UNIVERSITÀ DEL SALENTO
Facoltà di Ingegneria
Corso di Laurea Magistrale in
Ingegneria delle Telecomunicazioni

TESI DI LAUREA

SVILUPPO DEL SISTEMA DI ACQUISIZIONE PER UN ESPERIMENTO DI OUTREACH

Relatore:
Chiar.mo Prof. **Stefano D'Amico**
Chiar.mo Prof. **Marco Panareo**

Correlatore:
Chiar.mo Dott. **Alessandro Corvaglia**

Laureando:
Gianluigi Chiarello

ANNO ACCADEMICO 2011/2012

Alla mia famiglia,
che mi ha sostenuto,
aiutato e incoraggiato
in ogni momento,
e ha affrontato tanti
sacrifici per consentirmi di
raggiungere questo traguardo.

"Mi chiamarono pazzo nel 1896 quando annunciai la scoperta di raggi cosmici. Ripetutamente si presero gioco di me e poi, anni dopo, hanno visto che avevo ragione. Ora presumo che la storia si ripeterà quando affermo che ho scoperto una fonte di energia finora sconosciuta, un'energia senza limiti, che può essere incanalata."

Nikola Tesla

"La teoria è quando si sa tutto ma non funziona niente. La pratica è quando funziona tutto ma non si sa il perché. In ogni caso si finisce sempre con il coniugare la teoria con la pratica: non funziona niente e non si sa il perché."

Albert Einstein

INDICE

INTRODUZIONE

Il 2012 è l'anno in cui ricorre il centenario della scoperta dei raggi cosmici da parte di Victor Hess e Domenico Pacini. I raggi cosmici sono particelle di alta energia provenienti dallo spazio esterno, alle quali è continuamente esposta la Terra e qualunque altro corpo celeste. La loro scoperta, originata per spiegare la radioattività osservata sulla Terra, ha posto la base per lo sviluppo della fisica delle particelle.

In occasione di tale ricorrenza, l'Università del Salento, l'INFN (Istituto Nazionale di Fisica Nucleare) di Lecce e l'ASI (Agenzia Spaziale Italiana) in collaborazione con alcune Scuole Superiori del Salento hanno sviluppato un esperimento di outreach denominato CORAM [7] (Cosmic Ray Mission) per ripetere con strumenti moderni le misure effettuate da V. Hess nel 1912.

L'obiettivo della tesi è la realizzazione del sistema di acquisizione (DAQ, Data Acquition System) per questo esperimento. Il DAQ acquisisce i dati provenienti dai rivelatori di raggi cosmici, li elabora tramite una FPGA (Field Programmable Gate Array), opportunamente programmata attraverso il codice VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language), e li trasmette ad un microcontrollore. Il microcontrollore, una volta ottenute le informazioni dalla FPGA, gestisce le informazioni temporali e di posizione provenienti da un GPS (Global Positioning System), quelle di temperatura provenienti da opportuni sensori e quelle dell'orientazione dei rivelatori rispetto allo zenit e al nord, ottenute tramite un magnetometro e accelerometro; infine salva tutte le informazioni elaborate su una SD-CARD per poi inviarle al mondo esterno.

Si sono sviluppati due tipi di DAQ:

- Il primo, utilizzato per acquisire dati da quattro rivelatori, è stato sviluppato con delle demoboard allo scopo di verificare il corretto funzionamento del microcontrollore e della FPGA. Con questo primo prototipo è stata eseguita una campagna di misure presso Campo Imperatore (L'Aquila) e nei Laboratori Nazionali del Gran Sasso (LNGS) dell'INFN.
- Il secondo DAQ, sviluppato per acquisire dati da otto rivelatori, costituisce il dispositivo finale del progetto.

Scoperta dei raggi cosmici

I raggi cosmici [1] sono particelle di alta energia (100 eV - 100 GeV) che arrivano interrottamente sulla Terra e sono costituiti principalmente da elettroni, protoni e fotoni.

Le particelle di cui sono formati i raggi cosmici provengono dallo spazio extraterreste e sono originate dalle stelle (anche il sole genera raggi cosmici che arrivano sulla Terra), supernove, quasar e vengono successivamente accelerati ad alta energia attraverso opportuni meccanismi astrofisici.

I raggi cosmici vengono assorbiti e deviati dall'atmosfera terrestre; a causa degli urti con le particelle che compongono l'atmosfera, questi ne generano altre con un processo a cascata dando origine ad uno sciame che procede verso la superficie terrestre.

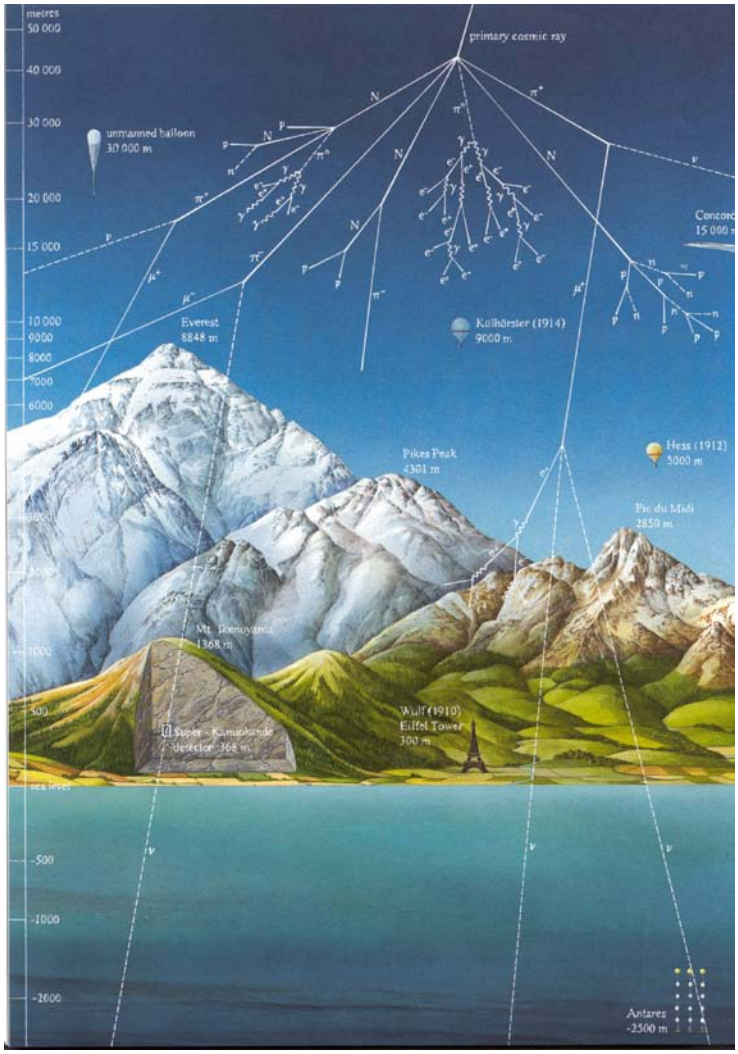


Figura 1:Generazione di uno sciame quando il raggio cosmico interagisce con l'atmosfera

Nel 1785 Charles Coulomb, per primo, osservò che un elettroscopio si scaricava spontaneamente in aria. Fu Micheal Faraday che in seguito studiò questo fenomeno, notando che il tempo di scarica aumentava al diminuire della pressione nel punto in cui erano presenti le foglie dell'elettroscopio. Da queste osservazioni egli dedusse che la scarica dell'elettroscopio era dovuta alla ionizzazione delle molecole presenti nell'aria.

