           // Vera se l'utente genera un commit
              static int control_ID;       // Vera sel'utente preme un tasto
              static int button_done=0;       // Vera sel'utente preme il tasto
Done
               static int button_cancel=0;   // Vera sel'utente preme il tasto
Cancel
              // Fine

              HidePanel(modifyModality);
              // Visualizza il pannello per l'impostazione del gruppo
              // di righe e colonne da modificare
              Input_range(whichmask,&start_row, &end_row, &start_col, &end_col);
              // Fine visualizzazione

              DisplayPanel (insertNewVal);
              if (strcmp (whichmask, "Threshold mask")==0) {
                   SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_TH,
                                     ATTR_VISIBLE, 1);
                 SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_EN,
                                   ATTR_VISIBLE, 0);
                 SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_T,
                                   ATTR_VISIBLE, 0);

              }

              if (strcmp (whichmask, "Enable mask")==0) {

                 SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_TH,
                                     ATTR_VISIBLE, 0);
                 SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_EN,
                                   ATTR_VISIBLE, 1);
                 SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_T,
                                   ATTR_VISIBLE, 0);

              }
```

```
            if (strcmp (whichmask, "Testbit mask")==0) {

               SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_TH,
                                 ATTR_VISIBLE, 0);
               SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_EN,
                                 ATTR_VISIBLE, 0);
               SetCtrlAttribute (insertNewVal, INSERT_NEW_VAL_T,
                                 ATTR_VISIBLE, 1);

            }

            do {
                 status = GetUserEvent (1, &insertNewVal, &control_ID);

                 button_done=(control_ID == INSERT_DONE);
                 if (status && button_done)  {
                      static int new_value;
                    static int start_row;
                    static int end_row;
                    static int start_col;
                    static int end_col;

              GetCtrlVal (inputRange, INPUTRANGE_STARTROW, &start_row);
              GetCtrlVal (inputRange, INPUTRANGE_ENDROW, &end_row);
              GetCtrlVal (inputRange, INPUTRANGE_STARTCOLUMN, &start_col);
              GetCtrlVal (inputRange, INPUTRANGE_ENDCOLUMN, &end_col);


                         if (strcmp (whichmask, "Enable mask")==0) {
                                  GetCtrlVal  (insertNewVal,  INSERT_NEW_VAL_EN,
&new_value);
                                modify_mask("enable_mask",  new_value,  start_row,
end_row,
                                            start_col, end_col);
                             HidePanel(insertNewVal);
                         }

                         if (strcmp (whichmask, "Testbit mask")==0) {
                                   GetCtrlVal  (insertNewVal,  INSERT_NEW_VAL_T,
&new_value);
                                 modify_mask("testbit_mask",  new_value,  start_row,
end_row,
                                            start_col, end_col);
                             HidePanel(insertNewVal);
                         }

                         if (strcmp (whichmask, "Threshold mask")==0) {
                                  GetCtrlVal  (insertNewVal,  INSERT_NEW_VAL_TH,
&new_value);
                              modify_mask("threshold_mask", new_value, start_row,
end_row,
                                            start_col, end_col);
                             HidePanel(insertNewVal);
                         }
                 }

                 button_cancel= (control_ID == INSERT_CANCEL);
```

```
                if (status && button_cancel)  {
                       HidePanel (insertNewVal);
                       DisplayPanel(modifyModality);
                    }
              } while (!button_done && !button_cancel);
                  DisplayPanel (modifyModality);

         }
/*****************************************************************************
**************     Fine bottone Modify group premuto     **********************
*****************************************************************************/



/*****************************************************************************
************          Inizio bottone Exit premuto          **********************
*****************************************************************************/
         button_exit=selection && (mod_Modality==modifyModality) &&
                           (button==M_MODALITY_MOD_EXIT);
         if (button_exit) {
          HidePanel(modifyModality);
         }
/*****************************************************************************
************           Fine bottone Exit premuto          **********************
*****************************************************************************/
   } while (!button_exit);

}


void CVICALLBACK Enable_graph (int menuBar, int menuItem, void *callbackData,
            int panel)
{   ColorMapEntry   colors[256];
    int hiColor, numColors;
    int i,j;
      int enable_mask[64][64];



    for(i=0;i<64;i++)
      {
            for(j=0;j<64;j++)
            {
                  enable_mask[i][j]=config_matrix[i][j]&1;
                  /* expand the enable_bit_mask
                  starting from "config_matrix"*/
            }
      }

      GetGraphMaskParams (enable_mask, colors, &hiColor, 2);

      PlotIntensity  (showMaskGraph,  SHOW_MASK_GRAPH,  enable_mask,  64,  64,
VAL_INTEGER,
                           colors, hiColor, 2, 0, 0);
}
```

```
void CVICALLBACK Testbit_graph (int menuBar, int menuItem, void *callbackData,
            int panel)
{   ColorMapEntry   colors[256];
    int hiColor, numColors;
    int i,j;
      int testbit_mask[64][64];



      for(i=0;i<64;i++)
      {
            for(j=0;j<64;j++)
            {
                  testbit_mask[i][j]=(config_matrix[i][j]&2)>>1;
                  /* expand the testbit_bit_mask
                   starting from "config_matrix"*/
            }
      }

      GetGraphMaskParams (testbit_mask, colors, &hiColor, 2);

      PlotIntensity  (showMaskGraph,  SHOW_MASK_GRAPH,  testbit_mask,  64,  64,
VAL_INTEGER,
                           colors, hiColor, 2, 0, 0);
}


void CVICALLBACK Threshold_graph (int menuBar, int menuItem, void *callbackData,
            int panel)
{   ColorMapEntry   colors[256];
    int hiColor, numColors;
    int i,j,k;
      int threshold_mask[64][64];



      /* expand the threshold_bit_mask
         starting from "config_matrix"*/
      for(i=0;i<64;i++)
      {
            for(j=0;j<64;j++)
            {
                  k=(config_matrix[i][j]&4)+((config_matrix[i][j]&8)>>2)+
                ((config_matrix[i][j]&16)>>4);

                              threshold_mask[i][j]=k;

                              /* expand the threshold_bit_mask
                              starting from "config_matrix"*/
            }
      }
      /* end expansion */

      GetGraphMaskParams (threshold_mask, colors, &hiColor, 8);

      PlotIntensity  (showMaskGraph,  SHOW_MASK_GRAPH,  threshold_mask,  64,  64,
VAL_INTEGER,
```

```
                            colors, hiColor, 8, 0, 0);
}


void GetGraphMaskParams (int m[64][64], ColorMapEntry colors[], int *hiColor,
               int numColors)
{
    int maxVal, minVal;
    unsigned char   loRed, hiRed, red, loGreen, hiGreen, green, loBlue, hiBlue,
                    blue;
    int             loCol, hiCol, i, j;
      int             min_val, max_val;
      int             screen_color;

      DisplayPanel(showMaskGraph);


    loRed = (unsigned char)((VAL_WHITE & 0x00FF0000) >> 16);
    hiRed = (unsigned char)((VAL_BLACK & 0x00FF0000) >> 16);
    loGreen = (unsigned char)((VAL_WHITE & 0x0000FF00) >> 8);
    hiGreen = (unsigned char)((VAL_BLACK & 0x0000FF00) >> 8);
    loBlue = (unsigned char)(VAL_WHITE & 0x000000FF);
    hiBlue = (unsigned char)(VAL_BLACK & 0x000000FF);

      minVal=0;                       // Valori minimi e massimi
      maxVal=numColors-1;

    for (i = 0; i < numColors; i++) {
         red = loRed + i * (hiRed - loRed) / (numColors-1);
         green = loGreen + i * (hiGreen - loGreen) / (numColors-1);
         blue = loBlue + i * (hiBlue - loBlue) / (numColors-1);
         colors[i].color = MakeColor ((int)red, (int)green, (int)blue);
         colors[i].dataValue.valInt =(int) (minVal + (i+1) * (maxVal - minVal)
                                           / (numColors));
    }

    *hiColor = MakeColor ((int)hiRed, (int)hiGreen, (int)hiBlue);

/*************************************************************************
***************   Disegna la legenda dei colori       ********************
*************************************************************************/
    CanvasClear (showMaskGraph, SHOW_MASK_COLOR_INDEX, VAL_ENTIRE_OBJECT);
      GetCtrlAttribute (showMaskGraph, SHOW_MASK_COLOR_INDEX,
                        ATTR_PEN_FILL_COLOR, &screen_color);


      for (i=minVal; i<=maxVal; i++) {
            static Rect square_box;
            static Point pos;
            static int top;
            static int offset=5;
        static int left=60;
        static char car[3];

            top=20*i;
            square_box=MakeRect(top+offset, left, 15, 15);
            SetCtrlAttribute (showMaskGraph, SHOW_MASK_COLOR_INDEX,
```

```
                                    ATTR_PEN_COLOR, 0);

            sprintf(car,"%d",i);  // Converte il valore in carattere
            pos = MakePoint (left-30, top+offset);
            SetCtrlAttribute (showMaskGraph, SHOW_MASK_COLOR_INDEX,
                              ATTR_PEN_FILL_COLOR, screen_color);
            CanvasDrawTextAtPoint (showMaskGraph, SHOW_MASK_COLOR_INDEX, car,
                                                    VAL_APP_META_FONT,    pos,
VAL_UPPER_LEFT);
            CanvasDrawRect (showMaskGraph, SHOW_MASK_COLOR_INDEX, square_box,
                                    VAL_DRAW_FRAME);
            for (j=left+1; j<left+14; j++) {
                  SetCtrlAttribute (showMaskGraph, SHOW_MASK_COLOR_INDEX,
                                             ATTR_PEN_COLOR, colors[i].color);
                  CanvasDrawLine (showMaskGraph, SHOW_MASK_COLOR_INDEX,
                                    MakePoint(j,top+offset+1),
MakePoint(j,top+offset+13));
            }


      }


/*****************************************************************************
*************   Fine disegno della legenda dei colori       ******************
*****************************************************************************/

}

int CVICALLBACK Quit_MaskGraph (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  HidePanel(showMaskGraph);
                  break;
      }
      return 0;
}


void  Input_range(char  *mask_name,  int  *s_row,  int  *e_row,  int  *s_col,  int
*e_col)
{  int status;            // Vera se l'utente preme ok
   int panelRange;              // Identificativo pannello inputRange
   int ok_button;         // Identificativo bottone OK
   int correctRange=0;    // Vera se i range di righe e colonne da visualizzare
                                    // sono corretti

   char Title[50];        // Titolo del pannello
   int start_row,end_row; // Prima e ultima riga
   int start_col,end_col; // Prima e ultima colonna


   sprintf (Title, "Insert Row/Column range: %s", mask_name);
   SetPanelAttribute (inputRange, ATTR_TITLE, Title);
   DisplayPanel (inputRange);
```

```
   do
   {

       status = GetUserEvent (1, &panelRange, &ok_button);
       if ((status==EVENT_COMMIT) &&
            (panelRange==inputRange) && (ok_button==INPUTRANGE_OK))

       // Range checking
       {   GetCtrlVal (inputRange, INPUTRANGE_STARTROW, &start_row);
         GetCtrlVal (inputRange, INPUTRANGE_ENDROW, &end_row);
         GetCtrlVal (inputRange, INPUTRANGE_STARTCOLUMN, &start_col);
         GetCtrlVal (inputRange, INPUTRANGE_ENDCOLUMN, &end_col);

         correctRange=(int) ((end_row>=start_row) && (end_col>=start_col));

         if(!correctRange)  MessagePopup ("Error !!", "Incorrect Row / Columns
Range");
       }
       // Fine range checking
       } while (!correctRange);

       *s_row=start_row;
       *e_row=end_row;
       *s_col=start_col;
       *e_col=end_col;
       HidePanel (inputRange);
}


/*****************************************************************************
************************* Fine Menu Mask  ***********************************
*****************************************************************************/

/*****************************************************************************
*****************************************************************************
*************************   Fine barra menu   *******************************
*****************************************************************************
*****************************************************************************/


/*****************************************************************************
*****************************************************************************
**************    Bottoni dell'area Acquisition parameter  ******************
*****************************************************************************
*****************************************************************************/

int CVICALLBACK Show_active_acq_params (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
       char message[150], trigmodstr[15];
       static char unit[2][3] = {"ms", "s"};

       switch (event) {
             case EVENT_COMMIT:
                   switch (trig_mode_act) {
                       case 0: sprintf(trigmodstr,"Internal"); break;
```

```
                    case 1: sprintf(trigmodstr,"External"); break;
                    case 2: sprintf(trigmodstr,"ANIN"); break;
                }
                sprintf(message,"Active acquisition parameters are:\n\n"
                                "tau0=%6d\ntau1=%6d %s\ntau2=%6d\n"
                                "Trigger Mode=%s",
                                     tau0_act,tau1_act,  &unit[TIME_UNIT][0],
tau2_act, trigmodstr);
                MessagePopup ("Show Active Acquisition Parameters", message);

                break;
      }
      return 0;
}

/*****************************************************************************
*************** Fine bottoni area Acquisition parameter  *********************
*****************************************************************************/


/*****************************************************************************
*****************************************************************************
**************    Bottoni dell'area VME Board Setting    *********************
*****************************************************************************
*****************************************************************************/

int CVICALLBACK Show_act_settings (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      char message[150],configmodstr[10];

      switch (event) {
            case EVENT_COMMIT:
                switch (config_mode_act) {
                    case 0: sprintf(configmodstr,"INDIRECT"); break;
                    case 1: sprintf(configmodstr,"DIRECT"); break;
                }
                sprintf(message,"Active VME Board Config pameters are:\n\n"
                                "Chip ID=%5d\nRAM offset=%8d\n"
                                "Config Mode=%s\nCycles=%8d",

chip_sel_act,ram_offset_act,configmodstr,cycles_act);
                MessagePopup  ("Show  Active  VME  Board  Config  Parameters",
message);


                break;
      }

      return 0;
}
/*, "these values are really loaded" */
/*****************************************************************************
***************   Fine bottoni area VME Board Setting   *********************
*****************************************************************************/
```

```
/*****************************************************************************
*****************************************************************************
**************    Bottoni dell'area ANIN Test parameter  ********************
*****************************************************************************
*****************************************************************************/

int CVICALLBACK show_act_anin (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      char message[150];
      static char unit[2][3] = {"ms", "s"};

      switch (event) {
            case EVENT_COMMIT:
                  sprintf(message,"Active  ANIN Test pameters are:\n\n"
                                  "ANIN Cycles=%5d\nSleep ANIN delay=%8d %s\n"
                                                            "Sleep   ANIN
Period=%5d",anin_cycles_act,sleep_anin_delay_act,
                                  &unit[TIME_UNIT][0], sleep_anin_period_act);
                  MessagePopup ("Show Active ANIN Test Parameters", message);
                  break;
      }
      return 0;
}


/*****************************************************************************
*************** Fine bottoni area ANIN Test parameter  ***********************
*****************************************************************************/


/*****************************************************************************
*****************************************************************************
*****************    Bottoni dell'area DAC parameter  ************************
*****************************************************************************
*****************************************************************************/

int CVICALLBACK show_act_DAC_params (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      char message[400],message1[20],DACmodstr[15];
      unsigned int i,jj;
      unsigned int switch_pos; // Posizione dello switch V-au


      read_dac();


      switch (event) {
            case EVENT_COMMIT:
              GetCtrlVal (panelHandle, PANEL_V_AU_SWITCH, &switch_pos);
              GetCtrlVal (panelHandle, PANEL_CHIP_ID, &chip_sel_tmp);
              if(chip_sel_tmp==0) { jj=0;}
              if(chip_sel_tmp==1) { jj=5;}
              if(switch_pos==0) {   // Switch in posizione au
```

```
                              sprintf  (message,  "Active  DAC  Bias  (a.u.)\n\nDAC
Bias:\n");
/*                          for (i=0; i<=9; i++) {

sprintf(message1,"DAC%2d=%5d\n",i+1,dac_bias_act[i]);
                                strcat (message, message1);
                            }

*/



sprintf(message1,"Vbias=%5d\n",dac_bias_act[0+jj]);
                                strcat (message, message1);

sprintf(message1,"Vcomp=%5d\n",dac_bias_act[1+jj]);
                                strcat (message, message1);
                           sprintf(message1,"Vth  =%5d\n",dac_bias_act[2+jj]);
                                strcat (message, message1);
                             sprintf(message1,"Vtha =%5d\n",dac_bias_act[3+jj]);
                                strcat (message, message1);
                             sprintf(message1,"Vdl  =%5d\n",dac_bias_act[4+jj]);
                                strcat (message, message1);


          }
              else { // Switch in posizione V
                    sprintf (message, "Active DAC Bias (V)\n\nDAC Bias:\n");
/*                        for (i=0; i<=9; i++) {
                            sprintf(message1,"DAC%2d=%5.2f\n",i+1,
                                    (float)dac_bias_act[i]/LSBCAL);
                          strcat (message, message1); }
*/
                                sprintf(message1,"Vbias=%5.2f\n",
                                        (float)dac_bias_act[0+jj]/LSBCAL);
                                strcat (message, message1);
                                  sprintf(message1,"Vcomp=%5.2f\n",
                                        (float)dac_bias_act[1+jj]/LSBCAL);
                                strcat (message, message1);
                                  sprintf(message1,"Vth  =%5.2f\n",
                                        (float)dac_bias_act[2+jj]/LSBCAL);
                                strcat (message, message1);
                                  sprintf(message1,"Vtha =%5.2f\n",
                                        (float)dac_bias_act[3+jj]/LSBCAL);
                                strcat (message, message1);
                                  sprintf(message1,"Vdl  =%5.2f\n",
                                        (float)dac_bias_act[4+jj]/LSBCAL);
                                strcat (message, message1);

                }
              MessagePopup ("Show Active DAC Parameters", message);
              break;
      }
      return 0;
}
```

```
int CVICALLBACK modify_dac_bias (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   int i;                              // Variabile contatore
    unsigned int switch_pos; // Posizione corrente dello switch V-au
    int value;               // Tensione del DAC (in Volt o a.u.)

      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal (panelHandle, PANEL_V_AU_SWITCH, &switch_pos);
                  if (switch_pos==0) { // Switch in posizione a.u.
                         DisplayPanel (modifyDACBias1);
                      // Caricamento delle impostazioni temporanee
                        SetCtrlVal       (modifyDACBias1,       MOD_BIAS_1_DAC0,
dac_bias_tmp[0]);
                        SetCtrlVal       (modifyDACBias1,       MOD_BIAS_1_DAC1,
dac_bias_tmp[1]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC2, dac_bias_tmp[2]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC3, dac_bias_tmp[3]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC4, dac_bias_tmp[4]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC5, dac_bias_tmp[5]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC6, dac_bias_tmp[6]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC7, dac_bias_tmp[7]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC8, dac_bias_tmp[8]);
                SetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC9, dac_bias_tmp[9]);
                      // Fine caricamento impostazioni temporanee

                  }
                  else {                          // Switch in posizione V
                         DisplayPanel (modifyDACBias2);
                         // Caricamento delle impostazioni temporanee
                        SetCtrlVal       (modifyDACBias2,       MOD_BIAS_1_DAC0,
dac_bias_tmp[0]/LSBCAL);
                        SetCtrlVal       (modifyDACBias2,       MOD_BIAS_1_DAC1,
dac_bias_tmp[1]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC2,
dac_bias_tmp[2]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC3,
dac_bias_tmp[3]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC4,
dac_bias_tmp[4]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC5,
dac_bias_tmp[5]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC6,
dac_bias_tmp[6]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC7,
dac_bias_tmp[7]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC8,
dac_bias_tmp[8]/LSBCAL);
                SetCtrlVal          (modifyDACBias2,          MOD_BIAS_1_DAC9,
dac_bias_tmp[9]/LSBCAL);
                      // Fine caricamento impostazioni temporanee
                  }

                  break;
      }
      return 0;
}
```

```
/*****************************************************************************
*************** Pulsante Ok (Switch in posizione au)  ************************
*****************************************************************************/

int CVICALLBACK Ok_Dac_Bias1 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int i;             // Variabile indice

      switch (event) {
            case EVENT_COMMIT:
                  // Aggiornamento delle variabili temporanee
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC0, &dac_bias_tmp[0]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC1, &dac_bias_tmp[1]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC2, &dac_bias_tmp[2]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC3, &dac_bias_tmp[3]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC4, &dac_bias_tmp[4]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC5, &dac_bias_tmp[5]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC6, &dac_bias_tmp[6]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC7, dac_bias_tmp[7]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC8, &dac_bias_tmp[8]);
            GetCtrlVal (modifyDACBias1, MOD_BIAS_1_DAC9, &dac_bias_tmp[9]);
                  // Fine Aggiornamento variabili temporanee
                  HidePanel (modifyDACBias1);
                  break;
      }
      return 0;
}

/*****************************************************************************
************** Pulsante Cancel (Switch in posizione au) **********************
*****************************************************************************/

int CVICALLBACK CancelDACBias1 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  HidePanel (modifyDACBias1);
                  break;
      }
      return 0;
}

/*****************************************************************************
************** Pulsante Ok (Switch in posizione Volt) ************************
*****************************************************************************/

int CVICALLBACK Ok_Dac_Bias2 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   int i;           // variabile indice
    float volt[10];   // tensione letta

      switch (event) {
            case EVENT_COMMIT:
                  // Aggiornamento delle variabili temporanee
```

```
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC0, &volt[0]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC1, &volt[1]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC2, &volt[2]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC3, &volt[3]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC4, &volt[4]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC5, &volt[5]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC6, &volt[6]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC7, &volt[7]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC8, &volt[8]);
                GetCtrlVal (modifyDACBias2, MOD_BIAS_2_DAC9, &volt[9]);
                for (i=0; i<10; i++) {
                    dac_bias_tmp[i]=(unsigned int) (volt[i]*LSBCAL);

                }
                // Fine Aggiornamento variabili temporanee
                HidePanel (modifyDACBias2);
                break;
      }
      return 0;
}


/*****************************************************************************
************** Pulsante Cancel (Switch in posizione Volt) *********************
*****************************************************************************/
int CVICALLBACK CancelDACBias2 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
            HidePanel (modifyDACBias2);
                  break;
      }
      return 0;
}

/*****************************************************************************
****************** Fine bottoni area DAC parameter  ***************************
*****************************************************************************/


/*****************************************************************************
*****************************************************************************
***********************    Timer principale  ********************************
*****************************************************************************
*****************************************************************************/
int CVICALLBACK Main_timer (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{

       int i;  //Valore di ritorno di un controllo
       // Variabili relative ai bottoni dell'area DAC Mode Parameters
       int count;
       char buff[10];
     char dacbias[30]; //Etichetta della lista
     // Variabili relative ai bottoni del pannello DAC Sequence
     int dimmed=0;     // Vera se il bottone deve essere disabilitato
```

```
	int num;		// Numero di elementi della lista
	char label[10];	// Etichetta corrispondente ad un item scelto

	switch (event) {
		case EVENT_TIMER_TICK:

/*****************************************************************************
*****************************************************************************
*********   Aggiornamento dei valori dell'area Acquisition Parameter  **********
*****************************************************************************
*****************************************************************************/
				GetCtrlVal (panelHandle, PANEL_TAU0, &tau0_tmp);
				GetCtrlVal (panelHandle, PANEL_TAU1, &tau1_tmp);
				GetCtrlVal (panelHandle, PANEL_TAU2, &tau2_tmp);
				GetCtrlVal	(panelHandle,	PANEL_TRIG_MODE,
&trig_mode_tmp);

/*****************************************************************************
*****************************************************************************
*****   Fine aggiornamento dei valori dell'area Acquisition parameter     ******
*****************************************************************************
*****************************************************************************/
/*****************************************************************************
*****************************************************************************
*********   Aggiornamento dei valori dell'area VME Board Setting     ***********
*****************************************************************************
*****************************************************************************/
			GetCtrlVal (panelHandle, PANEL_CHIP_ID, &chip_sel_tmp);
			GetCtrlVal (panelHandle, PANEL_CYCLES, &cycles_tmp);
			GetCtrlVal	(panelHandle,	PANEL_CONFIG_MODE,
&config_mode_tmp);
			GetCtrlVal	(panelHandle,	PANEL_RAM_OFFSET,
&ram_offset_tmp);
/*****************************************************************************
*****************************************************************************
*******   Fine aggiornamento dei valori dell'area VME Board Setting     ********
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
*****************************************************************************
*******   Aggiornamento dei valori dell'area ANIN Test Parameter     **********
*****************************************************************************
*****************************************************************************/
				GetCtrlVal	(panelHandle,	PANEL_ANIN_CYCLES,
&anin_cycles_tmp);
				GetCtrlVal (panelHandle, PANEL_SLEEP_ANIN_DELAY,
					&sleep_anin_delay_tmp);
				GetCtrlVal (panelHandle, PANEL_SLEEP_ANIN_FREQ,
					&sleep_anin_period_tmp);
/*****************************************************************************
*****************************************************************************
******   Fine aggiornamento dei valori dell'area ANIN Test Parameter     *******
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
```

```
*****************************************************************************
***********    Aggiornamento del valore dello switch Power Mode     ************
*****************************************************************************
*****************************************************************************/
                GetCtrlVal (panelHandle, PANEL_POWER_MODE, &power_mode_tmp);
                            power_mode_act=power_mode_tmp;

            if(power_mode_tmp==0)
              {
               SetCtrlAttribute(panelHandle,PANEL_STATE_ON,ATTR_DIMMED,1);
               SetCtrlAttribute(panelHandle,PANEL_STATE_OFF,ATTR_DIMMED,1);

               SetCtrlAttribute(panelHandle,PANEL_CHECK_POWER,ATTR_DIMMED,1);

              }
            else
              {
               SetCtrlAttribute(panelHandle,PANEL_STATE_ON,ATTR_DIMMED,0);
               SetCtrlAttribute(panelHandle,PANEL_STATE_OFF,ATTR_DIMMED,0);

               SetCtrlAttribute(panelHandle,PANEL_CHECK_POWER,ATTR_DIMMED,0);
              }
/*****************************************************************************
*****************************************************************************
*********   Fine aggiornamento del valore dello switch Power Mode     **********
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
***********  Controllo dei bottoni dell'area DAC  Parameters     ************
*****************************************************************************/
                        // Imposta i valori di DAC Bias
                        GetCtrlVal (panelHandle, PANEL_V_AU_SWITCH, &i);



                            switch (switch_on) {
                                  case 0:  // Medipix spento
                                        SetCtrlVal              (panelHandle,
PANEL_STATE_ON, 0);
                                SetCtrlVal (panelHandle, PANEL_STATE_OFF, 1);
            SetCtrlAttribute (panelHandle, PANEL_TURN_ON, ATTR_VISIBLE, 1);
            SetCtrlAttribute (panelHandle, PANEL_TURN_OFF, ATTR_VISIBLE, 0);
            SetCtrlAttribute (panelHandle, PANEL_LOAD_MASK, ATTR_DIMMED, 0);
            SetCtrlAttribute (panelHandle, PANEL_START_ACQ, ATTR_DIMMED, 0);
            SetCtrlAttribute (panelHandle, PANEL_SHOW_IMAGE, ATTR_DIMMED, 0);
                                        break;
                                  case 1:  // Medipix acceso
            SetCtrlVal (panelHandle, PANEL_STATE_ON, 1);
            SetCtrlVal (panelHandle, PANEL_STATE_OFF, 0);
            SetCtrlAttribute (panelHandle, PANEL_TURN_ON, ATTR_VISIBLE, 0);
            SetCtrlAttribute (panelHandle, PANEL_TURN_OFF, ATTR_VISIBLE, 1);
            SetCtrlAttribute (panelHandle, PANEL_LOAD_MASK, ATTR_DIMMED, 0);
            SetCtrlAttribute (panelHandle, PANEL_START_ACQ, ATTR_DIMMED, 0);
            SetCtrlAttribute (panelHandle, PANEL_SHOW_IMAGE, ATTR_DIMMED, 0);
            //SetCtrlAttribute (panelHandle, PANEL_NEW_ACQUISITION,
                                        //    ATTR_DIMMED, 0);
```

```
                                    break;
                               }

                               if (switch_on)  {
                                  switch (mask_status) {
                                       case  0:    //  Maschera  non  caricata  su
Medipix
                                     SetCtrlVal (panelHandle, PANEL_LOAD, 0);
                                   SetCtrlVal (panelHandle, PANEL_NOT_LOAD, 1);
                    SetCtrlAttribute (panelHandle, PANEL_LOAD_MASK, ATTR_DIMMED, 0);
                     SetCtrlAttribute (panelHandle, PANEL_START_ACQ, ATTR_DIMMED, 0);
                     SetCtrlAttribute (panelHandle, PANEL_SHOW_IMAGE, ATTR_DIMMED,0);
                                    break;
                                     case 1:  // Maschera caricata su Medipix
                     SetCtrlVal (panelHandle, PANEL_LOAD, 1);
                     SetCtrlVal (panelHandle, PANEL_NOT_LOAD, 0);
                     SetCtrlAttribute (panelHandle, PANEL_LOAD_MASK, ATTR_DIMMED, 0);
                     SetCtrlAttribute (panelHandle, PANEL_START_ACQ, ATTR_DIMMED, 0);
                     SetCtrlAttribute (panelHandle, PANEL_SHOW_IMAGE, ATTR_DIMMED, 0);
                                    break;
                                  }
                               }

                               if (!switch_on && !mask_status)  {
                                  SetCtrlVal (panelHandle, PANEL_LOAD, 0);
                                  SetCtrlVal (panelHandle, PANEL_NOT_LOAD, 1);
                               }

                               switch (image_ready) {
                                    case 0:
                     SetCtrlAttribute (panelHandle, PANEL_SHOW_IMAGE, ATTR_DIMMED, 0);
                     SetCtrlVal(panelHandle, PANEL_IMAGE_READY,0)
                                    break;
                                    case 1:
                     SetCtrlAttribute (panelHandle, PANEL_SHOW_IMAGE, ATTR_DIMMED, 0);
                     SetCtrlVal(panelHandle, PANEL_IMAGE_READY,1);
                                    break;
                               }

                               switch (acquisition_on) {
                                    case 0:
                     //SetCtrlAttribute (panelHandle, PANEL_SET_DAC, ATTR_DIMMED, 1);
                                SetCtrlVal(panelHandle, PANEL_END_ACQUISITION,0);

                     //    SetCtrlAttribute (panelHandle, PANEL_NEW_ACQUISITION,
                              //                    ATTR_DIMMED, 0);
                                    break;
                                    case 1:
                     // SetCtrlAttribute (panelHandle, PANEL_NEW_ACQUISITION,
                                //                 ATTR_DIMMED, 1);
                     //SetCtrlAttribute (panelHandle, PANEL_SET_DAC, ATTR_DIMMED, 0);
                     SetCtrlVal(panelHandle, PANEL_END_ACQUISITION,1);
                     SetCtrlAttribute (panelHandle, PANEL_LOAD_MASK, ATTR_DIMMED, 0);
                     SetCtrlAttribute (panelHandle, PANEL_START_ACQ, ATTR_DIMMED, 0);
                     SetCtrlAttribute (panelHandle, PANEL_SHOW_IMAGE, ATTR_DIMMED, 0);
                                    break;
                               }
```

```
/*****************************************************************************
*****************************************************************************
*********/
            break;
      }
      return 0;
}


/*****************************************************************************
*********************** Fine Timer principale  ******************************
*****************************************************************************/



/*****************************************************************************
*****************************************************************************
*************** Imposta i valori delle variabili temporanee ******************
*****************************************************************************
*****************************************************************************/

void        init_graph_interface(char        arc_file_default[200]        ,char
mask_file_default[512])
{
      set_arc_params(arc_file_default);
      set_mask_params(mask_file_default);
}

void set_arc_params(char arc_file[200])
{
    int sw_value;     // Valore dello switch V_au

      SetCtrlVal (panelHandle, PANEL_FILENAME, arc_file);

            // Impostazion area Acquisition parameter
            SetCtrlVal (panelHandle, PANEL_TAU0, tau0_tmp);
            if (TIME_UNIT==0)  // Unita di misura: millisecondi
            {
                  SetCtrlAttribute (panelHandle, PANEL_TAU1, ATTR_LABEL_TEXT,
                                    "Acquisition Time (ms)");

            }
            else                    // Unita di misura: secondi
            {
                  SetCtrlAttribute (panelHandle, PANEL_TAU1, ATTR_LABEL_TEXT,
                                    "Acquisition Time (s)");

            }

            SetCtrlVal (panelHandle, PANEL_TAU1, tau1_tmp);
            SetCtrlVal (panelHandle, PANEL_TAU2, tau2_tmp);
            SetCtrlVal (panelHandle, PANEL_TRIG_MODE, trig_mode_tmp);
            // Fine impostazioni area Acquisition paramenter

            // Impostazioni area ANIN Test parameter
            SetCtrlVal (panelHandle, PANEL_ANIN_CYCLES, anin_cycles_tmp);
```

```
            SetCtrlVal           (panelHandle,           PANEL_SLEEP_ANIN_FREQ,
sleep_anin_period_tmp);
            SetCtrlVal          (panelHandle,          PANEL_SLEEP_ANIN_DELAY,
sleep_anin_delay_tmp);
        // Fine impostazioni ANIN Test parameter

        // Impostazioni dello switch Power Mode
            SetCtrlVal (panelHandle, PANEL_POWER_MODE, power_mode_tmp);
        // Fine impostazioni dello switch Power Mode

            // Impostazioni area VME Board Settings
            SetCtrlVal (panelHandle, PANEL_CHIP_ID, chip_sel_tmp);
            SetCtrlVal (panelHandle, PANEL_CYCLES, cycles_tmp);
            SetCtrlVal (panelHandle, PANEL_CONFIG_MODE, config_mode_tmp);

            SetCtrlVal (panelHandle, PANEL_RAM_OFFSET, ram_offset_tmp);
        // Fine impostazioni VME Board Settings

}

void set_mask_params(char mask_file[200])
{

      SetCtrlVal (panelHandle, PANEL_MASKFILE, mask_file);

}


int CVICALLBACK turn_on_medipix (int panel, int control, int event,
                                 void *callbackData, int eventData1,
                                 int eventData2)
{
  char message[900]="";
  char message_act[500],message1_act[20];
  char message_tmp[500],message1_tmp[20];
  unsigned int i,imax,imin;
  unsigned int switch_pos; // Posizione dello switch V-au

  switch (event) {
      static int answer;    // Risposta alla domanda di conferma
                            // dei valori attivi e temporanei
    case EVENT_COMMIT:
        GetCtrlVal (panelHandle, PANEL_V_AU_SWITCH, &switch_pos);
        GetCtrlVal (panelHandle, PANEL_CHIP_ID, &chip_sel_tmp);
        if(chip_sel_tmp==0) {
          imin=0;
          imax=4;
        }
        if(chip_sel_tmp==1) {
          imin=5;
          imax=9;
        }
/*************************************************************************
******************** Inizio stampa valori attivi  ************************
*************************************************************************/
        if(switch_pos==0) {   // Switch in posizione au
// Inizio stampa valori attivi
```

```
          sprintf(message_act,
                  "Active DAC (last acquisition) parameters are:\n\n");
          strcat (message_act, "DAC Bias:\n");
          for (i=imin; i<=imax; i++) {
            sprintf(message1_act,"DAC%2d=%5d      ",i+1,dac_bias_act[i]);
            if (i==imax)
              strcat(message1_act,"\n");
            strcat (message_act, message1_act);
            }
// Fine stampa valori attivi
       }
       else {               // Switch in posizione V
//Inizio stampa valori attivi
             sprintf(message_act,"Active  DAC  (last  acquisition)  parameters
are:\n\n");
          strcat (message_act, "DAC Bias:\n");
          for (i=imin; i<=imax; i++) {
          sprintf(message1_act,"DAC%2d=%5.2f;  ",i+1,
                    (float)dac_bias_act[i]/LSBCAL);
            if (i==imax)
              strcat(message1_act,"\n");
            strcat (message_act, message1_act);
        }
//Fine stampa valori attivi
        }

/*************************************************************************
*********************  Fine stampa valori attivi   ***********************
*****************  Inizio stampa valori temporanei  *********************
*************************************************************************/
        if(switch_pos==0) {   // Switch in posizione au
// Inizio stampa valori temporanei
          sprintf(message_tmp,"\n\n\nTempoarary DAC parameters are:\n\n");
          strcat (message_tmp, "DAC Bias:\n");
          for (i=imin; i<=imax; i++) {
            sprintf(message1_tmp,"DAC%2d=%5d;   ",i+1,dac_bias_tmp[i]);
            if (i==imax)
                strcat(message1_tmp,"\n");
              strcat (message_tmp, message1_tmp);
            } // Fine stampa valori temporanei
       }
         else { // Switch in posizione V
//Inizio stampa valori temporanei
          sprintf(message_tmp,"\n\n\nTemporary DAC parameters are:\n\n");
          strcat (message_tmp, "DAC Bias:\n");
          for (i=imin; i<=imax; i++) {
            sprintf(message1_tmp,"DAC%2d=%5.2f;   ",i+1,
                    (float)dac_bias_tmp[i]/LSBCAL);
            if (i==imax)
              strcat(message1_tmp,"\n");
            strcat (message_tmp, message1_tmp);
            }  //Fine stampa valori temporanei
        }

/*************************************************************************
*****************  Fine stampa valori temporanei   **********************
```

```
*****************************************************************************/

        strcat (message, message_act);
        strcat (message, message_tmp);
        strcat (message,"\n\n Confirm these values and turn on Medipix?");
        answer = ConfirmPopup ("Active/temporary values", message);
        if (answer) {
          if (MISSING_HARDWARE==0) {
              if (power_mode_tmp==1)
                turn_on_power();
               else
                Print_error(14);
              switch_on=check_power();
              turn_on_dac();
              mask_status=0;
              image_ready=0;
          acquisition_on=0;
            }

          if (MISSING_HARDWARE==1) {
          turn_on_dac();
          switch_on=1; //Usata per la simulazione
          mask_status=0;
          image_ready=0;
          acquisition_on=0;
        }
      }
      break;
  }
  return 0;
}


int CVICALLBACK turn_off_medipix (int panel, int control, int event,
                                  void *callbackData, int eventData1,
                                  int eventData2)
{
  switch (event) {
      case EVENT_COMMIT:
      if (MISSING_HARDWARE==0) {
          turn_off_dac();
          if(power_mode_tmp==1)
            turn_off_power();
           else
            Print_error(15);
            switch_on=check_power();
            switch_on=0;
            mask_status=0;
      }
        if (MISSING_HARDWARE==1) {
          switch_on=0; //Usata per la simulazione
            mask_status=0;
            image_ready=0;
          acquisition_on=0;
        }
        break;
```

```
  }
  return 0;
}

/*****************************************************************************
******************************************************************************
************** Fine bottoni Turn-on/Turn-off Medipix  ************************
******************************************************************************
*****************************************************************************/



int CVICALLBACK Set_DAC (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   char message[900]="";
    char message_act[500],message1_act[20];
    char message_tmp[500],message1_tmp[20];
      unsigned int i,imax,imin;
      unsigned int switch_pos; // Posizione dello switch V-au

      switch (event) {
            static int answer;    // Risposta alla domanda di conferma
                                  // dei valori attivi e temporanei
            case EVENT_COMMIT:

                  GetCtrlVal (panelHandle, PANEL_V_AU_SWITCH, &switch_pos);
                GetCtrlVal (panelHandle, PANEL_CHIP_ID, &chip_sel_tmp);
                if(chip_sel_tmp==0) { imin=0; imax=4;}
                if(chip_sel_tmp==1) { imin=5; imax=9;}
                  read_dac();

/*****************************************************************************
******************** Inizio stampa valori attivi  ***************************
*****************************************************************************/
                if(switch_pos==0) {   // Switch in posizione au
                        // Inizio stampa valori attivi
                            sprintf(message_act,"Active  DAC  (last  acquisition)
settings are:\n\n");
                          strcat (message_act, "DAC Bias:\n");
                          for (i=imin; i<=imax; i++) {
                                                sprintf(message1_act,"DAC%2d=%5d
",i+1,dac_bias_act[i]);
                                if (i==imax) strcat(message1_act,"\n");
                              strcat (message_act, message1_act);

                          } // Fine stampa valori attivi
            }
                else { // Switch in posizione V
                        //Inizio stampa valori attivi
                            sprintf(message_act,"Active  DAC  (last  acquisition)
settings are:\n\n");
                          strcat (message_act, "DAC Bias:\n");
                          for (i=imin; i<=imax; i++) {
                                sprintf(message1_act,"DAC%2d=%5.2f;  ",i+1,
                                        (float)dac_bias_act[i]/LSBCAL);
                                if (i==imax) strcat(message1_act,"\n");
```

```
                            strcat (message_act, message1_act);
                          }  //Fine stampa valori attivi
                }

/*************************************************************************
*******************  Fine stampa valori attivi   ************************
*************************************************************************/



/*************************************************************************
****************** Inizio stampa valori temporanei  ***********************
*************************************************************************/
                if(switch_pos==0) {   // Switch in posizione au
                        // Inizio stampa valori temporanei
                         sprintf(message_tmp,"\n\n\nTemporary (next acquisition)
DAC settings are:\n\n");
                        strcat (message_tmp, "DAC Bias:\n");
                          for (i=imin; i<=imax; i++) {
                                              sprintf(message1_tmp,"DAC%2d=%5d;
",i+1,dac_bias_tmp[i]);
                                if (i==imax) strcat(message1_tmp,"\n");
                              strcat (message_tmp, message1_tmp);

                          } // Fine stampa valori temporanei
            }
                else { // Switch in posizione V
                        //Inizio stampa valori temporanei
                         sprintf(message_tmp,"\n\n\nTemporary (next acquisition)
DAC settings are:\n\n");
                        strcat (message_tmp, "DAC Bias:\n");
                          for (i=imin; i<=imax; i++) {
                                sprintf(message1_tmp,"DAC%2d=%5.2f;   ",i+1,
                                        (float)dac_bias_tmp[i]/LSBCAL);
                                if (i==imax) strcat(message1_tmp,"\n");
                              strcat (message_tmp, message1_tmp);
                          }  //Fine stampa valori temporanei
                }