La spiegazione del fenomeno della ionizzazione dell'aria si ebbe un secolo dopo, grazie alla scoperta della radioattività naturale da parte di Antoine Henri Becquerel e Pierre e Marie Curie che identificarono degli elementi soggetti a decadimenti radioattivi. L'elettroscopio carico in presenza di materiale radioattivo si

scarica più velocemente e ciò è dovuto alle ionizzazioni prodotte dalle particelle emesse dai materiali radioattivi. Nel 1899 Julius Elster e Hans Geitel scoprirono che gli elementi ionizzanti che provocavano la scarica, erano esterni al recipiente contenente l'elettroscopio, per cui si pensò che tale radiazione fosse dovuta ad elementi radioattivi presenti sulla terra.

Negli anni successivi, i risultati di diversi esperimenti suggerirono che la radiazione fosse di origine extraterrestre. Inoltre, attraverso questi esperimenti gli scienziati migliorarono gli strumenti di misura. In particolare Theodor Wulf realizzò un elettroscopio ad alta sensibilità che successivamente venne perfezionato da Hermann Ebert [1].

Figura 2:Elettroscopio di Wulf

Nel 1911 Domenico Pacini, per capire l'origine della radiazione, svolse delle misure di velocità di scarica dell'elettroscopio sott'acqua [1-2]. Dalle rilevazioni effettuate egli notò che le misure subacquee fornirono velocità di scarica inferiori rispetto a quelle effettuate in superficie; per cui la radiazione non poteva essere dovuta agli elementi radioattivi presenti sulla crosta terrestre. Questo esperimento, tuttavia, non fornisce la spiegazione dell'origine delle radiazioni, ma scartava solo una ipotesi, poiché l'origine della radiazione, sebbene comunque extraterrestre, poteva essere atmosferica. Per risolvere quest'ultimo dilemma si dovette aspettare l'anno successivo, quando Victor Hess fece alcune misure da un pallone aerostatico.

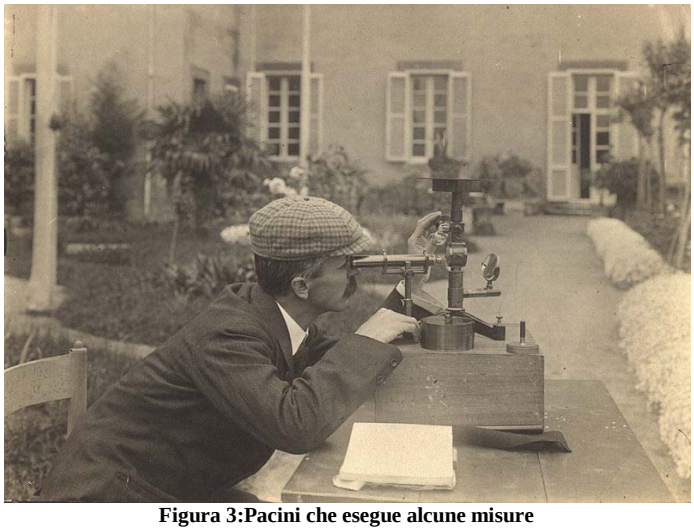


Figura 3:Pacini che esegue alcune misure

V. Hess effettuò diversi voli sui palloni aerostatici per osservare la variazione di densità della ionizzazione con la quota. Il volo decisivo avvenne nell'agosto del 1912 quando raggiunse i 5200 metri e poté osservare un aumento della densità concludendo che una radiazione altamente penetrante di origine extraterrestre entra nell'atmosfera terrestre producendo la ionizzazione [1-3]. Ciò portava ad escludere, inoltre, che la sorgente principale di questa radiazioni fosse il sole, in quanto non c'erano sostanziali variazioni tra le misure effettuate di giorno e quelle di notte. Per questa scoperta, nel 1936, gli fu assegnato il premio Nobel.

Figura 4:Hess in partenza per uno dei voli

Questi risultati furono confermati successivamente da Werner Kolhörster che eseguì dei voli fino alla quota di 9300 metri, verificando un aumento della densità delle ionizzazioni dieci volte superiore rispetto al livello del mare.

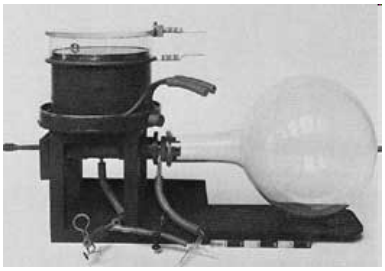


Figura 5: Camera a nebbia

Nel 1932 Carl Anderson, studiando la radiazione di origine cosmica, scoprì l'anti-materia e in particolare l'anti-elettrone, il positrone, utilizzando una camera a nebbia, cioè un dispositivo inventato da Charles Thomas Wilson che permetteva di rilevare la traiettoria di particelle cariche. Successivamente, sempre tramite i raggi cosmici, furono scoperte molteplici particelle. Attualmente si effettuano diversi esperimenti che studiano i raggi cosmici ad altissima energia, come quelli realizzati presso l'osservatorio di Auger, in Argentina, o Argo, in Tibet (Cina).

CAPITOLO 1

CORAM (COSMIC RAY MISSION)

1.1 L'esperimento

In occasione del centenario della scoperta della provenienza extraterrestre della radiazione ionizzante, l'INFN di Lecce e il Dipartimento di Matematica e Fisica dell'Università del Salento hanno proposto di ripetere, in chiave moderna, le misure effettuate da Victor Hess attraverso una attività di outreach e didattica. In questo progetto, la cui sigla è CORAM (Cosmic RAy Mission), sono stati coinvolti molteplici studenti degli Istituti di Istruzione Superiore del Salento.

Figura 1.1:Logo dell'esperimento CORAM.

Le misure effettuate da Victor Hess erano incentrate sulla misurazione del flusso delle particelle ionizzanti al variare dell'altitudine. L'andamento caratteristico del flusso viene identificato con il Pfotzer plot (Fig. 1.2), dal nome di uno dei fisici

che, per primi negli anni ’30, condussero una serie di esperimenti con palloni atmosferici usando strumentazione costituita da rivelatori di particelle messi in coincidenza, intervallati da opportuni strati di assorbitore.