/*************************************************************************
******************  Fine stampa valori temporanei   ***********************
*************************************************************************/

                strcat (message, message_act);
                strcat (message, message_tmp);
                  strcat (message,"\n\n Confirm these biases and send them to
Medipix?");

                   answer = ConfirmPopup ("Active/temporary values", message);
              if (answer)  {
                 if (MISSING_HARDWARE==0) {
                   turn_on_dac();
//                 mask_status=0;
//                 image_ready=0;
//                 end_acquisition=0;
                 }
              }
```

```
                break;
      }
      return 0;
}

/*****************************************************************************
*****************************************************************************
*********************** Bottone Load Mask  ***********************************
*****************************************************************************
*****************************************************************************/

int CVICALLBACK load_mask_on_Medipix (int panel, int control, int event,
                                      void *callbackData, int eventData1,
                                      int eventData2)
{
  switch (event) {
      case EVENT_COMMIT:
      load_mask();
        break;
  }
  return 0;
}


/*****************************************************************************
*****************************************************************************
********************** Fine bottone Load Mask  ***********************************
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
*****************************************************************************
******************** Bottone Start Acquisition  ***********************************
*****************************************************************************
*****************************************************************************/

int CVICALLBACK Start_acquisition (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   char filename[30];

      switch (event) {
            case EVENT_COMMIT:

                  image_ready=0;
                  acquisition_on=1;
                  SetCtrlVal(showImage,SHOW_FILE_READ,"");
                  SetCtrlVal(panelHandle, PANEL_IMAGE_READY,0);
            SetCtrlVal(panelHandle, PANEL_END_ACQUISITION,1);


                  if (MISSING_MEDIPIX==0) {
                     start_acq(RADIOGRAPHY);
                     download_data();                        //  Scrive  i  dati
acquisiti
                                                  // nella matrice data_matrix
                     LUT_conversion();            // Converte i dati in valori
```

```
                                                    // di conteggio
                }


                image_ready=1;
                acquisition_on=0;
            break;
      }
      return 0;
}

/*****************************************************************************
*****************************************************************************
*****************  Fine bottone Start Acquisition  ******************************
*****************************************************************************
*****************************************************************************/

/*****************************************************************************
**
*****************************************************************************
***********************  Bottone Show Image  ********************************
*****************************************************************************
*****************************************************************************/

int CVICALLBACK Show_image (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
  int min_Val,max_Val;

// Usate per la simulazione
  if(MISSING_HARDWARE==1) {
    FileToArray ("", data_matrix, VAL_INTEGER, 4096, 64,
                         VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_ROWS, VAL_ASCII);

    set_image_values();

    switch (event) {
        case EVENT_COMMIT:
        plot_image(data_matrix);
          break;
      }
  }  // Fine


        if(MISSING_MEDIPIX==0)  {
          int minVal, maxVal;

        cercaMinimo(data_matrix, &minVal);
          cercaMassimo(data_matrix, &maxVal);

              SetCtrlAttribute  (setColors,  SET_COLORS_MIN_VAL,  ATTR_MIN_VALUE,
minVal);
              SetCtrlAttribute  (setColors,  SET_COLORS_MIN_VAL,  ATTR_MAX_VALUE,
maxVal);
          SetCtrlVal (setColors, SET_COLORS_MIN_VAL, minVal);
```

```
              SetCtrlAttribute  (setColors,  SET_COLORS_MAX_VAL,  ATTR_MIN_VALUE,
minVal);
              SetCtrlAttribute  (setColors,  SET_COLORS_MAX_VAL,  ATTR_MAX_VALUE,
maxVal);
          SetCtrlVal (setColors, SET_COLORS_MAX_VAL, maxVal);

          switch (event) {
               case EVENT_COMMIT:
                    plot_image(data_matrix);
           break;
          }
        }
        return 0;
}


int CVICALLBACK Update_Cursor_Pos (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
  double x_pos, y_pos;
  int i, j;
  int counts;

  switch (event) {
      case EVENT_TIMER_TICK:
      GetGraphCursor(showImage, SHOW_IMAGE, 3, &y_pos, &x_pos);
        i = (int)x_pos;
        j = (int)y_pos;

        if(x_pos - (float)i > 0.5)
          i = i + 1;
        if(y_pos - (float)j > 0.5)
          j = j + 1;

        SetCtrlVal(showImage, SHOW_X_COORD, x_pos);
      SetCtrlVal(showImage, SHOW_Y_COORD, y_pos);
      SetCtrlVal(showImage, SHOW_COUNTS, data_matrix[i][j]);
      break;
  }
  return 0;
}


void set_image_values(void)
{ int min_Val;
  int max_Val;

  cercaMinimo(data_matrix, &min_Val);
  cercaMassimo(data_matrix, &max_Val);

  SetCtrlAttribute (setColors, SET_COLORS_MIN_VAL, ATTR_MIN_VALUE, min_Val);
  SetCtrlAttribute (setColors, SET_COLORS_MIN_VAL, ATTR_MAX_VALUE, max_Val);
  SetCtrlVal (setColors, SET_COLORS_MIN_VAL, min_Val);


  SetCtrlAttribute (setColors, SET_COLORS_MAX_VAL, ATTR_MIN_VALUE, min_Val);
  SetCtrlAttribute (setColors, SET_COLORS_MAX_VAL, ATTR_MAX_VALUE, max_Val);
```

```
  SetCtrlVal (setColors, SET_COLORS_MAX_VAL, max_Val);
}


void plot_image(int matrix[64][64])
{
  ColorMapEntry colors[256];
  int i, j, tot = 0;
  int d_matrix[64][64];
  int hiColor, numColors, interpColors, interpPixels;
  float media = 0;

  for (i=0; i<=63; i++)
      for (j=0; j<=63; j++) {
      d_matrix[i][j]=matrix[i][j];
      if(matrix[i][j] != 696969)
        tot += matrix[i][j];
    }

  media=(float)tot/4096.;

  SetCtrlVal(showImage, SHOW_MEDIA, media);
  SetCtrlVal(showImage, SHOW_INTEGRAL, tot);

  GetParams(d_matrix, colors, &hiColor, &numColors,
            &interpColors, &interpPixels);

  PlotIntensity(showImage, SHOW_IMAGE, d_matrix, 64, 64, VAL_INTEGER,
                     colors, hiColor, numColors, interpColors, interpPixels);
}


// Imposta i parametri dell' immagine
int GetParams (int m[64][64], ColorMapEntry colors[], int *hiColor,
               int *numColors, int *interpColors, int *interpPixels)
{
    int maxVal, minVal;
    unsigned char   loRed, hiRed, red, loGreen, hiGreen, green, loBlue, hiBlue,
                    blue;
    int             loCol, hiCol, i, j;
      int             min_val, max_val;
      // Variabili relative alla barra dei colori
       // Coordinate e numero divisioni della barra
       int Top=10;
       int Left=10;
       int Height=20;
       int Width=320;
       int Division=5;
       // Fine
       int x_pos=Left;   // Posizione all'interno della barra
       float base;  // Unità di misura dell'intervallo

       float value_start, value_end;
       int x_pos_start, x_pos_end;
       int color;

      DisplayPanel(showImage);
```

```
    GetCtrlVal (setColors, SET_COLORS_NUM, numColors);
    GetCtrlVal (setColors, SET_COLORS_LOW, &loCol);
    GetCtrlVal (setColors, SET_COLORS_HIGH, &hiCol);
    GetCtrlVal (setColors, SET_COLORS_INTERP_COL, interpColors);
    GetCtrlVal (setColors, SET_COLORS_INTERP_COL, interpPixels);

        loRed = (unsigned char)((loCol & 0x00FF0000) >> 16);
        hiRed = (unsigned char)((hiCol & 0x00FF0000) >> 16);
        loGreen = (unsigned char)((loCol & 0x0000FF00) >> 8);
        hiGreen = (unsigned char)((hiCol & 0x0000FF00) >> 8);
        loBlue = (unsigned char)(loCol & 0x000000FF);
        hiBlue = (unsigned char)(hiCol & 0x000000FF);

            cercaMinimo(m, &minVal);
            cercaMassimo(m, &maxVal);

            // Aggiusta i valori minimi e massimi
               GetCtrlVal (setColors, SET_COLORS_MIN_VAL, &min_val);
              GetCtrlVal (setColors, SET_COLORS_MAX_VAL, &max_val);

               for (i=0; i<=63; i++) {
                    for (j=0; j<=63; j++) {
                        if (m[i][j]==minVal)
                            minVal=m[i][j]-minVal+min_val;

                        if (m[i][j]==maxVal)
                            maxVal=m[i][j]-maxVal+max_val;
                   }
               }
        // Fine aggiustamento valori

//        if(trig_mode_act == 2)
//        {
//                maxVal=anin_cycles_act*2;
//          minVal=0;
//        }

        for (i = 0; i < *numColors; i++) {
            red = loRed + i * (hiRed - loRed) / (*numColors-1);
            green = loGreen + i * (hiGreen - loGreen) / (*numColors-1);
            blue = loBlue + i * (hiBlue - loBlue) / (*numColors-1);
            colors[i].color = MakeColor ((int)red, (int)green, (int)blue);
            colors[i].dataValue.valInt  =(int)  (minVal  +  (i+1)  *  (maxVal  -
minVal)
                / (*numColors));
            }
        *hiColor = MakeColor ((int)hiRed, (int)hiGreen, (int)hiBlue);

/*************************************************************************
***************  Disegna la barra dei colori  ********************************
*************************************************************************/
            CanvasClear (showImage, SHOW_COLOR_BAR, VAL_ENTIRE_OBJECT);
            SetCtrlAttribute (showImage, SHOW_COLOR_BAR, ATTR_PEN_COLOR, 0);
            CanvasDrawRect              (showImage,             SHOW_COLOR_BAR,
MakeRect(Top,Left,Height,Width),
                                    VAL_DRAW_FRAME);
```

```
            for (i=0; i<=Division; i++) {
                  static int base;
                  static float value;
                  static char valore[30];

              base=(float) (Width/Division);  // Unità di misura dell'intervallo

                  CanvasDrawLine           (showImage,           SHOW_COLOR_BAR,
MakePoint(i*base+Left,Top+Height),
                                    MakePoint(i*base+Left,Top+Height+5));

                  value=(float) (((float) (maxVal-minVal)/Division)*i+minVal);

                  sprintf(valore,"%.0f",value);

                  CreateMetaFont ("BarFont", VAL_APP_META_FONT, 10, 0, 0, 0, 0);
                  CanvasDrawTextAtPoint (showImage, SHOW_COLOR_BAR, valore,
                                                                        "BarFont",
MakePoint(i*base+Left,Top+Height+1),
                                                      VAL_UPPER_CENTER);
            }

            for (i=1; i<*numColors; i++) {

                  base=(float) (Width/((float) (maxVal-minVal)));
                                    // Coefficiente di conversione da valore
della matrice
                                    // dei dati a posizione sulla barra dei
colori

                  // Posizione punto precedente
                  value_start=(float) colors[i-1].dataValue.valInt;
                  x_pos_start=(int) (value_start*base+Left);

                  // Posizione punto corrente
                  value_end=(float) colors[i].dataValue.valInt;
                  x_pos_end=(int) (value_end*base+Left);

                  for (j=x_pos_start; j<x_pos_end; j++) {
                        color=colors[i-1].color;
                        SetCtrlAttribute       (showImage,       SHOW_COLOR_BAR,
ATTR_PEN_COLOR, color);
                        x_pos++;
                        CanvasDrawLine        (showImage,        SHOW_COLOR_BAR,
MakePoint(x_pos,Top+1),
                                          MakePoint(x_pos,Top+Height-2));
                  }

            }

            if (x_pos < Left+Width)
                  for (i=x_pos; i<Left+Width-1; i++) {
                        color=colors[*numColors-1].color;
                        SetCtrlAttribute       (showImage,       SHOW_COLOR_BAR,
ATTR_PEN_COLOR, color);
```

```
                        CanvasDrawLine        (showImage,        SHOW_COLOR_BAR,
MakePoint(i,Top+1),
                                              MakePoint(i,Top+Height-2));
                  }

/*****************************************************************************
************************  Fine Disegno  ************************************
*****************************************************************************/
    return 0;
}


void cercaMinimo(int matrix[64][64], int *min)
{   int i,j;

 *min = matrix[0][0];

 for (i=0; i<=63; i++) {
      for (j=0; j<=63; j++) {
            if (matrix[i][j]<*min)
               *min=matrix[i][j];
      }
 }
}

void cercaMassimo(int matrix[64][64], int *max)
{   int i,j;

 *max = matrix[0][0];

 for (i=0; i<=63; i++) {
      for (j=0; j<=63; j++) {
            if (matrix[i][j]>*max)
               *max=matrix[i][j];
      }
 }
}


/*****************************************************************************
*****************************************************************************
**********************  Fine bottone Show Image*****************************
*****************************************************************************
*****************************************************************************/
int CVICALLBACK Zoom_in (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            static double y2;
            static double x2;
            static double y1;
            static double x1;
            static double temp;

            case EVENT_COMMIT:
                  GetGraphCursor (showImage, SHOW_IMAGE, 1, &x1, &y1);
```

```
            GetGraphCursor (showImage, SHOW_IMAGE, 2, &x2, &y2);

            if (x1 >= x2) {
               temp=x2;
               x2=x1;
               x1=temp;
            }

            if (y1 >= y2) {
               temp=y2;
               y2=y1;
               y1=temp;
            }

                  SetAxisRange  (showImage,  SHOW_IMAGE,  VAL_MANUAL,  x1,  x2,
VAL_MANUAL,
                                       y1, y2);
                 break;
      }
      return 0;
}

int CVICALLBACK Restore (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  SetAxisRange  (showImage,  SHOW_IMAGE,  VAL_MANUAL,  0,  63,
VAL_MANUAL,
                                       0, 63);
                  break;
      }
      return 0;
}

int CVICALLBACK Colors (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            static int minVal;
            static int maxVal;
            static int button_plot=0;
            static int button_cancel=0;

            case EVENT_COMMIT:
                  DisplayPanel(setColors);

                  do { static int status;
                       static int Colors;
                       static int button;


                       status = GetUserEvent (1, &Colors, &button);

                       // Bottone Plot premuto
                       button_plot=(status==EVENT_COMMIT) && (Colors==setColors)
&&
```

```
                              (button==SET_COLORS_PLOT);
                        if (button_plot) {
                             HidePanel(setColors);
                             plot_image(data_matrix);
                        }
                        // Fine bottone Plot premuto

                        // Bottone Cancel premuto
                                    button_cancel=(status==EVENT_COMMIT)    &&
(Colors==setColors) &&
                                   (button==SET_COLORS_CANCEL);
                        if (button_cancel) {
                             HidePanel(setColors);
                        }
                        // Fine bottone Cancel premuto
                      } while(!button_plot && !button_cancel);
            break;
      }
      return 0;
}

int CVICALLBACK Profile (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   int value;
    int num_row;
      int start_row;
      int num_col;
      int start_col;
    double profile_array[64];
    double x1,x2,y1,y2,temp;

      switch (event) {
            static int status;
            static int select;
            static int panel;
            static int button;
            static int button_done=0;
            static int button_show_profile=0;
            static int exact_range=0;

            case EVENT_COMMIT:
                  SetCtrlAttribute (profile, PROFILE_DONE, ATTR_DIMMED, 0);
                  SetCtrlAttribute (profile, PROFILE_SHOW, ATTR_DIMMED, 0);
                  SetCtrlAttribute (profile, PROFILE_GRAPH, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (profile, PROFILE_OK, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (profile, PROFILE_SELECT, ATTR_VISIBLE, 1);
                  SetCtrlAttribute (profile, PROFILE_DONE, ATTR_VISIBLE, 1);
                  SetCtrlAttribute  (profile,  PROFILE_START_ROW,  ATTR_VISIBLE,
0);
                  SetCtrlAttribute  (profile,  PROFILE_START_COL,  ATTR_VISIBLE,
0);
                  SetCtrlAttribute (profile, PROFILE_NUM_ROW, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (profile, PROFILE_NUM_COL, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (profile, PROFILE_SHOW, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (profile, PROFILE_PIXEL, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (profile, PROFILE_COUNT, ATTR_VISIBLE, 0);
                  DisplayPanel(profile);
```

```
                do {
                   status = GetUserEvent (1, &panel, &button);

                   button_done=(status==EVENT_COMMIT) && (panel==profile) &&
                               (button==PROFILE_DONE);
                   if (button_done) {
                         GetCtrlVal (profile, PROFILE_SELECT, &value);
                         switch (value) {
                               case 0:  // Row Profile
                                     SetCtrlAttribute (profile, PROFILE_DONE,
ATTR_DIMMED, 1);
                                           SetCtrlAttribute    (profile,
PROFILE_START_ROW, ATTR_VISIBLE, 1);
                                           SetCtrlAttribute    (profile,
PROFILE_START_COL, ATTR_VISIBLE, 0);
                                           SetCtrlAttribute    (profile,
PROFILE_NUM_ROW, ATTR_VISIBLE, 1);
                                           SetCtrlAttribute    (profile,
PROFILE_NUM_COL, ATTR_VISIBLE, 0);
                                     SetCtrlAttribute (profile, PROFILE_SHOW,
ATTR_VISIBLE, 1);

                                    GetGraphCursor (showImage, SHOW_IMAGE, 1,
&x1, &y1);
                                    GetGraphCursor (showImage, SHOW_IMAGE, 2,
&x2, &y2);

                                    if (x1<x2) {
                                          temp=x1;
                                          x1=x2;
                                          x2=temp;
                                    }

                                     SetCtrlVal (profile, PROFILE_START_ROW,
(int) x2);
                                       SetCtrlVal  (profile,  PROFILE_NUM_ROW,
(int) (x1-x2+1.0));

                                    break;
                               case 1:  // Column Profile
                                     SetCtrlAttribute (profile, PROFILE_DONE,
ATTR_DIMMED, 1);
                                           SetCtrlAttribute    (profile,
PROFILE_START_ROW, ATTR_VISIBLE, 0);
                                           SetCtrlAttribute    (profile,
PROFILE_START_COL, ATTR_VISIBLE, 1);
                                           SetCtrlAttribute    (profile,
PROFILE_NUM_ROW, ATTR_VISIBLE, 0);
                                           SetCtrlAttribute    (profile,
PROFILE_NUM_COL, ATTR_VISIBLE, 1);
                                     SetCtrlAttribute (profile, PROFILE_SHOW,
ATTR_VISIBLE, 1);

                                    GetGraphCursor (showImage, SHOW_IMAGE, 1,
&x1, &y1);
```

```
                                        GetGraphCursor (showImage, SHOW_IMAGE, 2,
&x2, &y2);

                                        if (y1<y2) {
                                              temp=y1;
                                              y1=y2;
                                              y2=temp;
                                        }

                                         SetCtrlVal (profile, PROFILE_START_COL,
(int) y2);
                                           SetCtrlVal (profile, PROFILE_NUM_COL,
(int) (y1-y2+1.0));

                                        break;
                               }

                          }

                     } while (!button_done);

                do {
                     do {
                       status = GetUserEvent (1, &panel, &button);

                              button_show_profile=(status==EVENT_COMMIT)      &&
(panel==profile) &&
                                 (button==PROFILE_SHOW);

                       if (button_show_profile) {
                             static int value;

                             GetCtrlVal (profile, PROFILE_SELECT, &value);
                             switch (value) {
                                 case 0:
                                          GetCtrlVal (profile, PROFILE_START_ROW,
&start_row);
                                           GetCtrlVal (profile, PROFILE_NUM_ROW,
&num_row);
                                        if ((start_row+num_row)>64) {
                                            static int max_lim;
                                            static char message[50];

                                               sprintf(message,"Maximum number of
row is %d", (64-start_row));
                                                  MessagePopup ("Range error",
message);
                                             max_lim=(64-start_row);
                                                       SetCtrlVal      (profile,
PROFILE_NUM_ROW, max_lim);
                                             exact_range=0;
                                        } else {
                                          exact_range=1;
                                        }
                                 break;
                                 case 1:
```

```
                                        GetCtrlVal (profile, PROFILE_START_COL,
&start_col);
                                         GetCtrlVal (profile, PROFILE_NUM_COL,
&num_col);
                                    if ((start_col+num_col)>64) {
                                        static int max_lim;
                                        static char message[50];

                                           sprintf(message,"Maximum number of
column is %d", 64-start_col);
                                              MessagePopup ("Range error",
message);
                                          max_lim=(64-start_col);
                                                    SetCtrlVal      (profile,
PROFILE_NUM_COL, max_lim);
                                          exact_range=0;
                                    } else  {
                                      exact_range=1;
                                    }
                               break;
                           }
                     }

                  } while (!button_show_profile);
                  if (!exact_range)  {
                     button_show_profile=0;
                  }
            } while(!exact_range);


            SetCtrlAttribute (profile, PROFILE_SHOW, ATTR_DIMMED, 1);
            SetCtrlAttribute (profile, PROFILE_GRAPH, ATTR_VISIBLE, 1);
            SetCtrlAttribute (profile, PROFILE_PIXEL, ATTR_VISIBLE, 1);
            SetCtrlAttribute (profile, PROFILE_COUNT, ATTR_VISIBLE, 1);
            SetCtrlAttribute (profile, PROFILE_OK, ATTR_VISIBLE, 1);

            GetCtrlVal (profile, PROFILE_SELECT, &value);

            switch (value) {
                  case 0:
                        GetCtrlVal (profile, PROFILE_START_ROW, &start_row);
                        GetCtrlVal (profile, PROFILE_NUM_ROW, &num_row);
                         ProfileArray(data_matrix, value, start_row, num_row,
profile_array);
                        break;
                  case 1:
                        GetCtrlVal (profile, PROFILE_START_COL, &start_col);
                        GetCtrlVal (profile, PROFILE_NUM_COL, &num_col);
                         ProfileArray(data_matrix, value, start_col, num_col,
profile_array);
                        break;
            }


            SetCtrlAttribute (profile, PROFILE_GRAPH, ATTR_XNAME, "Pixels");

            DeleteGraphPlot (profile, PROFILE_GRAPH, -1, VAL_IMMEDIATE_DRAW);
```

```
              PlotY (profile, PROFILE_GRAPH, profile_array, 64, VAL_DOUBLE,
                           VAL_THIN_STEP,   VAL_EMPTY_SQUARE,   VAL_SOLID,   1,
VAL_RED);

              do {
                    status = GetUserEvent (1, &panel, &button);

                    button_done=(status==EVENT_COMMIT) && (panel==profile) &&
                                (button==PROFILE_OK);
                    if (button_done) {
                          HidePanel(profile);
                    }

              } while (!button_done);

            break;
      }
      return 0;
}


void ProfileArray(int matrix[64][64], int select, int start, int num, double
prof_array[64])
{   int i,j;
    int end=start+num;
      int sum[64];
      int d_matrix[64][64];

                  for (j=0; j<64; j++)
                        for (i=0; i<64; i++)
                              d_matrix[j][i]=matrix[i][j];

      switch (select) {
            case 0:              // Row Profile
                  for (j=0; j<64; j++) {
                        sum[j]=0;
                        for (i=start; i<end; i++) {
                            sum[j]=sum[j]+d_matrix[i][j];
                        }
                  }
                  break;
            case 1:              // Col Profile
                  for (i=0; i<64; i++) {
                        sum[i]=0;
                        for (j=start; j<end; j++) {
                            sum[i]=sum[i]+d_matrix[i][j];
                        }
                  }
                  break;
      }

      for (i=0; i<64; i++) {
            prof_array[i]=(double) (sum[i]/num);
      }

}
```

```
int CVICALLBACK Profile_timer (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            static double pixel;
            static double count;
            case EVENT_TIMER_TICK:
                  GetGraphCursor (profile, PROFILE_GRAPH, 1, &pixel, &count);

                  SetCtrlVal (profile, PROFILE_PIXEL, pixel);
                  SetCtrlVal (profile, PROFILE_COUNT, count);
                  break;
      }
      return 0;
}

int CVICALLBACK Load_image (int panel, int control, int event,
                                 void *callbackData, int eventData1,
                                 int eventData2)
{

  switch (event) {
    static int select;
      static char filename[300];
      char *prova;

      case EVENT_COMMIT:
        select = FileSelectPopup (Directory_Name, "*.dat", "*.dat", "Load File",
                                                 VAL_LOAD_BUTTON,  0,  0,  1,  0,
filename);
        if (select) {
            FileToArray (filename, data_matrix, VAL_INTEGER, 4096, 64,
                                  VAL_GROUPS_TOGETHER,     VAL_GROUPS_AS_ROWS,
VAL_ASCII);
          DeleteGraphPlot (showImage, SHOW_IMAGE, -1, VAL_IMMEDIATE_DRAW);
            set_image_values();
            plot_image(data_matrix);
            copiastr(filename);
        }
      break;
  }
  return 0;
}


int CVICALLBACK Save_image (int panel, int control, int event,
                           void *callbackData, int eventData1, int eventData2)
{
  char file0[300];
  char mask_file[300],arch_file[300];

  switch (event) {
      static int select;
      static char filename[300];

      case EVENT_COMMIT:
```

```
        select = FileSelectPopup (Directory_Name, "*.dat", "*.dat", "Save File",
                                                  VAL_SAVE_BUTTON,  0,  0,  1,  0,
filename);
        if (select)
          ArrayToFile (filename, data_matrix, VAL_INTEGER, 4096, 64,
                               VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_ROWS,
                               VAL_SEP_BY_TAB, 0,VAL_ASCII, VAL_TRUNCATE);
        sprintf(file0,"%s.info",filename);

        fpout=fopen(file0,"w");
        GetCtrlVal (panelHandle, PANEL_MASKFILE, mask_file);
        GetCtrlVal (panelHandle, PANEL_FILENAME, arch_file);
        fprintf(fpout,"loaded mask file = %s\n",mask_file);
        fprintf(fpout,"loaded archive file = %s\n",arch_file);

      fprintf(fpout,"tau1(ms) = %d",tau1_act);
        fprintf(fpout,"\ndac values (Volt) = %.3f %.3f %.3f %.3f %.3f\n",
                dac_bias_act[0]/LSBCAL, dac_bias_act[1]/LSBCAL,dac_bias_act[2]/
                LSBCAL,dac_bias_act[3]/LSBCAL,dac_bias_act[4]/LSBCAL);
        fclose(fpout);


        copiastr(filename);
        break;
  }
  return 0;
}


int CVICALLBACK Quit_Show_Image (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                HidePanel(showImage);
                HidePanel(showimage2);
                  break;
      }
      return 0;
}


/*********************************************************************
*************** Visualizzazione Autoradiografia *********************
*********************************************************************/
int CVICALLBACK Autoradiography (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{   int select,sel;
    int autorad;
    int button;
    int button_ok;
    int button_start_acq;
      int button_Exit;
      int total_time;
      int acquisition_time;
      int button_autorad_ok;
      double n;
```

```
      char filename[30];
      int i;


      switch(event) {
            case EVENT_COMMIT:
                     tau0_act=tau0_tmp;
                     tau1_act=tau1_tmp;
                     tau2_act=tau2_tmp;

                  DisplayPanel(showAutorad);
                  DeleteGraphPlot     (showAutorad,     AUTORAD_IMAGE,     -1,
VAL_IMMEDIATE_DRAW);
                  CanvasClear         (showAutorad,          AUTORAD_COLOR_BAR,
VAL_ENTIRE_OBJECT);
                  SetCtrlVal (showAutorad, AUTORAD_ACQUISITION_TIME, tau1_act);
                  SetCtrlAttribute (showAutorad, AUTORAD_OK, ATTR_DIMMED, 0);

               if (TIME_UNIT==0)    // Unita di misura: millisecondi
               {
                        SetCtrlAttribute   (showAutorad,   AUTORAD_TOTAL_TIME,
ATTR_LABEL_TEXT,
                                            "Total Time (ms)");

            }
                else                        // Unita di misura: secondi
                {
                        SetCtrlAttribute   (showAutorad,   AUTORAD_TOTAL_TIME,
ATTR_LABEL_TEXT,
                                      "Total Time (sec)");

                }


                  SetCtrlAttribute      (showAutorad,      AUTORAD_TOTAL_TIME,
ATTR_DFLT_VALUE,
                                         tau1_act);
                  SetCtrlAttribute      (showAutorad,      AUTORAD_TOTAL_TIME,
ATTR_MIN_VALUE,
                                         tau1_act);
                  SetCtrlAttribute      (showAutorad,      AUTORAD_TOTAL_TIME,
ATTR_INCR_VALUE,
                                         tau1_act);
                  SetCtrlAttribute      (showAutorad,      AUTORAD_SHOW_IMAGE,
ATTR_DIMMED, 1);
                  SetCtrlAttribute (showAutorad, AUTORAD_START_ACQ, ATTR_DIMMED,
1);
                  SetCtrlAttribute (showAutorad, AUTORAD_STOP_ACQ, ATTR_DIMMED,
1);
                  SetCtrlAttribute      (showAutorad,      AUTORAD_RESUME_ACQ,
ATTR_DIMMED, 1);
            SetCtrlAttribute (showAutorad, AUTORAD_EXIT, ATTR_DIMMED, 1);
            SetCtrlAttribute  (showAutorad,  AUTORAD_SLIDE_COUNT,  ATTR_VISIBLE,
0);
            SetCtrlAttribute (showAutorad, AUTORAD_TIME, ATTR_VISIBLE, 0);

                  do {
```

```
                    select = GetUserEvent (1, &autorad, &button);
                    button_ok=(select==EVENT_COMMIT) && (autorad==showAutorad)
&&
                              (button==AUTORAD_OK);

                      if (button_ok) {
                         // Calcola il numero di immagini visualizzate
                              GetCtrlVal   (showAutorad,   AUTORAD_TOTAL_TIME,
&total_time);
                  GetCtrlVal (showAutorad, AUTORAD_ACQUISITION_TIME,
                               &acquisition_time);
                  n=(total_time/acquisition_time);
                         SetCtrlVal (showAutorad, AUTORAD_SLIDE, n);
                          SetCtrlAttribute (showAutorad, AUTORAD_SLIDE_COUNT,
ATTR_MAX_VALUE, n);

                         // Chiede il tipo di autoradiografia
                         DisplayPanel (autoradType);

                         do
                         {
                            sel = GetUserEvent (1, &autorad, &button);
                                    button_autorad_ok=(sel==EVENT_COMMIT)   &&
(autorad==autoradType) &&
                                          (button==AUTOR_MOD_OK);
                            if (button_autorad_ok)
                            {
                              HidePanel (autoradType);
                            }

                         } while (!button_autorad_ok);


                         // Chiede il nome del file
                         PromptPopup ("", "Insert the filename where you want
to save the images",
                                                filename, 20);
                         SetCtrlVal (showAutorad, AUTORAD_FILENAME, filename);
                           SetCtrlAttribute  (showAutorad,  AUTORAD_START_ACQ,
ATTR_DIMMED, 0);
                               SetCtrlAttribute   (showAutorad,   AUTORAD_OK,
ATTR_DIMMED, 1);
                               SetCtrlAttribute   (showAutorad,   AUTORAD_EXIT,
ATTR_DIMMED, 0);
                    }

                  } while (!button_ok);

                  do {
                      select = GetUserEvent (1, &autorad, &button);
                                 button_start_acq=(select==EVENT_COMMIT)    &&
(autorad==showAutorad) &&
                                       (button==AUTORAD_START_ACQ);

                      if (button_start_acq) {
                          SetCtrlAttribute (showAutorad, AUTORAD_SLIDE_COUNT,
ATTR_VISIBLE, 1);
```

```
                           Start_acq_autoradiography();
                        }

                                          button_Exit=(select==EVENT_COMMIT)     &&
(autorad==showAutorad) &&
                                    (button==AUTORAD_EXIT);

                        if (button_Exit) {
                           HidePanel (showAutorad);
                           }
                  } while (!button_start_acq && !button_Exit);


                  break;
      }
      return 0;
}

/*********************************************************************
************Fine visualizzazione autoradiografia **********************
*********************************************************************/

void Start_acq_autoradiography (void)
{
  int i=0;
  int j,ii,jj;
  int a;
  int acq_time;
  float time=0.0;
  double slide_num;
  int n;
  int t_i,overtime_off=3;
  double t_f,overtime_s=1.06;

  int autorad_type;
  char filesaved[300];
  char filename[300];
  char file0[300];
  char num_file[10];
  int Exit=0;

  char mask_file[200], arch_file[200];

// Variabili grafiche
  int select;
  int autorad;
  int button;
  int button_stop;
  int sel;
  int button_resume;
  int button_show_image;
  int button_Exit=0;
  int data_matrix_tot[64][64];

  for (ii=0; ii<64; ii++)
    for (jj=0; jj<64; jj++)
      data_matrix_tot[ii][jj]=0;
```

```
  if (MISSING_HARDWARE==0) {
    SetCtrlAttribute (showAutorad, AUTORAD_START_ACQ, ATTR_DIMMED, 1);
    SetCtrlAttribute (showAutorad, AUTORAD_STOP_ACQ, ATTR_DIMMED, 0);
    SetCtrlAttribute (showAutorad, AUTORAD_RESUME_ACQ, ATTR_DIMMED, 1);

    SetCtrlVal (showAutorad, AUTORAD_ACQUISITION_ON, 1);
    SetCtrlVal (showAutorad, AUTORAD_STOP, 0);

    GetCtrlVal (showAutorad, AUTORAD_SLIDE, &slide_num);
    GetCtrlVal (showAutorad, AUTORAD_FILENAME, filename);

    n=(int) slide_num;

    do {
        start_acq(AUTORADIOGRAPHY);    //da modificare eliminando il polling

      time=0;
        sprintf(num_file,"%d",i);
        sprintf(filesaved,"%s%s_",PATHF,filename);
        strcat(filesaved,num_file);
        strcat(filesaved,".dat");

/*****************************************************************
******  Attende il tempo necessario al completamento della  *********
******    scrittura dei dati sulla RAM della scheda VME         *********
******          o che l'utente generi un evento             *********
*****************************************************************/
        if (TIME_UNIT==0)
            acq_time=(int) ((tau1_act/1000.0)+overtime_off);
         else {
          t_f= tau1_act*overtime_s;
          t_i= (int) t_f;
          if (tau1_act < 100)
            t_i++;
            acq_time=t_i + overtime_off;
        }

        SetCtrlAttribute (showAutorad, AUTORAD_TIME, ATTR_MAX_VALUE,
                          (double) acq_time);
      SetCtrlAttribute (showAutorad, AUTORAD_TIME, ATTR_VISIBLE, 1);
        SetCtrlVal (showAutorad, AUTORAD_SLIDE_COUNT, (float)(i));

        do {
          select = GetUserEvent (0, &autorad, &button);
/****************  Bottone Stop  ******************/
        button_stop=(select==EVENT_COMMIT) &&
                    (autorad==showAutorad) &&
                    (button==AUTORAD_STOP_ACQ);
            if (button_stop) {
              SetCtrlVal (showAutorad, AUTORAD_STOP, 1);
              if (i!=0)
// Il bottone Show image va visualizzazto dalla seconda acquisizione in poi
                 SetCtrlAttribute (showAutorad, AUTORAD_SHOW_IMAGE, ATTR_DIMMED,
0);

            SetCtrlAttribute (showAutorad, AUTORAD_STOP_ACQ, ATTR_DIMMED, 1);
```

```
          SetCtrlAttribute (showAutorad, AUTORAD_RESUME_ACQ, ATTR_DIMMED, 0);

            do {
            if (time<(acq_time-0.5)) {
                  Delay(0.5);
                    time=time+0.5;
                  }

            sel = GetUserEvent (0, &autorad, &button);
/***************  Bottone Resume  *****************/
            button_resume=(select==EVENT_COMMIT) &&
                          (autorad==showAutorad) &&
                          (button==AUTORAD_RESUME_ACQ);
                if (button_resume) {
                    SetCtrlVal (showAutorad, AUTORAD_STOP, 0);
                          SetCtrlAttribute    (showAutorad,    AUTORAD_STOP_ACQ,
ATTR_DIMMED, 0);
                        SetCtrlAttribute    (showAutorad,    AUTORAD_RESUME_ACQ,
ATTR_DIMMED, 1);
                        SetCtrlAttribute    (showAutorad,    AUTORAD_SHOW_IMAGE,
ATTR_DIMMED, 1);
                  }
/*************  Fine bottone Resume  **************/

/*************  Bottone Show Image  ***************/
            button_show_image=(select==EVENT_COMMIT) &&
                              (autorad==showAutorad) &&
                              (button==AUTORAD_SHOW_IMAGE);
                if (button_show_image) {
                    DisplayPanel (showImage);
                    set_image_values();
                    plot_image(data_matrix);
                  }
/************  Fine bottone Show Image  *************/

/****************  Bottone Exit  ******************/
            button_Exit=(sel==EVENT_COMMIT) &&
                        (autorad==showAutorad) &&
                        (button==AUTORAD_EXIT);
                if (button_Exit) {
                    button_resume=1;
                    HidePanel(showAutorad);
                  }
/**************  Fine bottone Exit  ***************/
          } while (!button_resume);

        }
/**************  Fine bottone Stop  ***************/

/****************  Bottone Exit  ******************/
        button_Exit=(select==EVENT_COMMIT) &&
                    (autorad==showAutorad) &&
                    (button==AUTORAD_EXIT);
            if (button_Exit) {
              Exit=1;
              HidePanel(showAutorad);
            }
```

```
/**************  Fine bottone Exit  ***************/

        Delay(0.5);
        time=time + 0.5;
            SetCtrlVal (showAutorad, AUTORAD_TIME, time);

        } while ((int) time<acq_time && !Exit);

/*****************************************************************
******************      Fine attesa      ************************
*****************************************************************/
      if ((a=modify_register(status_reg,0,bit_start,"R"))!=0) {
        Print_error(12);   // Error: Acquisition not completed!!
            Exit=1;
      }

        download_data();
        LUT_conversion();

        set_image_values();

        GetCtrlVal (autoradType, AUTOR_MOD, &autorad_type);
        if (autorad_type==0) {  // Modalita integrale
            for(ii=0;ii<64;ii++)
              for(jj=0;jj<64;jj++)
                data_matrix_tot[ii][jj]= data_matrix_tot[ii][jj] +
                                         data_matrix[ii][jj];
        plot_slide(data_matrix_tot);

/********** Salva Immagine  **************/
            if (SAVEDAT==0 || i == n-1)
              ArrayToFile (filesaved, data_matrix_tot, VAL_UNSIGNED_INTEGER,
                           4096, 64, VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_ROWS,
                           VAL_SEP_BY_TAB, 0, VAL_ASCII, VAL_TRUNCATE);

         }
         else     {                    // Modalita Slide
            plot_slide(data_matrix);
/********** Salva Immagine  **************/
            if(SAVEDAT==0 || i == n-1)
               ArrayToFile (filesaved, data_matrix, VAL_UNSIGNED_INTEGER, 4096,
64,
                           VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_ROWS,
                           VAL_SEP_BY_TAB, 0, VAL_ASCII, VAL_TRUNCATE);
        }

        sprintf(file0,"%s.info",filesaved);
      if (SAVEDAT==0 || i == n-1) {
        fpout=fopen(file0,"w");
        GetCtrlVal (panelHandle, PANEL_MASKFILE, mask_file);
      GetCtrlVal (panelHandle, PANEL_FILENAME, arch_file);
      fprintf(fpout,"loaded mask file = %s\n",mask_file);
      fprintf(fpout,"loaded archive file = %s\n",arch_file);

        fprintf(fpout,"tau1(ms) = %d",tau1_act);
          fprintf(fpout, "\ndac values (Volt) = %.3f %.3f %.3f %.3f %.3f\n",
                  dac_bias_act[0]/LSBCAL, dac_bias_act[1]/LSBCAL,
```

```
                dac_bias_act[2]/LSBCAL, dac_bias_act[3]/LSBCAL,
                dac_bias_act[4]/LSBCAL);
            fclose(fpout);
        }
        strcpy(filesaved,filename);
        i++;
        SetCtrlVal (showAutorad, AUTORAD_SLIDE_COUNT, (double) i);

    } while(i<n && !Exit);

    SetCtrlVal (showAutorad, AUTORAD_ACQUISITION_ON, 0);
      SetCtrlAttribute (showAutorad, AUTORAD_SHOW_IMAGE, ATTR_DIMMED, 0);
      SetCtrlAttribute (showAutorad, AUTORAD_STOP_ACQ, ATTR_DIMMED, 1);
      SetCtrlAttribute (showAutorad, AUTORAD_RESUME_ACQ, ATTR_DIMMED, 1);

      do {
        select = GetUserEvent (0, &autorad, &button);
/***************  Bottone Exit  ******************/
      button_Exit=(select==EVENT_COMMIT) &&
                  (autorad==showAutorad) &&
                  (button==AUTORAD_EXIT);
        if (button_Exit)
          HidePanel(showAutorad);

/**************  Fine bottone Exit  ***************/
/*************  Bottone Show Image  ***************/
        button_show_image = (select==EVENT_COMMIT) &&
                            (autorad==showAutorad) &&
                            (button==AUTORAD_SHOW_IMAGE);
        if (button_show_image) {
          DisplayPanel (showImage);
          set_image_values();
          plot_image(data_matrix);
        }
/***********  Fine bottone Show Image  ************/
    } while (!button_Exit && !Exit);
  }

}


/*********************************************************************
************Fine visualizzazione autoradiografia *********************
*********************************************************************/