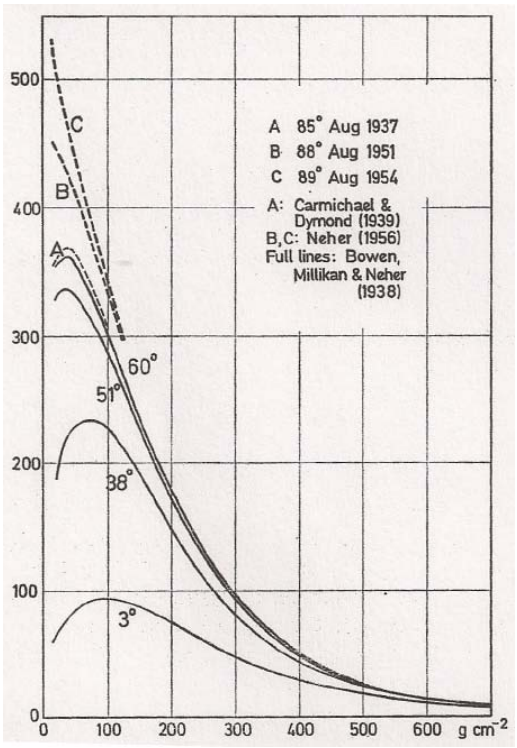


Figura 1.2:Pfotzer Plot

Dal Pfotzer plot [4] si nota che il flusso di particelle ionizzanti in atmosfera presenta un massimo ed un successivo assorbimento di tipo esponenziale (eccetto per i neutrini e per i muoni, a causa delle loro caratteristiche di interazione e/o decadimento e per i protoni, data la composizione del fascio di primari). La posizione del massimo di Pfotzer dipende dal tipo di componente del flusso (γ, e-, ecc.), dalla latitudine, dalla soglia di rivelazione e dal range angolare coperto, le varie componenti del flusso mostrano un massimo intorno alla quota di circa 18 km (100÷150 g/cm^2 di profondità atmosferica).

Lo scopo di CORAM è quello di condurre un esperimento simile a quelli effettuati da Victor Hess per ottenere l'andamento caratteristico del flusso di particelle ionizzanti ottenuto da G. Pfotzer; per far ciò si deve progettare, realizzare e testare un setup sperimentale in cui il rivelatore è costituito da un telescopio di scintillatori.

Per effettuare le misure ad alta quota, il DAQ sarà collocato a bordo di un pallone atmosferico fornito dall'ASI (Agenzia Spaziale Italiana). Il pallone può essere lanciato da diverse basi, localizzate a latitudini molto diverse, questo potrebbe fornirci i dati per lo studio degli effetti geomagnetici sui raggi cosmici. Il vettore per il trasporto del rivelatore oltre che dal pallone in polietilene è composto da :

- Un paracadute per il recupero del payload.
- Una navicella per il trasporto del payload e dei sottosistemi di volo.
- Una Catena di volo (Fig. 1.3), che rappresenta il sistema di connessione meccanica ed elettrica tra la navicella ed il pallone

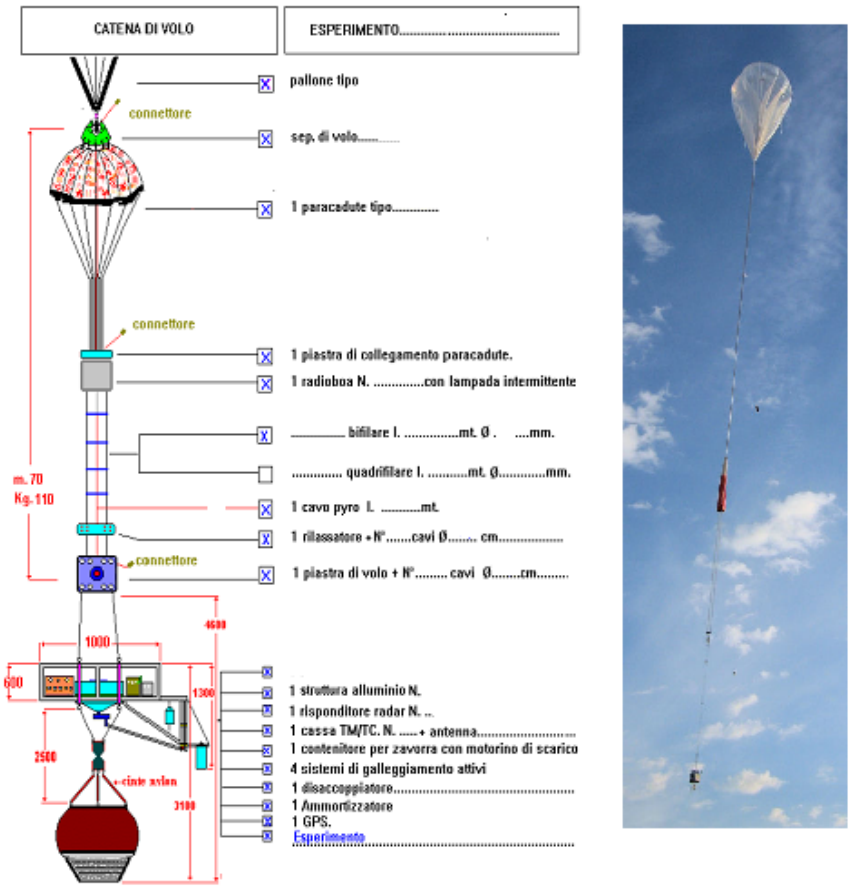


Figura 1.3:Catena di volo

La navicella, che ospita il rivelatore e il DAQ, sarà realizzata anche in relazione agli altri ospiti e in accordo con responsabili dei diversi sottosistemi di volo accomodati sulla stessa navicella. I principali sottosistemi (S/S) sono:

- S/S di potenza:
    - batterie con celle primarie al litio
- S/S di telemetria/telecontrollo:
    - trasmettitori di telemetria a modulazione di fase per il down-link con portanti RF a 400,17 MHz e 401,16 MHz e un encoder PCM (Pulse Code Modulation);
    - ricevitori supereterodina con discriminatori per FM sintonizzati sulla frequenza 444 MHz per l'up-link
    - antenne di trasmissione e di ricezione;
- S/S di zavorra motorizzata per il controllo della quota.
- S/S di localizzazione e di identificazione del pallone:
    - ricevitori GPS per la localizzazione in tempo reale;
    - radioboa, per la localizzazione della navicella in fase di discesa.
- S/S di separazione del pallone dal carico.
- S/S per l'atterraggio e il recupero:
    - sistema di galleggiamento;
    - sistema di smorzamento per l'atterraggio.

Figura 1.4:Struttura meccanica della navicella

Il setup dell'esperimento, alloggiato sulla navicella, deve interagire con un alcuni di questi sottosistemi perciò, prima del lancio, si deve verificare l'integrazione meccanica-elettronica con tutto il sistema. Il pallone può essere lanciato da differenti siti consentendo varie possibilità di misura. In particolare, è possibile effettuare il lancio da Trapani Milo per effettuare un volo locale della durata di circa 4 ore, oppure, un volo trans-mediterraneo con recupero in Spagna o, ancora, un lancio dalle isole Svalbard in Norvegia. In quest'ultima circostanza la traiettoria di volo sarebbe circolare e permetterebbe una misura a basso valore del taglio geomagnetico[1] ed in funzione della longitudine.