/*********************************************************************
*************   Stampa un messaggio di errore  **********************
*********************************************************************/
void Print_error(int err_code)
{
  switch (err_code) {
        case 1:
          MessagePopup ("Error !!", "VME write error on timer setup");
              break;
        case 2:
              MessagePopup ("Error !!", "VME write error on chip_sel setup");
```

```
              break;
        case 3:
              MessagePopup ("Error !!", "VME write error on RAM start address
setup");
              break;
        case 4:
               MessagePopup ("Error !!", "VME write error on cycle number
setup");
              break;
        case 5:
          MessagePopup ("Error !!", "VME write error on setup counters");

              break;
        case 6:
          MessagePopup ("Error !!", "Vdd is not on");
              break;
        case 7:
          MessagePopup ("Error !!", "Vdda is not on");
              break;
        case 8:
              MessagePopup ("Error !!", "VME write error on DAC channel");
              break;
        case 9:
          MessagePopup ("Error !!", "Vdd is not off");
              break;
        case 10:
              MessagePopup ("Error !!", "Vdda is not off");
              break;
        case 11:
              MessagePopup ("Error !!", "Acquisition Aborted");
              break;
        case 12:
              MessagePopup ("Error !!", "Acquisition not completed");
              break;
        case 13:
              MessagePopup ("Warning !!", "Check external power supply");
              break;
        case 14:
              MessagePopup ("Warning !!", "Turn on power supply");
              break;
        case 15:
              MessagePopup ("Warning !!", "Turn off external power supply");
              break;
  }

}


void plot_slide(int matrix[64][64])
{   ColorMapEntry   colors[256];
    int i,j;
      int d_matrix[64][64];
      int hiColor, numColors, interpColors, interpPixels;

      // Trasposta della matrice data_matrix
      for (i=0; i<=63; i++) {
            for (j=0; j<=63; j++) {
```

```
               d_matrix[i][j]=matrix[i][j];  /* trasposta della matrice */
            }
    }
    // Fine

      GetSlideParams (d_matrix, colors, &hiColor, &numColors,
               &interpColors, &interpPixels);

      PlotIntensity (showAutorad, AUTORAD_IMAGE, d_matrix, 64, 64, VAL_INTEGER,
                                 colors,   hiColor,   numColors,   interpColors,
interpPixels);
}

// Imposta i parametri dell' immagine
int GetSlideParams (int m[64][64], ColorMapEntry colors[], int *hiColor,
               int *numColors, int *interpColors, int *interpPixels)
{
    int maxVal, minVal;
    unsigned char   loRed, hiRed, red, loGreen, hiGreen, green, loBlue, hiBlue,
                    blue;
    int             loCol, hiCol, i, j;
      int             min_val, max_val;
      // Variabili relative alla barra dei colori
       // Coordinate e numero divisioni della barra
       int Top=10;
       int Left=10;
       int Height=20;
       int Width=320;
       int Division=5;
       // Fine
       int x_pos=Left;   // Posizione all'interno della barra
       float base;  // Unità di misura dell'intervallo

       float value_start, value_end;
       int x_pos_start, x_pos_end;
       int color;

//    DisplayPanel(showImage);

    GetCtrlVal (setColors, SET_COLORS_NUM, numColors);
    GetCtrlVal (setColors, SET_COLORS_LOW, &loCol);
    GetCtrlVal (setColors, SET_COLORS_HIGH, &hiCol);
    GetCtrlVal (setColors, SET_COLORS_INTERP_COL, interpColors);
    GetCtrlVal (setColors, SET_COLORS_INTERP_COL, interpPixels);



        loRed = (unsigned char)((loCol & 0x00FF0000) >> 16);
        hiRed = (unsigned char)((hiCol & 0x00FF0000) >> 16);
        loGreen = (unsigned char)((loCol & 0x0000FF00) >> 8);
        hiGreen = (unsigned char)((hiCol & 0x0000FF00) >> 8);
        loBlue = (unsigned char)(loCol & 0x000000FF);
        hiBlue = (unsigned char)(hiCol & 0x000000FF);

            cercaMinimo(m, &minVal);
            cercaMassimo(m, &maxVal);
```

```
            // Aggiusta i valori minimi e massimi
               GetCtrlVal (setColors, SET_COLORS_MIN_VAL, &min_val);
            GetCtrlVal (setColors, SET_COLORS_MAX_VAL, &max_val);
/*
               for (i=0; i<=63; i++) {
                    for (j=0; j<=63; j++) {
                        if (m[i][j]==minVal)
                            minVal=m[i][j]-minVal+min_val;

                        if (m[i][j]==maxVal)
                            maxVal=m[i][j]-maxVal+max_val;
                  }
               }
        // Fine aggiustamento valori

*/
        for (i = 0; i < *numColors; i++) {
            red = loRed + i * (hiRed - loRed) / (*numColors-1);
            green = loGreen + i * (hiGreen - loGreen) / (*numColors-1);
            blue = loBlue + i * (hiBlue - loBlue) / (*numColors-1);
            colors[i].color = MakeColor ((int)red, (int)green, (int)blue);
            colors[i].dataValue.valInt  =(int)  (minVal  +  (i+1)  *  (maxVal  -
minVal)
                / (*numColors));
            }
        *hiColor = MakeColor ((int)hiRed, (int)hiGreen, (int)hiBlue);

/*************************************************************************
***************  Disegna la barra dei colori  ********************************
*************************************************************************/
            CanvasClear (showAutorad, AUTORAD_COLOR_BAR, VAL_ENTIRE_OBJECT);
            SetCtrlAttribute  (showAutorad,  AUTORAD_COLOR_BAR,  ATTR_PEN_COLOR,
0);
            CanvasDrawRect           (showAutorad,           AUTORAD_COLOR_BAR,
MakeRect(Top,Left,Height,Width),
                                   VAL_DRAW_FRAME);

            for (i=0; i<=Division; i++) {
                  static int base;
                  static float value;
                  static char valore[10];

              base=(float) (Width/Division);  // Unità di misura dell'intervallo

                  CanvasDrawLine        (showAutorad,        AUTORAD_COLOR_BAR,
MakePoint(i*base+Left,Top+Height),
                                     MakePoint(i*base+Left,Top+Height+5));

                  value=(float) (((float) (maxVal-minVal)/Division)*i+minVal);

                  sprintf(valore,"%.0f",value);

                  CreateMetaFont ("BarFont", VAL_APP_META_FONT, 10, 0, 0, 0, 0);
                  CanvasDrawTextAtPoint (showAutorad, AUTORAD_COLOR_BAR, valore,
                                                                        "BarFont",
MakePoint(i*base+Left,Top+Height+1),
                                                  VAL_UPPER_CENTER);
```

```
            }


            for (i=1; i<*numColors; i++) {

                  base=(float) (Width/((float) (maxVal-minVal)));
                                    // Coefficiente di conversione da valore
della matrice
                                    // dei dati a posizione sulla barra dei
colori

                  // Posizione punto precedente
                  value_start=(float) colors[i-1].dataValue.valInt;
                  x_pos_start=(int) (value_start*base+Left);

                  // Posizione punto corrente
                  value_end=(float) colors[i].dataValue.valInt;
                  x_pos_end=(int) (value_end*base+Left);

                  for (j=x_pos_start; j<x_pos_end; j++) {
                        color=colors[i-1].color;
                        SetCtrlAttribute    (showAutorad,     AUTORAD_COLOR_BAR,
ATTR_PEN_COLOR, color);
                        x_pos++;
                        CanvasDrawLine      (showAutorad,     AUTORAD_COLOR_BAR,
MakePoint(x_pos,Top+1),
                                          MakePoint(x_pos,Top+Height-2));
                  }

            }


            if (x_pos < Left+Width) {
                  for (i=x_pos; i<Left+Width-1; i++) {
                        color=colors[*numColors-1].color;
                        SetCtrlAttribute    (showAutorad,     AUTORAD_COLOR_BAR,
ATTR_PEN_COLOR, color);
                        CanvasDrawLine      (showAutorad,     AUTORAD_COLOR_BAR,
MakePoint(i,Top+1),
                                          MakePoint(i,Top+Height-2));
                  }
            }


/*************************************************************************
************************  Fine Disegno  ***********************************
*************************************************************************/

    return 0;
}

int CVICALLBACK update_power (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
      {
            case EVENT_COMMIT:
```

```
                switch_on=check_power();
                break;
      }
      return 0;
}

int CVICALLBACK b_test (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
      {
            case EVENT_COMMIT:
            //    basic_test();
                  break;
      }
      return 0;
}


int CVICALLBACK Motor (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
      {   int x,y,z;
          int value;
          int move_type;     //Tipo di movimento:
                             // 1   movimento assoluto
                             // 2   movimento relativo
          char comando[300];

          int button;
          int select;
          int button_moveabs;
          int button_reset;
          int button_stop;
          int button_moverel;
          int button_tell_pos;
          int button_ok;
          int button_exit;

            case EVENT_COMMIT:

            if(MOTORFLAG == 0) return 0;

            motore(&x,&y,&z);

                  DisplayPanel (motor);
                  SetCtrlVal (motor, MOTOR_XAXIS, x);
                  SetCtrlVal (motor, MOTOR_YAXIS, y);
                  SetCtrlVal (motor, MOTOR_ZAXIS, z);

                  SetCtrlAttribute (motor, MOTOR_XCOORD, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (motor, MOTOR_YCOORD, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (motor, MOTOR_ZCOORD, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (motor, MOTOR_OK, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (motor, MOTOR_MOVE_TYPE, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (motor, MOTOR_DECORATION_3, ATTR_VISIBLE, 0);
```

```
                do
                {

                      select = GetUserEvent (1, &panel, &button);

                      button_stop=(select==EVENT_COMMIT) && (panel==motor) &&
                                              (button ==MOTOR_STOP);
                      if (button_stop)
                      {
                            motor_stop();
                            tell_position(&x,&y,&z);
                            SetCtrlVal (motor, MOTOR_XAXIS, x);
                            SetCtrlVal (motor, MOTOR_YAXIS, y);
                            SetCtrlVal (motor, MOTOR_ZAXIS, z);

                      }


                      button_reset=(select==EVENT_COMMIT) && (panel==motor) &&
                                              (button ==MOTOR_RESET_M);
                      if (button_reset)
                      {
                          mm2000();
//                          tell_position(&x,&y,&z);
                            SetCtrlVal (motor, MOTOR_XAXIS, x);
                            SetCtrlVal (motor, MOTOR_YAXIS, y);
                            SetCtrlVal (motor, MOTOR_ZAXIS, z);
                      }


                      button_tell_pos=(select==EVENT_COMMIT) && (panel==motor)
&&
                                              (button ==MOTOR_TELL_P);
                      if (button_tell_pos)
                      {
                            tell_position(&x,&y,&z);
                            SetCtrlVal (motor, MOTOR_XAXIS, x);
                            SetCtrlVal (motor, MOTOR_YAXIS, y);
                            SetCtrlVal (motor, MOTOR_ZAXIS, z);
                      }

                      button_moveabs=(select==EVENT_COMMIT) &&(panel==motor)
                                    && (button==MOTOR_MOVEABS);
                      if (button_moveabs)
                      {
                             SetCtrlVal  (motor,  MOTOR_MOVE_TYPE,  "Absolute
Position");

                         // impostazioni asse x
                                    SetCtrlAttribute    (motor,    MOTOR_XCOORD,
ATTR_LABEL_TEXT, "X-coor");
                                    SetCtrlAttribute    (motor,    MOTOR_XCOORD,
ATTR_MIN_VALUE, 0);
```

```
                                    SetCtrlAttribute    (motor,    MOTOR_XCOORD,
ATTR_MAX_VALUE, 50000);
                            SetCtrlVal (motor, MOTOR_XCOORD, x);

                            // impostazioni asse y
                                    SetCtrlAttribute    (motor,    MOTOR_YCOORD,
ATTR_LABEL_TEXT, "Y-coor");
                                    SetCtrlAttribute    (motor,    MOTOR_YCOORD,
ATTR_MIN_VALUE, 0);
                                    SetCtrlAttribute    (motor,    MOTOR_YCOORD,
ATTR_MAX_VALUE, 50000);
                            SetCtrlVal (motor, MOTOR_YCOORD, y);

                            // impostazioni asse z
                     SetCtrlAttribute (motor, MOTOR_ZCOORD, ATTR_LABEL_TEXT, "Z-
coor");
                    SetCtrlAttribute (motor, MOTOR_ZCOORD, ATTR_MIN_VALUE, 0);
                SetCtrlAttribute (motor, MOTOR_ZCOORD, ATTR_MAX_VALUE, 50000);
                            SetCtrlVal (motor, MOTOR_ZCOORD, z);

                        SetCtrlAttribute (motor, MOTOR_XCOORD, ATTR_VISIBLE, 1);
                        SetCtrlAttribute (motor, MOTOR_YCOORD, ATTR_VISIBLE, 1);
                        SetCtrlAttribute (motor, MOTOR_ZCOORD, ATTR_VISIBLE, 1);
                        SetCtrlAttribute (motor, MOTOR_OK, ATTR_VISIBLE, 1);
                                            SetCtrlAttribute        (motor,
MOTOR_MOVE_TYPE,ATTR_VISIBLE, 1);
                            SetCtrlAttribute (motor, MOTOR_DECORATION_3,
                                             ATTR_VISIBLE, 1);

                            move_type=0;   //Movimento assoluto

                        }

                        button_moverel=(select==EVENT_COMMIT) &&(panel==motor)
                                       && (button==MOTOR_MOVEREL);
                        if (button_moverel)
                        {
                               SetCtrlVal  (motor,  MOTOR_MOVE_TYPE,  "Relative
Displacement");

                            // impostazioni asse x
                            GetCtrlVal (motor, MOTOR_XAXIS, &value);
                                    SetCtrlAttribute    (motor,    MOTOR_XCOORD,
ATTR_LABEL_TEXT, "ðx");
                                    SetCtrlAttribute    (motor,    MOTOR_XCOORD,
ATTR_MIN_VALUE, -value);
                                    SetCtrlAttribute    (motor,    MOTOR_XCOORD,
ATTR_MAX_VALUE, 50000-value);
                            SetCtrlVal (motor, MOTOR_XCOORD, 0);

                            // impostazioni asse y
                            GetCtrlVal (motor, MOTOR_YAXIS, &value);
                                    SetCtrlAttribute    (motor,    MOTOR_YCOORD,
ATTR_LABEL_TEXT, "ðy");
                                    SetCtrlAttribute    (motor,    MOTOR_YCOORD,
ATTR_MIN_VALUE, -value);
```

```
                                    SetCtrlAttribute    (motor,    MOTOR_YCOORD,
ATTR_MAX_VALUE, 50000-value);
                           SetCtrlVal (motor, MOTOR_YCOORD, 0);

                           // impostazioni asse z
                           GetCtrlVal (motor, MOTOR_ZAXIS, &value);
                                    SetCtrlAttribute    (motor,    MOTOR_ZCOORD,
ATTR_LABEL_TEXT, "ðz");
                                    SetCtrlAttribute    (motor,    MOTOR_ZCOORD,
ATTR_MIN_VALUE, -value);
                                    SetCtrlAttribute    (motor,    MOTOR_ZCOORD,
ATTR_MAX_VALUE, 50000-value);
                           SetCtrlVal (motor, MOTOR_ZCOORD, 0);

                           SetCtrlAttribute (motor, MOTOR_XCOORD, ATTR_VISIBLE,
1);
                           SetCtrlAttribute (motor, MOTOR_YCOORD, ATTR_VISIBLE,
1);
                           SetCtrlAttribute (motor, MOTOR_ZCOORD, ATTR_VISIBLE,
1);
                           SetCtrlAttribute (motor, MOTOR_OK, ATTR_VISIBLE, 1);
                                        SetCtrlAttribute        (motor,
MOTOR_MOVE_TYPE,ATTR_VISIBLE, 1);
                           SetCtrlAttribute (motor, MOTOR_DECORATION_3,
                                     ATTR_VISIBLE, 1);

                           move_type=1;         // movimento relativo

                        }



                        button_ok=(select==EVENT_COMMIT) && (panel==motor)
                                  && (button==MOTOR_OK);
                        if (button_ok)
                        {
                           // Spostamento lungo l'asse x
                           switch (move_type)
                           {
                                 case 0:  // movimento assoluto
                               GetCtrlVal (motor, MOTOR_XCOORD, &value);
                                   sprintf(comando,"1pa%d",value);
                                   send_comman(comando);
                                       break;
                                 case 1:  // movimento relativo
                                GetCtrlVal (motor, MOTOR_XAXIS, &x); //posizione
x corrente
                               GetCtrlVal (motor, MOTOR_XCOORD, &value);
                                   value=value+x;
                                   sprintf(comando,"1pa%d",value);
                                   send_comman(comando);
                                       break;
                           }


                           // Spostamento lungo l'asse y
                           switch (move_type)
```

```
                        {
                              case 0:  // movimento assoluto
                            GetCtrlVal (motor, MOTOR_YCOORD, &value);
                                sprintf(comando,"2pa%d",value);
                                send_comman(comando);
                                    break;
                              case 1:  // movimento relativo
                             GetCtrlVal (motor, MOTOR_YAXIS, &y); //posizione
y corrente
                            GetCtrlVal (motor, MOTOR_YCOORD, &value);
                                value=value+y;
                                sprintf(comando,"2pa%d",value);
                                send_comman(comando);
                                    break;
                        }

                        // Spostamento lungo l'asse z
                        switch (move_type)
                        {
                              case 0:  // movimento assoluto
                            GetCtrlVal (motor, MOTOR_ZCOORD, &value);
                                sprintf(comando,"3pa%d",value);
                                send_comman(comando);
                                    break;
                              case 1:  // movimento relativo
                             GetCtrlVal (motor, MOTOR_ZAXIS, &z); //posizione
z corrente
                            GetCtrlVal (motor, MOTOR_ZCOORD, &value);
                                value=value+z;
                                sprintf(comando,"3pa%d",value);
                                send_comman(comando);
                                    break;
                        }

                       // Aggiornamento delle variabili
//                     tell_position(&x,&y,&z);
                       SetCtrlVal (motor, MOTOR_XAXIS, x);
                       SetCtrlVal (motor, MOTOR_YAXIS, y);
                       SetCtrlVal (motor, MOTOR_ZAXIS, z);

                         SetCtrlAttribute (motor, MOTOR_XCOORD, ATTR_VISIBLE,
0);
                         SetCtrlAttribute (motor, MOTOR_YCOORD, ATTR_VISIBLE,
0);
                         SetCtrlAttribute (motor, MOTOR_ZCOORD, ATTR_VISIBLE,
0);
                         SetCtrlAttribute (motor, MOTOR_OK, ATTR_VISIBLE, 0);
                                          SetCtrlAttribute        (motor,
MOTOR_MOVE_TYPE,ATTR_VISIBLE, 0);
                         SetCtrlAttribute (motor, MOTOR_DECORATION_3,
                                           ATTR_VISIBLE, 0);


                      }

                      button_exit=(select==EVENT_COMMIT) &&(panel==motor)
                                  && (button==MOTOR_EXIT);
```

```
                        if (button_exit)
                        {
                           HidePanel (motor);
                        }

                  } while (!button_exit);


                  break;
      }
      return 0;
}



int CVICALLBACK test_ANIN (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int i,j,I;
      int matrix2[64][64];
      int A[32800];

      switch (event)
      {
            case EVENT_COMMIT:

                  image_ready=0;
                  acquisition_on=1;
            SetCtrlVal(panelHandle, PANEL_END_ACQUISITION,1);

                  trig_mode_tmp=2;

            A[0]=696969;    /* unrealistic number to be read if counters output
            pseudorandom number 0 (impossible) */

            for(i=0;i<64;i++)
                  for(j=0;j<64;j++)
                 matrix2[i][j]=0;

            open_read_file("LUT_real.lut");
                  for (i=1;i<32768;i++)
                fscanf(fpin,"%d",&A[i]);
            fclose(fpin);


                  if (MISSING_MEDIPIX==0)
                  {

                  for(I=0;I<64;I++)
                  {

                        for(i=0;i<64;i++)
                              for(j=0;j<64;j++)
                                    {
                                    config_matrix[i][j]=(config_matrix[i][j]   &
0x1d);
```

```
                                    if(i==I)
config_matrix[i][j]=config_matrix[i][j]|2;
                                    }
                          load_mask();
                start_acq(RADIOGRAPHY);     /*start the acquisition */
                if(MISSING_MEDIPIX==0)download_data();
                  for(j=0;j<64;j++)
                        matrix2[I][j]=A[data_matrix[I][j]];

                  }

                  }
            for(i=0;i<64;i++)
                  for(j=0;j<64;j++)
                 data_matrix[i][j]=matrix2[i][j];

                  image_ready=1;
                  acquisition_on=0;

                  break;
      }

      return 0;
}

int CVICALLBACK weighing (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{

        char fil2[300],fil20[200];
        int count1[64][64],count2[64][64];
        int i,j,k,kk,tot1=0,tot2=0,erro;
        float med=0.,sig=0.,dist=0.,r_tot1=0.,r_count1[64][64],r_count2[64][64];


      switch (event)
      {
            case EVENT_COMMIT:

        for(i=0;i<64;i++)
         for(j=0;j<64;j++)
         {
                    count1[i][j]=data_matrix[i][j];
                    count2[i][j]=1;
            }

                  FileSelectPopup (Directory_Name, "*.dat", "*.dat", "Load File
for Calibration",
                                                      VAL_LOAD_BUTTON,  0,  0,
1, 0, fil2);

                     erro=FileToArray (fil2, count2, VAL_INTEGER, 4096, 64,
                                       VAL_GROUPS_TOGETHER,  VAL_GROUPS_AS_ROWS,
VAL_ASCII);
//                            DeleteGraphPlot    (showImage,    SHOW_IMAGE,    -1,
VAL_IMMEDIATE_DRAW);
//                   set_image_values();
```

```
//                  plot_image(data_matrix);

               if(erro !=0) break;

   for(i=0;i<64;i++)
      for(j=0;j<64;j++)
        {
            r_count2[i][j]=0.;
            r_count1[i][j]=0.;
            if(count2[i][j]==696969)count2[i][j]=0;
            if(count1[i][j]==696969)count1[i][j]=0;
                  tot1= tot1+count1[i][j];
                  tot2= tot2+count2[i][j];
               }

   for(i=0;i<64;i++)
      for(j=0;j<64;j++)
        {
                  r_count2[i][j]=(float)(count2[i][j])/(float)tot2 ;
                  if(r_count2[i][j]==0.) r_count1[i][j]=0.;
                  else r_count1[i][j]=(float)(count1[i][j])/r_count2[i][j] ;
               }

   for(i=0;i<64;i++)
      for(j=0;j<64;j++)
        {
                  r_tot1=r_tot1+r_count1[i][j];
               }

   for(i=0;i<64;i++)
      for(j=0;j<64;j++)
        {
                  count1[i][j]=(int)((r_count1[i][j]/(r_tot1))*tot1);
               }

               if(count1[i][j] > 0x0fffffff)
                              {
                              count1[i][j]=0;
                              r_count1[i][j]=0.;
                              }


        for(i=0;i<64;i++)
         for(j=0;j<64;j++)
                     data_matrix[i][j]=count1[i][j];



             plot_image(data_matrix);

                  break;
      }
      return 0;
}


int CVICALLBACK local_counts (int panel, int control, int event,
```

```
            void *callbackData, int eventData1, int eventData2)
{
            double y2;
            double x2;
            double y1;
            double x1;
            int i,j,i1,i2,j1,j2,tot=0,buff;
            float media=0;

      switch (event)
      {
            case EVENT_COMMIT:

                  GetGraphCursor (showImage, SHOW_IMAGE, 1, &y1, &x1);
            GetGraphCursor (showImage, SHOW_IMAGE, 2, &y2, &x2);

                  i1=(int) x1;
                  j1=(int) y1;
                  i2=(int) x2;
                  j2=(int) y2;

                  if(x1-(float)i1 > 0.5) i1=i1+1;
                  if(y1-(float)j1 > 0.5) j1=j1+1;
                  if(x2-(float)i2 > 0.5) i2=i2+1;
                  if(y2-(float)j2 > 0.5) j2=j2+1;

                  if (i1 > i2)
                        {
                        buff=i2;
                        i2= i1;
                        i1=buff;
                        }
                  if (j1 > j2)
                        {
                        buff=j2;
                        j2= j1;
                        j1=buff;
                        }

   tot=0;
   for(i=i1;i<i2+1;i++)
      for(j=j1;j<j2+1;j++)
               if(data_matrix[i][j] != 696969) tot=tot+data_matrix[i][j];

                  media=(float)tot/(float)((j2-j1+1)*(i2-i1+1));
            SetCtrlVal (showImage, SHOW_LOCAL_COUNTS_2, tot);
                  SetCtrlVal (showImage, SHOW_LOCAL_MEDIA, media);

                  break;

      }
      return 0;
}


int CVICALLBACK subtract (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
```

```
{

        char fil2[300],fil20[200];
        int count1[64][64],count2[64][64];
        int i,j,k,kk,tot1=0,tot2=0,erro;
        float med=0.,sig=0.,dist=0.,r_tot1=0.,r_count1[64][64],r_count2[64][64];


      switch (event)
      {
            case EVENT_COMMIT:

        for(i=0;i<64;i++)
         for(j=0;j<64;j++)
         {
                    count1[i][j]=data_matrix[i][j];
                    count2[i][j]=0;
            }

                  FileSelectPopup (Directory_Name, "*.dat", "*.dat", "Load file
to be subtracted",
                                                          VAL_LOAD_BUTTON,  0,  0,
1, 0, fil2);

                     erro=FileToArray (fil2, count2, VAL_INTEGER, 4096, 64,
                                       VAL_GROUPS_TOGETHER,  VAL_GROUPS_AS_ROWS,
VAL_ASCII);
//                             DeleteGraphPlot    (showImage,    SHOW_IMAGE,    -1,
VAL_IMMEDIATE_DRAW);
//                    set_image_values();
//                    plot_image(data_matrix);

               if(erro !=0) break;



        for(i=0;i<64;i++)
         for(j=0;j<64;j++)
                    {
                     data_matrix[i][j]=count1[i][j]-count2[i][j];
                        if(data_matrix[i][j]<0)
                              data_matrix[i][j]=0;
                    }
             plot_image(data_matrix);

                  break;
      }
      return 0;
}

/*********************************************************************/

void copiastr(char * stringa)
{
 int length,length2,i,j;
 char buffer[40]={0};
```

```
 length=StringLength(stringa);
 i=39;j=length;
 do
 {
  buffer[i]=stringa[j]; i--,j--;
 }
 while(stringa[j]!='\\');
 SetCtrlVal (showImage, SHOW_FILE_READ, &buffer[i+1]);

}



/*****************************************************************************
********* Sottoprogramma utilizzato per la chiusura di medipix         ********
********* mediante l'icona "X" in alto a destra del pannello principale *******
*****************************************************************************/
int CVICALLBACK Quit2 (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{

      switch (event) {
            case EVENT_COMMIT:
                   Close();
                  break;
      }
      return 0;
}


void Close(void)
{
  if (MISSING_HARDWARE==0)
   {
    turn_off_dac();
    turn_off_power();
    switch_on=check_power();
   }

  if (MISSING_HARDWARE==1)
   {
    switch_on=0;   //Usata per la simulazione
    mask_status=0;
    image_ready=0;
    acquisition_on=0;
   }

  QuitUserInterface(0);

}
/*****************************************************************************/


void CVICALLBACK Showimage2 (int menuBar, int menuItem, void *callbackData,
            int panel)
{
 DisplayPanel(showimage2);
```

```
}


/*****************************************************************************
******************** Callback per richiamare il vecchio basic test ************
*****************************************************************************/

void CVICALLBACK BasicAccess (int menuBar, int menuItem, void *callbackData,
            int panel)
{
      basic_test();
      SetStdioWindowVisibility (0);
}
/*****************************************************************************/




void CVICALLBACK Version (int menuBar, int menuItem, void *callbackData,
            int panel)
{
 int status, shell, ok_button;

 DisplayPanel(versione);
 do
  status=GetUserEvent(1,&shell,&ok_button);
 while(!(status==EVENT_COMMIT&&shell==versione&&ok_button==VERSION_QUIT));
 HidePanel(versione);
}
```

Main.h

```
#include "medipix.h"
#include <nivxi.h>
#include <ansi_c.h>


char FILE_ARCH[200];   // Nome completo del file di archivio
char FILE_MASK[200];   // Nome completo del file di maschere

/* external variables : register addresses*/
unsigned int *status_reg;
unsigned int *analog_reg;
unsigned int *chip_sel_reg;
unsigned int *cycles_reg;
unsigned int *ram_offset_reg;
unsigned int *tau0_reg;
unsigned int *tau1_reg;
unsigned int *tau2_reg;
unsigned int *dac0_reg;
unsigned int *dac1_reg;
unsigned int *config_reg;
unsigned int *data_reg;
unsigned int *config_ram;
unsigned int *data_fifo;
unsigned int *IBad; /* gpib */

//unsigned int ibsta, ibcnt;

/* external "file" variables */
char out_data_file[FILENAMELEN];
FILE *fpin;
FILE *fpout;
char *selected_file;
char Directory_Name[DIRNAMELEN];

/* external data values */
int data_matrix[64][64];
//char config_matrix[64][64];
int config_matrix[64][64];
int pattern_matrix[64][64];

unsigned short cold_start;
char mask_status;    // Vera se la maschera è caricata su medipix
int image_ready;     // Vero se l'immagine è pronta per essere visualizzata
int switch_on;       // Vero se medipix è acceso
int acquisition_on; // Vera se l'acquisizione è completata
int charged_values[33];


/* external variables : temporary configuration values */
unsigned int power_mode_tmp;
unsigned int tau0_tmp;
unsigned int tau1_tmp;
```

```
unsigned int tau2_tmp;
unsigned int anin_cycles_tmp;
unsigned int sleep_anin_period_tmp;
unsigned int sleep_anin_delay_tmp;
unsigned int trig_mode_tmp;
unsigned int dac_bias_tmp[10];
unsigned int chip_sel_tmp;
unsigned int cycles_tmp;
unsigned int config_mode_tmp;
unsigned int ram_offset_tmp;

/* external variables : active configuration values */
unsigned int power_mode_act;
unsigned int tau0_act;
unsigned int tau1_act;
unsigned int tau2_act;
unsigned int anin_cycles_act;
unsigned int sleep_anin_period_act;
unsigned int sleep_anin_delay_act;
unsigned int trig_mode_act;
unsigned int dac_bias_act[10];
unsigned int chip_sel_act;
unsigned int cycles_act;
unsigned int config_mode_act;
unsigned int ram_offset_act;

/***************************************/

int  basic_r_a, ram_fifo, Test_Counters, Test_Mask, E_Mask, Add_Subt_Files,
     Pixels_Cal, Vth_Cal, PSelect, analisi, versione, showimage2;
```

Medipix.h

```
/* MACRO definition of register address */
/* REGISTER ADDRESS = VMEbase <OR> REGISTER SUBADDRESS */
#define VMEbase         0xf0000000  /*VMEbase address */
                                    /*
                                    0xe0000000 for VMEboard1 during test
                                    0xf0000000 for VMEboard2
                                    0xf0000000 for VMEboard (Napoli)
                                    0x01000000 for VMEboard (CERN)
                                    0xf0000000  for 68040 Pisa (Amendolia)
                                    0x04000000   for 68020 Pisa (Morsani)
                                    */
#define STATUS    0x0               /*status_reg subaddress (R/W) */
#define ANALOG_REG      0x4             /*analog_reg subaddress (R/W)
                                          for analog bias control*/
#define CHIP_SEL  0x8               /*chip_sel_reg, select Medipix */
#define CYCLE_NUM 0xc               /*cycles_reg, number of W/R cycles */
#define RAM_START 0x10        /*ram_offset_reg, offset on RAM start address */
#define TIMER_TAU0      0x14        /*time register tau0_reg */
#define TIMER_TAU1      0x18        /*time register tau1_reg */
#define TIMER_TAU2      0x1c        /*time register tau2_reg */
#define DAC_VALUE0      0x40        /*dac0_reg (5) subaddress (R/W)
                                          for DAC values*/
#define DAC_VALUE1      0x60        /*dac1_reg(5) subaddress (R/W)
                                          for DAC values*/
#define CONFIG_REG  0x10000         /*config_reg(8) subaddress (R/W)
                                                for   direct   access   to
FPGA*/
#define DATA_REG  0x10020           /*data_reg subaddress (R/W)
                                          to load "zero" on medipix.
                                          Follow: 15 registers (R) for
                                          direct accesss to FPGA*/
#define CONFIG_RAM  0x20000         /*config_ram RAM area (2k)*/
#define DATA_FIFO 0x40000           /*data_fifo subaddress,
                                                (2k FIFO)*/


/* turn on/off bit */
#define turn_on 0xffffffff
#define turn_off 0x0

/* status_reg bits */
#define bit_config_wr   0x1   /*bit  0,enable  indirect  write  config  data  RAM-
>chip */
#define bit_direct_config_wr  0x81/*bit 7 and bit 1, enable write
                                          config   data   FPGA->chip
direct mode****/
#define bit_config_rd   0x2         /*bit 1,enable read config data chip->RAM */
#define bit_zero_wr           0x4         /*bit   2,enable   write   "zero"   to
counters */
#define bit_data_rd           0x8         /*bit  3,enable  read  counters  data
chip->FIFO */
#define bit_start       0x10  /*bit 4,internal acq-start: start gate/shutter*/
#define bit_shutter           0x20  /*bit 5,set continuous shutter on */
```

```
#define bit_ANIN        0x40  /*bit 6,generate ANIN signal */
#define bit_resetb            0x100   /*bit 8,reset MEDIPIX */
#define bit_resetVME    0x200   /*bit 9,reset VMEboard (except DACs and FIFOs)
*/
#define bit_Ret               0x400   /*bit 10,retransmit FIFOs */
#define bit_resetFIFO   0x800   /*bit 11,reset FIFOs */
#define bit_all         0xffffffff  /* all bits */

/* dac_ctrl bits */
#define bit_Vdd_en      0x1         /*bit 0, enable Vdd output*/
#define bit_Vdda_en     0x2         /*bit 1, enable Vdda output*/
#define bit_DAC0_en           0x4         /*bit 2, enable DAC0 output*/
#define bit_DAC0_res    0x8         /*bit 3, reset DAC0*/
#define bit_DAC1_en           0x10  /*bit 4, enable DAC1 output*/
#define bit_DAC1_res    0x20  /*bit 5, reset DAC1*/

/* additional MACRO definitions */
#define TYPE unsigned int
#define RADIOGRAPHY 0         // radiography mode
#define AUTORADIOGRAPHY 1     // autoradiography mode
#define SAVEDAT 0             //    0=saves    all    intermediate    files    in
autoradiography mode
#define ZERO 32767                  //   starting   number   loaded   on   counters
(32767=pseudorandom zero)
#define SAVE_WITH_CONTROL 1
#define SAVE_WITHOUT_CONTROL 0
#define START_FROM_0 0
#define START_FROM_1 1
#define TIME_UNIT 1     /* Time unit (related to VME clock selecting jumper,
                                                    Laben   default   is
"milliseconds"):
                           0:  milliseconds
                           1:  seconds   */
#define BAD_RAM 0             /*  default  =0.  Some  RAM  showed  a  problem  that
required
                                    overwriting  of  config[0][0],config[0][16],
etc
                                    If you have a defaulting RAM, you should set
this flag  =1. */
#define LSBCAL 825.14   // for DAC calibration
#define OFFS -0.044           // for DAC calibration
                                    /* dac_bias=Volt*LSBCAL, no OFFS*/
                                    /* 1147.315 if Vrefh - Vrefl = 3.57 V */
                                    /* 803.14   if Vrefh - Vrefl = 5.1 V */

                                    /* V= dac_bias * LSB + OFFS
                                    /* LSB=(4.83+0.108)/4096
                                    /* LBSCAL = 1/LSB
                                    /* Naples VMEboard test:
                                    /* 829.5  if Vrefh - Vrefl = 5 V */
                                    /* OFFS = - 0.108 */

                                    /*  nuovo  setting  feb  1999  =  internal  bias
*/
                                    /* Vrefh - Vrefl = 4.96 - 0 V (da bus VME)
*/
                                    /* V= dac_bias * LSB + OFFS
```

```
                                    /* LSB=(4.92+0.044)/4096 = 0.0012
                                    /* LBSCAL = 1/LSB  = 825.14
                                    /* OFFS = - 0.044 */

/* additional external hardware MACRO definitions */
#define MOTORFLAG 1           // =1, Microcontrole Scanning system present; =0
absent
#define MOTOROPT 1            //   This   line   should   be   present   ONLY   if
MOTORFLAG=1:
                                    //  if  the  Microcontrole  mm2000  board  is
present,
                                    // consequently, Mm2k_32.lib and MM2k_32dll
should be
                                    //  present  in  the  user  directory,  and
Mm2k_32.lib included
                                    // in project build.
#define GPIBFLAG 1            // =0, no GPIB Board
                                    // =1, GPIB-VME Board present
                                    // =2, GPIB-PCI Board present
#define PULFLAG 1       // =0, no GPIB controlled pulse generator present
                                    // =1, HP8130A pulse generator present
                                    // =2, HP8110A pulse generator present
#define Gpib_ad 0x1000           // GPIB-VME Interface Board VMEbus-address if
present
#define GPIB_TIMEOUT 0xfffff   // VME-GPIB timeout (esp488.c), default 0xfffff
#define GPIBADD 10               // GPIB address of pulser generator if present
#define FACT 10.0                     // attenuation factor on Pulser signal
output

/* VXI+PC hardware  */
#define VMEadd16 0x1          /* for MapVXIAddress                  */
#define VMEadd32 0x3          /*    =     0x81  A16, Intel order
                                          =     0x83  A32, Intel order
                                          =     0x1         A16,  Motorola
order       DEFAULT
                                          =     0x3         A32,  Motorola
order       DEFAULT
                                          =   0x43    A32, Motorola, Owner
*/
#define init_VME  InitVXIlibrary()
#define close_VME       CloseVXIlibrary()
#define MAKE_ADDRESS MapVXIAddress
#define MAKE_ADDRESS_SIZE MapVXIAddressSize
#define PC_VXI 1
#define PATHF "files/"
#define PATHF2 "files\\"

/* flags for missing parts */
#define MISSING_HARDWARE 0     /* = 0, if all hardware is present;
                                          =  1,  if  all  hardware  is  NOT
present:
                               VME Board, Motherboard and Medipix are simulated
*/
#define MISSING_MEDIPIX 0      /* = 0, if Motherboard and Medipix are present
                                            = 1, if Motherboard and Medipix are
NOT present
                                       but VMEboard is present */
```

```
// GRAPHICS AND FILES MACRO
#define FILENAMELEN 256
#define DIRNAMELEN 512

// HARDWARE PROTOTYPES

/****** GPIB & pulser *****/
void Pulser(float AMPL, float period, float length);
void Pulser_Disable(void);
void Pulser_Enable(void);

/****** GPIB-VME interface *****/
int Ibonl(int);
int Ibsic(void);
int Ibsre(int);
int Ibcmd(char*,int);
int Ibwrt(char*,int);

/****** microprecision 3D scansion (motor) *****/
void motore(int *, int *, int *);
void mm2000(void);
void mm_send(char *);
void tell_position(int *,int *, int *);
void send_comman(char *);
void motor_stop(void);

/* function_VME : basic VME access functions */
void make_address(void);
unsigned int *make_address_DUMMY(unsigned int, unsigned int,
      int, unsigned int *, short int *);
int make_address_DUMMY_size(unsigned int);
int modify_register(TYPE *,unsigned int,unsigned int,char *);
void read_data(TYPE *,unsigned int *,int,char *);
void write_data(TYPE *,unsigned int *,int,char *);
int setup_tau(void);
int setup_chip(void);
int setup_RAM_start(void);
int setup_cycles(void);
int setup_counters(unsigned int);
int reset_and_setup(void);
int init_board(void);
int check_power(void);
void reset_counters(unsigned int);
int turn_on_dac(void);
int turn_off_dac(void);
int turn_off_power(void);
int turn_on_power(void);
void load_config_bits(void);
void start_acq(int);
void download_data(void);
void LUT_conversion(void);
void sleep_soft(int);

/* main display functions */
void main_menu(void);
void sys_setting(void),DAQ_TEST(void),show(void),
```

```
        load_file_arc(char*),      // loads from file temporary values
        handle_files(void),
        mod_settings(void),
        check_sys(void),
        bad_option(int i,int j),
        set_DAC(void),
        check_mod_mask(void),
        handle_daq(void),
        load_mask(void),
        get_file(void),
        Save_File(void),
        save_file(char*),          // saves filename
        get_pattern(void),
        save_pattern(void),
        get_mask(void),
        Save_Mask(void),           // saves mask file - only in user_test
        save_mask(char*),           // saves mask file
            modify_mask_values(char*),
        modify_mask(char*, int, int, int,
                    int, int),      // modifies mask
            Input_range(char *mask_name, int *start_row,
                        int *end_row, int *start_col, int *end_col),
                                        // sets the display range (mask)
        show_current_mask(void),
        modify_pattern(void),
        do_test(void),
        display_values(void),
        display_charged_values(void),
        modify_dac_values(void),
        modify_opmode_values(void),
        show_active_values(void);
void set_default(char mask_file_default[512]);
                                                 // sets default values from
files:
                                                // default.arc and default.msk
int open_read_file(char *);
int open_write_file(char *,int);
int read_file_arc(char *,int);
int read_file_mask(char *,int m1[64][64],int m2[64][64],int m3[64][64]);
int read_file_pattern(char *,int mat[64][64]);
void display_matrix_values(int m[64][64], char *);
void change_mask_values(int m[64][64], char *, int, int, int, int, int);
void   set_params(void);           //   sets   parameters   loaded   from   file
// maschera selezionata
void cercaMinimo(int matrix[64][64], int *);
void cercaMassimo(int matrix[64][64], int *);
void ProfileArray(int matrix[64][64], int, int, int, double prof_array[64]);
void show_color_bar(void);
void read_dac(void);
void  read_act(void);        //  Reads  setting  presently  set  on  the  VME  board
(active)
void Print_error(int err_code);