Una volta lanciato, il pallone viene controllato da terra per mezzo del collegamento radio; alcuni canali di comunicazione sono adoperati per la localizzazione e per la gestione operativa del volo, mentre i rimanenti sono a disposizione per la gestione e trasmissione dei dati del payload.

Il lavoro qui presentato si inserisce in CORAM nell'ambito della progettazione e sviluppo del sistema di acquisizione (DAQ Data Acquisition System) che ha quale principale obiettivo quello di fornire e salvare le informazioni sul flusso dei raggi cosmici da analizzare successivamente. Una prima analisi dei dati acquisiti viene eseguita in real time tramite un programma realizzato con la piattaforma LabView.

[1] Il taglio geomagnetico è l'effetto del campo magnetico terrestre che impedisce ai raggi cosmici a bassa energia di arrivare sulla superficie terrestre, deflettendoli fuori dall'atmosfera

Figura 1.5:Pallone atmosferico dell'ASI

1.2 Il rivelatore

Per misurare il flusso dei raggi cosmici si utilizzano degli scintillatori. Lo scintillatore è costituito da un materiale che emette luce quando è attraversato da una particella ionizzante; l'emissione di luce è dovuta all'eccitazione degli atomi costituenti il materiale scintillante. La luce prodotta è raccolta da una fibra ottica che, mediante trasduttori a semiconduttore, è convertita in un segnale di corrente.

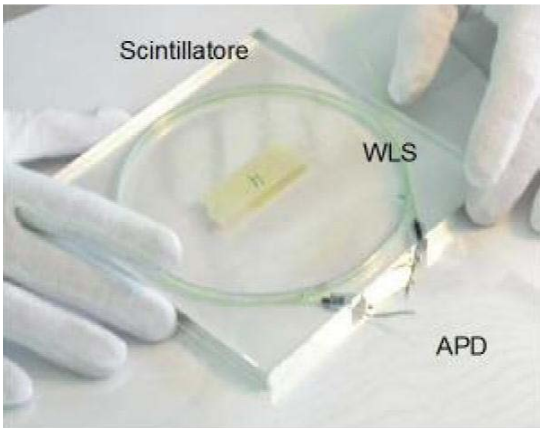


Figura 1.6:APD, WLS e materiale scintillante

Gli scintillatori usati per il rivelatore dell'esperimento sono piani di materiale plastico *BC-412 [8-10]*. Per raccogliere i fotoni prodotti dal materiale scintillante e trasportarli verso i dispositivi a semiconduttore si utilizzano due fibre ottiche (WLS wavelength-shifting) del diametro di circa 1 mm. Per migliorare la sensibilità dello scintillatore le fibre ottiche sono inserite in un percorso a spira circolare. Ciascuna

fibra ad una estremità è accoppiata al dispositivo a semiconduttore e all'altra termina con un materiale riflettente. Per semplificare il montaggio delle due WLS (Fig.1.6) il materiale scintillante plastico è fresato per ottenete una scanalatura circolare.

I trasduttori a semiconduttore che utilizza lo scintillatore sono APD (Avalanche Photo-Diodes), dispositivi che sfruttano l'effetto a valanga per la rivelazione della radiazione luminosa. Gli APD (Fig. 1.6) per convertire la luce in corrente utilizzano una matrice di microcelle di area 1 mm^2 con struttura MRS (Metal-Resistance-Semiconductor). La struttura MRS è formata da un substrato di silicio di tipo *p*. La parte del substrato collegata al contatto elettrico viene ulteriormente drogata per ottenere un substrato fortemente drogato ($p+$) per avere una riduzione della resistenza elettrica che si forma tra il metallo del contatto e il substrato. Sul substrato di tipo *p* viene depositato uno strato di silicio drogato $n+$, che forma una giunzione *p*-*n* in cui può essere generato il forte campo elettrico necessario a generare valanghe locali. A causa della foto-ionizzazione della microcella, un'ulteriore generazione di valanghe è originata dalla caduta di tensione dovuta allo strato resistivo che si trova sotto l'elettrodo metallico superiore. La conversione della luce avviene quando il fotone colpisce la struttura MRS con una energia sufficiente a creare una coppia elettrone-lacuna libera che, per effetto valanga, creerà un numero maggiore di coppie elettrone-lacuna, elettroni che formeranno la corrente associata al segnale ottico.

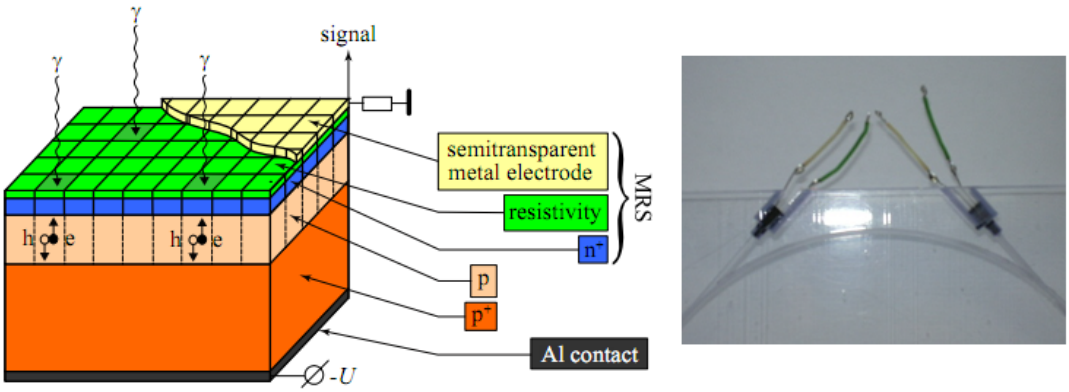


Figura 1.7:A sinistra:Struttura MRS. A destra: ADP e WLS

L'uso delle fibre WLS è necessario per effettuare uno shifting della lunghezza d'onda allo scopo di migliorare la sensibilità del rilevatore in quanto il materiale scintillante ha un massimo di emissione della radiazione alla lunghezza d'onda di 430 nm (blu), mentre la sensibilità spettrale degli APD è massima lunghezze d'onda superiori a 520 nm (verde). Negli scintillatori si utilizzano due APD per diminuire il rumore di background, infatti questo rumore utilizzando un unico APD è di circa 1-10 KHz con una soglia fissata a 3-4 fotoelettroni. Utilizzando due APD con una finestra di coincidenza tra i due di 10 - 20 ns il rumore di backgroud si riduce a circa 0,01 Hz, una quantità trascurabile.