/* basic test calls */
void thres_adjus(void);
void filtro(void);
```

```
/* user_test calls */
void basic_test(void);

// FUNCTIONAL MACROS
#define for_any_pixel for (i = 0; i < 64; i++) for (j = 0; j < 64; j ++)
```

Motore.c

```
#include "function.h"

#ifdef MOTOROPT
int _stdcall MMReadRespNTO(char *lpResponse, int sAddress);
int _stdcall MMSendCmdNTO(char *lpCommand, int sAddress);
int _stdcall MMReadResp(char *lpResponse, int sAddress);
int _stdcall MMSendCmd(char *lpCommand, int sAddress);
#else
int MMReadRespNTO(char *lpResponse, int sAddress);
int MMSendCmdNTO(char *lpCommand, int sAddress);
int MMReadResp(char *lpResponse, int sAddress);
int MMSendCmd(char *lpCommand, int sAddress);

int MMReadRespNTO(char *lpResponse, int sAddress) {return 0;}
int MMSendCmdNTO(char *lpCommand, int sAddress){return 0;}
int MMReadResp(char *lpResponse, int sAddress){return 0;}
int MMSendCmd(char *lpCommand, int sAddress){return 0;}
#endif

char cmd[80];


void motore(int *x, int *y, int *z)
{
  int i = 0, j = 0;

  char *cmd_imm[2] = {
    "1mf;2mf;3mf",
    "end"
  };

  while (strcmp(cmd_imm[i], "end") != 0) {
      strcpy(cmd, cmd_imm[i]);
      mm_send(cmd);
      ++i;
  }

  tell_position(x, y, z);

}       // motore.c


void mm2000(void)
{
  int i = 0, j = 0;
  char *cmd_start[5] = {
    "1mo;2mo;3mo",
    "1fm00;2fm00;3fm00",
    "1ml-;2ml-;3ml-",
    "1ws;2ws;3ws;1dh;2dh;3dh",
    "end"
  };
```

```
  char *cmd_init[8] = {
    "1SL-10;1SL+50000",
    "2SL-10;2SL+50000",
    "3SL-10;3SL+45000",
    "1fm02;2fm02;3fm02",
    "1pa5000;2pa30000;3pa20000",
    "1ws;2ws;3ws",
    "1fo00;2fo00;3fo00",
    "end"
  };

  while (strcmp(cmd_start[i], "end") != 0) {
      strcpy(cmd, cmd_start[i]);
      mm_send(cmd);
      ++i;
  }

  while (strcmp(cmd_init[j], "end") != 0) {
      strcpy(cmd, cmd_init[j]);
      mm_send(cmd);
      ++j;
  }

//tell_position(x, y, z);
}


void mm_send(char *cmd0)
{
  char cmd2[80];

  strcat(strcpy(cmd2, cmd0), "\r");
  MMSendCmd(cmd2, 0);
}

void motor_stop(void)
{
  char *cmd_st = "#";

  send_comman(cmd_st);
}

void send_comman(char *comma)
{
  int i = 0;
  char *cmd_send[3] = {
    "                                    ",
    "ws",
    "end"
  };

  strcpy(cmd_send[0], comma);
  while (strcmp(cmd_send[i], "end") != 0) {
      strcpy(cmd, cmd_send[i]);
      mm_send(cmd);
      ++i;
```

```
  }
}


void tell_position(int *x, int *y, int *z)
{
  int j;
  char resp1[80], resp2[80], resp3[80];
  char *cmd_tp1[3] = {
    "1fo00",
    "1tp",
    "end"
  };

  char *cmd_tp2[3] = {
    "2fo00",
    "2tp",
    "end"
  };

  char *cmd_tp3[3] = {
    "3fo00",
    "3tp",
    "end"
  };

  j = 0;
  while (strcmp(cmd_tp1[j], "end") != 0) {
      strcpy(cmd, cmd_tp1[j]);
      mm_send(cmd);
      ++j;
  }

  while ((j = MMReadResp(resp1, 0)) != 1);

  j = 0;
  while (strcmp(cmd_tp2[j], "end") != 0) {
      strcpy(cmd, cmd_tp2[j]);
      mm_send(cmd);
      ++j;
  }
  while ((j = MMReadResp(resp2, 0)) != 1);
  j = 0;
  while (strcmp(cmd_tp3[j], "end") != 0) {
      strcpy(cmd, cmd_tp3[j]);
      mm_send(cmd);
      ++j;
  }
  while ((j = MMReadResp(resp3, 0)) != 1);

/*printf("\nPosition axis 1 =%s",resp1);
printf("\nPosition axis 2 =%s",resp2);
printf("\nPosition axis 3 =%s",resp3);  */
  *x = strtol (resp1, NULL, 10);
  *y = strtol (resp2, NULL, 10);
  *z = strtol (resp3, NULL, 10);
}
```

Pulser.c

```
/* PULSER.C - Functions to remotely control Impulsers */

#include <userint.h>
#include "function.h"

int gwrite (char write_buffer[]);
int Pul_HP8110(float AMPL, float period, float length);
int Pul_HP8130(float AMPL, float period, float length);

int device;
char NIT = '@', NIL = ' ', pul_list = 040 + GPIBADD;
float fact = FACT;

 /*  example of GPIB addressing:
   NI-VME Board has access=non-privileged, VMEaddressA16=1000.
   NI-VME Board has GPIBaddress= 0:
      talk address= 0100 (octal)   or '@'
          listen address=  040 (octal)    or ' '
   if the HP 8130A Pulser has GPIBaddress 10:
      talk address=     112 (octal)  or 'J'
          listen address=     52 (octal)  or '*'
   */
//  NIT='@';
//  NIL=' ';
//  pul_talk=0100+GPIBADD;
//  pul_list=040+GPIBADD;

/*****************************************************************************
********************** GPIB & Pulse Generator         **********************
*****************************************************************************/
void Pulser_Disable(void)
{
  char cmd[100];
  if (PULFLAG == 0);                  // no GPIB controlled pulse generator

  if (PULFLAG == 1) {                                    //if   you   are   using
HP8130A
      if (GPIBFLAG == 1) {                        //VME, Napoli
        Ibonl(1);                           // initialize the interface
        Ibsic();                            // send IFC to clear the GPIB
        Ibsre(1);  // set REN to prepare for placing the instrument in remote
mode
        sprintf(cmd, "\021%c\024%c", pul_list, NIT);
        Ibcmd(cmd, 4);
        gwrite(":OUTP:PULS:STAT OFF");
        gwrite(":OUTP:TRIG:STAT OFF");
      Delay(0.010);
        Ibonl(0);
      }
      if (GPIBFLAG == 2) {                          //PCI, CERN
        device = ibdev (0, GPIBADD, NO_SAD, T10s, 1, 0);
        gwrite(":OUTP:PULS:STAT OFF");
```

```
        gwrite(":OUTP:TRIG:STAT OFF");
        Delay(0.010);
    }
  }


  if (PULFLAG == 2);                              //if you are using  HP8110A
}


void Pulser_Enable(void)
{
  char cmd[100];

  if (PULFLAG == 0);                  // no GPIB controlled pulse generator

  if (PULFLAG == 1) {                           //if you are using HP8130A
      if (GPIBFLAG == 1) {                      //VME, Napoli
        Ibonl(1);                            // initialize the interface
        Ibsic();                                      // send IFC to clear the
GPIB
        Ibsre(1); // set REN to prepare for placing the instrument in remote
mode
        sprintf(cmd, "\021%c\024%c", pul_list, NIT);
        Ibcmd(cmd, 4);
        gwrite(":OUTP:PULS:STAT ON");
        gwrite(":OUTP:TRIG:STAT ON");
        Delay(0.010);
        Ibonl(0);
      }
      if (GPIBFLAG == 2) {                        //PCI, CERN
        device = ibdev (0, GPIBADD, NO_SAD, T10s, 1, 0);
        gwrite(":OUTP:PULS:STAT ON");
        gwrite(":OUTP:TRIG:STAT ON");
        Delay(0.010);
      }
  }

  if (PULFLAG == 2);                              //if you are using  HP8110A

}


void Pulser(float AMPL,float peri,float widt)
{

  if (PULFLAG == 0);                  // no GPIB controlled pulse generator

  if (PULFLAG == 1)                                     //if you are using GPIB +
HP8130A
      Pul_HP8130(AMPL, peri, widt);

  if (PULFLAG == 2)                                     //if you are using GPIB +
HP8110A
      Pul_HP8110(AMPL, peri, widt);
}                                        // end of Pulser function
```

```
/*************************************************************************
********************** Pulse Generator HP 8130A ************************
*************************************************************************/
int Pul_HP8130(float AMPL, float peri, float widt)
{
  char cmd[100];
  int amp;

  if (GPIBFLAG == 1) {
      Ibonl(1);                                    //     initialize     the
interface
      Ibsic();                                     // send IFC to clear the
GPIB
      Ibsre(1); // set REN to prepare for placing the instrument in remote mode
      sprintf(cmd, "\021%c\024%c", pul_list, NIT);
      Ibcmd(cmd, 4);
  }

  if (GPIBFLAG == 2)
      device = ibdev (0, GPIBADD, NO_SAD, T10s, 1, 0);

  gwrite(":INP:TRIG:STAT ON");
  gwrite(":INP:TRIG:MODE TRIG");
  gwrite(":INP:TRIG:SLOP POS");
  gwrite(":INP:TRIG:THR 1.0V");
  gwrite(":PULS:EDGE:LEAD 1ns");
  gwrite(":PULS:EDGE:TRA 1ns");
  gwrite(":OUTP:PULS:STAT ON");
  sprintf(cmd, ":PULS:TIM:PER %fms", peri);
  gwrite(cmd);
  sprintf(cmd, ":PULS:TIM:WIDT %fns", widt);
  gwrite(cmd);
  gwrite(":PULS:LEV:LOW 0.0V");            // Bettina MIKULEC, 18.1.2000
  amp = (int)(fact*AMPL); /* fact is the attenuation factor AFTER the HP*/
  sprintf(cmd, ":PULS:LEV:HIGH %dmV", amp);  // Bettina MIKULEC, 18.1.2000
//sprintf(cmd,":PULS:LEV:AMPL %dmV",amp);
  gwrite(cmd);

  Delay(0.002);

  if (GPIBFLAG == 1)                        //if you are using GPIB-VME
    Ibonl(0);                              // disable the interface

  if (GPIBFLAG == 2);                      //if you are using GPIB-PCI

  return(0);
}


int gwrite (char write_buffer[100])
{

  if (GPIBFLAG == 1)
    Ibwrt(write_buffer, strlen(write_buffer));
  if (GPIBFLAG == 2)
    ibwrt(device, write_buffer, strlen(write_buffer));
```

```
  return (0);
}


/*************************************************************************
********************** Pulse Generator HP 8110A ************************
*************************************************************************/
int Pul_HP8110(float AMPL, float peri, float widt)
{
  char cmd[100];
  float vamp;

  if (GPIBFLAG == 1) {
      Ibonl(1);         // initialize the interface
      Ibsic();          // send IFC to clear the GPIB
      Ibsre(1); // set REN to prepare for placing the instrument in remote mode
      sprintf(cmd, "\021%c\024%c", pul_list, NIT);
      Ibcmd(cmd, 4);
  }
  if (GPIBFLAG == 2)
    device = ibdev(0, GPIBADD, NO_SAD, T10s, 1, 0);

  gwrite(":ARM:SOUR EXT");        // Set trigger source
  gwrite(":ARM:LEV 2.5V");        // Set EXT threshold level
  gwrite(":OUTP1 ON");            // Turn out1 on
  gwrite(":SOUR:VOLT:OFFS 0.0V"); // Set Offset

//   gwrite (":PULS:WIDT1 200ns");
//   gwrite (":PULS:PER 133us");
  sprintf(cmd, ":PULS:WIDT1 %fns", widt);
  gwrite(cmd);
  sprintf(cmd, ":PULS:PER %fms", peri);
  gwrite(cmd);
  gwrite(":PULS:TRAN1 2NS");
  gwrite(":PULS:TRAN1:TRA 2NS");
  gwrite(":SOUR:VOLT1 0.10V");

  vamp = fact * AMPL / 1000.;
  sprintf(cmd, "SOURCE:VOLT1 +%fV", vamp);
//   sprintf(vstr,"%5.3f",vampl);
//   strncat(sendstr,vstr,5);
//   strncat(sendstr,"V",1);
  gwrite(cmd);

  return(0);
}
```

User_test.c

```
#include <userint.h>
#include "Interface.h"

/* basic test functions */
void bus_access_us(void);
void FIFO_RAM_us(void);
void sleep_cal_us(void );
void test_counters_us(void );
void mask_load_test_us(void );
void cern7_us(void );
void no_noise_thres_us(void );
void subt_us(void);
void eff_cal_us(void);
void off_noise_us(void);
void wait_keypress(void);

/* file basic_test.c */
/*****************************************************************************/
/*          ____________________________________________________             */
/*         |                                                    |            */
/*         |                 MEDISOFT SOFTWARE                  |            */
/*         |                                                    |            */
/*         |                 by:                                |            */
/*         |                     S.R. Amendolia                 |            */
/*         |                     M. Conti                       |            */
/*         |                     P. Delogu                      |            */
/*         |                     S. Stumbo                      |            */
/*         |                                                    |            */
/*         |                              May 1997              |            */
/*         |____________________________________________________|            */
/*                                                                           */
/*****************************************************************************/

# include "function.h"
void pulser(float,float,float);

void basic_test(void)
{
  int choice;
  char buff[80], buffer[20];
  int c, i, c2;

  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

/*********** BASIC TEST MENU ***************/

ONE:
  printf("\n\n\n\n\n\n");
  printf("\n\n\n\n\nBASIC TEST Menu");

  printf(" .... possible choices are:\n\n\n");
```

```
  printf("0)  basic register access;\n");
  printf("1)  RAM or FIFO test;\n");
  printf("2)  sleep_soft time calibration ;\n");
  printf("3)  counter test;\n");
  printf("4)  mask_loading_test;\n");
  printf("5)  min thres & mask (output: .msk);\n");
  printf("6)  thres_cal with pulse generator (output: .dat);\n");
  printf("7)  background subtraction;\n");
  printf("8)  efficiency pixel calibration;\n");
  printf("9)  turn off noisy pixels;\n");
  printf("10) empty;\n");
  printf("11)    for the following better go to 'dslab';\n");
  printf("12)    - thres_cal analysis (output: .pix,.sgl);\n");
  printf("13)    - thresh_mask calculation (output: final .msk);\n");
  printf("14) empty;\n");
  printf("15) Return to Main Menu;\n\n\n");

  printf("Now you must make a choice: ");

  choice = atoi(gets(buff));

  if (choice == 0)
    bus_access_us();
   else if (choice == 1)
    FIFO_RAM_us();
   else if (choice == 2)
    sleep_cal_us();
   else if (choice == 3)
    test_counters_us();
   else if (choice == 4)
    mask_load_test_us();
   else if (choice == 5)
    no_noise_thres_us();
   else if (choice == 6)
    cern7_us();
   else if (choice == 7)
    subt_us();
   else if (choice == 8)
    eff_cal_us();
   else if (choice == 9)
    off_noise_us();
   else if (choice >= 10 && choice <=14);
   else if (choice == 15)
    goto THREE;
   else
      bad_option(16, START_FROM_1);

  goto ONE;

THREE: {     }

}


/****************** FUNCTION bus_access ***********************/
void bus_access_us()
{
```

```
  int choice;
  int choice0, choice1, choice2, choice3, choice4;
  unsigned int *choice00, *choice11, *choice22, choice33, choice44;
  unsigned int letto;
  char buff[80], buffer[20];
  int c, i, c2;

  printf("\n\nDirect access to a VMEboard register:");
  printf("\nModify only a bit or write/read ?(1 o 2)");

  choice1 = atoi(gets(buff));

  if (choice1 == 1) {
      printf("\nRegister address (HEX): ");
      scanf("%x", &choice22);
      c = getchar();
      printf("\nturn on=1,turn off=0: ");
      choice3 = atoi(gets(buff));

      if (choice3 == 1)
        printf("\nselect bit (mask 1 and 0): ");
       else if (choice3 == 0)
        printf("\nselect bit: ");
    choice4 = atoi(gets(buff));
    choice44 = (unsigned int)choice4;
    modify_register(choice22, choice3 * turn_on, choice44, "W");


      letto = modify_register(choice22, choice3, choice4, "R");
      printf("read value is :%x", letto);
      wait_keypress();

   }
   else if (choice1 == 2) {
      printf("\nwrite or read (1,2)?: ");
      choice0 = atoi(gets(buff));

    printf("\nRegister address (HEX): ");
    scanf("%x", &choice22);
    c = getchar();
    if (choice0 == 1) {
      printf("\nValue to be written: ");
        scanf("%x", &choice33);
        c = getchar();
      write_data(choice22, &choice33, 1, "s");
        read_data(choice22, &letto, 1, "s");
        printf("Value written:%x", letto);
       }
       else if (choice0 == 2) {
        read_data(choice22, &letto, 1, "s");
        printf("Read value :%x", letto);
    }
    wait_keypress();
  }
}
/************* END ********************/
```

```
/************** FUNCTION CERN1 ****************/

void FIFO_RAM_us()
{
// read_write_array, writes an array in a RAM or FIFO position, reads, compares
  int choice;
  char buff[80];
  int i, n, c;
  unsigned int offs, dati_rd[4096], dati_wr[4096];

  for (i = 0; i < 4096; i++) {
      dati_wr[i] = i;
    dati_rd[i] = 0;
  }

  printf("\nRAM or FIFO test? (0=RAM, 1=FIFO)");
  choice = atoi(gets(buff));
  if (choice == 0) {
      printf("\nOffset on RAM start address (HEX): ");
      scanf("%x", &offs);
      c = getchar();
      printf("\nNumber of data to be written: ");
      scanf("%d", &n);
      c = getchar();
      if (n > 2048) {
        printf("\nerror: RAM size is 2048 words!");
        wait_keypress();
        return;
      }
      write_data(config_ram + offs, dati_wr, n, "m");
      read_data(config_ram + offs, dati_rd, n, "m");

      for(i = 0; i < n; i++)
        if (dati_rd[i] != dati_wr[i]) {
            printf("\n RAM test failure: %d datum is wrong", i);
            printf("\n End of test, failure");
            wait_keypress;
            return;
        }
   }
   else {
      printf("\nNumber of data to be written: ");
      scanf("%d", &n);
      c = getchar();
      if (n >= 4096) {
        printf("\nerror: FIFO depth is 4096 !");
        wait_keypress;
        return;
      }
      write_data(data_fifo, dati_wr, n, "s");
      read_data(data_fifo, dati_rd, n, "s");
/*          for (i=0; i<n; i++)
            {
                  VXIpoke(data_fifo, 4, dati_wr[i]);
            }
```

```
            for (i=0; i<n; i++)
            {
                  VXIpeek (data_fifo, 4, &dati_rd[i]);
            } */

      for(i = 0; i < n; i++)
        if (dati_rd[i] != dati_wr[i]) {
            printf("\n FIFO test failure: %d datum is wrong", i);
            printf("\nEnd of FIFO test, failure");
            wait_keypress();
            return;
        }
  }
  printf("\nEnd of test, successful ");
  wait_keypress();
}
/************** END ********************/


/************** FUNCTION CERN2 ***************/
void sleep_cal_us()
{
/* calibration of the sleep time for DAC and ANIN,
using the VMEboard internal clock (timer tau0):
the usual setup for tau0 is 1 MHz/1024, therefore the value tau0_ms
written on tau0_reg is in ms.
*/
  int cpu_tempo, i, c;
  unsigned int tau0_ms = 10000, tau1_ms = 10000, tau2_ms = 10000, tau0_left = 0;

  printf("\nSleep cycles to be written: ");
  scanf("%d", &cpu_tempo);
  c = getchar();

  modify_register(status_reg, turn_on, bit_resetVME, "W");
  write_data(tau0_reg, &tau0_ms, 1, "s");
  write_data(tau1_reg, &tau1_ms, 1, "s");
  write_data(tau2_reg, &tau2_ms, 1, "s");

  modify_register(status_reg, turn_on, bit_start, "W");

  sleep_soft(cpu_tempo);

  read_data(tau0_reg, &tau0_left, 1, "s");
  i = (int)tau0_ms - (int)tau0_left;
  printf("\n%d sleep cycles correspond to %d ms", cpu_tempo, i);

  wait_keypress();
}
/************** END ********************/


void mask_load_test_us () // ****************** mask_loading_test
{
  char buff[20];
  int choice, c;
  int i, ii, a, j, k, kk, x = 0, err = 0, W_mode;
```

```
  int c_m_o[64][64], offs = 1024;
  unsigned int config_input[1024], config_output[1024], cycles_dummy, ram_dummy;
  char SF[20];

  get_mask();
  load_mask();
// ********* mask is read

  ram_dummy = ram_offset_tmp;
  ram_offset_tmp = offs;
  reset_and_setup();
  ram_offset_tmp = 0;

  modify_register(status_reg, turn_on, bit_config_rd, "W");

//  start polling to verify that configuration bits have been read from medipix
  for(ii = 0; ; ii++)
      if ((a = modify_register(status_reg, 0, bit_config_rd, "R")) == 0)
        break;

  read_data(config_ram + offs, config_output, 1024, "m");

  for(k = 0; k < 4; k++)
      for(i = 0; i < 64; i++)
        for(kk = 0; kk < 4; kk++) {
            c_m_o[i][kk * 4 + k] = config_output[x] & 0x1f;
            c_m_o[i][(4 + kk) * 4 + k] = (config_output[x] >> 8) & 0x1f;
            c_m_o[i][(8 + kk) * 4 + k] = (config_output[x] >> 16) & 0x1f;
            c_m_o[i][(12 + kk) * 4 + k] = (config_output[x] >> 24) & 0x1f;
            x++;
        }

  for_any_pixel
      if (c_m_o[i][j] != config_matrix[i][j]) {
        err++;
        if (BAD_RAM == 1 && i == 0 && ((j & 15) == 0))
          err--;
    }

  if (err == 0) {
    printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
    printf("Test ok !");
    printf("\nBoth arrays are saved in file 'mask.error'");

    open_write_file("mask.error", SAVE_WITHOUT_CONTROL);

      fprintf(fpout, "/******** mask matrix (written) *******/\n");

      for(i = 0; i < 64; i++) {
        for(j = 0; j < 64; j++)
            fprintf(fpout, "%x ", config_matrix[i][j]);
        fprintf(fpout,"\n");
      }

      fprintf(fpout, "\n/******** mask matrix (read) *******/\n");

      for(i = 0; i < 64; i++) {
```

```
        for(j = 0; j < 64; j++)
            fprintf(fpout, "%x ", c_m_o[i][j]);
        fprintf(fpout,"\n");
      }

    fclose(fpout);
  }

  if (err != 0) {
    printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
    printf("Read Data array is different from Loaded Data array");
    printf("\nBoth arrays are saved in file 'mask.error'");

    open_write_file("mask.error", SAVE_WITHOUT_CONTROL);

    fprintf(fpout, "/******** mask matrix (written) *******/\n");

    for(i = 0; i < 64; i++) {
        for(j = 0; j < 64; j++)
            fprintf(fpout, "%x ", config_matrix[i][j]);
      fprintf(fpout,"\n");
      }

      fprintf(fpout, "\n/******** mask matrix (read) *******/\n");

      for(i = 0; i < 64; i++) {
        for(j = 0; j < 64; j++)
            fprintf(fpout, "%x ", c_m_o[i][j]);
        fprintf(fpout,"\n");
      }

    fclose(fpout);
  }

  printf("\nEnd of Mask Loading Test, press return to continue");
  wait_keypress();
}
/************** END ********************/


void no_noise_thres_us()
{
/*** funzione per il calcolo automatico di Vth minima (maurizio) ******/
  int i, j, k, ii, iii, I, jj, c, s;
  int Vth, MINC, tempo, mask_only;
  float PERC;
  int Vb_val, Vc_val, Vdl_val;
  char buff[20];
  int A[32768], C1 = 0, C2 = 63, R1 = 0, R2 = 63;
  int matrix2[64][64], enable_mask[64][64], CHIP;

  CHIP = chip_sel_tmp;

  A[0]=696969;    /* unrealistic number to be read if counters output
                     pseudorandom number 0 (impossible) */

  open_read_file("LUT_real.lut");
```

```
  for (i = 1; i < 32768; i++)
    fscanf(fpin, "%d", &A[i]);
  fclose(fpin);

  printf("\n Insert Vbias number :");
  Vb_val = (int)(LSBCAL * atof(gets(buff)));
  printf("\n Insert Vcomp number :");
  Vc_val = (int)(LSBCAL * atof(gets(buff)));
  printf("\n Insert Vdl:");
  Vdl_val = (int)(LSBCAL * atof(gets(buff)));
  printf("\n Insert acquisition time (ms):");
  tempo = atoi(gets(buff));
  printf("\nInsert the fraction of bad pixels admitted:");
  PERC = atof(gets(buff));
  printf("\nInsert the minimum bad count admitted:");
  MINC = atoi(gets(buff));
  printf("\nmask only ? (1=yes):");
  mask_only = atoi(gets(buff));

  trig_mode_tmp = 0;
  tau1_tmp = tempo;
  dac_bias_tmp[0 + CHIP * 5] = Vb_val;
  dac_bias_tmp[1 + CHIP * 5] = Vc_val;
  dac_bias_tmp[4 + CHIP * 5] = Vdl_val;
  for_any_pixel
      config_matrix[i][j] = config_matrix[i][j] & 0xfe;

/******* chiede di definire la zona per il calcolo di Vth ******/

  printf("\n Vth calculation \n");
  if (mask_only == 1) {
    printf("\n Insert Vth :");
    Vth = (int)(LSBCAL * atof(gets(buff)));
      for_any_pixel
        enable_mask[i][j] = 0;
      dac_bias_tmp[2 + CHIP * 5] = Vth;
      turn_on_dac();
      load_mask();
      start_acq(RADIOGRAPHY);     //start the acquisition
    if (MISSING_MEDIPIX == 0)
      download_data();
                                    // il calcolo e' fatto
      c = 0;
    for_any_pixel {
        matrix2[i][j] = A[data_matrix[i][j]];
        if (matrix2[i][j] >= MINC) {
          c++;
          enable_mask[i][j] = 1;
      }
    }
   }
   else {
      Vth = 512;
    while (Vth <= 4096) {
        for_any_pixel
            enable_mask[i][j] = 0;
        dac_bias_tmp[2 + CHIP * 5] = Vth;
```

```
        turn_on_dac();
        load_mask();
        start_acq(RADIOGRAPHY);     //start the acquisition
      if (MISSING_MEDIPIX == 0)
        download_data();
                                    // il calcolo e' fatto
        c = 0;
        for_any_pixel {
            matrix2[i][j] = A[data_matrix[i][j]];
            if(matrix2[i][j] >= MINC) {
              c++;
              enable_mask[i][j] = 1;
            }
        }
      if (c <= (PERC * ((R2 - R1 + 1) * (C2 - C1 + 1))))
        break;
      Vth += 32;
      }
  }

  for_any_pixel
      config_matrix[i][j] = config_matrix[i][j] + enable_mask[i][j];
  Save_Mask();
  printf("\nVthmin =%.3f \t Noisy pixels:%d", Vth / LSBCAL, c);
  printf("\nEnd of the minimum threshold function ");
  wait_keypress();

}
/************** END ********************/


void test_counters_us  (void)
{
  int choice, c;
  char buff[80];

ONE:
  printf("\n\n\n\n\n\n\n\n\n\n\n");
  display_values();
  printf("\nThis is the TEST_COUNTERS menu");

  printf(" .... possible choices are:\n\n");
  printf("1) Get a file pattern;\n\n");
  printf("2) Save a file pattern;\n\n");
  printf("3) Modify pattern;\n\n");
  printf("4) Show current pattern;\n\n");
  printf("5) DO test;\n\n");
  printf("6) Return to Basic_test Menu.\n\n");
  printf("Now you must make a choice: ");

TWO:
  choice = atoi(gets(buff));

  if (choice == 1)
    get_pattern();
   else if (choice == 2)
    save_pattern();
```

```
   else if (choice == 3)
    modify_pattern();
   else if (choice == 4) {
    display_matrix_values(pattern_matrix, "pattern_matrix");
            // actually this shows the pattern and not mask values
    printf("\n\n\t\tPress 'return' to continue:");
      wait_keypress();
   }
   else if (choice == 5)
    do_test();
   else if (choice == 6) {
    printf("\nBYE!!!!\n");
    goto THREE;
   }
   else {
      bad_option(6, START_FROM_1);
    goto ONE ;
  }

  goto ONE;

THREE:   {}

}


void get_pattern(void)
{
  int c, choice;
  char SF[20], buff[4];

  printf("\n\n\n\n");
  printf("\n\n\n\n\n\n\n\n\nthis is the GET PATTERN menu\n\n\n\n\n\n");
  printf("\n\n\nthis is the view_pattern_files menu\n\n\n\n\n\n");
  printf("\nInsert the file name that you want to load: ");

  gets(SF);

  selected_file = (char *)SF;

  if (read_file_pattern(selected_file, pattern_matrix) == 0) {
      printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

UNO:
    printf("Do you want to display 'pattern'? ('1'=yes,'2'=no)");
      choice = atoi(gets(buff));

      if (choice == 1) {
      display_matrix_values(pattern_matrix, "pattern_matrix");
               // in reality this show the pattern and not the mask values
      goto UNO;
       }
       else if (choice == 2)
        goto THREE;
       else {
      bad_option(2, START_FROM_1);
      goto UNO;
```

```
      }
   }
   else {
  }
/* this means "read_file_pattern(selected_file,pattern_matrix)=1"
   and the user has decided to not change a wrong name for the pattern file */

THREE:      {}

}


int read_file_pattern(char *filename, int mat[64][64])
{
  char buff[50];
  int i, j;

  if (open_read_file(filename) == 0) {
           // this must be=0 if the file to read exists
      for_any_pixel
        fscanf(fpin, "%d", &mat[i][j]);   // gets the pattern_matrix

      fclose(fpin);

      return(0);
   }
   else
    return(1);
/* this condition is true if the user selects not to try to load a file,
   and changes the name, in the case that the specified name is not valid */
}


void save_pattern(void)
{
  int i, j, k;
  char buff[30], pathname[FILENAMELEN];

/**** questa funzione serve a salvare un file di tipo ".msk"
a partire dalla attuale maschera in ram (config_matrix)****/

  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("You can save current 'pattern matrix'");
  printf("in an archive (filename.pat) file\n\n");
  printf("Insert the 'filename.pat', please: ");
  gets(buff);

  if (open_write_file(buff, SAVE_WITH_CONTROL) == 0) {
      for(i = 0; i < 64; i++) {
      for(j = 0; j < 64; j++)
           fprintf(fpout,   "%d   ",   pattern_matrix[i][j]);//   write   the
pattern_matrix
        fprintf(fpout, "\n");
      }
      fclose(fpout);

      sprintf(pathname, "%spattern.names", PATHF);
```

```
      fpout = fopen(pathname, "a");
      fprintf(fpout, "%s\n", buff);
      fclose(fpout);
// here the file with the list of "*.pat" files is updated
   }
   else;
}


void modify_pattern(void)
{
  int C1, C2, L1, L2, choice;
  char buff[30];
  int c, i, j, k;

  printf("\n\n\n\n");
  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("\nYou can change the pattern from:");
  printf("\nrows L1 to L2 (0-63);\ncolumns C1 to C2 (0-3)\n\n");

ZER:
  printf("Please, insert new value (0-%d): ", 32767);

  choice = atoi(gets(buff));

  if (choice < 0 || choice > 32767) {
      printf("\n\n\n\n\n\n\n\n\n\n");
      printf("\t\t\t\tBAD OPTION!!\n\n\n");
      printf("\t\t\t Possible values are (0-%d)", 32767);
      printf("\n\n\t\tPress 'return' and then Re-type:\n\n\n\n\n\n\n\n\n");
      wait_keypress();

      goto ZER;
  }

ONE:
  printf("Please, insert L1: ");

  L1 = atoi(gets(buff));

  if (L1 < 0 || L1 > 63) {
      printf("\n\n\n\n\tBAD pixel coordinate");
      printf("\n\n\t\tPress 'return' and then Re-type:\n\n\n\n\n\n\n\n\n");
      wait_keypress();
      printf("\n\n\n\n\n\n\n");
      goto ONE;
  }

TWO:
  printf("Please, insert L2 (>=%d): ", L1);

  L2 = atoi(gets(buff));

  if(L2 < L1 || L2 > 63) {
    printf("\n\n\n\n\tBAD pixel coordinate");
      printf("\n\n\t\tPress 'return' and then Re-type:\n\n\n\n\n\n\n\n\n");
      wait_keypress();
```

```
      printf("\n\n\n\n\n\n\n");
      goto TWO;
  }

THREE:
  printf("Please, insert C1: ");

  C1 = atoi(gets(buff));

  if (C1 < 0 || C1 > 3) {
      printf("\n\n\n\n\tBAD pixel coordinate");
      printf("\n\n\t\tPress 'return' and then Re-type:\n\n\n\n\n\n\n\n\n\n");
      wait_keypress();
      printf("\n\n\n\n\n\n");
      goto THREE;
  }

FOUR:
  printf("Please, insert C2 (>=%d): ", C1);

  C2 = atoi(gets(buff));

  if (C2 < C1 || C2 > 3) {
      printf("\n\n\n\n\tBAD pixel coordinate");
      printf("\n\n\t\tPress 'return' and then Re-type:\n\n\n\n\n\n\n\n\n\n");
      wait_keypress();
      printf("\n\n\n\n\n\n");
      goto FOUR;
  }

  for(i = L1; i <= L2; i++)            // changes only selected _bit_pattern
    for(k = 0; k < 16; k++)
        for(j = C1; j <= C2; j++)
            pattern_matrix[i][j + (k * 4)] = choice;
}


void do_test(void)
{
  int i, j, cycles_dummy, err = 0, i1, j1, c;

  cycles_dummy = cycles_tmp;
  cycles_tmp = 1;
  if (check_power() != 1)
    printf("\n\n\t\tMedipix is off!\n\n");
/*
            reset_and_setup();
            reset_counters((unsigned int)pattern_matrix[1][1]);

   */
      /* patterns are written on medipix        */
  for(j = 3; j >= 0; j--)
      for(i = 63; i >= 0; i--) {
        cycles_tmp = 64 * j + 1;
        reset_and_setup();
        reset_counters((unsigned int)pattern_matrix[i][j]);
      }
```

```
  cycles_tmp=cycles_dummy;

  if (MISSING_MEDIPIX == 0)
    download_data();

  for_any_pixel
      if (pattern_matrix[i][j] != data_matrix[i][j])
        err++;

  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");

  if (err != 0)
    printf("Read Data array is different from Loaded Data array");
   else
    printf("test ok!, no error ");

  printf("\nBoth arrays are saved in file 'pat.error'");

  open_write_file("pat.error", SAVE_WITHOUT_CONTROL);
  fprintf(fpout,"/******** pattern matrix *******/\n");
  for(i1 = 0; i1 < 64; i1++) {
    for(j1 = 0; j1 < 64; j1++)
      fprintf(fpout, "%d ", pattern_matrix[i1][j1]);
    fprintf(fpout,"\n");
  }

  fprintf(fpout,"\n/******** data matrix *******/\n");
  for(i1 = 0; i1 < 64; i1++) {
    for(j1 = 0; j1 < 64; j1++)
      fprintf(fpout, "%d ", data_matrix[i1][j1]);
    fprintf(fpout,"\n");
  }
  fclose(fpout);

  printf("\n\n\t\tPress 'return' to continue:\n\n");
  wait_keypress();
}


/************** FUNCTION CERN7 ***************/
void cern7_us()
{
/***************** funzione elena per acq. automatiche  ******************/
  int i, j, k, ii, iii, I, jj, c, cc, crepeat = 11, s, choice = 0, choiceAr = 0,
      pulseM = 100;
  int VBi, VCi, Nvb, Nvc, Vth, choicePul = 0, MINC = 2, Mrep, mrep, Irep;
  float pulseW = 300, pulseT = 0.1, PERC = 0.1, pulseV = 10, pulseST = 10,
        pulseS = 10;
  int Vb_val[30], Vc_val[30], Vth_val[30][30];
  char buff[FILENAMELEN], fil[FILENAMELEN], filepath[FILENAMELEN],
       lista[FILENAMELEN], file0[FILENAMELEN], plist[FILENAMELEN];
  char maskfile[FILENAMELEN], maschera[FILENAMELEN];
  int A[32768], R1 = 0, R2 = 63, C1 = 0, C2 = 63;
  int matrix2[64][64], config_matrix0[64][64], m1[64][64], m2[64][64],
      m3[64][64];
  int trash, CHIP;
  int exist, size;
```

```
  CHIP = chip_sel_tmp;

  A[0] = 696969;    /* unrealistic number to be read if counters output
                       pseudorandom number 0 (impossible) */
  for_any_pixel
    config_matrix0[i][j] = 0;

  open_read_file("LUT_real.lut");
  for (i = 1; i < 32768; i++)
    fscanf(fpin, "%d", &A[i]);
  fclose(fpin);

ENTER_FILE_NAME:
  printf("\nWhat is the file Prefix for this run ?");
  gets(fil);

/*  sprintf(file0,"%sthresh2/%s.thr",PATHF,fil);
  exist=GetFileInfo(file0,&size);
  if (exist) {
    static char message[300]
      sprintf(message,"%s FILE PREFIX ALREADY EXISTS!!"
              "\n\nDo you really want to OVERWRITE it?",fil);
      answer = ConfirmPopup ("Warning !!!", message);
      if (!answer)
      goto ENTER_FILE_NAME;
  }*/

  printf("\n Do you want to charge a special enable/thres mask file? (1=yes)");
  if ((c = atoi(gets(buff))) == 1) {
    printf("\n Insert mask filename: ");
    gets(buff);
    sprintf(maskfile, "%s%s", PATHF,buff);
    read_file_mask(maskfile, m1, m2, m3);

/* enable, test, thr */
      for_any_pixel
        config_matrix0[i][j] = ((m3[i][j] & 1) << 4) +
                                    ((m3[i][j] & 2) << 2) +
                                      ((m3[i][j] & 4)) +
                                      (m2[i][j] << 1) +
                                 m1[i][j];
  }

  printf("\nDo you want to charge a special threshold mask?");
  printf("\nonly thr_adj=0-7, 10=8 repeated acquisition thr_adj, 11=use loaded
.msk: ");
  crepeat = atoi(gets(buff));
  if (crepeat == 10) {
      printf("\nfrom adj # ? ");
      mrep = atoi(gets(buff));
      printf("\nto adj # ? ");
      Mrep = atoi(gets(buff));
   }
   else if (crepeat < 10) {
    mrep = Mrep = crepeat;
   }
```

```
   else if (crepeat == 11) {
     mrep = Mrep = 0;
   }

   printf("\nName will be %s_Q#_Vbias_Vcomp.dat", fil);
   printf("\n(Q is number of test pulse) ");
   printf("\nTest Pulse Width (ns,real)? ");
   pulseW = atof(gets(buff));

   printf("\nTest Pulse Period (ms,real)? ");
   pulseT = atof(gets(buff));

   printf("\nNumber of different Vbias:");
   Nvb = atoi(gets(buff));

   printf("\nNumber of different Vcomp:");
   Nvc = atoi(gets(buff));

   for(i = 0; i < Nvb; i++) {
     printf("\n Insert Vbias #%d (V):", i);
     Vb_val[i] = (int)(LSBCAL * atof(gets(buff)));
   }

   for(i = 0; i < Nvc; i++) {
     printf("\n Insert Vcomp #%d (V):", i);
     Vc_val[i] = (int)(LSBCAL * atof(gets(buff)));
   }

   printf("\n Insert Vdl (V):");
   dac_bias_tmp[4 + CHIP * 5] = (int)(LSBCAL * atof(gets(buff)));

   printf("\nVth auto (=1) or manual (=2)?:");
   choice = atoi(gets(buff));

   if (choice == 1) {
     printf("\nOnly Vth measurement (=1) or calibration too (=2) ?:");
     choiceAr = atoi(gets(buff));
   }

   if (choiceAr != 1) {
     printf("\nPulse Generator auto (=1) or manual (=2):");
     choicePul = atoi(gets(buff));
   }

   if(choicePul == 1) {
     printf("\nFirst Test Pulse Height (mV, min=10mV) ?");
     pulseST = atof(gets(buff));
       if (pulseST < 10)
         pulseST = 10;
     printf("\nTest Pulse Step (mV) ?");
     pulseS = atof(gets(buff));
     printf("\nMax Test Pulse Height (mV) ?");
     pulseM = atof(gets(buff));
   }

/*****************  Vth automatico  **********************/
   if(choice == 1) {
```

```
/******* chiede di definire la zona per il calcolo di Vth ******/
      trig_mode_tmp = 0;
      printf("\nInsert the fraction of bad pixels admitted:");
      PERC = atof(gets(buff));
      printf("\nInsert the minimum bad count admitted (default = 2):");
      MINC = atoi(gets(buff));

    printf("\n Vth matrix (changing Vbias&Vcomp)\n");
    for(VBi = 0; VBi < Nvb; VBi++) {
      dac_bias_tmp[0 + CHIP * 5] = Vb_val[VBi];
      for(VCi = 0; VCi < Nvc; VCi++) {
        dac_bias_tmp[1 + CHIP * 5] = Vc_val[VCi];
            Vth = 512;
        for (ii = 0; Vth <= 4096; ii++) {
          dac_bias_tmp[2 + CHIP * 5] = Vth;
              turn_on_dac();
              c = 0;