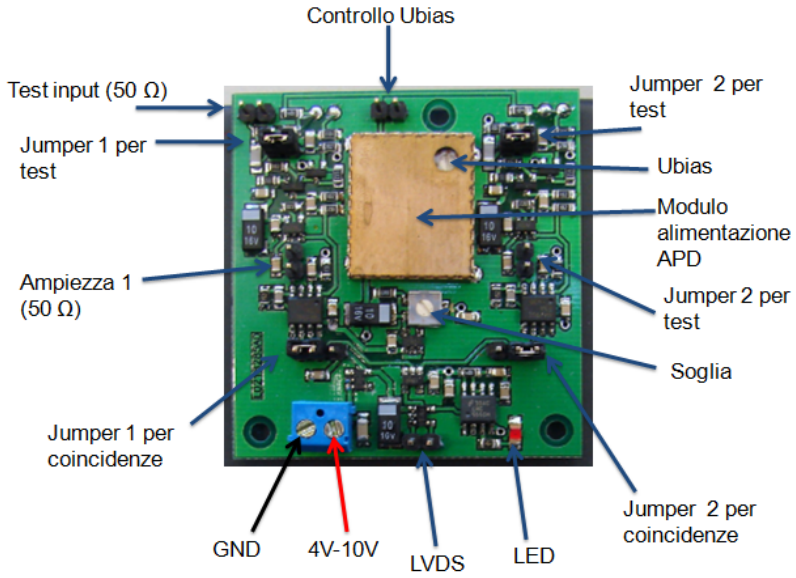


Figura 1.8FEE dello scintillatore

Agli APD è collegato un front-end (Fig. 1.8) per l'elaborazione del segnale che amplifica gli impulsi prodotti e li discrimina con una soglia regolabile. Dopo la discrimina dei segnali il front-end verifica la coincidenza tra i due APD con una porta NAND e li digitalizza in formato ECL[2] o LVDS[3] con durata di 60 ns.

[2] Eemitter-Coupled Logic è una logica non saturata per applicazioni ad alta velocità.

[3] Low-Voltage Differential Signaling è uno standard digitale differenziale ad alta velocità.

Figura 1.9:Scintillatore con front end

Gli scintillatori utilizzati per il rivelatore hanno dimensione 30 cm x 30 cm x 1 cm e densità media pari a 1,032 g/cm^3 (BC-412) (in Fig. 1.9 è mostrato un prototipo del singolo piano di rivelazione). Il rivelatore (Fig. 1.10) è costituito da otto scintillatori di cui quattro intervallati con piani di ferro spesso 2 cm e piani di *foan plastico* di 1 cm posti orizzontalmente in una struttura a telescopio; altri quatto posizionati sulle facce laterali del rivelatore.

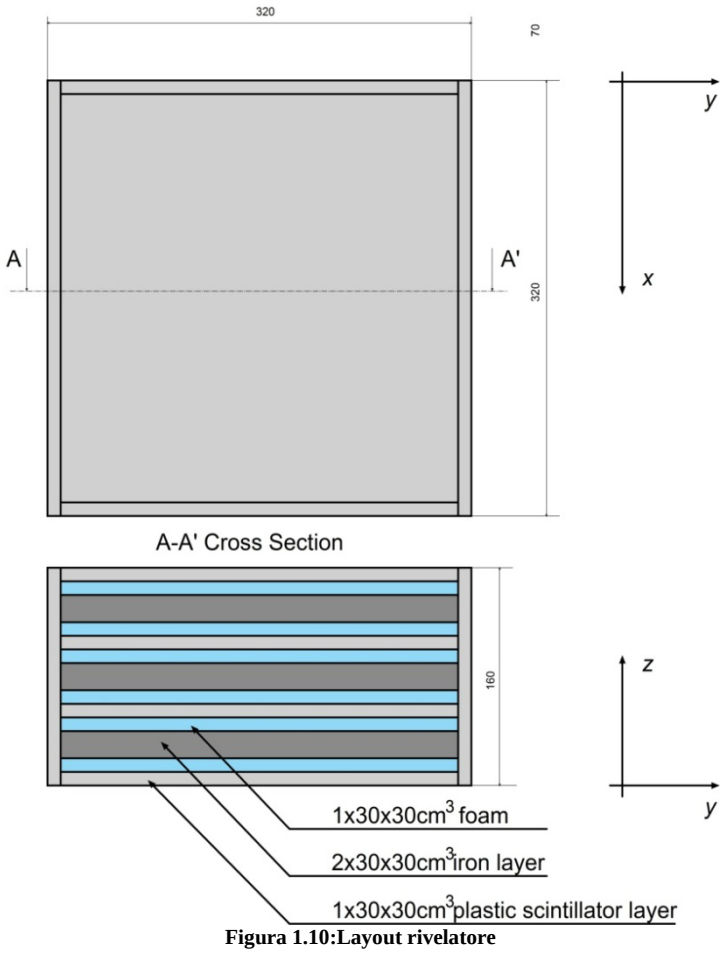


Figura 1.10:Layout rivelatore

Il setup scelto per il rivelatore permette di avere sufficiente stabilità, di rilevare il flusso orizzontale di raggi cosmici tramite gli scintillatori laterali e di intercettare particelle con energia minima intorno a 100 MeV per un m.i.p (Minimum Ionizing Particle) tramite una coincidenza quadrupla. La massa del rivelatore è di circa 60 Kg.

Prima dell'assemblaggio del rivelatore, gli scintillatori devono essere calibrati agendo sulla soglia per massimizzare l'efficienza. Per effettuare questa operazione si inseriscono tre scintillatori a telescopio (Fig. 1.11), lo scintillatore centrale sarà quello da calibrare e i due scintillatori esterni con soglia pre-impostata in base al conteggio di singola.

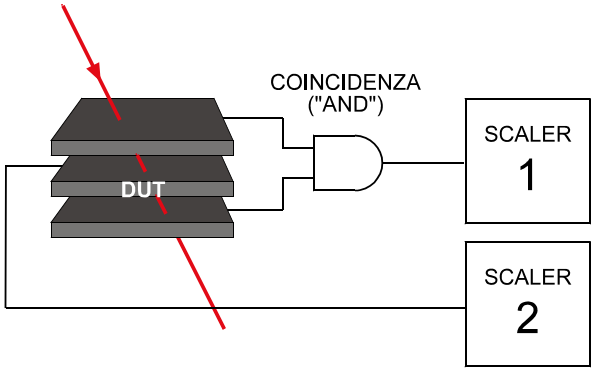


Figura 1.11:Struttura per la taratura dello scintillatore

Per calcolare l'efficienza dello scintillatore sotto test si calcola il rapporto tra il conteggio di coincidenza dello scintillatore da calibrare (C1) e il conteggio di coincidenza degli scintillatori esterni (C2).

$$Efficenza\% = \frac{C1}{C2} \times 100$$

L'efficienza è stata misurata al variare della tensione di soglia del discriminatore dello scintillatore sotto test,(in Fig. 1.12: è riportato efficienza e il rate di singola in funzione della Tensione di soglia). La tensione di soglia scelta, in questo caso, è 130 mV (Fig. 1.12: linea rossa) che garantisce una buona efficienza (Fig. 1.12: linea nera) e il conteggio di singola (Fig. 1.12: linea blu) in accordo con il valore atteso dal Pfotzer plot sul livello del mare.

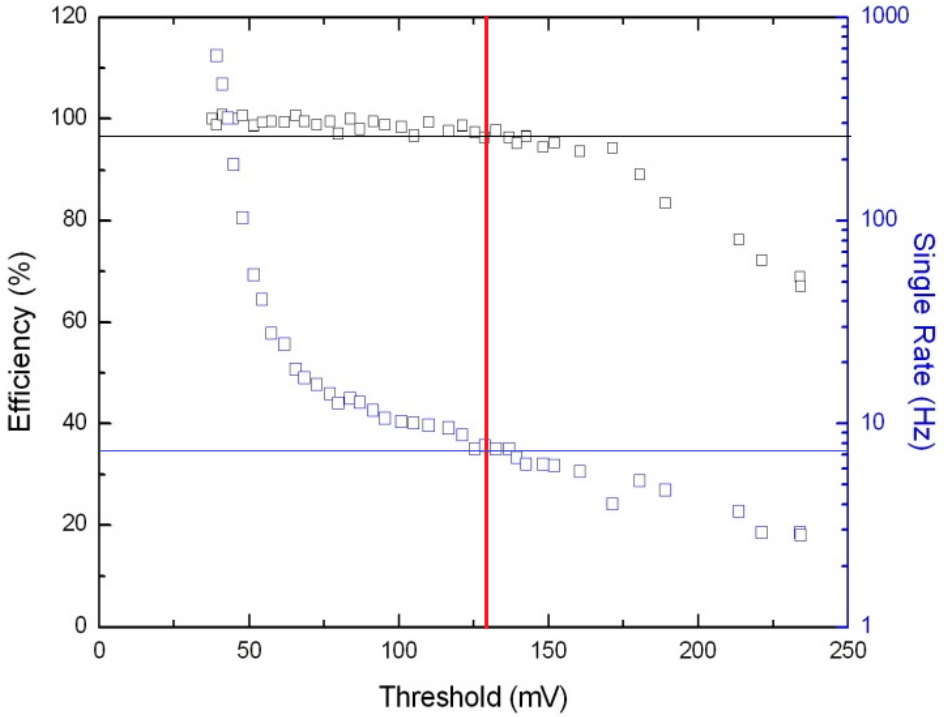


Figura 1.12:Efficienza-Rate vs Tensione di soglia

1.3 Il Data AcQuisition system (DAQ)

Lo scopo principale del DAQ è quello di fornire le informazioni sul flusso dei raggi cosmici, ottenute mediante il conteggio delle singole (il numero di volte che lo scintillatore rileva una particella in un intervallo di tempo prestabilita) e delle coincidenze degli scintillatori (due o più scintillatori che rilevano contemporaneamente le particelle). Per avere una maggiore versatilità del dispositivo il conteggio viene effettuato tramite una FPGA, in cui vengono implementati i contatori necessari per considerare tutte le possibili coincidenze tra i diversi scintillatori. L'incremento dei contatori viene effettuato dopo aver valutato il tipo di coincidenza. L'FPGA invia le informazioni accumulate sul flusso dei raggi cosmici al microcontrollore con una frequenza definibile da remoto (tramite il microcontrollore e la telemetria del pallone). L'FPGA utilizzata per il DAQ è la Xilinx SPARTAN 3E 250k [24-26].

Il microcontrollore inizia il processo di acquisizione delle informazioni quando riceve il segnale dall'FPGA e associa alle informazioni ottenute i dati provenienti dal GPS (data, ora, latitudine, longitudine e altitudine). Il GPS, oltre a fornire informazioni sulla quota del rivelatore e informazioni temporali sul conteggio, ci offre la possibilità di studiare gli effetti geomagnetici della terra sui raggi cosmici come l'effetto latitudine, dovuto al fatto che siccome i raggi cosmici sono formati da particelle cariche (scoperta attribuita a Arthur Holly Compton nel 1930), il loro flusso diminuisce avvicinandosi all'equatore. Il microcontrollore oltre a comunicare con l'FPGA comunica con un altro microcontrollore per acquisire informazioni sulla temperatura ambientale e sul rollio del rilevatore. Quest'ultimo dato è importante per analizzare l'effetto est-ovest (scoperto da Bruno Rossi nel 1930) dovuto al campo magnetico terrestre. Questo effetto, conseguenza dell'azione della forza di Lorentz sulle particelle fa si che il flusso dei raggi cosmici è maggiore lungo la direzione da ovest ad est, in quanto costituito in prevalenza di particelle positive. Le informazioni relative alla temperatura servono per avere un controllo più completo del DAQ.

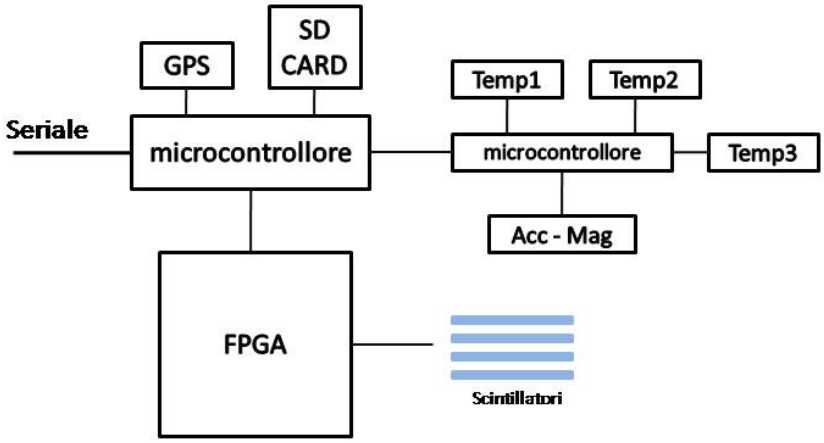


Figura 1.13:Schema generale del DAQ

Il microcontrollore principale, quello che comunica con l'FPGA, dopo aver raccolto tutte le informazioni, effettua il salvataggio locale su SD CARD e le invia al sotto sistema di telemetria del pallone tramite una porta seriale con un baudrate di 9600 baud che le trasmetterà alla stazione di controllo a terra. Si è scelto come microcontrollore principale il PIC18F87J50 [18] della Microchip e come

microcontrollore secondario il PIC16F1847 [19] della Microchip. Il motivo delle scelte dei componenti verrà descritto nel paragrafo successivo.

I tre componenti descritti rappresentano il cuore del DAQ. In seguito si vedrà come comunicano tra di loro e con il S/S di telemetria del pallone.

Il DAQ, per essere alimentato, è collegato al S/S di potenza che fornisce una tensione di 12 V a tutti gli esperimenti presenti nel payload del pallone; da questa tensione tramite quattro regolatori, si ottengono le tensioni necessarie per il corretto funzionamento del DAQ. In particolare l'FPGA lavora con le tensioni 1.2 V, 2.5 V e 3.3 V, mentre il rivelatore, il microcontrollore principale, il GPS e l'accelerometro-magnetometro lavorano con una tensione di 3.3 V infine il microcontrollore secondario e i sensori di temperatura lavorano con una tensione di 5 V.