/******** scrittura delle maschere ************/
            for_any_pixel
/******* la maschera tiene conto della zona di acq. *****/
/* config_matrix = 0 -> enable on, test off
                 = 1 -> enable off, test off
                 = 2 -> enable on, test on
                 = 3 -> enable off, test on  */

            if(i >= R1 && i <= R2 && j >= C1 && j <= C2)
                config_matrix[i][j] = 0;
               else
                config_matrix[i][j] = 1;

            load_mask();
          start_acq(RADIOGRAPHY);     //start the acquisition
          if (MISSING_MEDIPIX == 0)
            download_data();

// il calcolo e' fatto
            for_any_pixel {
                  matrix2[i][j] = A[data_matrix[i][j]];
                  if (matrix2[i][j] >= MINC)
                    c++;
            }
          if (c <= (PERC * ((R2 - R1 + 1) * (C2 - C1 + 1))))
            break;
          Vth += 32;
        }

          Vth_val[VBi][VCi] = Vth;

          printf("Vbias =%.3f   Vcomp =%.3f   Vthmin =%.3f\n",
                 Vb_val[VBi] / LSBCAL, Vc_val[VCi] / LSBCAL, Vth / LSBCAL);
        sprintf(file0, "%sthresh2/%s.thr", PATHF, fil);
// Mind the ATTRIBUTE, must be "w".
      fpout = fopen(file0, "a");
          fprintf(fpout, "Vbias =%.3f   Vcomp =%.3f   Vthmin =%.3f\n",
                  Vb_val[VBi] / LSBCAL, Vc_val[VCi] / LSBCAL, Vth / LSBCAL);
```

```
          fclose(fpout);
        }
        printf("\n");
      }

      printf("\nEnd of the min-thres calculation ");
      wait_keypress();
    }

    if (choiceAr == 1)
      return;

    /************* Vth manuale ***********************/
    trig_mode_tmp = 2;

    if (choice == 2) {
      for(i = 0; i < Nvb; i++)
        for(j = 0; j < Nvc; j++) {
          printf("\n Vbias=%.3f\tVcomp=%.3f",Vb_val[i] / LSBCAL,
                 Vc_val[j] / LSBCAL);
            printf("\n Insert Vth:");
            Vth_val[i][j] = (int)(LSBCAL * atof(gets(buff)));
        }

      printf("\n Insert the value of Vtha:");
      dac_bias_tmp[3 + CHIP * 5] = (int)(LSBCAL * atof(gets(buff)));
    }

/***************** fine impostazione dei dati **********/

    for (Irep = mrep; Irep <= Mrep; Irep++) {
        for_any_pixel{
        c = Irep;
        if (crepeat == 11)
          c = m3[i][j];
        cc = ((c & 1) << 2) + (c & 2) + ((c & 4) >> 2);
        cc = cc * 4;
          config_matrix0[i][j] = (config_matrix0[i][j] & 3) + cc;
      }

      pulser(pulseST, pulseT, pulseW);
      pulseV = pulseST;
      turn_off_dac();
      sleep_soft(10000000);

      for(jj = 0; pulseV <= pulseM; jj++) {  // for principale sugli impulsi
        if (choicePul == 1)
          pulser(pulseV, pulseT, pulseW);
         else {
/* manual pulser */
          printf("\nTest Pulse Height (mV) ?");
          pulseV = atof(gets(buff));
          printf("\nReady with pulse generator? (go=0 exit=1)");
          if ((c = atoi(gets(buff))) == 1)
            break;
        }
```

```
      for(VBi = 0; VBi < Nvb; VBi++) {       // for sul Vbias
        dac_bias_tmp[0 + CHIP * 5] = Vb_val[VBi];
        for(VCi = 0; VCi < Nvc; VCi++) {     // for sul Vcomp
            dac_bias_tmp[1 + CHIP * 5] = Vc_val[VCi];
            dac_bias_tmp[2 + CHIP * 5] = Vth_val[VBi][VCi];
            turn_on_dac();
            if (crepeat == 11)
              trash = 11;
             else
              trash = Irep ;
            sprintf(filepath, "%sthresh1/%s_%d_Q%d_%.2f_%.2f.dat", PATHF, fil,
                    trash, jj, dac_bias_act[0 + CHIP * 5] / LSBCAL,
                    dac_bias_act[1 + CHIP * 5] / LSBCAL);
        if (jj == 0) {
                sprintf(plist, "%sthresh1/%s_%d.lst", PATHF, fil, trash);
// Mind the attribute, must be "w"!
            fpout = fopen(plist, "a");
                fprintf(fpout, "%s_%d_%.2f_%.2f\n", fil, trash,
                        dac_bias_act[0 + CHIP * 5] / LSBCAL,
                        dac_bias_act[1 + CHIP * 5] / LSBCAL);
                fclose(fpout);
            }

            printf("\n#%d, test pulse=%.1fmV\tVbias=%.3f\tVcomp=%.3f\tVth=%.3f",
                   jj + 1, pulseV, dac_bias_act[0 + CHIP * 5] / LSBCAL,
                   dac_bias_act[1 + CHIP * 5] / LSBCAL,
                   dac_bias_act[2 + CHIP * 5] / LSBCAL);

/********* scrittura della lista ************/
          sprintf(lista, "%sthresh1/%s_%d_%.2f_%.2f.lst", PATHF, fil, trash,
                  dac_bias_act[0 + CHIP * 5] / LSBCAL,
                  dac_bias_act[1 + CHIP * 5] / LSBCAL);
// Mind the ATTRIBUTE, must be "w"
            fpout = fopen(lista,"a");
            fprintf(fpout, "%s\n", filepath);
            fclose(fpout);

               for(I = 0; I < 64; I++) {               /***** for sulle righe
******/
/******** scrittura delle maschere ************/
              for_any_pixel
                    if(i == I && i >= R1 && i <= R2 && j >= C1 && j <= C2)
                      config_matrix[i][j] = config_matrix0[i][j] + 2;
                     else
                      config_matrix[i][j] = 1;

                load_mask();
/****** mask is loaded **/

            start_acq(RADIOGRAPHY);     /*start the acquisition */
            if (MISSING_MEDIPIX == 0)
              download_data();

/*********** convert and save *************/
            for(j = 0; j < 64; j++) {
              matrix2[I][j] = A[data_matrix[I][j]];
              if (matrix2[I][j] >= 32800)
```

```
                printf("\nfunny number! error:%x in pixel %d %d ",
                       data_matrix[I][j], I, j);
            }
                if (I == 0) {
                sprintf(file0, "%sthresh1/%s_%d_Q%d_%.2f_%.2f.info",
                             PATHF, fil, trash, jj, dac_bias_act[0 + CHIP*5] /
LSBCAL,
                      dac_bias_act[1 + CHIP * 5] / LSBCAL);
                fpout = fopen(file0, "w");
              fprintf(fpout, "anin cycles=%d, anin period=%d, anin delay=%d\n",
                      anin_cycles_act, sleep_anin_period_act,
                      sleep_anin_delay_act);
              fprintf(fpout,  "Pulse  Period=%.3fms,  Pulse  Width=%.1fns,  Pulse
Height=%.3fmV\n", pulseT, pulseW, pulseV);
              fprintf(fpout, "DAC values (V) : Vbias=%.3f, Vcomp=%.3f, Vth=%.3f,
Vtha=%.3f,Vdl=%.3f\n",
                      dac_bias_act[0 + CHIP * 5] / LSBCAL,
                      dac_bias_act[1 + CHIP * 5] / LSBCAL,
                      dac_bias_act[2 + CHIP * 5] / LSBCAL,
                      dac_bias_act[3 + CHIP * 5] / LSBCAL,
                      dac_bias_act[4 + CHIP * 5] / LSBCAL);
                fclose(fpout);
                }
// Mind the attribute, must be "w"
              fpout = fopen(filepath, "a");
            for(j = 0; j < 64; j++)
              fprintf(fpout, "%d ", matrix2[I][j]);
            fprintf(fpout, "\n");
            fclose(fpout);
          }                                       // end rows loop
          }                              /******** end of Vcomp loop ******/
      }                        /*********** end of Vbias loop ********/
      printf("\nEnd of loop with pulse=%.3f mV (acquisition %d)", pulseV, Irep);

      if (choicePul == 2) {
        printf("\nDo you want to stop here? (yes=1)");
        c = atoi(gets(buff));
        if (c == 1)
          break;
      }

/*auto pulser*/
      if (choicePul == 1)
        pulseV = pulseV + pulseS;
    }                                            /* end pulse loop */

    printf("\n\nEnd of acquisition #%d\n", Irep);
  } /*end of repeated acquisition loop */

  printf("\nEnd of the elena function ");
  wait_keypress();
}
/************** END ********************/


/* GPIB, Pulse Generator HP 8130A */
void pulser(float AMPL, float peri, float widt)
```

```
{
  char cmd[50];
  int amp, i, k;
  float fact = 10;

/* indirizzi GPIB:
NI-VME Board ha indirizzo 0, access non-privileged, A16=1000,
      cioe' 0100   (ottale)      o '@'    talk address
      cioe' 040   o ' ' listen address
l'impulsatore ha indirizzo  10
      cioe' 112 o 'J'   talk address 10
      cioe' 52  o '*'   listen address 10
*/

  char NIT, NIL;
  char PULT, PULL;

  NIT = '@';
  NIL = ' ';

  Ibonl(1);       // initialize the interface
  Ibsic();        //send IFC to clear the GPIB
  Ibsre(1);  // set REN to prepare for placing the instrument in remote mode
// reset

  Ibcmd("\021*\024@", 4);

  sprintf(cmd, ":INP:TRIG:MODE TRIG");
  Ibwrt(cmd, strlen(cmd));
  sprintf(cmd, ":INP:TRIG:SLOP POS");
  Ibwrt(cmd, strlen(cmd));
  sprintf(cmd, ":INP:TRIG:THR 1.0V");
  Ibwrt(cmd, strlen(cmd));
  sprintf(cmd, ":PULS:EDGE:LEAD 1ns");
  Ibwrt(cmd, strlen(cmd));
  sprintf(cmd, ":PULS:EDGE:TRA 1ns");
  Ibwrt(cmd, strlen(cmd));
  sprintf(cmd, ":PULS:TIM:PER %fms", peri);
  Ibwrt(cmd, strlen(cmd));
  sprintf(cmd, ":PULS:TIM:WIDT %fns", widt);
  Ibwrt(cmd, strlen(cmd));

  amp = (int)(fact * AMPL); // fatt. correzione display valore in mV
  sprintf(cmd, ":PULS:LEV:AMPL %dmV", amp);
  Ibwrt(cmd, strlen(cmd));

  sleep_soft(1000000);

/* disable the interface       */
  Ibonl(0);
}


void Save_Mask(void)
{
  int i, j, k;
  char buff[30], filepath[FILENAMELEN];
```

```
/**** questa funzione serve a salvare un file di tipo "msk"
 a partire dalla attuale maschera in ram (config_matrix)****/

  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  printf("You can save current 'configuration MASK' in an archive ");
  printf("(filename.msk) file\n\n");
  printf("Insert the 'filename.msk', please: ");
  gets(buff);

  if (open_write_file(buff, SAVE_WITH_CONTROL) == 0) {
      fprintf(fpout, "* enable mask *\n");
      fprintf(fpout, "\n");
      for(i = 0; i < 64; i++) {
      for(j = 0; j < 64; j++)
            fprintf(fpout, "%d ", (config_matrix[i][j] & 1));
                                                  // write the enable_bit_mask
      fprintf(fpout, "\n");
      }

      fprintf(fpout, "\n");
      fprintf(fpout, "* test mask *\n");
      fprintf(fpout, "\n");
      for(i = 0; i < 64; i++) {
      for(j = 0; j < 64; j++)
            fprintf(fpout, "%d ", (config_matrix[i][j] & 2) >> 1);
                                                  // write the test_bit_mask
      fprintf(fpout, "\n");
      }

      fprintf(fpout, "\n");
      fprintf(fpout, "* threshold mask *\n");
      fprintf(fpout, "\n");
      for(i = 0; i < 64; i++) {
      for(j = 0; j < 64; j++) {
            k = (config_matrix[i][j] & 4) +
                ((config_matrix[i][j] & 8) >> 2) +
                ((config_matrix[i][j] & 16) >> 4);
            fprintf(fpout,"%d ", k);                          //  write  the
threshold_bit_mask
        }
        fprintf(fpout, "\n");
      }
      fclose(fpout);

      sprintf(filepath, "%smask.names", PATHF);
// Mind the attribute, must be "w"
      fpout = fopen(filepath, "a");
      fprintf(fpout, "%s\n", buff);
      fclose(fpout);
/* here the file with the list of "*.msk" files is updated */

   }
   else;

}
```

```
void bad_option(int i, int k)
{
  int c, j;
  printf("\n\n\n\n\n\n\n\n\n\n\n");
  printf("\t\t\t\tBAD OPTION!!\n\n\n");
  printf("\t\t\t Possible keys are");
  if (k == 1)
      for(j = 1; j <= i; j++)
        printf(" %d,", j);
   else
      for(j = 0; j <= i; j++)
        printf(" %d,", j);

  printf("\n\n\t\tPress        'return'        and        then        Re-type
correctely:\n\n\n\n\n\n\n\n\n\n");
  wait_keypress();
}


void display_values(void)
{

  printf("these are the ACTIVE values:\n");
  printf("\t tau0=%dms; tau1(acq time)=%dms; tau2=%dms; anin_cycles=%d;\n",
       tau0_act, tau1_act, tau2_act, anin_cycles_act);
  printf("\t trig_mode=%d; cycles=%d; config_mode=%d.\n", trig_mode_act,
         cycles_act, config_mode_act);

  printf("these are the TEMPORARY values:\n");
  printf("\t tau0=%dms; tau1(acq time)=%dms; tau2=%dms; anin_cycles=%d;\n",
         tau0_tmp, tau1_tmp, tau2_tmp, anin_cycles_tmp);
  printf("\t trig_mode=%d; cycles=%d; config_mode=%d.\n", trig_mode_tmp,
         cycles_tmp, config_mode_tmp);
}


void get_mask(void)
{
  int i, j, threshold_mask[64][64], testbit_mask[64][64], enable_mask[64][64];
  int c, choice;
  char SF[20], buff[4];
  char maskfile[FILENAMELEN];

  if (cold_start == 1) {
    sprintf(maskfile, "%sdefault.msk", PATHF);

      read_file_mask(maskfile, enable_mask, testbit_mask, threshold_mask);
        /* here config_matrix is build starting from
           enable_mask, testbit_mask,threshold_mask */

      for_any_pixel
        config_matrix[i][j] = ((threshold_mask[i][j] & 1) << 4) +
                              ((threshold_mask[i][j] & 2) << 2) +
                                    ((threshold_mask[i][j] & 4)) +
                                    (testbit_mask[i][j] << 1) +
                              enable_mask[i][j];
```

```
   }
   else {
    printf("\n\n\n\n");
      printf("\n\n\n\n\n\n\n\n\nthis is the GET MASK menu\n\n\n\n\n");
    printf("\n Insert mask filename: ");
    gets(SF);
    sprintf(maskfile,"%s%s", PATHF, SF);

      if (read_file_mask(maskfile, enable_mask, testbit_mask, threshold_mask) ==
0) {
        printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
UNO:
      printf("Do you want to visualize the \n'enable-mask',");
        printf("'testbit-mask' or 'threshold-mask'?");
        printf("('1','2','3','4'=no)");
        choice = atoi(gets(buff));
        if (choice == 1) {
            display_matrix_values(enable_mask, "enable_mask");
        goto UNO;
       }
       else if (choice == 2) {
        display_matrix_values(testbit_mask, "testbit_mask");
        goto UNO;
       }
       else if (choice == 3) {
        display_matrix_values(threshold_mask, "threshold_mask");
                              // va fatta tutta la visualizzazione
        goto UNO;
       }
       else if (choice == 4) {
        goto FOUR;
       }
       else {
            bad_option(4, START_FROM_1);
        goto UNO;
        }

FOUR:
      printf("\n\n\n\n\n\n\t\tDo you want to confirm ");
        printf("these masks? ('1'=yes,'2'=no)\n\n\n");

        choice = atoi(gets(buff));
      if (choice == 1) {
            for_any_pixel
          config_matrix[i][j] = ((threshold_mask[i][j] & 1) << 4) +
                                       ((threshold_mask[i][j] & 2) << 2) +
                                               ((threshold_mask[i][j] & 4)) +
                                               (testbit_mask[i][j] << 1) +
                                               enable_mask[i][j];

       }
       else if (choice == 2);
       else {
            bad_option(2, START_FROM_1);
            goto UNO;
        }
```

```
      }
      else;
           /* this means "read_file_mask(selected_file,threshold_mask,
           enable_mask,testbit_mask)=1"
           and the user has decided to not change a wrong name
           for the mask file */
  }
}


void subt_us(void)
{
  FILE *fpi;
  FILE *fpo;
  char c, buff[20];
  char fil1[FILENAMELEN], fil2[FILENAMELEN], fil3[FILENAMELEN];
  char fil10[FILENAMELEN], fil20[FILENAMELEN], fil30[FILENAMELEN];
  int count1[64][64], count2[64][64], count3[64][64];
  int i, j, fact1 = 1, fact2 = 1;

  printf("\n insert the data file:");
  gets(fil1);

  printf("\n insert the background file:");
  gets(fil2);

  printf("\n insert the output file:");
  gets(fil3);

  printf("\n insert the data file multiplication factor (default=1):");
  fact1 = atoi(gets(buff));
  printf("\n insert the back file multiplication factor (default=1):");
  fact2 = atoi(gets(buff));

  sprintf(fil10, "%s%s", PATHF, fil1);
  sprintf(fil20, "%s%s", PATHF, fil2);
  sprintf(fil30, "%s%s", PATHF, fil3);

  FileToArray (fil10, count1, VAL_INTEGER, 4096, 64, VAL_GROUPS_TOGETHER,
               VAL_GROUPS_AS_COLUMNS, VAL_ASCII);

  FileToArray (fil20, count2, VAL_INTEGER, 4096, 64, VAL_GROUPS_TOGETHER,
               VAL_GROUPS_AS_COLUMNS, VAL_ASCII);

  for_any_pixel {
//              count3[i][j] = count1[i][j] || count2[i][j]  ;
      count3[i][j] = fact1 * count1[i][j] + fact2 * count2[i][j];
      if (count3[i][j] < 0)
        count3[i][j] = 0;
  }

  ArrayToFile (fil30, count3, VAL_UNSIGNED_INTEGER, 4096, 64,
               VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS, VAL_SEP_BY_TAB, 10,
               VAL_ASCII, VAL_TRUNCATE);

  printf("\nEnd of data subtraction function ");
  wait_keypress;
```

```
}


void eff_cal_us(void)
{
  FILE *fpi;
  FILE *fpo;
  char buff[80];
  char c;
  char fil1[FILENAMELEN], fil2[FILENAMELEN], fil3[FILENAMELEN];
  char fil10[FILENAMELEN], fil20[FILENAMELEN],fil30[FILENAMELEN];
  int count1[64][64], count2[64][64];
  int i, j, k, kk, tot1 = 0, tot2 = 0;
  float med = 0., sig = 0., dist = 0., r_tot1 = 0.,
        r_count1[64][64], r_count2[64][64];

  printf("\n insert the data file:");
  gets(fil1);

  printf("\n insert the background file:");
  gets(fil2);

  printf("\n insert the output calibrated file:");
  gets(fil3);

//      printf("\n insert the factor (divisore di integrale):");
//      FACT=atoi(gets(buff));

  sprintf(fil10, "%s%s", PATHF, fil1);
  sprintf(fil20, "%s%s", PATHF, fil2);
  sprintf(fil30, "%s%s", PATHF, fil3);

  FileToArray (fil10, count1, VAL_INTEGER, 4096, 64, VAL_GROUPS_TOGETHER,
               VAL_GROUPS_AS_COLUMNS, VAL_ASCII);
  FileToArray (fil20, count2, VAL_INTEGER, 4096, 64, VAL_GROUPS_TOGETHER,
               VAL_GROUPS_AS_COLUMNS, VAL_ASCII);


  for_any_pixel {
    r_count2[i][j] = 0.;
    r_count1[i][j] = 0.;
    if (count2[i][j] == 696969)
      count2[i][j] = 0;
    if (count1[i][j] == 696969)
      count1[i][j] = 0;
      tot1 = tot1 + count1[i][j];
      tot2 = tot2 + count2[i][j];
  }

  for_any_pixel {
    r_count2[i][j] = (float)(count2[i][j]) / (float)tot2 ;
      if (r_count2[i][j] == 0.)
        r_count1[i][j] = 0.;
       else
        r_count1[i][j] = (float)(count1[i][j]) / r_count2[i][j] ;
  }
```

```
  for_any_pixel
      r_tot1 = r_tot1 + r_count1[i][j];

  for_any_pixel
      count1[i][j] = (int)((r_count1[i][j] / (r_tot1)) * tot1);

  for_any_pixel {
    med = 0.;
    sig = 0.;
      dist = 0.;
      if (i > 0 & i < 63 & j > 0 & j < 63) {
        med = r_count1[i - 1][j - 1] + r_count1[i][j - 1] +
                  r_count1[i + 1][j - 1] + r_count1[i - 1][j] +
                  r_count1[i + 1][j] + r_count1[i - 1][j + 1]+
                  r_count1[i][j + 1] + r_count1[i + 1][j + 1];
        med = (med / 8.);
        sig = (r_count1[i - 1][j - 1] - med) * (r_count1[i - 1][j - 1] - med) +
                  (r_count1[i][j - 1] - med) * (r_count1[i][j - 1] - med) +
                  (r_count1[i + 1][j - 1] - med) * (r_count1[i + 1][j - 1] -
med) +
                  (r_count1[i - 1][j] - med) * (r_count1[i - 1][j] - med) +
                  (r_count1[i + 1][j] - med) * (r_count1[i + 1][j] - med) +
                  (r_count1[i - 1][j + 1] - med) * (r_count1[i - 1][j + 1] -
med) +
                  (r_count1[i][j + 1] - med) * (r_count1[i][j + 1] - med) +
                  (r_count1[i + 1][j + 1] - med) * (r_count1[i + 1][j + 1] -
med);
        sig = sqrt(sig / 8.);
        dist = r_count1[i][j] - med;
        if (dist < 0)
          dist = -dist;
        if(dist > (2 * sig) | count1[i][j] == 0) {
            count1[i][j] = (int)(med * tot1 / r_tot1);
            r_count1[i][j] = med;
        }
      }
  }
  tot2 = 0;
  for_any_pixel
      tot2 = tot2 + count1[i][j];

  ArrayToFile(fil30, count1, VAL_UNSIGNED_INTEGER, 4096, 64,
              VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS, VAL_SEP_BY_TAB, 10,
              VAL_ASCII, VAL_TRUNCATE);

  printf("\nIntegral counts: %d",tot1);
  printf("\nIntegral counts after normalization: %d", tot2);

  printf("\nEnd of data efficiency calibration ");
  wait_keypress();
}


void off_noise_us()
{
/*** funzione per il calcolo automatico di Vth minima (maurizio) ******/
  int i, j, k, ii, iii, I, jj, c = 0, s;
```

```
  int Vth, MINC, tempo, mask_only;
  int tot, MAX;
  char buff[20];
  int A[32768], C1 = 0, C2 = 63, R1 = 0, R2 = 63;
  int matrix2[64][64], enable_mask[64][64];

  A[0] = 696969;    /* unrealistic number to be read if counters output
                       pseudorandom number 0 (impossible) */
  open_read_file("LUT_real.lut");
  for (i = 1; i < 32768; i++)
    fscanf(fpin, "%d", &A[i]);
  fclose(fpin);

  for_any_pixel
      enable_mask[i][j] = 0;

  printf("\nInsert the maximum count value:");
  MAX = atoi(gets(buff));
/******* chiede di definire la zona per il calcolo di Vth ******/

  start_acq(RADIOGRAPHY);     /*start the acquisition */
  if (MISSING_MEDIPIX == 0)
    download_data();
                                    /**** il calcolo e' fatto ****/

  for_any_pixel
      matrix2[i][j] = A[data_matrix[i][j]];

  for_any_pixel {
      data_matrix[i][j] = matrix2[i][j];
      if (matrix2[i][j] < MAX)
        tot = tot + matrix2[i][j];
  }

  tot = tot / 4096;

  for_any_pixel
      if (matrix2[i][j] >= 2 * tot) {
        c++;
        enable_mask[i][j] = 1;
      }

  for_any_pixel
      config_matrix[i][j] = config_matrix[i][j] | enable_mask[i][j];

  Save_Mask();
  printf("\nAverage counts:%d", tot);
  printf("\nNoisy pixels:%d", c);
  printf("\nEnd of function ");
  wait_keypress();
}
/************** END ********************/


void wait_keypress(void)
{
  char c;
```

```
  while (c = getchar() == NULL);
}
```

V_test.c

```
#include <utility.h>
#include <formatio.h>
#include <ansi_c.h>
#include <string.h>
#include <userint.h>
#include "Interface.h"
#include "main.h"

extern int panelHandle;

void Close(void);

/*****************************************************************************
********************** inizio opzione "basic access register" ***************
*****************************************************************************/

void start_B_A(void);
unsigned * registro(int);

int basic_r_a;

void CVICALLBACK Bus_access (int menuBar, int menuItem, void *callbackData,
            int panel)
{
   start_B_A();
   DisplayPanel(basic_r_a);
}


int CVICALLBACK ModifyBit (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  ProcessSystemEvents ();

                  SetCtrlAttribute (basic_r_a, BASIC_R_A_MSG3,ATTR_VISIBLE, 1);
                  SetCtrlAttribute (basic_r_a, BASIC_R_A_MSG1,ATTR_VISIBLE, 0);
                  SetCtrlAttribute (basic_r_a, BASIC_R_A_MSG4,ATTR_VISIBLE, 0);

                  SetCtrlAttribute     (basic_r_a,     BASIC_R_A_VALUE_WRITTEN,
ATTR_DIMMED, 1);
                  SetCtrlAttribute       (basic_r_a,       BASIC_R_A_VALUE_READ,
ATTR_DIMMED, 1);
                  SetCtrlAttribute (basic_r_a, BASIC_R_A_RUN2, ATTR_VISIBLE, 0);
                  SetCtrlAttribute       (basic_r_a,       BASIC_R_A_WRITE_READ,
ATTR_DIMMED, 1);
                  SetCtrlAttribute (basic_r_a, BASIC_R_A_REGISTER, ATTR_DIMMED,
0);
                  SetCtrlAttribute      (basic_r_a,      BASIC_R_A_TURN_ON_OFF,
ATTR_DIMMED, 0);
                  SetCtrlAttribute (basic_r_a, BASIC_R_A_MASK, ATTR_DIMMED, 0);
```

```
                  SetCtrlAttribute (basic_r_a, BASIC_R_A_RUN, ATTR_VISIBLE, 1);

                  break;
      }
      return 0;
}

int CVICALLBACK ModifyWord (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int risul;

      switch (event) {
            case EVENT_COMMIT:
                  ProcessSystemEvents ();
                  GetCtrlVal(basic_r_a,BASIC_R_A_WRITE_READ,&risul);
                  if(risul==1)
                   {
                       SetCtrlAttribute   (basic_r_a,   BASIC_R_A_VALUE_WRITTEN,
ATTR_DIMMED, 0);
                     SetCtrlAttribute  (basic_r_a,  BASIC_R_A_MSG4,ATTR_VISIBLE,
1);
                     SetCtrlAttribute  (basic_r_a,  BASIC_R_A_MSG1,ATTR_VISIBLE,
0);
                     SetCtrlAttribute  (basic_r_a,  BASIC_R_A_MSG3,ATTR_VISIBLE,
0);
                   }
                  else
                  {
                        SetCtrlAttribute   (basic_r_a,   BASIC_R_A_VALUE_WRITTEN,
ATTR_DIMMED, 1);
                 SetCtrlAttribute (basic_r_a, BASIC_R_A_MSG6,ATTR_DIMMED, 0);
                  }
                      SetCtrlAttribute    (basic_r_a,    BASIC_R_A_TURN_ON_OFF,
ATTR_DIMMED, 1);
                   SetCtrlAttribute (basic_r_a, BASIC_R_A_MASK, ATTR_DIMMED, 1);
                   SetCtrlAttribute (basic_r_a, BASIC_R_A_RUN, ATTR_VISIBLE, 0);

                            SetCtrlAttribute   (basic_r_a,   BASIC_R_A_REGISTER,
ATTR_DIMMED,0);
                    SetCtrlAttribute  (basic_r_a,  BASIC_R_A_RUN2,  ATTR_VISIBLE,
1);
                       SetCtrlAttribute    (basic_r_a,     BASIC_R_A_WRITE_READ,
ATTR_DIMMED, 0);
                     SetCtrlAttribute    (basic_r_a,    BASIC_R_A_DECORATION_5,
ATTR_DIMMED, 0);
                  break;
      }
      return 0;
}

int CVICALLBACK run_only_bit (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
  int choice_bit_1,choice_bit_3;
  unsigned * address;
  unsigned int choice_bit_2;
```

```
  int status,shell,ok_button;

      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal (basic_r_a, BASIC_R_A_REGISTER, &choice_bit_1);
                  GetCtrlVal (basic_r_a, BASIC_R_A_TURN_ON_OFF, &choice_bit_2);
                  GetCtrlVal (basic_r_a, BASIC_R_A_MASK, &choice_bit_3);
                  address=registro(choice_bit_1);
                  modify_register(address,choice_bit_2,choice_bit_3,"W");
                  SetCtrlAttribute(basic_r_a,BASIC_R_A_MSG2, ATTR_VISIBLE, 1);
                  SetCtrlAttribute(basic_r_a,BASIC_R_A_MSG3, ATTR_VISIBLE, 0);
                  SetCtrlAttribute(basic_r_a,BASIC_R_A_BACK, ATTR_DIMMED, 0);
                  SetCtrlAttribute(basic_r_a,BASIC_R_A_BIT, ATTR_DIMMED, 1);
                  SetCtrlAttribute(basic_r_a,BASIC_R_A_WORD, ATTR_DIMMED, 1);

                  SetCtrlAttribute (basic_r_a, BASIC_R_A_RUN, ATTR_VISIBLE, 0);
                  SetCtrlAttribute (basic_r_a, BASIC_R_A_RUN2, ATTR_VISIBLE, 0);

                  do
                   status=GetUserEvent(1,&shell,&ok_button);
                  while(!((status==EVENT_COMMIT)&&(((shell==basic_r_a)&&

((ok_button==BASIC_R_A_BACK)||(ok_button==BASIC_R_A_QUIT)))||
                     ((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT))));
                  if(ok_button==BASIC_R_A_BACK)
                    start_B_A();
                  if(ok_button==BASIC_R_A_QUIT)
                    HidePanel(basic_r_a);
                  break;
      }
      return 0;
}

int CVICALLBACK run_word (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
 int choice_word_1, choice_word_2;
 unsigned int choice_word_3, read;
 unsigned int * address;
 int status,shell,ok_button;

      switch (event) {
            case EVENT_COMMIT:

                   GetCtrlVal (basic_r_a, BASIC_R_A_WRITE_READ, &choice_word_1);
                   SetCtrlAttribute(basic_r_a,BASIC_R_A_BACK, ATTR_DIMMED, 0);
                   SetCtrlAttribute(basic_r_a,BASIC_R_A_BIT, ATTR_DIMMED, 1);
                   SetCtrlAttribute(basic_r_a,BASIC_R_A_WORD, ATTR_DIMMED, 1);
                   if(choice_word_1==1)
                    {
                     GetCtrlVal (basic_r_a, BASIC_R_A_REGISTER, &choice_word_2);
                            GetCtrlVal   (basic_r_a,   BASIC_R_A_VALUE_WRITTEN,
&choice_word_3);
                     address=registro(choice_word_2);
                     write_data(address,&choice_word_3,1,"s");
                          SetCtrlAttribute   (basic_r_a,   BASIC_R_A_VALUE_READ,
ATTR_DIMMED, 0);
```

```
                    read_data(address,&read,1,"s");
                    SetCtrlVal (basic_r_a, BASIC_R_A_VALUE_READ, read);

                   }
                  else if(choice_word_1==0)
                   {
                         SetCtrlAttribute   (basic_r_a,BASIC_R_A_VALUE_READ   ,
ATTR_DIMMED, 1);
                       SetCtrlAttribute  (basic_r_a,  BASIC_R_A_VALUE_WRITTEN,
ATTR_DIMMED, 1);
                    GetCtrlVal (basic_r_a, BASIC_R_A_REGISTER, &choice_word_2);
                    address=registro(choice_word_2);
                    read_data(address,&read,1,"s");
                         SetCtrlAttribute   (basic_r_a,BASIC_R_A_VALUE_READ   ,
ATTR_DIMMED, 0);
                    SetCtrlVal (basic_r_a, BASIC_R_A_VALUE_READ, read);

                   }

                    SetCtrlAttribute  (basic_r_a,  BASIC_R_A_RUN,  ATTR_VISIBLE,
0);
                   SetCtrlAttribute (basic_r_a,  BASIC_R_A_RUN2,  ATTR_VISIBLE,
0);
                       SetCtrlAttribute   (basic_r_a,   BASIC_R_A_WRITE_READ,
ATTR_DIMMED, 1);
              SetCtrlAttribute (basic_r_a,  BASIC_R_A_DECORATION_5,  ATTR_DIMMED,
1);

                  do
                   status=GetUserEvent(1,&shell,&ok_button);
                 while(!((status==EVENT_COMMIT)&&(((shell==basic_r_a)&&

((ok_button==BASIC_R_A_BACK)||(ok_button==BASIC_R_A_QUIT)))||
                    ((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT)))));
                  if(ok_button==BASIC_R_A_BACK)
                    start_B_A();
                  if(ok_button==BASIC_R_A_QUIT)
                    HidePanel(basic_r_a);


            break;
      }
      return 0;
}


int CVICALLBACK ctrlWrite (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int test;
      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal (basic_r_a, BASIC_R_A_WRITE_READ, &test);
                  if(test==1)
                    {
                         SetCtrlAttribute  (basic_r_a,  BASIC_R_A_VALUE_WRITTEN,
ATTR_DIMMED, 0);
```

```
                        SetCtrlAttribute   (basic_r_a,   BASIC_R_A_VALUE_READ,
ATTR_DIMMED, 1);
                  }
                else if(test==0)
                      SetCtrlAttribute  (basic_r_a,  BASIC_R_A_VALUE_WRITTEN,
ATTR_DIMMED, 1);
                break;
      }
      return 0;
}


/*****************************************************************************
************************ inizio opzione "RAM or FIFO test" ********************
*****************************************************************************/


unsigned data_wr[4096], data_rd[4096]={0};
int ram_fifo;

void run_RAM_FIFO(void);
void start_R_F(void);
void error(int);
void erase_message(void);

void CVICALLBACK Ram_Fifo_test (int menuBar, int menuItem, void *callbackData,
            int panel)
{

 int i;

 for(i=0;i<4096;i++)
   data_wr[i]=i;

 start_R_F();
 DisplayPanel(ram_fifo);

}



int CVICALLBACK ram (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  erase_message();
                   SetCtrlAttribute (ram_fifo,  RAM_FIFO_RESULT6,  ATTR_VISIBLE,
1);
             SetCtrlAttribute(ram_fifo, RAM_FIFO_READ, ATTR_DIMMED, 0);
             SetCtrlAttribute(ram_fifo, RAM_FIFO_OFFSET, ATTR_DIMMED, 0);
                  SetCtrlAttribute (ram_fifo, RAM_FIFO_RAM, ATTR_DIMMED, 1);
                  SetCtrlAttribute (ram_fifo, RAM_FIFO_FIFO, ATTR_DIMMED, 1);
                  SetCtrlAttribute(ram_fifo, RAM_FIFO_BACK, ATTR_DIMMED, 0);
                   SetCtrlAttribute (ram_fifo,  RAM_FIFO_RUN_FIFO,  ATTR_VISIBLE,
0);
```

```
                    SetCtrlAttribute (ram_fifo, RAM_FIFO_RUN_RAM, ATTR_VISIBLE,
1);
                    run_RAM_FIFO();

                   break;
      }
      return 0;
}

int CVICALLBACK fifo (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  erase_message();
                  SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT7, ATTR_VISIBLE, 1);
            SetCtrlAttribute(ram_fifo, RAM_FIFO_OFFSET, ATTR_DIMMED, 1);
            SetCtrlAttribute(ram_fifo, RAM_FIFO_READ, ATTR_DIMMED, 0);
                  SetCtrlAttribute (ram_fifo, RAM_FIFO_RAM, ATTR_DIMMED, 1);
                SetCtrlAttribute (ram_fifo, RAM_FIFO_FIFO, ATTR_DIMMED, 1);
                SetCtrlAttribute(ram_fifo, RAM_FIFO_BACK, ATTR_DIMMED, 0);
                  SetCtrlAttribute (ram_fifo, RAM_FIFO_RUN_RAM, ATTR_VISIBLE,
0);
                  SetCtrlAttribute (ram_fifo, RAM_FIFO_RUN_FIFO, ATTR_VISIBLE,
1);
                  run_RAM_FIFO();
                  break;
      }
      return 0;
}

int CVICALLBACK QuitRF (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  HidePanel(ram_fifo);
                  SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_VISIBLE,0);
                  break;
      }
      return 0;
}


void run_RAM_FIFO(void)
{
 int status=0;
 int exit=1, panel, ok_button, i=0, n;
 unsigned int offs;
 char message[60];


 do
  status = GetUserEvent (1, &panel, &ok_button);
 while (!((status==EVENT_COMMIT)&&((panel==ram_fifo)&&
       ((ok_button==RAM_FIFO_RUN_RAM)||(ok_button==RAM_FIFO_RUN_FIFO)
       ||(ok_button==RAM_FIFO_QUIT)||(ok_button==RAM_FIFO_BACK))||
```

```
          ((panel==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT)))));

  if(ok_button==RAM_FIFO_RUN_RAM)
       {
                  SetCtrlAttribute(ram_fifo, RAM_FIFO_BACK, ATTR_DIMMED, 1);

                  GetCtrlVal (ram_fifo, RAM_FIFO_OFFSET, &offs);
                  GetCtrlVal (ram_fifo, RAM_FIFO_READ, &n);
                  if(n>2048)
                    {
                     erase_message();
                      SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT4, ATTR_VISIBLE,
1);
                      SetCtrlAttribute(ram_fifo, RAM_FIFO_RUN_RAM, ATTR_VISIBLE,
0);
                     SetCtrlAttribute (ram_fifo, RAM_FIFO_RAM, ATTR_DIMMED, 0);
                   SetCtrlAttribute (ram_fifo, RAM_FIFO_FIFO, ATTR_DIMMED, 0);
                   SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT,ATTR_VISIBLE,0);
                     SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_VISIBLE,1);

                    }
                  else
                   {
                         erase_message();
                              SetCtrlAttribute(ram_fifo,     RAM_FIFO_RESULT7_2,
ATTR_VISIBLE, 1);
                         SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT,ATTR_DIMMED,1);
                     SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_DIMMED,1);
                     ProcessDrawEvents();

                       write_data(config_ram+offs,data_wr,n,"m");
                       read_data(config_ram+offs,data_rd,n,"m");
                       do
                         i++;
                       while((data_rd[i-1]==data_wr[i-1])&&(i<n));

                       if(data_rd[i-1]!=data_wr[i-1])
                         {
                              sprintf(message, "RAM TEST FAILURE: %d datum is
wrong\n\n",i);
                          MessagePopup ("Error", message);
                          start_R_F();
                         }
                        else
                         {
                          erase_message();
                                   SetCtrlAttribute(ram_fifo,  RAM_FIFO_RESULT2,
ATTR_VISIBLE, 1);
                                   SetCtrlAttribute(ram_fifo,  RAM_FIFO_RUN_RAM,
ATTR_VISIBLE, 0);
                         }
                     SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT,ATTR_VISIBLE,0);
                     SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_VISIBLE,1);
                    }


       }
```

```
   if(ok_button==RAM_FIFO_RUN_FIFO)
       {
          SetCtrlAttribute(ram_fifo, RAM_FIFO_BACK, ATTR_DIMMED, 1);

          GetCtrlVal(ram_fifo, RAM_FIFO_READ, &n);
        if(n>=4096)
                  {
                   erase_message();
                    SetCtrlAttribute(ram_fifo,  RAM_FIFO_RESULT1,  ATTR_VISIBLE,
1);
                    SetCtrlAttribute(ram_fifo,  RAM_FIFO_RUN_FIFO,  ATTR_VISIBLE,
0);
                   SetCtrlAttribute (ram_fifo, RAM_FIFO_RAM, ATTR_DIMMED, 0);
                 SetCtrlAttribute (ram_fifo, RAM_FIFO_FIFO, ATTR_DIMMED, 0);
                 SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT,ATTR_VISIBLE,0);
                   SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_VISIBLE,1);
                  }
             else
             {

                erase_message();
                SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT7_2, ATTR_VISIBLE, 1);
                SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT,ATTR_DIMMED,1);
                SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_DIMMED,1);
                ProcessDrawEvents();

                write_data(data_fifo,data_wr,n,"s");
                read_data(data_fifo,data_rd,n,"s");
                do
                  i++;
                while((data_rd[i-1]==data_wr[i-1])&&(i<n));

                if (data_rd[i-1] != data_wr[i-1])
                     {
                          sprintf(message,  "FIFO  TEST  FAILURE:  %d  datum  is
wrong\n\n",i);
                      MessagePopup("Error", message);
                      start_R_F();
                     }
                else
                  {
               erase_message();
               SetCtrlAttribute(ram_fifo, RAM_FIFO_RESULT3, ATTR_VISIBLE, 1);
                     SetCtrlAttribute(ram_fifo,  RAM_FIFO_RUN_FIFO,  ATTR_VISIBLE,
0);
                  }
                SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT,ATTR_VISIBLE,0);
                SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_VISIBLE,1);
             }

       }
      SetCtrlAttribute (ram_fifo, RAM_FIFO_RAM, ATTR_DIMMED, 0);
    SetCtrlAttribute (ram_fifo, RAM_FIFO_FIFO, ATTR_DIMMED, 0);
    SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT,ATTR_DIMMED,0);
    SetCtrlAttribute(ram_fifo,RAM_FIFO_QUIT_RF,ATTR_DIMMED,0);

      if(ok_button==RAM_FIFO_QUIT)
```

```
        HidePanel(ram_fifo);

    if(ok_button==RAM_FIFO_BACK)
      start_R_F();

}
/*****************************************************************************
********** fine RAM or FIFO text ********************************************
*****************************************************************************/




/*****************************************************************************
*************** inizio TEST COUNTERS*****************************************
*****************************************************************************/

void gdisplay_matrix_values(int [64][64],char *);
void Get_Pattern(void);
void Read_File_Pattern(char *, int[64][64]);
void Save_Pattern(void);
void Modify_Pattern(void);
void Do_Test(void);
void save_files(void);
int Open_Read_File(char *);
void start (void);

int Test_Counters;

/****************************** CVICALLBACK FUNCTIONS *********************/
void CVICALLBACK Counter_Test (int menuBar, int menuItem, void *callbackData,
            int panel)
{
  start();
  DisplayPanel(Test_Counters);
  SetCtrlAttribute (Test_Counters, COUNTERS_SAVE, ATTR_DIMMED, 1);
  SetCtrlAttribute (Test_Counters, COUNTERS_SHOW, ATTR_DIMMED, 1);
  SetCtrlAttribute (Test_Counters, COUNTERS_GO_TEST, ATTR_DIMMED, 1);
}



int CVICALLBACK Create (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int status, shell, ok_button;

      switch (event) {
            case EVENT_COMMIT:

             SetCtrlAttribute  (Test_Counters,  COUNTERS_RUN_SAVE,  ATTR_VISIBLE,
0);
             SetCtrlAttribute  (Test_Counters,  COUNTERS_RUN_LOAD,  ATTR_VISIBLE,
0);

             SetCtrlAttribute (Test_Counters, COUNTERS_SAVE, ATTR_DIMMED, 1);
```

```
            SetCtrlAttribute (Test_Counters, COUNTERS_LOAD, ATTR_DIMMED, 1);
            SetCtrlAttribute (Test_Counters, COUNTERS_CREATE, ATTR_DIMMED, 1);
            SetCtrlAttribute (Test_Counters, COUNTERS_GO_TEST, ATTR_DIMMED, 1);

            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG1,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG2,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG3,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG4,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG5,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG6,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG7,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG9,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_FILE_NAME,ATTR_VISIBLE,0);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_4,ATTR_VISIBLE,0);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_2,ATTR_VISIBLE,0);

                    SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_CREATE,
ATTR_VISIBLE, 1);
                    SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_CREATE,
ATTR_DIMMED,0);
                   SetCtrlAttribute   (Test_Counters,   COUNTERS_START_COLUMN,
ATTR_VISIBLE, 1);
                    SetCtrlAttribute    (Test_Counters,    COUNTERS_END_COLUMN,
ATTR_VISIBLE, 1);
                     SetCtrlAttribute    (Test_Counters,    COUNTERS_START_ROW,
ATTR_VISIBLE, 1);
                      SetCtrlAttribute     (Test_Counters,     COUNTERS_END_ROW,
ATTR_VISIBLE, 1);
                       SetCtrlAttribute     (Test_Counters,      COUNTERS_VALUE,
ATTR_VISIBLE, 1);

                  SetCtrlAttribute (Test_Counters, COUNTERS_BACK, ATTR_VISIBLE,
1);
                  SetCtrlAttribute (Test_Counters, COUNTERS_QUIT, ATTR_VISIBLE,
0);

                  do
                    status = GetUserEvent (1, &shell, &ok_button);
                  while(!((status==EVENT_COMMIT)&&((shell==Test_Counters)&&

((ok_button==COUNTERS_RUN_CREATE)||(ok_button==COUNTERS_BACK))

||((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT)))));
                  if (ok_button==COUNTERS_BACK)

                     start();

                  else if(ok_button==COUNTERS_RUN_CREATE)
                    {
                         SetCtrlAttribute  (Test_Counters,  COUNTERS_RUN_CREATE,
ATTR_DIMMED,1);
                      Modify_Pattern();
                      start();
                             SetCtrlAttribute   (Test_Counters,   COUNTERS_SAVE,
ATTR_DIMMED,0);
```

```
                              SetCtrlAttribute  (Test_Counters,  COUNTERS_SHOW,
ATTR_DIMMED,0);
                        }

                    break;
      }
      return 0;
}


int CVICALLBACK Load (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
  int status, shell, ok_button;


      switch (event) {
            case EVENT_COMMIT:

                        SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_SAVE,
ATTR_VISIBLE, 0);
                       SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_CREATE,
ATTR_VISIBLE, 0);
                        SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_LOAD,
ATTR_DIMMED, 0);

             SetCtrlAttribute (Test_Counters, COUNTERS_LOAD, ATTR_DIMMED, 1);
             SetCtrlAttribute (Test_Counters, COUNTERS_CREATE, ATTR_DIMMED, 1);


SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG1,ATTR_VISIBLE,0);
             SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG2,ATTR_VISIBLE,0);
             SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG3,ATTR_VISIBLE,0);
             SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG4,ATTR_VISIBLE,0);
             SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG5,ATTR_VISIBLE,0);
             SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG7,ATTR_VISIBLE,0);
             SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG9,ATTR_VISIBLE,0);

                       SetCtrlAttribute   (Test_Counters,   COUNTERS_START_COLUMN,
ATTR_VISIBLE, 0);
                        SetCtrlAttribute    (Test_Counters,    COUNTERS_END_COLUMN,
ATTR_VISIBLE, 0);
                        SetCtrlAttribute    (Test_Counters,    COUNTERS_START_ROW,
ATTR_VISIBLE, 0);
                         SetCtrlAttribute    (Test_Counters,    COUNTERS_END_ROW,
ATTR_VISIBLE, 0);
                          SetCtrlAttribute     (Test_Counters,     COUNTERS_VALUE,
ATTR_VISIBLE, 0);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_3,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_2,ATTR_VISIBLE,1);


SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG6,ATTR_VISIBLE,1);
                  SetCtrlVal(Test_Counters, COUNTERS_FILE_NAME,"*.pat");
```

```
                        SetCtrlAttribute    (Test_Counters,    COUNTERS_FILE_NAME,
ATTR_VISIBLE, 1);
                         SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_LOAD,
ATTR_VISIBLE, 1);

                     SetCtrlAttribute (Test_Counters, COUNTERS_BACK, ATTR_VISIBLE,
1);
                     SetCtrlAttribute (Test_Counters, COUNTERS_QUIT, ATTR_VISIBLE,
0);

                     do
                      status=GetUserEvent(1,&shell,&ok_button);
                     while(!((status==EVENT_COMMIT)&&((shell==Test_Counters)&&

(ok_button==COUNTERS_RUN_LOAD)||(ok_button==COUNTERS_BACK))

||((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT))));
                     if(ok_button==COUNTERS_BACK)
                        {
                         start();
                                SetCtrlAttribute  (Test_Counters,  COUNTERS_SAVE,
ATTR_DIMMED, 1);
                   SetCtrlAttribute  (Test_Counters,  COUNTERS_SHOW,  ATTR_DIMMED,
1);
                   SetCtrlAttribute (Test_Counters, COUNTERS_GO_TEST, ATTR_DIMMED,
1);
                  }
                     else if(ok_button==COUNTERS_RUN_LOAD)
                       {
                        Get_Pattern();
                        start();

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG7,ATTR_VISIBLE,0);

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG9,ATTR_VISIBLE,1);
                 }

                    break;
      }
      return 0;
}


int CVICALLBACK Save (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
   int status, shell, ok_button;


      switch (event) {
            case EVENT_COMMIT:

                        SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_CREATE,
ATTR_VISIBLE, 0);
             SetCtrlAttribute  (Test_Counters,  COUNTERS_RUN_LOAD,  ATTR_VISIBLE,
0);
```

```
            SetCtrlAttribute (Test_Counters, COUNTERS_SAVE, ATTR_DIMMED, 0);
            SetCtrlAttribute (Test_Counters, COUNTERS_LOAD, ATTR_DIMMED, 1);
            SetCtrlAttribute (Test_Counters, COUNTERS_CREATE, ATTR_DIMMED, 1);
            SetCtrlAttribute (Test_Counters, COUNTERS_GO_TEST, ATTR_DIMMED, 1);

            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG1,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG2,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG3,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG4,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG6,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG7,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG9,ATTR_VISIBLE,0);

                     SetCtrlAttribute   (Test_Counters,   COUNTERS_START_COLUMN,
ATTR_VISIBLE, 0);
                      SetCtrlAttribute    (Test_Counters,    COUNTERS_END_COLUMN,
ATTR_VISIBLE, 0);
                       SetCtrlAttribute    (Test_Counters,    COUNTERS_START_ROW,
ATTR_VISIBLE, 0);
                        SetCtrlAttribute     (Test_Counters,     COUNTERS_END_ROW,
ATTR_VISIBLE, 0);
                         SetCtrlAttribute      (Test_Counters,      COUNTERS_VALUE,
ATTR_VISIBLE, 0);
                       SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_SAVE,
ATTR_VISIBLE, 1);
                       SetCtrlAttribute    (Test_Counters,    COUNTERS_RUN_SAVE,
ATTR_DIMMED, 0);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_2,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_3,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_4,ATTR_VISIBLE,1);
                                  SetCtrlAttribute               (Test_Counters,
COUNTERS_TEXTMSG5,ATTR_VISIBLE,1);
                  SetCtrlVal(Test_Counters, COUNTERS_FILE_NAME,".pat");
                     SetCtrlAttribute    (Test_Counters,    COUNTERS_FILE_NAME,
ATTR_VISIBLE, 1);

                  SetCtrlAttribute (Test_Counters, COUNTERS_BACK, ATTR_VISIBLE,
1);
                  SetCtrlAttribute (Test_Counters, COUNTERS_QUIT, ATTR_VISIBLE,
0);

                  do
                    status=GetUserEvent(1,&shell,&ok_button);
                  while(!((status==EVENT_COMMIT)&&((shell==Test_Counters)&&

(ok_button==COUNTERS_RUN_SAVE)||(ok_button==COUNTERS_BACK))

||((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT))));
                  if(ok_button==COUNTERS_BACK)
                     start();
                  else if(ok_button==COUNTERS_RUN_SAVE)
                   {
                    Save_Pattern();
                    start();
```

```
            }

                break;
      }
      return 0;
}


char select_file[25];

int CVICALLBACK Show (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:

                  gdisplay_matrix_values(pattern_matrix,select_file);

                  break;
      }
      return 0;
}


int CVICALLBACK Go_Test (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int status,ok_button,shell;
      switch (event) {
            case EVENT_COMMIT:

                   start();

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG10,ATTR_VISIBLE,1);
                   SetCtrlAttribute(Test_Counters,COUNTERS_BACK,ATTR_VISIBLE,1);
                   SetCtrlAttribute(Test_Counters,COUNTERS_QUIT,ATTR_VISIBLE,0);
                   SetCtrlAttribute(Test_Counters,COUNTERS_LOAD,ATTR_DIMMED,1);
                   SetCtrlAttribute(Test_Counters,COUNTERS_SAVE,ATTR_DIMMED,1);

SetCtrlAttribute(Test_Counters,COUNTERS_CREATE,ATTR_DIMMED,1);
                   SetCtrlAttribute(Test_Counters,COUNTERS_SHOW,ATTR_DIMMED,1);

SetCtrlAttribute(Test_Counters,COUNTERS_GO_TEST,ATTR_DIMMED,1);
                   SetCtrlAttribute(Test_Counters,COUNTERS_BACK,ATTR_DIMMED,1);
                   ProcessDrawEvents();
                   Do_Test();

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG10,ATTR_VISIBLE,0);
                   SetCtrlAttribute(Test_Counters,COUNTERS_BACK,ATTR_DIMMED,0);

                   do
                     status=GetUserEvent(1,&shell,&ok_button);
                   while(!((status==EVENT_COMMIT)&&(shell=Test_Counters)&&

((ok_button==COUNTERS_BACK)||(ok_button==PANEL_TOTAL_QUIT))));
                   start();
            break;
```

```
      }
      return 0;
}
/*****************************************************************************
*****/


void Modify_Pattern(void)
{
      int C1,C2,L1,L2,choice,i,j,k;

      GetCtrlVal (Test_Counters, COUNTERS_VALUE, &choice);
      GetCtrlVal (Test_Counters, COUNTERS_START_ROW, &L1);
      GetCtrlVal (Test_Counters, COUNTERS_END_ROW, &L2);

      GetCtrlVal (Test_Counters, COUNTERS_START_COLUMN, &C1);
      GetCtrlVal (Test_Counters, COUNTERS_END_COLUMN, &C2);
      if(L2<L1)
        {
            SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG1,ATTR_VISIBLE,1);
            SetCtrlAttribute(Test_Counters,
COUNTERS_DECORATION_4,ATTR_VISIBLE,1);

      SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_2,ATTR_VISIBLE,1);
        }
      else if(C2<C1)
        {
          SetCtrlAttribute(Test_Counters, COUNTERS_TEXTMSG2,ATTR_VISIBLE,1);
          SetCtrlAttribute(Test_Counters, COUNTERS_DECORATION_4,ATTR_VISIBLE,1);
          SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_2,ATTR_VISIBLE,1);
        }
        else
          {
           for(k=0;k<16;k++)
             {
                for(i=L1;i<=L2;i++)               /*   changes   only   selected
_bit_pattern*/
                     {
                          for(j=C1;j<=C2;j++)
                                    pattern_matrix[i][j+(k*4)]=choice;

                     }
             }
          }
}



void Get_Pattern(void)
{
      GetCtrlVal (Test_Counters, COUNTERS_FILE_NAME, select_file);
      Read_File_Pattern(select_file, pattern_matrix);
}



void Read_File_Pattern(char *filename,int mat[64][64])
```

```
{
      char buff[50];
    int i=0,j=0,k=0;

  if(Open_Read_File(filename)==0)
      {

         for(i=0;i<64;i++)
            {

                  for(j=0;j<4;j++)
                  {
                        fscanf(fpin,"%d",&mat[i][j]);

                         for(k=1;k<16;k++)
                        mat[i][j+(k*4)]=mat[i][j];
                  }
            }
        fclose(fpin);
    }

  else
    error(1);

}


void Save_Pattern(void)
{
  int i,j,k;
  char pathname[512];

  GetCtrlVal (Test_Counters, COUNTERS_FILE_NAME, select_file);
  if(!open_write_file(select_file, SAVE_WITH_CONTROL)) {
      for(i=0;i<64;i++) {
        for(j=0;j<4;j++)
            fprintf(fpout,"%d ",pattern_matrix[i][j]);
      fprintf(fpout,"\n");
      }
      fclose(fpout);

  }

}


void Do_Test(void)
{
       int i,j,cycles_dummy,err=0,i1,j1,c;

       cycles_dummy=cycles_tmp;
       cycles_tmp=1;

       if(check_power()!=1)
            error(2);
       else
         {
```

```
            for(j=3;j>=0;j--)
//for(j=0;j>=0;j--)
              {
                  for(i=63;i>=0;i--)
//for(i=0;i>=0;i--)
                  {
                        cycles_tmp=64*j+1;
                        reset_and_setup();
                        reset_counters((unsigned int)pattern_matrix[i][j]);
                    }
              }
            cycles_tmp=cycles_dummy;
            if(MISSING_MEDIPIX==0)download_data();


            for(i=0;i<64;i++)
               {
                 for(j=0;j<64;j++)
                    {
                       if(pattern_matrix[i][j]!=data_matrix[i][j])
                         err++;
                        }
                 }


              if(err!=0)
                  {
                   save_files();

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_4,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_2,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG3,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG7,ATTR_VISIBLE,0);

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG10,ATTR_VISIBLE,0);

                  }


              if(err==0)
                  {
                   save_files();

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_4,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_DECORATION_2,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG4,ATTR_VISIBLE,1);

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG7,ATTR_VISIBLE,0);

SetCtrlAttribute(Test_Counters,COUNTERS_TEXTMSG10,ATTR_VISIBLE,0);
                  }
```

```
           SetCtrlAttribute (Test_Counters, COUNTERS_DECORATION_5, ATTR_VISIBLE,
1);
         SetCtrlAttribute (Test_Counters, COUNTERS_VIEW, ATTR_VISIBLE, 1);
         SetCtrlAttribute (Test_Counters, COUNTERS_TEXTMSG, ATTR_VISIBLE, 1);


          }


}
/*****************************************************************************
******************** fine TEST COUNTER*****************************************
*****************************************************************************/



/*****************************************************************************
*************************** inizio TEST MASK ********************************
*****************************************************************************/


int Get_Mask(void);
int Read_File_Mask(char *,int [64][64],int [64][64],int [64][64]);
void save_matrix(void);
void Run_Test_Mask(void);
void get_array(int[64][64]);
int Load_Mask(void);

int Test_Mask;

int i,j,threshold_mask[64][64],testbit_mask[64][64],enable_mask[64][64];
int c_m_o[64][64];



/*********************  CALLBACK FUNCTIONS  *********************************/

void CVICALLBACK Mask_Test (int menuBar, int menuItem, void *callbackData,
            int panel)
{

  DisplayPanel(Test_Mask);
  Run_Test_Mask();
}

int CVICALLBACK Threshold (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  display_matrix_values(threshold_mask,"threshold_mask");
                  break;
      }
      return 0;
}

int CVICALLBACK Testbit (int panel, int control, int event,
```

```
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                  display_matrix_values(testbit_mask,"testbit_mask");
                  break;
      }
      return 0;
}

int CVICALLBACK Enable (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                   display_matrix_values(enable_mask,"enable_mask");
                  break;
      }
      return 0;
}



int CVICALLBACK go_test (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{


int status,shell,ok_button;


int i,ii,a,j,k,kk,x=0,err=0;
int offs=1024;
unsigned int config_input[1024],config_output[1024],cycles_dummy,ram_dummy;
char SF[20];

switch (event)
 {
  case EVENT_COMMIT:

   SetCtrlAttribute(Test_Mask,MASK_RUN,ATTR_DIMMED,1);
   ProcessDrawEvents();

   if(check_power()!=1)
           {
            error(2);
            SetCtrlAttribute(Test_Mask,MASK_BACK2,ATTR_DIMMED,0);
           }
   else
   {

   if (Load_Mask())
    {
       ram_dummy=ram_offset_tmp;
       ram_offset_tmp=offs;
       reset_and_setup();
       ram_offset_tmp=0;
```

```
       modify_register(status_reg ,turn_on,bit_config_rd,"W");

       while(modify_register(status_reg,0,bit_config_rd,"R")!=0);


       read_data(config_ram+offs,config_output,1024,"m");

     for(k=0;k<4;k++)
        {
            for(i=0;i<64;i++)
            {
                  for(kk=0;kk<4;kk++)
                  {
                        c_m_o[i][kk*4+k]=(config_output[x]&255) & 0x1f;
                        c_m_o[i][(4+kk)*4+k]=((config_output[x]>>8)&255) & 0x1f;
                        c_m_o[i][(8+kk)*4+k]=((config_output[x]>>16)&255)      &
0x1f;
                        c_m_o[i][(12+kk)*4+k]=((config_output[x]>>24)&255)     &
0x1f;
                        x++;
                  }
            }
       }

     for(i=0;i<64;i++)
        {
         for(j=0;j<64;j++)
            {
                  if(c_m_o[i][j]!=config_matrix[i][j])
                    {
                      err++;
                        if(BAD_RAM == 1 && i==0 && (j==0||j==16||j==32||j==48))
                              err--;

                    }
              }
        }


     if(err==0)
        {
          save_matrix();
          SetCtrlAttribute(Test_Mask,MASK_MSG3,ATTR_VISIBLE,1);
          SetCtrlAttribute(Test_Mask,MASK_MSG1,ATTR_VISIBLE,0);
          SetCtrlAttribute(Test_Mask,MASK_MSG2,ATTR_VISIBLE,0);
          SetCtrlAttribute(Test_Mask,MASK_MSG4,ATTR_VISIBLE,0);
        }



     if(err!=0)
        {
          save_matrix();
          SetCtrlAttribute(Test_Mask,MASK_MSG4,ATTR_VISIBLE,1);
            SetCtrlAttribute(Test_Mask,MASK_MSG1,ATTR_VISIBLE,0);
            SetCtrlAttribute(Test_Mask,MASK_MSG2,ATTR_VISIBLE,0);
```

```
            SetCtrlAttribute(Test_Mask,MASK_MSG3,ATTR_VISIBLE,0);

        }
      }
   }
      SetCtrlAttribute(Test_Mask,MASK_BACK2,ATTR_DIMMED,0);
      SetCtrlAttribute(Test_Mask,MASK_QUIT,ATTR_VISIBLE,1);
      SetCtrlAttribute(Test_Mask,MASK_QUIT_2,ATTR_VISIBLE,0);

      SetCtrlAttribute (Test_Mask, MASK_DECORATION_5, ATTR_VISIBLE, 1);
    SetCtrlAttribute (Test_Mask, MASK_VIEW, ATTR_VISIBLE, 1);
    SetCtrlAttribute (Test_Mask, MASK_TEXTMSG, ATTR_VISIBLE, 1);

      do
       status=GetUserEvent(1,&shell,&ok_button);
      while(!((status==EVENT_COMMIT)&&((shell==Test_Mask)&&
           ((ok_button==MASK_BACK2)||(ok_button==MASK_QUIT))||
           ((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT)))));

      if(ok_button==MASK_QUIT)
        HidePanel(Test_Mask);

      else if(ok_button==MASK_BACK2)
        Run_Test_Mask();

    else if (ok_button==PANEL_TOTAL_QUIT)
      Close();
  break;
 }

return(0);
}
/******************** fine CALLBACK  ****************************************/



void Run_Test_Mask (void)
{

  int status,shell,ok_button;

  SetCtrlVal(Test_Mask,MASK_FILENAME,"*.msk");
  SetCtrlAttribute (Test_Mask, MASK_DECORATION_5, ATTR_VISIBLE, 0);
  SetCtrlAttribute (Test_Mask, MASK_VIEW, ATTR_VISIBLE, 0);
  SetCtrlAttribute (Test_Mask, MASK_TEXTMSG, ATTR_VISIBLE, 0);
  SetCtrlAttribute(Test_Mask,MASK_RUN,ATTR_DIMMED,0);
  SetCtrlAttribute(Test_Mask,MASK_MSG1,ATTR_VISIBLE,1);
  SetCtrlAttribute(Test_Mask,MASK_MSG2,ATTR_VISIBLE,0);
  SetCtrlAttribute(Test_Mask,MASK_MSG3,ATTR_VISIBLE,0);
  SetCtrlAttribute(Test_Mask,MASK_MSG4,ATTR_VISIBLE,0);
  SetCtrlAttribute(Test_Mask,MASK_QUIT,ATTR_VISIBLE,1);
  SetCtrlAttribute(Test_Mask,MASK_QUIT_2,ATTR_VISIBLE,0);
  SetCtrlAttribute(Test_Mask,MASK_BACK2,ATTR_DIMMED,1);
  SetCtrlAttribute(Test_Mask,MASK_GO_TEST,ATTR_DIMMED,1);
  SetCtrlAttribute(Test_Mask,MASK_ENABLE,ATTR_DIMMED,1);
  SetCtrlAttribute(Test_Mask,MASK_TESTBIT,ATTR_DIMMED,1);
  SetCtrlAttribute(Test_Mask,MASK_THRESHOLD,ATTR_DIMMED,1);
```

```
  do
   status=GetUserEvent(1,&shell,&ok_button);
  while(!((status==EVENT_COMMIT)&&((shell==Test_Mask)&&
           ((ok_button==MASK_QUIT)||(ok_button==MASK_RUN))||
           ((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT)))));

  if(ok_button==MASK_QUIT)
     HidePanel(Test_Mask);

  else if(ok_button==MASK_RUN)
     {
      if (Get_Mask())
        {
         SetCtrlAttribute(Test_Mask,MASK_MSG1,ATTR_VISIBLE,0);
         SetCtrlAttribute(Test_Mask,MASK_MSG2,ATTR_VISIBLE,1);
           SetCtrlAttribute(Test_Mask,MASK_ENABLE,ATTR_DIMMED,0);
           SetCtrlAttribute(Test_Mask,MASK_TESTBIT,ATTR_DIMMED,0);
           SetCtrlAttribute(Test_Mask,MASK_THRESHOLD,ATTR_DIMMED,0);
           SetCtrlAttribute(Test_Mask,MASK_QUIT,ATTR_VISIBLE,0);
           SetCtrlAttribute(Test_Mask,MASK_QUIT_2,ATTR_VISIBLE,1);
           SetCtrlAttribute(Test_Mask,MASK_GO_TEST,ATTR_DIMMED,0);
          }
        else
          Run_Test_Mask();
       }
}

void save_matrix(void)
{
    int i,j;

      open_write_file("mask.error",SAVE_WITHOUT_CONTROL);

      fprintf(fpout,"/******** mask matrix (written) *******/\n");

      for(i=0;i<64;i++)
        {
         for(j=0;j<64;j++)
           {
              fprintf(fpout,"%x ",config_matrix[i][j]);
            }
         fprintf(fpout,"\n");
        }

      fprintf(fpout,"\n/******** mask matrix (read) *******/\n");

      for(i=0;i<64;i++)
        {
         for(j=0;j<64;j++)
           {
              fprintf(fpout,"%x ",c_m_o[i][j]);
            }
         fprintf(fpout,"\n");
        }
```

```
      fclose(fpout);

}


int Get_Mask(void)
{

  char SF[20];
  char maskfile[80];
  int result;

  GetCtrlVal(Test_Mask,MASK_FILENAME,SF);
  result = CompareStrings ("*.msk", 0, SF, 0, 0);
  if(result==0)
    {
       error(4);
       return(0);
    }

  sprintf(maskfile,"%s%s",PATHF,SF);

  if(Read_File_Mask(maskfile,enable_mask,testbit_mask,threshold_mask)==0)
      {
      for(i=0;i<64;i++)
            {
         for(j=0;j<64;j++)
           {
                  config_matrix[i][j]=((threshold_mask[i][j]&1)<<4)+
                                      ((threshold_mask[i][j]&2)<<2)+
                                                ((threshold_mask[i][j]&4))+

      (testbit_mask[i][j]<<1)+enable_mask[i][j];
           }
        }



       return(1);
    }

    else
     return(0);

}

/*****************************************************************************
************************* fine TEST MASK ************************************
*****************************************************************************/


/*****************************************************************************
******************** inizio ADD_SUBTRACT ************************************
*****************************************************************************/

int Add_Subt_Files;
int control_1=0, control_2=0;
```

```
void failure(void);

void CVICALLBACK add_subt (int menuBar, int menuItem, void *callbackData,
            int panel)
{

   ResetTextBox(Add_Subt_Files,ADD_SUBT_TEXTBOX,"");
   InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"");
   InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"Enter origin file names
( I, II ),");
   InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"target    file,    two
multiplication");
   InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"factors   and   select
Add");
   InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"or Subt.");
   InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"");
   InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"Then   press   the   Run
button.");

   SetCtrlAttribute(Add_Subt_Files,ADD_SUBT_CHECKADD,ATTR_DIMMED,0);
   SetCtrlAttribute(Add_Subt_Files,ADD_SUBT_CHECKSUBT,ATTR_DIMMED,0);

   SetCtrlVal(Add_Subt_Files,ADD_SUBT_CHECKADD,0);
   SetCtrlVal(Add_Subt_Files,ADD_SUBT_CHECKSUBT,0);

   DisplayPanel(Add_Subt_Files);

}



int CVICALLBACK Add_Subt (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{

    char fil1[50],fil2[50],fil3[50];
    char fil10[200],fil20[200],fil30[200];
    int count1[64][64],count2[64][64], count3[64][64];
    int i,j,fact1=1,fact2=1, ctrl_1=0, ctrl_2=0;
      int status1,status2;
      char message[30];

      switch (event) {
            case EVENT_COMMIT:


          GetCtrlVal(Add_Subt_Files,ADD_SUBT_CHECKADD,&ctrl_1);
            GetCtrlVal(Add_Subt_Files,ADD_SUBT_CHECKSUBT,&ctrl_2);
            if (!(ctrl_1||ctrl_2))
              {
               error(3);
               return 0;
              }
          GetCtrlVal(Add_Subt_Files,ADD_SUBT_FILENAME1,fil1);
          GetCtrlVal(Add_Subt_Files,ADD_SUBT_FILENAME2,fil2);
          GetCtrlVal(Add_Subt_Files,ADD_SUBT_FILENAME3,fil3);
```

```
          GetCtrlVal(Add_Subt_Files,ADD_SUBT_FACTOR1,&fact1);
          GetCtrlVal(Add_Subt_Files,ADD_SUBT_FACTOR2,&fact2);


            sprintf(fil10,"%s%s",PATHF2,fil1);
            sprintf(fil20,"%s%s",PATHF2,fil2);
            sprintf(fil30,"%s%s",PATHF2,fil3);

          status1=FileToArray (fil10, count1, VAL_INTEGER, 4096, 64,
                               VAL_GROUPS_TOGETHER,        VAL_GROUPS_AS_COLUMNS,
VAL_ASCII);

          if(status1==-1)
            {
             sprintf(message,"%s%s",fil10," NOT FOUND !");
             MessagePopup("Error!",message);
             failure();
             return 0;
            }

          status2=FileToArray (fil20, count2, VAL_INTEGER, 4096, 64,
                               VAL_GROUPS_TOGETHER,        VAL_GROUPS_AS_COLUMNS,
VAL_ASCII);

          if(status2==-1)
            {
             sprintf(message,"%s%s",fil20," NOT FOUND !");
             MessagePopup("Error",message);
             failure();
             return 0;
            }



          ResetTextBox(Add_Subt_Files,ADD_SUBT_TEXTBOX,"");
          for(i=1; i<5;i++)
            InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"");
          InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"
RUNNING !");
            ProcessDrawEvents();


          if (control_1==1)
            {
             for(i=0;i<64;i++)
             for(j=0;j<64;j++)
               {
                                    count3[i][j]   =   (fact1*count1[i][j])   +
(fact2*count2[i][j]);
                        if (count3[i][j] < 0)
                            count3[i][j]=0;
                    }
            }
          if (control_2==1)
              {
               for(i=0;i<64;i++)
             for(j=0;j<64;j++)
```

```
                {
                 count3[i][j] = (fact1*count1[i][j]) - (fact2*count2[i][j]);
                        if (count3[i][j] < 0)
                            count3[i][j]=0;
                     }


                }

          ArrayToFile (fil30, count3, VAL_UNSIGNED_INTEGER, 4096,
                        64, VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
                            VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_TRUNCATE);


          ResetTextBox(Add_Subt_Files,ADD_SUBT_TEXTBOX,"");
          InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"");
          InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"");
          InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"");
          InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"     End of 'Add
or Subt files'.");


      break;
      }
  return 0;
}



int CVICALLBACK dimmed_checkAdd (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int ctrl_1;

      switch (event) {
            case EVENT_COMMIT:
                  control_1=0;
                  control_2=0;
                  GetCtrlVal(Add_Subt_Files,ADD_SUBT_CHECKADD,&ctrl_1);
                  if(ctrl_1==0)
                    {

      SetCtrlAttribute(Add_Subt_Files,ADD_SUBT_CHECKSUBT,ATTR_DIMMED,0);
                        control_1=0;

                    }
                  else
                    {

SetCtrlAttribute(Add_Subt_Files,ADD_SUBT_CHECKSUBT,ATTR_DIMMED,1);
                     control_1=1;
                    }

                 break;
      }
      return 0;
}
```

```
int CVICALLBACK dimmed_checkSubt (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int ctrl_2;

      switch (event) {
            case EVENT_COMMIT:
                     control_1=0;
                     control_2=0;
                     GetCtrlVal(Add_Subt_Files,ADD_SUBT_CHECKSUBT,&ctrl_2);
                     if(ctrl_2==0)
                       {

SetCtrlAttribute(Add_Subt_Files,ADD_SUBT_CHECKADD,ATTR_DIMMED,0);
                        control_2=0;
                       }
                     else
                       {

SetCtrlAttribute(Add_Subt_Files,ADD_SUBT_CHECKADD,ATTR_DIMMED,1);
                         control_2=1;
                        }
                  break;
      }
      return 0;
}


void failure(void)
{
     ResetTextBox(Add_Subt_Files,ADD_SUBT_TEXTBOX,"");
     for(i=1;i<4;i++)
       InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"");
     InsertTextBoxLine(Add_Subt_Files,ADD_SUBT_TEXTBOX,-1,"
FAILURE !");
       ProcessDrawEvents();
}
/*****************************************************************************
*****************************************************************************
*****************************************************************************/


/*****************************************************************************
***************************** inizio Enable Mask   ***************************
*****************************************************************************/

int E_Mask, controll;

void start_E_M(void);
void check(void);
void InputValues(int);
void savemask(char *);
```

```
int load_mask_file(void);

void CVICALLBACK Enable_Mask (int menuBar, int menuItem, void *callbackData,
                              int panel)
{

  if (TIME_UNIT==0)      // Unita di misura: millisecondi
      SetCtrlAttribute (E_Mask, ENABLE_NUMERIC4, ATTR_LABEL_TEXT,
                        "Acq. time (ms)");
   else                        // Unita di misura: secondi
      SetCtrlAttribute (E_Mask, ENABLE_NUMERIC4, ATTR_LABEL_TEXT,
                                "Acq. time (s)");

  start_E_M();
  DisplayPanel(E_Mask);

}


int CVICALLBACK Best_Vthr (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                    controll=2;
                    check();
                    SetCtrlAttribute(E_Mask, ENABLE_NUMERIC6, ATTR_VISIBLE, 1);

               ProcessSystemEvents();

                  break;
      }
      return 0;
}

int CVICALLBACK Vthr (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                    controll=1;
                    check();
                    SetCtrlAttribute(E_Mask, ENABLE_NUMERIC7, ATTR_VISIBLE, 1);

                    ProcessSystemEvents();


                  break;
      }
      return 0;
}

char file[30]={0};

int CVICALLBACK Run_E_Mask (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
```

```
       int i,j,ii,c=0,result;
     int A[32800], CHIP, MAXC;
     int matrix2[64][64],enable_mask[64][64];
       char res1[20], res2[20];// file[30]={0};
       float Vth=0, PERC=0;


      switch (event) {
            case EVENT_COMMIT:

GetCtrlVal(E_Mask,ENABLE_FILEMASK,file);
result = CompareStrings ("*.msk", 0, file, 0, 0);
if(result==0)
  {
      error(4);
      return(0);
  }
if(check_power()!=1)
            error(2);

else
 {
            ResetTextBox(E_Mask,ENABLE_TEXTBOX,"");
                              InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,"\n\n\n
RUNNING");
          ProcessDrawEvents();

        A[0]=696969;    /* unrealistic number to be read if counters output
        pseudorandom number 0 (impossible) */

        open_read_file("LUT_real.lut");

      for (i=1;i<32768;i++)
        {
                fscanf(fpin,"%d",&A[i]);
        }
        fclose(fpin);

    CHIP=chip_sel_tmp;
    InputValues(CHIP);

    GetCtrlVal(E_Mask, ENABLE_NUMERIC6, &PERC);
      GetCtrlVal(E_Mask, ENABLE_NUMERIC5, &MAXC);

      if (controll==1)
       {

      GetCtrlVal(E_Mask, ENABLE_NUMERIC7, &Vth);

/* inserted here the loading of an enable mask to be used in noise calculation
*/

load_mask_file();
```

```
      for(i=0;i<64;i++)
          for(j=0;j<64;j++)
               enable_mask[i][j]=0;


                dac_bias_tmp[2+CHIP*5]=(int)(Vth*LSBCAL);

                turn_on_dac();
              load_mask();
                start_acq(RADIOGRAPHY);     /*start the acquisition */
          if (MISSING_MEDIPIX == 0) download_data();
                                  /**** il calcolo e' fatto ****/
                c=0;
            for(i=0;i<64;i++)
              for(j=0;j<64;j++)
                      {
                            matrix2[i][j]=A[data_matrix[i][j]];
                            if(matrix2[i][j]>=MAXC)
                                  {
                                  c++;
                                  enable_mask[i][j] = 1;
                                  }
                            }
}

if (controll==2)
{
          Vth=512;
      for (ii=0; Vth<=4096; ii++)
          {
            for(i=0;i<64;i++)
              for(j=0;j<64;j++)
                            enable_mask[i][j] = 0;

                dac_bias_tmp[2+CHIP*5]=(int)Vth;

                turn_on_dac();
              load_mask();
                start_acq(RADIOGRAPHY);     /*start the acquisition */
          if (MISSING_MEDIPIX == 0) download_data();
                                  /**** il calcolo e' fatto ****/
                c=0;
            for(i=0;i<64;i++)
              for(j=0;j<64;j++)
                      {
                            matrix2[i][j]=A[data_matrix[i][j]];
                            if(matrix2[i][j]>=MAXC)
                                  {
                                  c++;
                                  enable_mask[i][j] = 1;
                                  }
                            }
          if(c<=(int)(PERC*4096)) break;
          Vth+=32;

                }
```

```
      }

            for(i=0;i<64;i++)
                for(j=0;j<64;j++)
                        config_matrix[i][j]=        config_matrix[i][j]        |
enable_mask[i][j];
              //GetCtrlVal(E_Mask,ENABLE_FILEMASK,file);
              savemask(file);
              sprintf(res2,"  Bad Pixels= %d",c);
              ResetTextBox(E_Mask,ENABLE_TEXTBOX,"");
              InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,"");


  if(controll==1)
             sprintf(res1,"Vth= %.3f",Vth);
  else if(controll==2)
             sprintf(res1,"Vth Min= %.3f",Vth/LSBCAL);

          InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,res1);
                  InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,res2);
                  InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,"\nIn file:");
                  InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,       file);
                  InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,"bad   pixels   are
turn off");
                  InsertTextBoxLine(E_Mask,ENABLE_TEXTBOX,-1,"\nEnd Noise&Enable
Mask");
  }

   break;


      }
      return 0;
}



void InputValues(int par_CHIP)
{

        float Vb_val, Vc_val, Vdl_val, Vth, Vtha_val;
      int tempo;


         SetCtrlAttribute(panelHandle,PANEL_TIMER,ATTR_ENABLED,0);  //disabilita
il timer della funzione timer principale

      GetCtrlVal(E_Mask, ENABLE_NUMERIC1, &Vb_val);
        GetCtrlVal(E_Mask, ENABLE_NUMERIC2, &Vc_val);
        GetCtrlVal(E_Mask, ENABLE_NUMERIC3, &Vdl_val);
        GetCtrlVal(E_Mask, ENABLE_NUMERIC7_2, &Vtha_val);

        GetCtrlVal(E_Mask, ENABLE_NUMERIC4, &tempo);

        trig_mode_tmp=0;
      tau1_tmp=tempo;
```

```
      dac_bias_tmp[0+par_CHIP*5]=((int)(Vb_val*LSBCAL));
        dac_bias_tmp[1+par_CHIP*5]=(int)(Vc_val*LSBCAL);
      dac_bias_tmp[4+par_CHIP*5]=(int)(Vdl_val*LSBCAL);
      dac_bias_tmp[3+par_CHIP*5]=(int)(Vtha_val*LSBCAL);
              for(i=0;i<64;i++)
                for(j=0;j<64;j++)
                            config_matrix[i][j]=config_matrix[i][j] & 0xfe;

}



void start_E_M(void)
{

  SetPanelAttribute (E_Mask, ATTR_WIDTH, 381);
  SetPanelPos (E_Mask,VAL_AUTO_CENTER ,VAL_AUTO_CENTER );

  ResetTextBox(E_Mask, ENABLE_TEXTBOX,"");
  InsertTextBoxLine (E_Mask, ENABLE_TEXTBOX, -1,"");
  InsertTextBoxLine (E_Mask, ENABLE_TEXTBOX, -1,"                  Select");
  InsertTextBoxLine  (E_Mask,  ENABLE_TEXTBOX,  -1,"\n             Turn  off  bad
pixels");
  InsertTextBoxLine (E_Mask, ENABLE_TEXTBOX, -1,"                    or");
  InsertTextBoxLine (E_Mask, ENABLE_TEXTBOX, -1,"             Best Vth min");


  SetCtrlAttribute(E_Mask, ENABLE_DECORATION2, ATTR_VISIBLE, 0);
  //SetCtrlAttribute(E_Mask, ENABLE_DECORATION6, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_DECORATION7, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_DECORATION8, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_DECORATION10, ATTR_VISIBLE, 0);
  //SetCtrlAttribute(E_Mask, ENABLE_DECORATION11, ATTR_VISIBLE, 0);

  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC1, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC2, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC3, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC4, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC5, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC7, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC6, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_NUMERIC7_2, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_FILEMASK, ATTR_VISIBLE, 0);
  //SetCtrlAttribute(E_Mask, ENABLE_NUMERIC8, ATTR_VISIBLE, 0);

  SetCtrlAttribute(E_Mask, ENABLE_RUN, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_BACK, ATTR_VISIBLE, 0);
  SetCtrlAttribute(E_Mask, ENABLE_DECORATION5, ATTR_VISIBLE, 0);