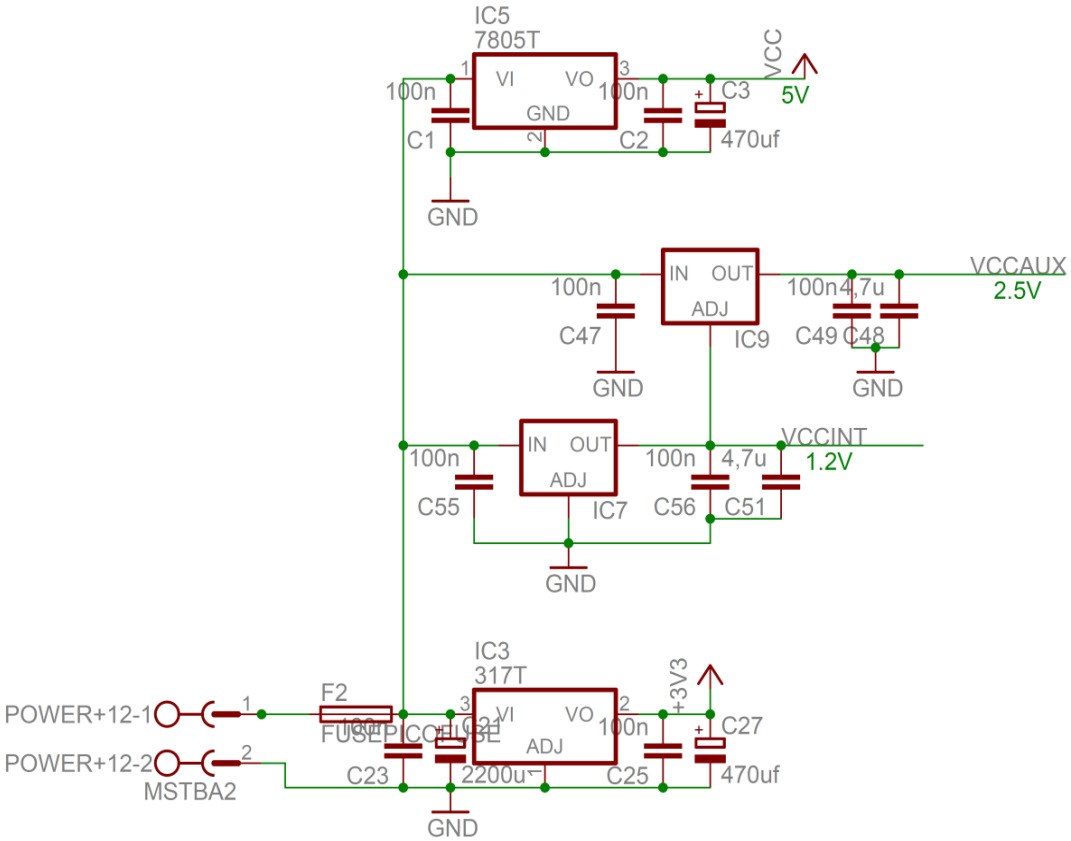


Figura 1.14:Sistema di alimentazione tramite regolatori di tensione

Per ottenere le informazioni sui raggi cosmici l'FPGA è collegata al rivelatore mediante otto bus LVDS (un bus per ogni scintillatore). Questo tipo di bus, per poter

lavorare, necessita di una resistenza di terminazione da 100 Ω (Fig. 1.15). Sullo stesso bus in cui è presente il segnale LVDS è presente l'alimentazione dello scintillatore e la massa. Come si nota dalla Figura 1.15, nel DAQ si è previsto un numero superiore di ingressi LVDS per avere maggior flessibilità nel setup del rivelatore (si può pensare di costruire un rivelatore con un massimo di 10 scintillatori senza cambiare DAQ) e per avere degli ingressi liberi in caso di malfunzionamento di uno degli altri ingressi LVDS.

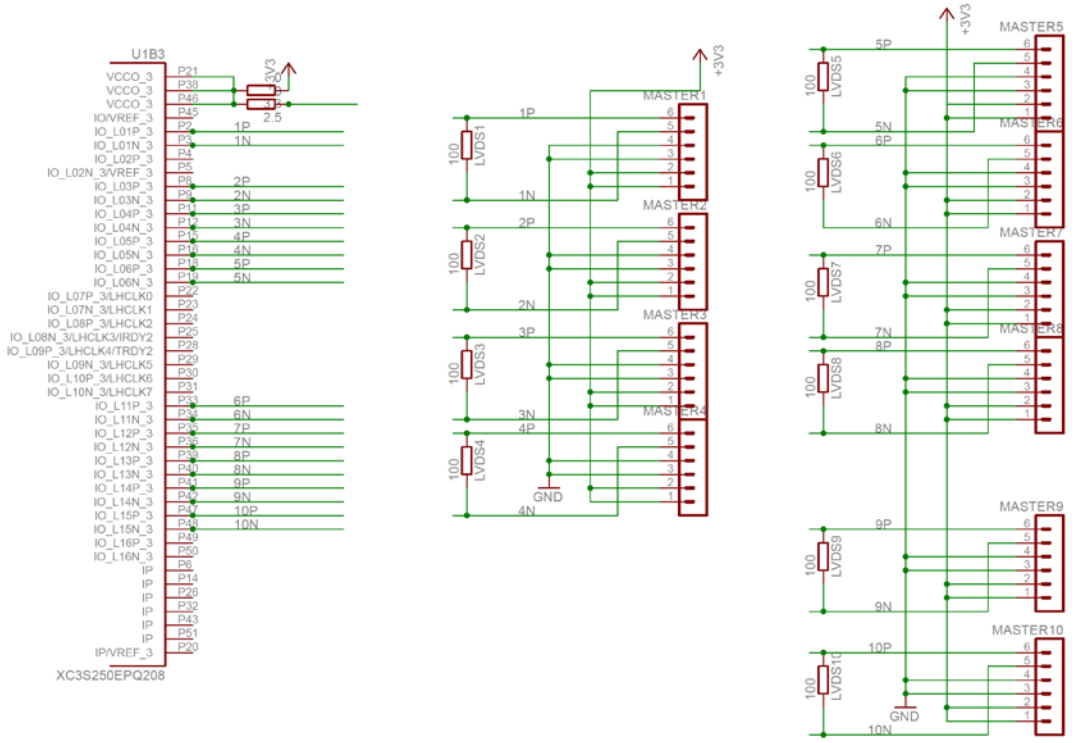


Figura 1.15:Collegamento LVDS tra rivelatore e FPGA

Per l'FPGA, oltre agli I/O[4] di collegamento con il microcontrollore principale, sono previsti altre otto linee di I/O di cui quattro, configurate nel firmware come uscite di singola, doppia, tripla e quadrupla degli scintillatori della struttura telescopica del rivelatore e le altre quattro come ingresso di trigger per altri rivelatori (per maggiori informazioni si veda Capitolo 2); le uscite possono essere riconfigurate per nuove esigenze riprogrammando l'FPGA.