  SetCtrlAttribute(E_Mask, ENABLE_CHECK_VTH, ATTR_VISIBLE, 1);
  SetCtrlAttribute(E_Mask, ENABLE_CHECK_BEST_VTH, ATTR_VISIBLE, 1);
  //SetCtrlAttribute(E_Mask, ENABLE_CHECK_OFF_NOISE, ATTR_VISIBLE, 1);
  SetCtrlAttribute(E_Mask, ENABLE_DECORATION3, ATTR_VISIBLE, 1);
  SetCtrlAttribute(E_Mask, ENABLE_DECORATION4, ATTR_VISIBLE, 1);
  SetCtrlAttribute(E_Mask, ENABLE_DECORATION9, ATTR_VISIBLE, 1);

}
```

```
void check(void)
{
   SetCtrlAttribute(E_Mask, ENABLE_CHECK_VTH, ATTR_VISIBLE, 0);
   SetCtrlAttribute(E_Mask, ENABLE_CHECK_BEST_VTH, ATTR_VISIBLE, 0);
  // SetCtrlAttribute(E_Mask, ENABLE_CHECK_OFF_NOISE, ATTR_VISIBLE, 0);
   SetCtrlAttribute(E_Mask, ENABLE_DECORATION3, ATTR_VISIBLE, 0);
   SetCtrlAttribute(E_Mask, ENABLE_DECORATION4, ATTR_VISIBLE, 0);

   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC7, ATTR_VISIBLE, 0);
   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC6, ATTR_VISIBLE, 0);

   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC1, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC2, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC3, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC4, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC5, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_NUMERIC7_2, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_FILEMASK, ATTR_VISIBLE, 1);
   SetCtrlVal(E_Mask, ENABLE_FILEMASK, "*.msk");

   SetCtrlAttribute(E_Mask, ENABLE_DECORATION5, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_DECORATION2, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_DECORATION7, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_DECORATION8, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_DECORATION10, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_RUN, ATTR_VISIBLE, 1);
   SetCtrlAttribute(E_Mask, ENABLE_BACK, ATTR_VISIBLE, 1);

   SetPanelAttribute (E_Mask, ATTR_WIDTH, 636);
   SetPanelPos (E_Mask,VAL_AUTO_CENTER ,VAL_AUTO_CENTER );

   SetCtrlVal(E_Mask, ENABLE_CHECK_VTH, 0);
   SetCtrlVal(E_Mask, ENABLE_CHECK_BEST_VTH, 0);

   ResetTextBox(E_Mask,ENABLE_TEXTBOX,"");
   InsertTextBoxLine (E_Mask, ENABLE_TEXTBOX, -1,"");
   InsertTextBoxLine (E_Mask, ENABLE_TEXTBOX, -1,"\n\n                    Fill
out");
   InsertTextBoxLine (E_Mask, ENABLE_TEXTBOX, -1,"\n         and press RUN.");

}

int CVICALLBACK Back_E_Mask (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                     start_E_M();



                  break;
      }
      return 0;
}
```

```
int CVICALLBACK Enable_Quit (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                        SetCtrlAttribute(panelHandle,PANEL_TIMER,ATTR_ENABLED,1);
//abilita il timer della funzione timer principale
                      HidePanel(E_Mask);
                   break;
      }
      return 0;
}


void savemask(char *filemask)
{
      int i,j,k;
      char buff[30],filepath[50], filename[30];

/**** questa funzione serve a salvare un file di tipo "msk"
a partire dalla attuale
maschera in ram (config_matrix)**********/

            sprintf(filename,PATHF2"%s",filemask);
            fpout = fopen (filename, "w");
            fprintf(fpout,"* enable mask *\n");
            fprintf(fpout,"\n");
            for(i=0;i<64;i++)
                  {
                    for(j=0;j<64;j++)
                    {
                        fprintf(fpout,"%d ",(config_matrix[i][j]&1));
                                        /* write the enable_bit_mask*/
                    }
                        fprintf(fpout,"\n");
                  }

            fprintf(fpout,"\n");
            fprintf(fpout,"* test mask *\n");
            fprintf(fpout,"\n");
            for(i=0;i<64;i++)
                  {
                        for(j=0;j<64;j++)
                        {
                          fprintf(fpout,"%d ",(config_matrix[i][j]&2)>>1);
                                              /* write the test_bit_mask*/
                        }
                        fprintf(fpout,"\n");
                  }

            fprintf(fpout,"\n");
            fprintf(fpout,"* threshold mask *\n");
            fprintf(fpout,"\n");
            for(i=0;i<64;i++)
                  {
            for(j=0;j<64;j++)
```

```
                        {
                              k=(config_matrix[i][j]&4)+
                                ((config_matrix[i][j]&8)>>2)+
                                ((config_matrix[i][j]&16)>>4);
                                    fprintf(fpout,"%d ",k);
                                    /* write the threshold_bit_mask*/
                        }
                        fprintf(fpout,"\n");
                  }
            fclose(fpout);
}


/*************************************************************************
*************************************************************************/

/************************** fine Enable_Mask ******************************/



/*************************************************************************
******* Questa CallBack viene utilizzata per chiudere qualsiasi ***********
*******  pannello aperto dal menu di basic access ************************
*************************************************************************/

int Pixels_Cal;

int CVICALLBACK GeneralQuit (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{

  switch (event) {
            case EVENT_COMMIT:

      if(panel==basic_r_a) HidePanel (basic_r_a);

      if(panel==ram_fifo) HidePanel (ram_fifo);

      if(panel==Test_Counters) HidePanel (Test_Counters);

      if(panel==Test_Mask) HidePanel(Test_Mask);

      if(panel==Add_Subt_Files) HidePanel(Add_Subt_Files);

      if(panel==Pixels_Cal) HidePanel(Pixels_Cal);

     break;
       }
 return 0;
}

/*************************************************************************
*************************************************************************
/


int Load_Data_File(void);
```

```
int Load_Back_File(void);


void CVICALLBACK Calib (int menuBar, int menuItem, void *callbackData,
                        int panel)
{
  int r;

  r = Load_Data_File();
  if (r)
    return;
  r = Load_Back_File();
  if (r)
    return;
  DisplayPanel(Pixels_Cal);
}


int CVICALLBACK Run (int panel, int control, int event, void *callbackData,
                     int eventData1, int eventData2)
{
  char c;
  char fil1[512],fil2[512],fil3[512];
  char fil10[512],fil20[512],fil30[512];
  int count1[64][64],count2[64][64];
  int i,j,k,kk,tot1=0,tot2=0;
  float med=0.,sig=0.,dist=0.,r_tot1=0.,r_count1[64][64],r_count2[64][64];
  char message1[50];
  char message2[50];

  switch (event) {
      case EVENT_COMMIT:
      GetCtrlVal(Pixels_Cal,PIXELS_CAL_STRING1,fil1);
        GetCtrlVal(Pixels_Cal,PIXELS_CAL_STRING2,fil2);
        GetCtrlVal(Pixels_Cal,PIXELS_CAL_STRING3,fil3);

        sprintf(fil10,"%s%s",PATHF2,fil1);
        sprintf(fil20,"%s%s",PATHF2,fil2);
        sprintf(fil30,"%s%s",PATHF2,fil3);

      FileToArray (fil10, count1, VAL_INTEGER, 4096, 64, VAL_GROUPS_TOGETHER,
                   VAL_GROUPS_AS_COLUMNS, VAL_ASCII);
      FileToArray (fil20, count2, VAL_INTEGER, 4096, 64, VAL_GROUPS_TOGETHER,
                   VAL_GROUPS_AS_COLUMNS, VAL_ASCII);

      ResetTextBox(Pixels_Cal,PIXELS_CAL_TEXTBOX,"");
      InsertTextBoxLine(Pixels_Cal,PIXELS_CAL_TEXTBOX,-1,"");
      InsertTextBoxLine(Pixels_Cal,PIXELS_CAL_TEXTBOX,-1,
                        "              RUNNING !");
      for(i=0;i<64;i++)
      for(j=0;j<64;j++) {
        r_count2[i][j]=0.;
        r_count1[i][j]=0.;
        if(count2[i][j]==696969)
          count2[i][j]=0;
        if(count1[i][j]==696969)
          count1[i][j]=0;
```

```
            tot1= tot1+count1[i][j];
            tot2= tot2+count2[i][j];
        }

      for(i=0;i<64;i++)
      for(j=0;j<64;j++) {
            r_count2[i][j]=(float)(count2[i][j])/(float)tot2 ;
            if(r_count2[i][j]==0.)
              r_count1[i][j]=0.;
             else
              r_count1[i][j]=(float)(count1[i][j])/r_count2[i][j] ;
         }

      for(i=0;i<64;i++)
      for(j=0;j<64;j++)
            r_tot1=r_tot1+r_count1[i][j];

      for(i=0;i<64;i++)
      for(j=0;j<64;j++)
            count1[i][j]=(int)((r_count1[i][j]/(r_tot1))*tot1);

      tot2=0;
      for(i=0;i<64;i++)
      for(j=0;j<64;j++)
            tot2=tot2+count1[i][j];

      ArrayToFile (fil30, count1, VAL_UNSIGNED_INTEGER, 4096, 64,
                  VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
                        VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_TRUNCATE);

      ResetTextBox(Pixels_Cal,PIXELS_CAL_TEXTBOX,"");
      InsertTextBoxLine(Pixels_Cal,PIXELS_CAL_TEXTBOX,-1,"");
      InsertTextBoxLine(Pixels_Cal,PIXELS_CAL_TEXTBOX,-1,
                        " End of Pixel Calibration.");
      sprintf(message1,"\n Integral counts: %d",tot1);
      sprintf(message2," Integral counts after\n normalization: %d",tot2);
      InsertTextBoxLine(Pixels_Cal,PIXELS_CAL_TEXTBOX,-1,message1);
      InsertTextBoxLine(Pixels_Cal,PIXELS_CAL_TEXTBOX,-1,message2);

        break;
  }
  return 0;
}


/*****************************************************************************
*************************   CERN 7     ******************************
*****************************************************************************/

int Vth_Cal;
int i, j, k, ii, iii, I, jj, c, cc, crepeat, s, choice = 0, choiceAr = 0;
                          //pulseM=100; modificato gual da int a float
int VBi,VCi,Nvb,Nvc,Vth,choicePul=0,MINC=2, Mrep, mrep, Irep;
float pulseW=300,pulseT=0.1,PERC=0.1,pulseV=10,pulseST=10,pulseS=10;
float pulseM=100;
int Vb_value[30],Vc_value[30];
int Vth_val[30][30];
```

```
char buff[50],fil[512],filepath[512],lista[512],file0[512],plist[512];
char maskfile[512], maschera[80];
int A[32768];
int matrix2[64][64],config_matrix0[64][64],m1[64][64],m2[64][64],m3[64][64];
int trash,CHIP,stop=0;
int ctrl_ring1, ctrl_ring2;
int AUTO_VTH, ONLY_VOLTAGE;
int PSelect;
int Check=0;
float Vb_value_buff[30], Vc_value_buff[30];
int Time, appoggio;

int Voltage_Calibra(int,int,int);
void Input_Voltage(void);
void Select_Voltage(void);
int Insert_Filename(void);
void Disable(int);
void Input_Val(int);
void Load_LUT(void);
void Dimmed(void);
void enable(void);
int load_mask_file (void);
void thr_adj_bits_set(void);
void Thr_Adj_Bits_Set(void);
void adj_loaded_msk_file(void);


void CVICALLBACK vth_cal (int menuBar, int menuItem, void *callbackData,
                          int panel)
{
  SetCtrlAttribute(panelHandle, PANEL_TIMER, ATTR_ENABLED, 0);
// disabilita la funzione main timer nel file main.c

  Dimmed();
  enable();
  DisplayPanel(Vth_Cal);
  DisplayPanel(PSelect);
  CHIP = chip_sel_tmp;
  ONLY_VOLTAGE = 0;
  AUTO_VTH = 1;
  if (check_power() != 1) {
    error(2);
    SetPanelAttribute(PSelect, ATTR_DIMMED, 1);
  }
}


int CVICALLBACK Run_Voltage_Aut (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
  int r;

      switch (event) {
            case EVENT_COMMIT:

                     ONLY_VOLTAGE=1;
                     r = Insert_Filename();
```

```
                    if (r==1) {
                      HidePanel(Vth_Cal);
                      return 1;
                    }
                    Load_LUT();
/* September 1999, inserted here this line, the call to loadmsk, in order to use
a loaded mask also for
voltage calculation */
                    // if(!(load_mask_file()))/* end of inserted line */
                    load_mask_file();
                       Voltage_Calibra(1,1,0);

                 break;
      }
      return 0;
}

int CVICALLBACK Run_Calibration (int panel, int control, int event,
                                 void *callbackData, int eventData1,
                                 int eventData2)
{
  int r;

  switch (event) {
      case EVENT_COMMIT:
      r = Insert_Filename();
        if (r==1) {
          HidePanel(Vth_Cal);
          return 1;
        }
        Load_LUT();
                    // if(!(load_mask_file()))
        load_mask_file();
        Voltage_Calibra(2,0,0);
        break;
  }
  return 0;
}


int Voltage_Calibra(int ctrl_run, int ctrl_led, int ctrl_vth)
{
  int status, shell, ok_button, length, length1, i, j, k, h=0;
  char message[80];
  char buff1[60]={0};
 // float Vb_value_buff[30], Vc_value_buff[30];


  if(ctrl_run==1)
    {

      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM1,ATTR_DIMMED, 1);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM2,ATTR_DIMMED, 1);
      if((ctrl_vth==1)||(ONLY_VOLTAGE))
        {
         SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM1,ATTR_DIMMED, 0);
         SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM2,ATTR_DIMMED, 0);
```

```
      }
    SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM3,ATTR_DIMMED, 0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM4,ATTR_DIMMED, 0);

    if (TIME_UNIT==0)  // Unita di misura: millisecondi
                SetCtrlAttribute (Vth_Cal, VTH_CAL_NUM4_2, ATTR_LABEL_TEXT,
                                  "Acquisition time (ms)");
      else                        // Unita di misura: secondi
                SetCtrlAttribute (Vth_Cal, VTH_CAL_NUM4_2, ATTR_LABEL_TEXT,
                                       "Acquisition time (s)");

    SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM4_2,ATTR_DIMMED, 0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON1,ATTR_DIMMED, 0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED1,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED2,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED3,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED4,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED5,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED6,ATTR_DIMMED,0);
    if(ctrl_led==1)
      SetCtrlVal(Vth_Cal,VTH_CAL_LED3,1);
  }

if(ctrl_run==2)
  {
      SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED, 0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM6,ATTR_DIMMED, 0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED, 0);
    SetCtrlVal(Vth_Cal,VTH_CAL_CHECKNUM5, 0);
    SetCtrlVal(Vth_Cal,VTH_CAL_CHECKNUM6, 0);
    SetCtrlVal(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE, 0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED1,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED2,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED3,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED4,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED5,ATTR_DIMMED,0);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED6,ATTR_DIMMED,0);
    SetActiveCtrl(Vth_Cal,VTH_CAL_CHECKNUM5);
    return 0;
    }
do
    status=GetUserEvent(1,&shell,&ok_button);
while(!(status==EVENT_COMMIT&&(shell==Vth_Cal&&
     ((ok_button==VTH_CAL_BUTTON1)||(ok_button==VTH_CAL_QUIT)))
     ||((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT))));

if(ok_button==VTH_CAL_BUTTON1)
  {
    SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON1,ATTR_DIMMED, 1);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_TEXTBOX1,ATTR_DIMMED, 0);
    GetCtrlVal(Vth_Cal,VTH_CAL_NUM1, &PERC);
    GetCtrlVal(Vth_Cal,VTH_CAL_NUM2, &MINC);
    GetCtrlVal(Vth_Cal,VTH_CAL_NUM3, &Nvb);
    GetCtrlVal(Vth_Cal,VTH_CAL_NUM4, &Nvc);

    GetCtrlVal(Vth_Cal,VTH_CAL_NUM4_2, &Time);
    tau1_tmp=Time;
```

```
      Disable(1);
      ResetTextBox(Vth_Cal,VTH_CAL_TEXTBOX1,"");

      for(i=1; i<=Nvb; i++)
         {
             Fmt(message,"%s<%s%d%s ","\nInsert #",i," Vbias: ");
               length = StringLength (message);
             SetCtrlVal(Vth_Cal,VTH_CAL_TEXTBOX1,message);
               SetActiveCtrl (Vth_Cal, VTH_CAL_TEXTBOX1);
               do
                 status=GetUserEvent(1,&shell,&ok_button);

while(!((status==EVENT_COMMIT)&&(shell=Vth_Cal)&&((ok_button==VTH_CAL_TEXTBOX1)|
|(ok_button==VTH_CAL_QUIT))));
               if(ok_button==VTH_CAL_TEXTBOX1)
                 {
                  GetTextBoxLine (Vth_Cal, VTH_CAL_TEXTBOX1,i, buff1);
                  length1 = StringLength (buff1);
                  CopyString (buff1, 0, buff1, length-1,-1);
                  Vb_value_buff[i-1]=atof(buff1);
                  Vb_value[i-1]=(int)(LSBCAL*atof(buff1));
                 }
             }

        k=i;

        for(i=1; i<=Nvc; i++)
           {
             Fmt(message,"%s<%s%d%s ","\nInsert #",i," Vcomp: ");
             length = StringLength (message);

             SetCtrlVal(Vth_Cal,VTH_CAL_TEXTBOX1,message);
             SetActiveCtrl (Vth_Cal, VTH_CAL_TEXTBOX1);

               do
                 status=GetUserEvent(1,&shell,&ok_button);

while(!((status==EVENT_COMMIT)&&(shell=Vth_Cal)&&((ok_button==VTH_CAL_TEXTBOX1)|
|(ok_button==VTH_CAL_QUIT))));
               if(ok_button==VTH_CAL_TEXTBOX1)
               {
                GetTextBoxLine (Vth_Cal, VTH_CAL_TEXTBOX1,i+k-1, buff1);
                length1 = StringLength (buff1);
                CopyString (buff1, 0, buff1, length-1,-1);
                Vc_value_buff[i-1]=atof(buff1);
                Vc_value[i-1]=(int)(LSBCAL*atof(buff1));
               }
             }

             if(ctrl_vth==2)
               {
                ResetTextBox(Vth_Cal,VTH_CAL_TEXTBOX1,"");
                  for(i=0;i<Nvb;i++)
                     for(j=0;j<Nvc;j++)
                        {
                          h+=3;
```

```
                                                    Fmt(message,"%s<%s    %f       %s
%f","\n\nVbias=",Vb_value_buff[i],"Vcomp=",Vc_value_buff[j]);
                        SetCtrlVal(Vth_Cal,VTH_CAL_TEXTBOX1,message);
                        Fmt(message,"%s<%s","\nInsert Vth: ");
                        SetCtrlVal(Vth_Cal,VTH_CAL_TEXTBOX1,message);
                        length = StringLength (message);
                        SetActiveCtrl (Vth_Cal, VTH_CAL_TEXTBOX1);

                          do
                            status=GetUserEvent(1,&shell,&ok_button);

while(!((status==EVENT_COMMIT)&&(shell=Vth_Cal)&&((ok_button==VTH_CAL_TEXTBOX1)|
|(ok_button==VTH_CAL_QUIT))));
                          if(ok_button==VTH_CAL_TEXTBOX1)
                            {
                           GetTextBoxLine (Vth_Cal, VTH_CAL_TEXTBOX1,h, buff1);
                           length1 = StringLength (buff1);
                             CopyString (buff1, 0, buff1, length-1,-1);
                             Vth_val[i][j]=(int)(LSBCAL*atof(buff1));
                            }
                        }

                }
                SetCtrlVal(Vth_Cal,VTH_CAL_TEXTBOX1,"\n");
                        SetCtrlAttribute(Vth_Cal,VTH_CAL_TEXTBOX1,ATTR_CTRL_MODE,
VAL_INDICATOR);

                if(ctrl_vth!=0)
                  Input_Val(1);
                else
                  {
                   SetCtrlAttribute(Vth_Cal,VTH_CAL_RUN,ATTR_DIMMED,0);
               SetActiveCtrl(Vth_Cal,VTH_CAL_RUN);
                  }
      }
 return 0;
}



int CVICALLBACK En_Loaded_msk (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{

      int ctrl;

      switch (event) {
            case EVENT_COMMIT:

                  GetCtrlVal(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,&ctrl);
                  if(ctrl==1)
                   {
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED,1);
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM6,ATTR_DIMMED,1);
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_LED1,ATTR_DIMMED,0);
                    SetCtrlVal(Vth_Cal,VTH_CAL_LED1,0);
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED,0);
```

```
                    Check=3;
      /*            do
                       status=GetUserEvent(0,&shell,&ok_button);

while(!(status==EVENT_COMMIT&&shell==Vth_Cal&&((ok_button==VTH_CAL_OK_LOADED_MAS
K_FILE)||(ok_button==VTH_CAL_QUIT))));
                    if(ok_button==VTH_CAL_OK_LOADED_MASK_FILE)
                    {
               SetCtrlVal(Vth_Cal,VTH_CAL_LED1,1);
               crepeat=11;
               mrep=0;
                     Mrep=0;
                     Select_Voltage();
                    } */
                   }
                 else
                   {
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED,0);
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM6,ATTR_DIMMED,0);
                    Check=0;
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED,1);
                   }
            break;
      }
      return 0;
}

int CVICALLBACK En_rep_acq (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int ctrl, status, shell, ok_button;

      switch (event) {
            case EVENT_COMMIT:

                  GetCtrlVal(Vth_Cal,VTH_CAL_CHECKNUM6,&ctrl);
                  if(ctrl==1)
                   {
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED,1);

SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED,1);
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_1,ATTR_DIMMED,0);
                    SetCtrlVal(Vth_Cal,VTH_CAL_NUM6_1,0);
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_DIMMED,0);
                    SetCtrlVal(Vth_Cal,VTH_CAL_NUM6_2,7);
                    //SetActiveCtrl (Vth_Cal, VTH_CAL_NUM6_1);
                    Check=2;
                    SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED,0);

              /*
                    do
                     status=GetUserEvent(1,&shell,&ok_button);

while(!(status==EVENT_COMMIT&&shell==Vth_Cal&&((ok_button==VTH_CAL_NUM6_1)

||(ok_button==VTH_CAL_CHECKNUM6)||(ok_button==VTH_CAL_QUIT))));
```

```
                    if(ok_button==VTH_CAL_NUM6_1)
                       {
                        GetCtrlVal(Vth_Cal,VTH_CAL_NUM6_1,&mrep);

SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_1,ATTR_CTRL_MODE,VAL_INDICATOR);
                        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_DIMMED,0);
                        SetActiveCtrl (Vth_Cal, VTH_CAL_NUM6_2);
                        SetCtrlVal(Vth_Cal,VTH_CAL_NUM6_2,7);

SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_MIN_VALUE,mrep);
                        do
                          status=GetUserEvent(1,&shell,&ok_button);

while(!(status==EVENT_COMMIT&&shell==Vth_Cal&&ok_button==VTH_CAL_NUM6_2));
                        GetCtrlVal(Vth_Cal,VTH_CAL_NUM6_2,&Mrep);

SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_CTRL_MODE,VAL_INDICATOR);
                        Select_Voltage();
                       }
      //             else if(ok_button==VTH_CAL_CHECKNUM6)
      //                {
      //
SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED,0);
      //
SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED,0);
      //                 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_1,ATTR_DIMMED,1);
      //                 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_DIMMED,1);
      //                }
        */
                    }
                  else
                    {
                     SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED,0);

SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED,0);
                     SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_1,ATTR_DIMMED,1);
                     SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_DIMMED,1);
                     Check=0;
                     SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED,1);
                    }
            break;
      }
      return 0;
}

int CVICALLBACK En_Thr_adj (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      int ctrl, status, shell, ok_button;

      switch (event) {
            case EVENT_COMMIT:

             GetCtrlVal(Vth_Cal,VTH_CAL_CHECKNUM5,&ctrl);
                   if(ctrl==1)
                    {
```

```
SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED,1);
                      SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM6,ATTR_DIMMED,1);
                      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM5,ATTR_DIMMED,0);
                      SetActiveCtrl (Vth_Cal, VTH_CAL_NUM5);

      /*
                      do
                        status=GetUserEvent(1,&shell,&ok_button);

while(!(status==EVENT_COMMIT&&shell==Vth_Cal&&((ok_button==VTH_CAL_NUM5)||(ok_bu
tton==VTH_CAL_QUIT))));
                      if(ok_button==VTH_CAL_NUM5)
                        {
                         GetCtrlVal(Vth_Cal,VTH_CAL_NUM5,&crepeat);
                         mrep=crepeat;
                         Mrep=crepeat;
//
SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM5,ATTR_CTRL_MODE,VAL_INDICATOR);
//
SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_CTRL_MODE,VAL_INDICATOR);
                         Select_Voltage();
                        }   */

                      Check=1;
                      SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED,0);
                     }

                   else
                     {

SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED,0);
                      SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM6,ATTR_DIMMED,0);
                      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM5,ATTR_DIMMED,1);
                      SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED,1);
                      Check=0;
                     }
            break;
      }
      return 0;
}



void Select_Voltage(void)
{
  int status, shell, ok_button;

  SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM6,ATTR_DIMMED, 1);
  SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED, 1);
  SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED, 1);
  SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED, 1);
  SetCtrlAttribute(Vth_Cal,VTH_CAL_RING1,ATTR_DIMMED, 0);
  SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON4,ATTR_DIMMED, 0);
  if(Check==1)
    thr_adj_bits_set();
```

```
  if(Check==2)
    Thr_Adj_Bits_Set();
  if(Check==3)
    adj_loaded_msk_file();
  do
    status=GetUserEvent(1,&shell,&ok_button);
  while(!(status==EVENT_COMMIT&&(shell==Vth_Cal&&((ok_button==VTH_CAL_BUTTON4)||
        (ok_button==VTH_CAL_QUIT)))||
        ((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QUIT))));
  if(ok_button==VTH_CAL_BUTTON4) {
    GetCtrlVal(Vth_Cal,VTH_CAL_RING1,&ctrl_ring1);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON4,ATTR_DIMMED,1);
    SetCtrlAttribute(Vth_Cal,VTH_CAL_RING1,ATTR_CTRL_MODE,VAL_INDICATOR);
    if(ctrl_ring1==1) {                // Auto Vth
      SetCtrlVal(Vth_Cal,VTH_CAL_LED3,1);
      Voltage_Calibra(1,0,ctrl_ring1);
     }
     else if(ctrl_ring1==2)   {                         // Manual Vth
       AUTO_VTH=0;
       SetCtrlVal(Vth_Cal,VTH_CAL_LED2,1);
       Voltage_Calibra(1,0,ctrl_ring1);
    }
  }
}


int Insert_Filename(void)
{
  char message[60];
  int ldn = 0, lfn = 0, select;
  char fil0[512], fil1[512];
  long *size;

  HidePanel(PSelect);

  sprintf(fil1, "%s\\thresh1\\", Directory_Name);
  select = FileSelectPopup(fil1, "*.lst", "*.lst", "FILE PREFIX FOR THIS RUN",
                           VAL_SAVE_BUTTON, 0, 0, 1, 0, fil0);
  if (select == VAL_NO_FILE_SELECTED) {
    printf("\a");
    MessagePopup("Warning!",
                 "NO  FILE  SELECTED!\nI  shall  take  you  back  to  the  previous
Panel.");
    return 1;
  }

  ldn = strrchr(fil0, '\\') - &fil0[0];
  size = &ldn;
  ldn++;
  strncpy(fil1, fil0, ldn);
  fil1[ldn] = '\0';
  lfn = strlen(fil0);
  for (i = ldn; i < lfn - 4; i++)
    fil[i - ldn] = fil0[i];
  fil[lfn - 4] = '\0';

  for (i = 0; i <= 11; i++) {
```

```
    sprintf(fil0, "%s%s_%d.lst", fil1, fil, i);
    ldn = GetFileInfo(fil0, size);
    if (ldn == 1) {
      printf("\a");
      MessagePopup("Warning!",
                   "ALREADY EXISTING FILE SELECTED!\nI shall take you back to
the previous Panel.");
      return 1;
    }
  }

  if (ONLY_VOLTAGE == 0)
    Fmt(message, "%s<Name will be %s_Q#_Vbias_Vcomp.dat", fil);
   else if (ONLY_VOLTAGE == 1)
    Fmt(message, "%s<Name will be %s.thr", fil);

  MessagePopup("", message);
  return 0;
}


void Disable(int select)
{
  switch(select)
    {
     case 1:
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM1,  ATTR_CTRL_MODE, VAL_INDICATOR );
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM2,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM3,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM4,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM4_2,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_LED1,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_LED2,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_LED3,ATTR_CTRL_MODE, VAL_INDICATOR);
     break;
     case 2:
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM7,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM8,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM10,ATTR_CTRL_MODE, VAL_INDICATOR);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM11,ATTR_CTRL_MODE, VAL_INDICATOR);
     break;
     case 3:
      SetCtrlAttribute(Vth_Cal,VTH_CAL_RING2,ATTR_CTRL_MODE, VAL_INDICATOR);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM12,ATTR_CTRL_MODE, VAL_INDICATOR);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM13,ATTR_CTRL_MODE, VAL_INDICATOR);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM14,ATTR_CTRL_MODE, VAL_INDICATOR);
       break;
   }
}


void enable(void)
{
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM1,  ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM2, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM3, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM4, ATTR_CTRL_MODE, VAL_HOT);
```

```
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM4_2, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM5, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM6_1, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM6_2, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_CHECKNUM5, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_CHECKNUM6, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal,    VTH_CAL_CHECKLOADED_MASK_FILE,    ATTR_CTRL_MODE,
VAL_HOT);

  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM7, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM8, ATTR_CTRL_MODE, VAL_HOT);

  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM10, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM11, ATTR_CTRL_MODE, VAL_HOT);

  SetCtrlAttribute(Vth_Cal, VTH_CAL_RING2, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_RING1, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM12, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM13, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_NUM14, ATTR_CTRL_MODE, VAL_HOT);
  SetCtrlAttribute(Vth_Cal, VTH_CAL_TEXTBOX1, ATTR_CTRL_MODE, VAL_HOT);
}


void Input_Val(int select)
{
 int status, shell, ok_button;
 float input_Vdl, input_Vtha;

    if(select==1)
      {
        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM7,ATTR_DIMMED,0);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM8,ATTR_DIMMED,0);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM10,ATTR_DIMMED,0);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM11,ATTR_DIMMED,0);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON2,ATTR_DIMMED,0);
        SetActiveCtrl (Vth_Cal, VTH_CAL_NUM7);
      }
    if(select==2)
      {
            SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM12,ATTR_DIMMED,0);
            SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM13,ATTR_DIMMED,0);
            SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM14,ATTR_DIMMED,0);
            SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON3,ATTR_DIMMED,0);
            SetActiveCtrl (Vth_Cal, VTH_CAL_NUM12);
        }

 do
   status=GetUserEvent(1,&shell,&ok_button);
 while(!(status==EVENT_COMMIT&&shell==Vth_Cal&&

((ok_button==VTH_CAL_BUTTON2)||(ok_button==VTH_CAL_BUTTON3)||(ok_button==VTH_CAL
_QUIT))));

 if(ok_button==VTH_CAL_BUTTON2)
   {
    SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON2,ATTR_DIMMED, 1);
```

```
    Disable(2);
      GetCtrlVal(Vth_Cal,VTH_CAL_NUM7,&pulseW);
      GetCtrlVal(Vth_Cal,VTH_CAL_NUM8,&pulseT);
      GetCtrlVal(Vth_Cal,VTH_CAL_NUM10,&input_Vdl);

      dac_bias_tmp[4+CHIP*5]=(int)(LSBCAL*input_Vdl);

      GetCtrlVal(Vth_Cal,VTH_CAL_NUM11,&input_Vtha);

      dac_bias_tmp[3+CHIP*5]=(int)(LSBCAL*input_Vtha);

      SetCtrlAttribute(Vth_Cal,VTH_CAL_RING2,ATTR_DIMMED,0);
      SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON5,ATTR_DIMMED, 0);

      do
        status=GetUserEvent(1,&shell,&ok_button);
      while(!(status==EVENT_COMMIT&&(shell==Vth_Cal&&((ok_button==VTH_CAL_BUTTON
5)

||(ok_button==VTH_CAL_QUIT))||((shell==panelHandle)&&(ok_button==PANEL_TOTAL_QU
IT))));

      if(ok_button==PANEL_TOTAL_QUIT)
        return;

      if(ok_button==VTH_CAL_QUIT)
        return;

      if(ok_button==VTH_CAL_BUTTON5);
      {
      SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON5,ATTR_DIMMED,1);
      GetCtrlVal(Vth_Cal,VTH_CAL_RING2,&ctrl_ring2);
      if(ctrl_ring2==1)
        {
         SetCtrlAttribute(Vth_Cal,VTH_CAL_RING2,ATTR_CTRL_MODE,VAL_INDICATOR);
         SetCtrlVal(Vth_Cal,VTH_CAL_LED4,1);
         choicePul=1;
         Input_Val(2);
        }
      else if(ctrl_ring2==2)
        {
         SetCtrlAttribute(Vth_Cal,VTH_CAL_RING2,ATTR_CTRL_MODE,VAL_INDICATOR);
         SetCtrlVal(Vth_Cal,VTH_CAL_LED5,1);
         choicePul=2;
        }
      }
  }
  if(ok_button==VTH_CAL_BUTTON3) {
    SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON3,ATTR_DIMMED, 1);
    Disable(3);
    GetCtrlVal(Vth_Cal,VTH_CAL_NUM12,&pulseST);
    GetCtrlVal(Vth_Cal,VTH_CAL_NUM13,&pulseS);
    GetCtrlVal(Vth_Cal,VTH_CAL_NUM14,&pulseM);
  }
  SetCtrlAttribute(Vth_Cal,VTH_CAL_RUN,ATTR_DIMMED,0);
  SetActiveCtrl(Vth_Cal,VTH_CAL_RUN);
```

```
}


void Load_LUT(void)
{

  open_read_file("LUT_real.lut");
  for(i=1;i<32768;i++)
    fscanf(fpin,"%d",&A[i]);
  fclose(fpin);
  A[0]=696969;
}


int CVICALLBACK Start (int panel, int control, int event, void *callbackData,
                       int eventData1, int eventData2)
{
  int taub0,taub1;
  char mask_file[512], arch_file[512];

  switch (event) {
      case EVENT_COMMIT:
      if(stop)
        SetCtrlVal(Vth_Cal,VTH_CAL_LED6,0);
        SetCtrlVal(Vth_Cal,VTH_CAL_OUTPUT,0);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_OUTPUT,ATTR_DIMMED,0);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_QUIT,ATTR_DIMMED,1);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_RUN,ATTR_DIMMED,1);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_STOP,ATTR_DIMMED,0);
        ProcessDrawEvents();
        Cls();

        for(i=0;i<64;i++)
          for(j=0;j<64;j++)
            config_matrix0[i][j]=config_matrix[i][j];

        if(AUTO_VTH) {

/** Here I am going to write the pulser output disable routine for the auto cal.
**********************************************************************/
                  Pulser_Disable();

//SetCtrlAttribute(Vth_Cal,VTH_CAL_STOP,ATTR_DIMMED,0);
//SetCtrlAttribute(Vth_Cal,VTH_CAL_OUTPUT,ATTR_DIMMED,0);
//ProcessDrawEvents();
//ProcessSystemEvents();

          sprintf(file0,"%sthresh2/%s.thr",PATHF,fil);
          fpout=fopen(file0,"w");

        for(VBi=0;VBi<Nvb;VBi++) {
          dac_bias_tmp[0+CHIP*5]=Vb_value[VBi];
          for(VCi=0;VCi<Nvc;VCi++) {
              dac_bias_tmp[1+CHIP*5]=Vc_value[VCi];
                Vth=512;
            for (ii=0; Vth<=4096; ii++) {
              dac_bias_tmp[2+CHIP*5]=Vth;
```

```
                turn_on_dac();
                c=0;

/******** scrittura delle maschere ************/
/*  Modified Sept 10, 1999:
    We want to use the enable mask already in use to select the area
    where to calculate Vth min or calibration.
    Old lines are therefore removed:

                            for(iii=0;iii<64;iii++)
                              {
                               ProcessSystemEvents();
                                          if(stop) break;   // controllo sullo
stop
                               for(j=0;j<64;j++)
                                 {
                                  if(iii>=0 && iii<=63 && j>=0 && j<=63)
                                    {
                                     config_matrix[iii][j]=0;
                                                   ProcessDrawEvents();
                                    }
                                  else config_matrix[iii][j]=1;
                                 }
                              }

follow initialization of config_matrix0: */


              load_mask();
                  ProcessSystemEvents();
                  if(stop)
                    break;  // controllo sullo stop
                  taub0=tau0_tmp;
                  taub1=tau1_tmp;
                  tau0_tmp=0;
                  tau2_tmp=0;

            start_acq(RADIOGRAPHY);     /*start the acquisition */
            if(MISSING_MEDIPIX==0)
              download_data();

                  tau0_tmp=taub0;
                  tau1_tmp=taub1;

/**** il calcolo e' fatto ****/
            for(i=0;i<64;i++) {
                ProcessSystemEvents();
                      if(stop)
                        break;  // controllo sullo stop
              for(j=0;j<64;j++) {
                      matrix2[i][j]=A[data_matrix[i][j]];
                      ProcessDrawEvents();
                      if(matrix2[i][j]>=MINC)
                        c++;
              }
            }
            if(c<=(PERC*(4096)))
```

```
                break;
              Vth+=32;
            }

            ProcessSystemEvents();
                if(stop)
                  break;  // controllo sullo stop

              Vth_val[VBi][VCi]=Vth;
                printf("Vbias =%.3f   Vcomp =%.3f   Vthmin =%.3f\n",
                       Vb_value_buff[VBi],Vc_value_buff[VCi],Vth/LSBCAL);
/* printf("Vbias =%.3f   Vcomp =%.3f   Vthmin =%.3f\n",
          Vb_value[VBi]/LSBCAL,Vc_value[VCi]/LSBCAL,Vth/LSBCAL);*/

              fprintf(fpout,"Vbias =%.3f   Vcomp =%.3f   Vthmin =%.3f\n",
                      Vb_value[VBi]/LSBCAL,Vc_value[VCi]/LSBCAL,Vth/LSBCAL);

            }

            ProcessSystemEvents();
            if(stop)
              break;  // controllo sullo stop
            printf("\n");
          }
              fclose(fpout);

          if (stop)
            printf("\n\nVth min calculation STOPPED !");
           else
            printf("\nEnd of Vth min calculation !");

/** Here I am going to write the pulser enable-on routine for the auto cal.
*************************************************************************/
      Pulser_Enable();

      }

          if(ONLY_VOLTAGE) {
            stop=0;
            SetCtrlAttribute(Vth_Cal,VTH_CAL_QUIT,ATTR_DIMMED,0);
            SetCtrlAttribute(Vth_Cal,VTH_CAL_RUN,ATTR_DIMMED,0);
            ProcessDrawEvents();
            return 0;
          }

      trig_mode_tmp=2;
      for (Irep=mrep; Irep<=Mrep; Irep++) {
        SetCtrlAttribute(Vth_Cal,VTH_CAL_STOP,ATTR_DIMMED,0);
        SetCtrlAttribute(Vth_Cal,VTH_CAL_OUTPUT,ATTR_DIMMED,0);
        ProcessDrawEvents();
        ProcessSystemEvents();
        for(i=0;i<64;i++)
          for(j=0;j<64;j++) {
            c=Irep;
            if(crepeat==11)
              c=m3[i][j];
            cc=((c&1)<<2) + (c&2) + ((c&4)>>2);
```

```
            cc=cc*4;
              config_matrix0[i][j]=(config_matrix0[i][j] & 3)+cc;
          }

          Pulser(pulseST,pulseT,pulseW);
          pulseV=pulseST;
          turn_off_dac();

          Delay(0.5);
//sleep_soft(10000000);

        for(jj=0; pulseV<=pulseM ;jj++) { /* for principale sugli impulsi */
          if(choicePul == 1)              /* auto Pulser */
            Pulser(pulseV,pulseT,pulseW);
           else if(choicePul==2) {        /* manual Pulser */
            PromptPopup ("Manual Pulser", " Insert Test Pulse Height (mV): ",
                         buff,10);
            pulseV=atof(buff);
            c = ConfirmPopup ("Manual Pulser",
                              "Ready to Go with Pulse Generator?");
            if(c==0)
              break;
          }

          for(VBi=0;VBi<Nvb;VBi++) {           /**** for sul Vbias ******/
            dac_bias_tmp[0+CHIP*5]=Vb_value[VBi];
            for(VCi=0;VCi<Nvc;VCi++) {         /**** for sul Vcomp ******/
              dac_bias_tmp[1+CHIP*5]=Vc_value[VCi];
              dac_bias_tmp[2+CHIP*5]=Vth_val[VBi][VCi];
              turn_on_dac();
              if(crepeat==11)
                trash=11;
               else
                trash = Irep ;
              sprintf(filepath,"%sthresh1/%s_%d_Q%d_%.2f_%.2f.dat",PATHF,fil,
                      trash,jj,dac_bias_act[0+CHIP*5]/LSBCAL,
                      dac_bias_act[1+CHIP*5]/LSBCAL);
              if(jj==0) {
                sprintf(plist,"%sthresh1/%s_%d.lst",PATHF,fil,trash);
                fpout=fopen(plist,"a");
                fprintf(fpout,"%s_%d_%.2f_%.2f\n",fil,trash,
                        dac_bias_act[0+CHIP*5]/LSBCAL,
                        dac_bias_act[1+CHIP*5]/LSBCAL);
                fclose(fpout);
              }

              printf("\n#%d,                                                    test
pulse=%.3fmV\tVbias=%.3f\tVcomp=%.3f\tVth=%.3f",
                       jj+1,pulseV,dac_bias_act[0+CHIP*5]/LSBCAL,
                       dac_bias_act[1+CHIP*5]/LSBCAL,
                       dac_bias_act[2+CHIP*5]/LSBCAL);
                    /********* scrittura della lista ************/
              sprintf(lista,"%sthresh1/%s_%d_%.2f_%.2f.lst",PATHF,fil,trash,
                      dac_bias_act[0+CHIP*5]/LSBCAL,
                      dac_bias_act[1+CHIP*5]/LSBCAL);

// Attribute must be "w"
```

```
              fpout=fopen(lista,"a");
              fprintf(fpout,"%s\n",filepath);
              fclose(fpout);

/***** for sulle righe ******/

 /* righe copiate da test */
//                trig_mode_tmp=2;
              for(I=0;I<64;I++) {
                for(i=0;i<64;i++)
                      for(j=0;j<64;j++) {
                          config_matrix[i][j]=(config_matrix0[i][j] & 0x1d);
                            if(i==I)
                              config_matrix[i][j]=config_matrix0[i][j]|2;
                          }

                    load_mask();
/****** mask is loaded **/

                start_acq(RADIOGRAPHY);     /*start the acquisition */
                if(MISSING_MEDIPIX==0)
                  download_data();

/*********** convert and save *************/
                for(j=0;j<64;j++) {
                  matrix2[I][j]=A[data_matrix[I][j]];
                  if(matrix2[I][j]>=32800)
                    printf("\nfunny number! error:%x in pixel %d %d ",
                           data_matrix[I][j],I,j);
                }

                if(jj==0 && I==0) {
                    sprintf(file0,"%sthresh1/%s_%d_Q%d_%.2f_%.2f.info",PATHF,
                            fil,trash,jj,dac_bias_act[0+CHIP*5]/LSBCAL,
                            dac_bias_act[1+CHIP*5]/LSBCAL);
                    fpout=fopen(file0,"w");
                  GetCtrlVal (panelHandle, PANEL_MASKFILE, mask_file);
                        GetCtrlVal (panelHandle, PANEL_FILENAME, arch_file);
                        fprintf(fpout,"loaded mask file = %s\n",mask_file);
                        fprintf(fpout,"loaded archive file = %s\n",arch_file);
                  fprintf(fpout,
                          "anin cycles=%d, anin period=%d, anin delay=%d\n",
                          anin_cycles_act,sleep_anin_period_act,
                          sleep_anin_delay_act);
                  fprintf(fpout,        "Pulse        Period=%.3fms,        Pulse
Width=%.1fns\n",pulseT,pulseW);
                  fprintf(fpout,      "First      Pulse      Height=%.3fmV,Pulse
Step=%.3fmV,Maximum Pulse Height=%.3fmV\n",pulseST,pulseS,pulseM);
                  fprintf(fpout,  "DAC  values  (V)  :  Vbias=%.3f,  Vcomp=%.3f,
Vth=%.3f, Vtha=%.3f,Vdl=%.3f\n",
                          dac_bias_act[0+CHIP*5]/LSBCAL,
                          dac_bias_act[1+CHIP*5]/LSBCAL,
                          dac_bias_act[2+CHIP*5]/LSBCAL,
                          dac_bias_act[3+CHIP*5]/LSBCAL,
                          dac_bias_act[4+CHIP*5]/LSBCAL);
                      fclose(fpout);
                }
```

```
// Attribute must be "w"
                fpout=fopen(filepath,"a");
                for(j=0;j<64;j++)
                  fprintf(fpout,"%d ",matrix2[I][j]);
                fprintf(fpout,"\n");
                fclose(fpout);

                        ProcessSystemEvents();
                        if(stop)
                          break;  // controllo sullo stop
              }  /***** end rows loop *****/

              ProcessSystemEvents();
                    if(stop)
                      break;    // controllo sullo stop

            }   /******** end of Vcomp loop *****/
            ProcessSystemEvents();
                if(stop)
                  break;        // controllo sullo stop

          }   /*********** end of Vbias loop *********/

          if(crepeat==11)
            printf("\nEnd  of  loop  with  pulse=%.3f  mV  (adj-bits  as  in  loaded
mask)",pulseV);
           else
            printf("\nEnd of loop with pulse=%.3f mV (adj-bits set to #%d)",
                   pulseV,Irep);

          if(choicePul == 2) {         /* Manual Pulser */
              c = ConfirmPopup ("Manual Pulser","Do you want to stop here?");
            if(c==1)
              break;
          }

          if(choicePul == 1)          /*auto Pulser*/
            pulseV = pulseV+pulseS;

          ProcessSystemEvents();
                if(stop)
                  break;        // controllo sullo stop

        } /* end pulse loop */
        if(crepeat==11)
          printf("\n\nEnd of acquisition (adj-bits as in loaded mask)");
         else
          printf("\n\nEnd of acquisition with adj-bits set to #%d\n",Irep);

            ProcessSystemEvents();
            if(stop)
              break;      // controllo sullo stop

      } /*end of repeated acquisition loop */

      printf("\n   END ");
```

```
      if(stop)
        SetCtrlVal(Vth_Cal,VTH_CAL_LED6,1);

/************** END ********************/

      break;
  }

  stop=0;
  SetCtrlAttribute(Vth_Cal,VTH_CAL_QUIT,ATTR_DIMMED,0);
  SetCtrlAttribute(Vth_Cal,VTH_CAL_RUN,ATTR_DIMMED,0);
  ProcessDrawEvents();
  return 0;
}


void Dimmed(void)
{
 SetStdioWindowSize (289, 628);
 SetStdioWindowOptions (2000, 0, 0);
 SetStdioWindowPosition (75, 61);

 SetPanelAttribute (PSelect, ATTR_DIMMED, 0);

 SetCtrlAttribute(Vth_Cal,VTH_CAL_OUTPUT,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_STOP,ATTR_DIMMED,1);

 SetCtrlAttribute(Vth_Cal,VTH_CAL_GO,ATTR_DIMMED,1);

 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM1,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM2,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM3,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM4,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM4_2,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM5,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_1,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM7,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM8,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM10,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM11,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM12,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM13,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM14,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM6,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKLOADED_MASK_FILE,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_RING1,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_RING2,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_LED1,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_LED2,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_LED3,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_LED4,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_LED5,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_LED6,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON1,ATTR_DIMMED,1);
```

```
 SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON2,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON3,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON4,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_BUTTON5,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_RUN,ATTR_DIMMED,1);
 SetCtrlAttribute(Vth_Cal,VTH_CAL_TEXTBOX1,ATTR_DIMMED,1);
 SetCtrlVal(Vth_Cal,VTH_CAL_LED1,0);
 SetCtrlVal(Vth_Cal,VTH_CAL_LED2,0);
 SetCtrlVal(Vth_Cal,VTH_CAL_LED3,0);
 SetCtrlVal(Vth_Cal,VTH_CAL_LED4,0);
 SetCtrlVal(Vth_Cal,VTH_CAL_LED5,0);
 ResetTextBox(Vth_Cal,VTH_CAL_TEXTBOX1,"");
}


int CVICALLBACK Exit (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:

                     HidePanel(Vth_Cal);
                     SetStdioWindowVisibility (0);
                                       SetCtrlAttribute(panelHandle,PANEL_TIMER,
ATTR_ENABLED,1);// abilita la funzione timer principale nel file main.c

                  break;
      }
      return 0;
}

/*****************************************************************************
****************************** fine CERN7  ***************************
*****************************************************************************/



/*****************************************************************************
****************** opzione Load&Upload del menu Mask ************************
*****************************************************************************/

void set_mask_params(char*);

void CVICALLBACK Load_Upload (int menuBar, int menuItem, void *callbackData,
            int panel)
{

   char filename[300];     // Path selezionato dall'utente
   int i,j;
   int enable_mask[64][64];
   int testbit_mask[64][64];
   int threshold_mask[64][64];


   if(FileSelectPopup (Directory_Name, "*.msk", "*.msk", "Open Mask File",
                                                VAL_LOAD_BUTTON,  0,  0,  1,  0,
filename))
```

```
      {
        Read_File_Mask(filename, enable_mask, testbit_mask, threshold_mask);
        set_mask_params(filename);
      }

   for(i=0;i<64;i++)
      {
       for(j=0;j<64;j++)
          {
               config_matrix[i][j]= ((threshold_mask[i][j]&1)<<4)+
                                    ((threshold_mask[i][j]&2)<<2)+

((threshold_mask[i][j]&4))+(testbit_mask[i][j]<<1)+
                                    enable_mask[i][j];
          }
      }

   load_mask();
}



int CVICALLBACK view (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{



//    char                                pat_error[100]="D:\\Medipix\\Wordpad.exe
D:\\Medipix\\files\\pat.error";
//    char                               mask_error[100]="D:\\Medipix\\Wordpad.exe
D:\\Medipix\\files\\mask.error";
//    char file_mask[100]="D:\\Medipix\\Wordpad.exe D:\\Medipix\\files\\";
char pat_error[100];
char mask_error[100];
char file_mask[100];

      switch (event) {
            case EVENT_COMMIT:
                  if(panel==Test_Counters)
                    {
                     sprintf(pat_error,"Wordpad.exe %spat.error",PATHF);
                     LaunchExecutable (pat_error);
                       SetCtrlAttribute (Test_Counters, COUNTERS_DECORATION_5,
ATTR_VISIBLE, 0);
                SetCtrlAttribute  (Test_Counters,  COUNTERS_VIEW,  ATTR_VISIBLE,
0);
                SetCtrlAttribute (Test_Counters, COUNTERS_TEXTMSG, ATTR_VISIBLE,
0);
               }
                   else if(panel==Test_Mask)
                     {
                      sprintf(mask_error,"Wordpad.exe %smask.error",PATHF);
                      LaunchExecutable (mask_error);
                             SetCtrlAttribute   (Test_Mask,   MASK_DECORATION_5,
ATTR_VISIBLE, 0);
                SetCtrlAttribute (Test_Mask, MASK_VIEW, ATTR_VISIBLE, 0);
```

```
                SetCtrlAttribute (Test_Mask, MASK_TEXTMSG, ATTR_VISIBLE, 0);
               }
             else if(panel==E_Mask)
               {
                sprintf(file_mask,"%s",file);
                LaunchExecutable (file_mask);
                SetCtrlAttribute  (Test_Mask,  MASK_DECORATION_5,  ATTR_VISIBLE,
0);
                SetCtrlAttribute (Test_Mask, MASK_VIEW, ATTR_VISIBLE, 0);
                SetCtrlAttribute (Test_Mask, MASK_TEXTMSG, ATTR_VISIBLE, 0);
               }
                  break;
      }
      return 0;
}

int CVICALLBACK Stop (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:

                        stop=1;

                  break;
      }
      return 0;
}

int CVICALLBACK Show_StandardInputOutput (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{

      int val;

      switch (event) {
            case EVENT_COMMIT:
                  GetCtrlVal(Vth_Cal,VTH_CAL_OUTPUT,&val);
                  if(val==1)

                    SetStdioWindowVisibility (1);

                  else if(val==0)

                    SetStdioWindowVisibility (0);

                 break;
      }
      return 0;
}


int load_mask_file (void)
{
  int ret;
  int select;    // Error Code (see "userint.h", FileSelectPopup return values)
```

```
  select = FileSelectPopup(Directory_Name, "*.msk", "*.msk", "Load a Mask File",
                           VAL_LOAD_BUTTON, 0, 0, 1, 0, maskfile);
  if (select == VAL_NO_FILE_SELECTED) {
    printf("\a");
    MessagePopup("Warning!",
                 "NO FILE SELECTED!\n Default mask file shall be loaded.");
    GetCtrlVal(panelHandle, PANEL_MASKFILE, maskfile);
  }

/*  PromptPopup("", "Load .msk file (if none - or incorrect - previous mask file
will be used):", buff, 50);

// Questa dovrebbe andar bene...
  if (buff = "")
    maskfile = mask_file_default;
   else
    sprintf(maskfile, "%s%s", PATHF2, buff);
// No? */

  if (!(ret = Read_File_Mask(maskfile, m1, m2, m3)))
    for(i = 0; i < 64; i++)
      for(j = 0; j < 64; j++)
            config_matrix[i][j] = ((m3[i][j] & 1) << 4) +
                                        ((m3[i][j] & 2) << 2) +
                                        ((m3[i][j] & 4)) +
                                        (m2[i][j] << 1) +
                                  m1[i][j];

  return ret;
}


/*****************************************************************************
***************** ANALISI ****************************************************
*****************************************************************************/


static char filename_act [30];
int ANIN;

char line [30];


void sog_b (char fil1[512], int PUL, float IMP, float STEP, int PMAX,
            int GRAF_FLAG)
{

  static int ACQ=1, THADJ=0,conteggio[200],npix[200],NPIX[200],sum;
  static int i,ii,a=0,j,k,l,m,I,R1=0,R2=63,C1=0,C2=63;
  static int soglia_mat[64][64];
  static int curva_pix[100][4096];
  static float SIGM[4096];
  static char A[1200][50];
  static char butta[150], buff[100];
  static char filename[512], lista[512], fil[512], soglia[512], pixel[512];
  static char grafico[512], plist[512], sigma[512], summa[512];
  static float h;
```

```
  FILE *fpi;
  FILE *fpo;
  char c;

     /*******************************/

  sprintf(lista,"%sthresh1/%s.lst",PATHF,fil1);
//     sprintf(plist,"%sthresh1/%s.lst",PATHF,fil1);

  if((fpi=fopen(lista,"r"))==NULL) {
    error(1);
    return;
  }

  for(I=0;I<ACQ;I++) {  /*********** for principale ************/
//    fpi=fopen(plist,"r");
//    for(i=0;i<I;i++)
//      fgets(butta,100,fpi);
//    fscanf(fpi,"%s",fil);
//    fclose(fpi);

//    sprintf(lista,"%sthresh1/%s.lst",PATHF,fil);

/**** inizializza soglia_mat e SIGM *****/
    for(i=0;i<64;i++)
      for(j=0;j<64;j++){
            soglia_mat[i][j]=0;
        SIGM[(64*i)+j]=0;
      }

/***** inizializza npix *********/

      for(k=0;k<PUL;k++) {
        npix[k]=0;
        conteggio[k]=0;
        NPIX[k]=0;
      }

// printf("\n%d\t%s\n",(ACQ-I),lista);

    fpi=fopen(lista,"r");
    for(i=0;i<PUL;i++)
      fscanf(fpi,"%s",A[i]);
    fclose (fpi);

      sprintf(soglia,"%sthresh2/%s.sgl",PATHF,fil1);
      sprintf(pixel,"%sthresh2/%s.pix",PATHF,fil1);
      sprintf(grafico,"%sthresh2/%s.graf",PATHF,fil1);
    sprintf(sigma,"%sthresh2/%s.sgm",PATHF,fil1);
    sprintf(summa,"%sthresh2/%s.sum",PATHF,fil1);
      sprintf(filename_act,"%s",fil1);

    for(k=0;k<PUL;k++) {
        fpi=fopen(A[k],"r"); /**** apre il file della lista *****/
        for(i=0;i<64;i++) {
        for(j=0;j<64;j++) {
              fscanf(fpi,"%d",&conteggio[k]);   /*carica valore del conteggio*/
```

```
      if(i>=R1 && i<=R2 && j>=C1 && j<=C2) {
        curva_pix[k][(64*i)+j]=conteggio[k];
            if(conteggio[k]>(ANIN+20)) {
                NPIX[k]++;
                if(k==0)
                  soglia_mat[i][j]=0.;
              }

            if( k==(PUL-1) && conteggio[k]==0)
              soglia_mat[i][j]=(PMAX+10)*1000;

            if(conteggio[k]>=(int)(ANIN/2) &&
               conteggio[k]<=(ANIN+20) &&
               soglia_mat[i][j]==0.)
               soglia_mat[i][j]=(int)(IMP*1000)+(int)(k*STEP*1000);
        }
      }
  }
  fclose(fpi);
}

fpo=fopen(soglia,"w");
for(i=0;i<64;i++) {
  for(j=0;j<64;j++)
  fprintf(fpo,"%.3f ",((float)(soglia_mat[i][j]))/1000.);

  fprintf(fpo,"\n");
}
fclose(fpo);

  for(k=0;k<PUL;k++)
    for(i=0;i<64;i++)
        for(j=0;j<64;j++)
          if(soglia_mat[i][j]==(int)(IMP*1000)+(int)(k*STEP*1000))
              npix[k]++;

  fpo=fopen(pixel,"w");
sum=0;
h=0.000;
  for(k=0;k<PUL;k++) {
    h=IMP+(k*STEP);
    fprintf(fpo,"%.3f\t%d\t%d\n",h,npix[k],NPIX[k]);
    sum+=npix[k];
}
fclose(fpo);

fpo=fopen(summa,"w");
fprintf(fpo,"sum of counting pixel: %d\n",sum);
fclose(fpo);

  if (GRAF_FLAG==1) {
    fpo=fopen(grafico,"w");
    for(k=0;k<PUL;k++)
      fprintf(fpo,"%d ",(int)(IMP+(k*STEP)+0.5));
    fprintf(fpo,"\n");
    for(i=0;i<4096;i++) {
        for(k=0;k<PUL;k++)
```

```
              fprintf(fpo,"%d ",curva_pix[k][i]);
            fprintf(fpo,"\n");
        }
        fclose(fpo);
      }

/* calcolo di sigma */

      for(i=0;i<4096;i++) {
        a=0;
      for(k=0;k<PUL;k++)
          if(curva_pix[k][i]>1.3 && curva_pix[k][i]<998.7)
          a++;
        SIGM[i]=(a*STEP)/6.;
/* value corresponds to 1 sigma */
      }

/* stampa di sigma */

      fpo=fopen(sigma,"w");
      for(i=0;i<64;i++) {
        for(j=0;j<64;j++)
            fprintf(fpo,"%.3f ",SIGM[(64*i)+j]);
        fprintf(fpo,"\n");
      }
      fclose(fpo);

  } /******* END of the main Loop *******/

}


void plot_distr(int nr_of_pulses)
{
   float x[100], x_max, mean, devsta;
   int y[100], y_max, buff[100];

   char input [30];
   char filename[50];
   char *pt;
   int i,j,sum_pix,sum_pix0;
   long int sum;
   FILE *fpo;


   sprintf(filename,"%sthresh2/%s.pix",PATHF,filename_act);


   if((fpo=fopen(filename,"r"))==NULL)
            {
        fclose(fpo);
            return;
            }

   else
          {
        fpo=fopen(filename,"r");
```

```
          for (i=0; i<=nr_of_pulses; i++)
              fscanf (fpo, "%f\t%d\t%d\n", &x[i], &y[i], &buff[i]);
          fclose(fpo);
            }



   y_max=0;
   sum_pix=0;
   sum=0;
   devsta=0.;
   sum_pix0=0;

   for (i=0; i<nr_of_pulses; i++) {
         if (y[i]>y_max) {
           y_max=y[i]; x_max=x[i];
         }
         sum_pix+=y[i];
         if (i>0) {                // we exclude from statistical  calculation
(mean, devsta)
                                   // the first bin of the histogram:
                                   //  it  could  include  all  pixels  that
have a threshold
                                   // lower than the first test pulse.
           sum+=x[i]*y[i];
           devsta+=y[i]*x[i]*x[i];
         }
   }
   sum_pix0 = sum_pix - y[0];

   mean = (float) sum/sum_pix0;

   devsta=devsta-sum_pix0*mean*mean;
   if (sum_pix0 == 0)
     return;

   devsta=devsta/sum_pix0;
   devsta=sqrt(devsta);

   SetCtrlAttribute (analisi, ANALISI_GRAPH, ATTR_DIMMED, 0);

   SetAxisScalingMode (analisi, ANALISI_GRAPH, VAL_LEFT_YAXIS,
                               VAL_MANUAL, 0.0, y_max);
   SetAxisScalingMode (analisi, ANALISI_GRAPH, VAL_XAXIS, VAL_MANUAL,
                               0.0, x[nr_of_pulses-1]);


   PlotXY (analisi, ANALISI_GRAPH, x, y, nr_of_pulses, VAL_FLOAT,
               VAL_INTEGER, VAL_VERTICAL_BAR, VAL_EMPTY_SQUARE, VAL_SOLID, 1,
               VAL_RED);


   SetCtrlAttribute (analisi, ANALISI_NOISE, ATTR_DIMMED, 0);
   SetCtrlAttribute (analisi, ANALISI_MEAN, ATTR_DIMMED, 0);
   SetCtrlAttribute (analisi, ANALISI_SUM, ATTR_DIMMED, 0);
   SetCtrlAttribute (analisi, ANALISI_DEV, ATTR_DIMMED, 0);
```

```
   SetCtrlVal (analisi, ANALISI_NOISE, buff[0]);
   SetCtrlVal (analisi, ANALISI_SUM, sum_pix);
   SetCtrlVal (analisi, ANALISI_MEAN, mean);
   SetCtrlVal (analisi, ANALISI_DEV, devsta);

}


void CVICALLBACK Analisi (int menuBar, int menuItem, void *callbackData,
                          int panel)
{
  char fil[512],fil0[512],fil1[512],filn[512],butta[512],c;
  static char plist[512],plist0[512],plist1[512],ext[10];
  FILE *fpi;
  FILE *fpo;
  int nr_of_meas=0,ANIN=0,line_num=0;
  float buttaf;
  float pulseS,pulseST,pulseM;
  int ldn, lfn;                  // Length of Directory Name and Length of File
Name
  int select;    // Error Code (see "userint.h", FileSelectPopup return values)
  int test, lext, *size;
  char message[100], tempchar[512];
  float Vb, Vc;

OPEN_LOAD_MASK:
  sprintf(fil0, "%s\\thresh1\\", Directory_Name);
  select=FileSelectPopup(fil0, "*.lst", "*.lst", "Open a .lst File",
                         VAL_LOAD_BUTTON, 0, 0, 1, 0, plist);
  if (select == VAL_NO_FILE_SELECTED) {
    printf("\a");
    MessagePopup("Warning!",
                 "NO FILE SELECTED!\nI shall take you back to the Main Panel.");
    return;
  }
  lfn = strlen(plist);
  ldn = (strrchr(plist, '.') - &plist[0]);
  ldn++;
  lext = lfn - ldn;
  for (i=ldn; i<=lfn; i++)
    ext[i-ldn] = plist[i];
  ext[lfn-ldn+1] = '\0';
  test = strcmp(ext,"lst\0");
  if (test) {
    strcat(plist, ".lst");
    test = GetFileInfo(plist, size);
    if (test == 0) {
      printf("\a");
      MessagePopup("Warning!",
                   "THE  FILE  YOU  SELECTED  DOES  NOT  EXIST!\nChoose  another
one.");
      goto OPEN_LOAD_MASK;
    }
  }

  ldn=(strrchr(plist,'\\')-&plist[0]);
  ldn++;
```

```
  strncpy(fil0, plist, ldn);
  fil0[ldn]='\0';
  lfn = strlen(plist) - 5;
  for (i=ldn;i<=lfn;i++)
    fil[i-ldn] = plist[i];
  fil[lfn-ldn+1]='\0';

  lfn=strlen(fil);
  for (i=lfn-4; i<lfn; i++)
    tempchar[i-lfn+4]=fil[i];
  tempchar[4]='\0';
  Vc=atof(tempchar);
  for (i=lfn-9; i<lfn-5; i++)
    tempchar[i-lfn+9]=fil[i];
  Vb=atof(tempchar);

  SetCtrlVal (analisi, ANALISI_DATA_DIR, fil0);
  SetCtrlVal (analisi, ANALISI_FILENAME, fil);
  SetCtrlVal (analisi, ANALISI_V_bias, Vb);
  SetCtrlVal (analisi, ANALISI_V_comp, Vc);

  DisplayPanel (analisi);

//               OPERAZIONIIIIIII
// *    sprintf(plist,"%sthresh1/%s.lst",PATHF,fil);
//    fscanf(fpi,"%s",plist1);
//    fclose(fpi);
//    sprintf(plist0,"%sthresh1/%s.lst",PATHF,plist1);

//*    fpi=fopen(plist,"r");

  fpi=fopen(plist,"r");
  fscanf(fpi,"%s",fil0);
  fscanf(fpi,"%s",fil1);
  while (fscanf(fpi,"%s",filn) != EOF)
    nr_of_meas++;
  fclose(fpi);
  nr_of_meas+=2;

  memset(filn,'\0',strlen(fil0));
  strncpy(filn,fil0,strlen(fil0)-3);
  strcat(filn,"info");

  fpi=fopen(filn,"r");
  while ((c = getc(fpi)) != EOF) {
    if (c == '\n')
      line_num++;
  }
  fclose(fpi);

  fpi=fopen(filn,"r");
  if (line_num > 3)
    for (i=1; i<=2; i++)
      while ((c = fgetc(fpi)!='\n'));         //Salta una linea
  fgets(butta,13,fpi);
  fscanf(fpi,"%d",&ANIN);
  while ((c = fgetc(fpi)!='\n'));
```

```
  switch (line_num) {
    case 3: case 5:
      fgets(butta,14,fpi);
        fscanf(fpi,"%f",&buttaf);
      fgets(butta,17,fpi);
        fscanf(fpi,"%f",&buttaf);
      fgets(butta,18,fpi);
        fscanf(fpi,"%f",&pulseST);
        fclose(fpi);

      memset(filn,'\0',strlen(fil1));
        strncpy(filn,fil1,strlen(fil1)-3);
        strcat(filn,"info");

        fpi=fopen(filn,"r");
        if (line_num > 3)
            for (i=1; i<=2; i++)
          while ((c = fgetc(fpi)!='\n'));     //Salta una linea
      while ((c = fgetc(fpi)!='\n'));
      fgets(butta,14,fpi);
        fscanf(fpi,"%f",&buttaf);
      fgets(butta,17,fpi);
        fscanf(fpi,"%f",&buttaf);
      fgets(butta,18,fpi);
        fscanf(fpi,"%f",&pulseS);
        fclose(fpi);

        pulseS=pulseS-pulseST;
        pulseM=pulseST+pulseS*(nr_of_meas-1);
      break;
      case 6:
        while ((c = fgetc(fpi)!='\n'));
        fgets(butta,20,fpi);
        fscanf(fpi,"%f",&pulseST);
        fgets(butta,15,fpi);
        fscanf(fpi,"%f",&pulseS);
        fgets(butta,25,fpi);
        fscanf(fpi,"%f",&pulseM);
        fclose(fpi);
      break;
  }

  SetCtrlAttribute (analisi, ANALISI_PULSER_STEP, ATTR_DIMMED, 0);
  SetCtrlAttribute (analisi, ANALISI_PULSER_MIN, ATTR_DIMMED, 0);
  SetCtrlAttribute (analisi, ANALISI_PULSER_MAX, ATTR_DIMMED, 0);
  SetCtrlAttribute (analisi, ANALISI_NUMBER, ATTR_DIMMED, 0);
  SetCtrlAttribute (analisi, ANALISI_PULSES_ANIN, ATTR_DIMMED, 0);

  SetCtrlVal (analisi, ANALISI_PULSER_STEP, pulseS);
  SetCtrlVal (analisi, ANALISI_PULSER_MIN, pulseST);
  SetCtrlVal (analisi, ANALISI_PULSER_MAX, pulseM);
  SetCtrlVal (analisi, ANALISI_NUMBER, nr_of_meas);
  SetCtrlVal (analisi, ANALISI_PULSES_ANIN, ANIN);

}
```

```
int CVICALLBACK start_analisi (int panel, int control, int event,
                                    void *callbackData, int eventData1,
                                    int eventData2)
{
  char fil [512];
  float pulseS,pulseST,pulseM;
  int nr_of_meas;
  int GRAF_FLAG;

  switch (event) {
      case EVENT_COMMIT:
        GetCtrlVal (analisi, ANALISI_FILENAME, fil);
        GetCtrlVal (analisi, ANALISI_PULSER_STEP, &pulseS);
        GetCtrlVal (analisi, ANALISI_PULSER_MIN, &pulseST);
      GetCtrlVal (analisi, ANALISI_PULSER_MAX, &pulseM);
        nr_of_meas = (int)(1+(pulseM-pulseST)/pulseS);
        SetCtrlVal (analisi, ANALISI_NUMBER, nr_of_meas);
        GetCtrlVal (analisi, ANALISI_GRAF_FLAG, &GRAF_FLAG);
        GetCtrlVal (analisi, ANALISI_PULSES_ANIN, &ANIN);
//GetCtrlVal (analisi, ANALISI_DATA_DIR, &PATHF);

      SetWaitCursor (1);
      sog_b (fil, nr_of_meas, pulseST, pulseS, pulseM, GRAF_FLAG);
      plot_distr(nr_of_meas);
        SetWaitCursor (0);
        break;
  }
  return 0;
}


int CVICALLBACK quit (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event)
            {
            case EVENT_COMMIT:
                     HidePanel(analisi);
                  break;
            }
      return 0;
}

/*****************************************************************************
*************** Threshold Adjustment Calculation ***************************
*****************************************************************************/

void thres_adjus(void)
{

int MAXTHR=110;   /*IMP+(PUL-1)*STEP; */
float tha[8][64][64];
int m1[64][64],m2[64][64],m3[64][64];
FILE *filein;
FILE *fileout;
int i,j,k,NOISY_PIX=0,DEAD_PIX=0;
```

```
float Vbias,Vcomp;
char file[50], file0[50], filemask[50],filemask0[50];
float Vmed,dist,mindist;
int vthamask[64][64];
char c,trash[300];

printf("\nThis program processes 8 files .sgl with the following format:");
printf("\n prefix_##.sgl , where ## goes from 0 to 7");

printf("\ninsert input mask file (only enable part is passed in output):");
gets(trash);
sprintf(filemask,"%s%s",PATHF,trash);

printf("\ninsert output mask file :");
gets(trash);
sprintf(filemask0,"%s%s",PATHF,trash);

printf("\ninsert prefix :");
gets(file0);

printf("\ninsert Vbias : ");
Vbias=atof(gets(trash));
printf("\ninsert Vcomp : ");
Vcomp=atof(gets(trash));

printf("\ninsert center of threshold distribution (mV): ");
Vmed=atof(gets(trash));

printf("\n max test pulse (mV):");
MAXTHR =atoi(gets(trash));

MAXTHR=MAXTHR+10;


   filein=fopen(filemask,"r");

      for(i=0;i<64;i++)
      {
            for(j=0;j<64;j++)
            {
                  m1[i][j]=0;
                  m2[i][j]=0;
                  m3[i][j]=0;
            }
      }


      for (i=1; i<=2; i++) {
         while ((c = fgetc(filein)!='\n')) { }      //Salta una linea
      }


    for(i=0;i<64;i++)
            {
                  for(j=0;j<64;j++)
                  {
                        fscanf(filein,"%d",&m1[i][j]);
```

```
                        /* gets the enable_bit_mask*/
                }
        }

      for (i=1; i<=3; i++) {
         while ((c = fgetc(filein)!='\n')) { }      //Salta una linea
      }


    for(i=0;i<64;i++)               /* gets the test_bit_mask*/
            {

                  for(j=0;j<64;j++)
                  {
                        fscanf(filein,"%d",&m2[i][j]);
                  }

            }


      for (i=1; i<=3; i++) {
         while ((c = fgetc(filein)!='\n')) { }      //Salta una linea
      }


    for(i=0;i<64;i++)           /* gets the threshold_bit_mask*/
            {

                  for(j=0;j<64;j++)
                  {
                        fscanf(filein,"%d",&m3[i][j]);
                  }

            }

  fclose(filein);



 for(i=0;i<64;i++)
      {
      for(j=0;j<64;j++)
        {
          vthamask[i][j]=8;
        }
      }

for(k=0;k<8;k++)
  {
sprintf(file,"%sthresh2/%s_%d_%.2f_%.2f.sgl",PATHF,file0,k,Vbias,Vcomp);


    filein=fopen(file,"r");
    for(i=0;i<64;i++)
      {
      for(j=0;j<64;j++)
        {
```

```
          fscanf(filein,"%f",&tha[k][i][j]);
        }
      }
      fclose(filein);
  }

for(i=0;i<64;i++)
      for(j=0;j<64;j++)
            for(k=0;k<=7;k++)
                  {
                  dist= tha[k][i][j]-Vmed;
                  if(dist < 0) dist=-dist;
                  if (k==0) mindist = dist;
                  if (dist <= mindist)
                  {
                  mindist = dist;
                  vthamask[i][j]=k;
                  }
                  if(k==7 && tha[k][i][j]==0.)   m1[i][j]=1;

                  }


for(i=0;i<64;i++)
      for(j=0;j<64;j++)
                  {
                  NOISY_PIX=NOISY_PIX + m1[i][j];
                  if(tha[0][i][j] >= ((float)MAXTHR)) DEAD_PIX++;
                  }

printf("\nNoisy pixels=%d, Dead pixels=%d",NOISY_PIX,DEAD_PIX);

fileout=fopen(filemask0,"w");
                  fprintf(fileout,"* enable mask *\n");
                  fprintf(fileout,"\n");
                  for(i=0;i<64;i++)
                        {
                              for(j=0;j<64;j++)
                                          fprintf(fileout,"%d ",m1[i][j]);
/* write the enable_bit_mask*/
                              fprintf(fileout,"\n");
                        }

                  fprintf(fileout,"\n");
                  fprintf(fileout,"* test mask *\n");
                  fprintf(fileout,"\n");
                  for(i=0;i<64;i++)
                        {

                              for(j=0;j<64;j++)
                                    {
                                          fprintf(fileout,"%d ",m2[i][j]);
                                                /* write the test_bit_mask*/
                                    }
                              fprintf(fileout,"\n");
                        }
```

```
                  fprintf(fileout,"\n");
                  fprintf(fileout,"* threshold mask *\n");
                  fprintf(fileout,"\n");
                  for(i=0;i<64;i++)
                        {

                              for(j=0;j<64;j++)
                                    {
                                          fprintf(fileout,"%d ",vthamask[i][j]);
                                          /* write the threshold_bit_mask*/
                                    }
                              fprintf(fileout,"\n");
                        }
            fclose(fileout);

printf("\nEnd of mask generating function ");
                while((c=getchar())==(int)NULL){}

}




void   CVICALLBACK   Thres_Equalization   (int   menuBar,   int   menuItem,   void
*callbackData,
            int panel)
{
 thres_adjus();
}


void filtro(void)
{
float med=0.,sig=0., dist=0.;
int count1[64][64];
float r_count1[64][64];
int tot1=0;
float r_tot1=0;
char fil10[50], fil20[50];


FileSelectPopup (Directory_Name, "*.dat", "*.dat", "Load File to be filtered",
                                                      VAL_LOAD_BUTTON,  0,  0,
1, 0, fil10);

FileToArray (fil10, count1, VAL_INTEGER, 4096, 64,
                              VAL_GROUPS_TOGETHER,        VAL_GROUPS_AS_COLUMNS,
VAL_ASCII);

   for(i=0;i<64;i++)
      for(j=0;j<64;j++)
               {
                r_count1[i][j]=(float)(count1[i][j]);
                tot1=tot1+count1[i][j];
                r_tot1=r_tot1+r_count1[i][j];
               }
```

```
        for(i=0;i<64;i++)
      for(j=0;j<64;j++)
        {
            med=0.;
            sig=0.;
            dist=0.;
                  if (i>0 & i<63 & j>0 & j<63)
                  {
                        med=r_count1[i-1][j-1]+r_count1[i][j-1]+
                                                r_count1[i+1][j-1]+r_count1[i-
1][j]+
                                                  r_count1[i+1][j]+r_count1[i-
1][j+1]+

r_count1[i][j+1]+r_count1[i+1][j+1];
                        med=(med/8.);
                        sig= (r_count1[i-1][j-1]-med)*(r_count1[i-1][j-1]-med)+
                               (r_count1[i][j-1]-med)*(r_count1[i][j-1]-med)+
                                   (r_count1[i+1][j-1]-med)*(r_count1[i+1][j-1]-
med)+
                               (r_count1[i-1][j]-med)*(r_count1[i-1][j]-med)+
                               (r_count1[i+1][j]-med)*(r_count1[i+1][j]-med)+
                                   (r_count1[i-1][j+1]-med)*(r_count1[i-1][j+1]-
med)+
                               (r_count1[i][j+1]-med)*(r_count1[i][j+1]-med)+
                                   (r_count1[i+1][j+1]-med)*(r_count1[i+1][j+1]-
med);

                        sig=sqrt(sig/8.);
                        dist=  r_count1[i][j]-med;
                        if (dist < 0) dist=-dist;
                        if( dist > (2*sig) | count1[i][j] == 0)
                              {
                              count1[i][j]=(int)(med*tot1/r_tot1);
                              r_count1[i][j]=med;
                              }
                  }
               }

FileSelectPopup (Directory_Name, "*.dat", "*.dat", "Filtered output file",
                                                        VAL_SAVE_BUTTON,  0,  0,
1, 0, fil20);


ArrayToFile (fil20, count1, VAL_UNSIGNED_INTEGER, 4096,
                  64, VAL_GROUPS_TOGETHER, VAL_GROUPS_AS_COLUMNS,
                  VAL_SEP_BY_TAB, 10, VAL_ASCII, VAL_TRUNCATE);

}

void CVICALLBACK LP_filtro (int menuBar, int menuItem, void *callbackData,
            int panel)
{
```

```
 filtro();
}



void thr_adj_bits_set(void)
{

                        GetCtrlVal(Vth_Cal,VTH_CAL_NUM5,&crepeat);
                        mrep=crepeat;
                        Mrep=crepeat;

SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM5,ATTR_CTRL_MODE,VAL_INDICATOR);

SetCtrlAttribute(Vth_Cal,VTH_CAL_CHECKNUM5,ATTR_CTRL_MODE,VAL_INDICATOR);

}


void Thr_Adj_Bits_Set(void)
{

                         GetCtrlVal(Vth_Cal,VTH_CAL_NUM6_1,&mrep);

SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_1,ATTR_CTRL_MODE,VAL_INDICATOR);
                         GetCtrlVal(Vth_Cal,VTH_CAL_NUM6_2,&Mrep);

SetCtrlAttribute(Vth_Cal,VTH_CAL_NUM6_2,ATTR_CTRL_MODE,VAL_INDICATOR);

}

void adj_loaded_msk_file(void)
{

                SetCtrlVal(Vth_Cal,VTH_CAL_LED1,1);
                crepeat=11;
                mrep=0;
                      Mrep=0;
}


int CVICALLBACK Sel_Voltage (int panel, int control, int event,
            void *callbackData, int eventData1, int eventData2)
{
      switch (event) {
            case EVENT_COMMIT:
                    Select_Voltage();
                  break;
      }
      return 0;
}

int CVICALLBACK load_thr_name (int panel, int control, int event,
                                    void *callbackData, int eventData1,
                                    int eventData2)
{
  switch (event) {
```

```
      case EVENT_COMMIT:
      HidePanel(analisi);
      Analisi(MENU_BASIC, MENU_BASIC_analysis, callbackData, panel);
      break;
  }
  return 0;
}

int CVICALLBACK New_Data_File (int panel, int control, int event,
                               void *callbackData, int eventData1,
                                 int eventData2)
{
  int r;

  switch (event) {
      case EVENT_LEFT_CLICK:
      r = Load_Data_File();
      break;
  }
  return r;
}


int Load_Data_File()
{
  char fil[512], fil0[512];
  static char plist[512];
  int ldn, lfn;                  // Length of Directory Name and Length of File
Name
  int select;    // Error Code (see "userint.h", FileSelectPopup return values)
  char message[100];

  sprintf(fil0, "%s\\", Directory_Name);
  select = FileSelectPopup(fil0, "*.dat", "*.dat", "Open a Data File",
                           VAL_LOAD_BUTTON, 0, 0, 1, 0, plist);
  if (select==VAL_NO_FILE_SELECTED) {
    printf("\a");
    MessagePopup("Warning!",
                 "NO  FILE  SELECTED!\nI  shall  take  you  back  to  the  previous
Panel.");
    return 1;
  }
  ldn = strrchr(plist, '\\') - &plist[0];
  ldn++;
  strncpy(fil0, plist, ldn);
  fil0[ldn] = '\0';
  lfn = strlen(plist);
  for (i = ldn; i <= lfn; i++)
    fil[i - ldn] = plist[i];

  SetCtrlVal (Pixels_Cal, PIXELS_CAL_STRING1, fil);
  return 0;
}

int CVICALLBACK New_Back_File (int panel, int control, int event,
                               void *callbackData, int eventData1,
                                 int eventData2)
```

```
{
  int r;

  switch (event) {
      case EVENT_LEFT_CLICK:
        r = Load_Back_File();
        break;
  }
  return r;
}


int Load_Back_File()
{
  char fil0[512], fil1[512];
  static char plist[512];
  int ldn, lfn;                  // Length of Directory Name and Length of File
Name
  int select;    // Error Code (see "userint.h", FileSelectPopup return values)
  char message[100];

  sprintf(fil0, "%s\\", Directory_Name);
  select = FileSelectPopup(fil0, "*.dat", "*.dat", "Open a Background File",
                           VAL_LOAD_BUTTON, 0, 0, 1, 0, plist);
  if (select==VAL_NO_FILE_SELECTED) {
    printf("\a");
    MessagePopup("Warning!",
                 "NO  FILE  SELECTED!\nI  shall  take  you  back  to  the  previous
Panel.");
    return 1;
  }
  ldn = strrchr(plist, '\\') - &plist[0];
  ldn++;
  strncpy(fil0, plist, ldn);
  fil0[ldn] = '\0';
  lfn = strlen(plist);
  for (i = ldn; i <= lfn; i++)
    fil1[i-ldn] = plist[i];

  SetCtrlVal (Pixels_Cal, PIXELS_CAL_STRING2, fil1);
  return 0;
}
```