[4] I/O: Input/Output

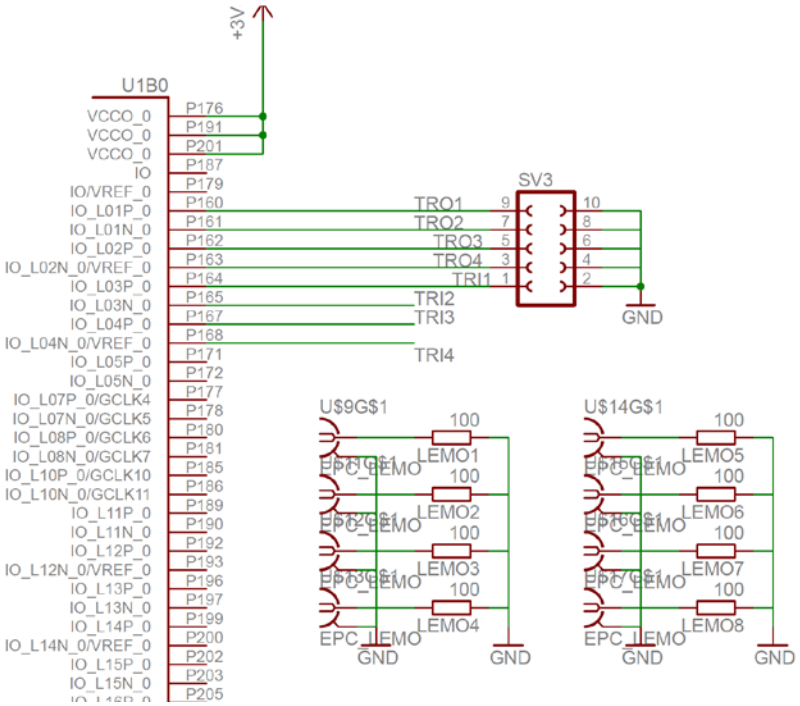


Figura 1.16:Uscite di Trigger

L'FPGA è una matrice di celle composte da flip-flop e porte logiche; in questa matrice agisce il firmware che connette tra di loro le varie celle a seconda della loro funzione. Quando l'FPGA viene spenta si perdono le informazioni sulle connessioni delle celle per cui alla riaccensione questa deve essere riprogrammata. La riprogrammazione è automatica se nel circuito del DAQ si prevede una memoria PROM (Programmable Read Only Memory) in cui si salva il firmware. La PROM scelta, compatibile con l'FPGA, è l'XCF04 della Xilinx con 4 KB di memoria per il firmware (Fig. 1.17).

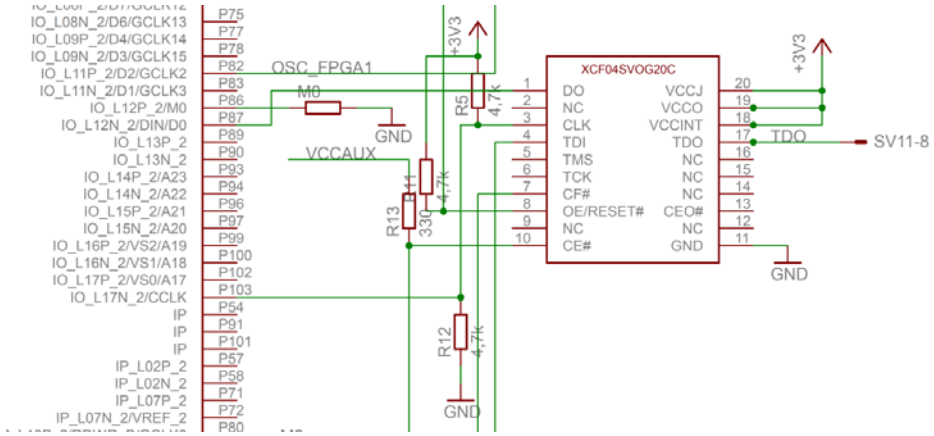


Figura 1.17:Collegamento PROM e FPGA

La connessione tra microcontrollore principale e FPGA avviene tramite un bus di sei linee: tre linee (Fig 1.18: riquadro rosso) sono utilizzate per trasmettere all'FPGA la frequenza di invio dei dati e quindi la finestra temporale di conteggio del flusso dei raggi cosmici, impostata da remoto tramite la porta seriale; le altre tre linee del bus di connessione (Fig 1.18: riquadro blu) sono utilizzate per implementare il protocollo di comunicazione sincrono tra FPGA e microcontrollore. Il protocollo, che verrà analizzato in maniera dettagliata nel prossimo capitolo, oltre alla linea dati e alla linea di clock per il trasferimento ne utilizza una terza controllata dall'FPGA che fornisce al microcontrollore il segnale per iniziare la procedura di acquisizione dati. Nella figura sottostante si è evidenziato il connettore e le linee di programmazione del microcontrollore

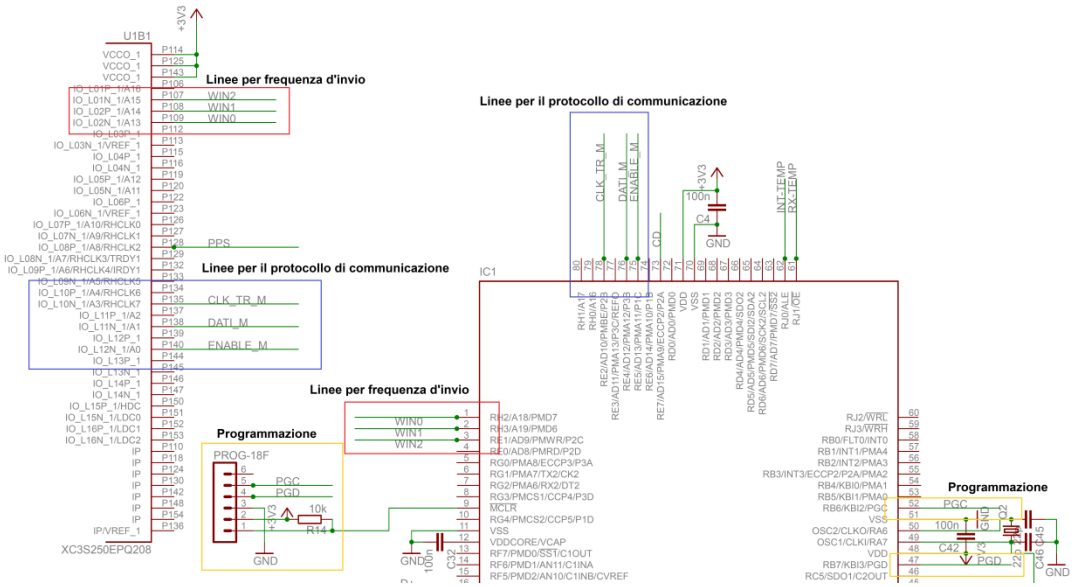


Figura 1.18:Connessione tra microcontrollore principale e FPGA

Il microcontrollore principale, dopo aver ricevuto le informazioni dell'FPGA, comunica con gli altri dispositivi per ottenere tutti i dati necessari (time stamping, posizione, rollio del rivelatore....) associati al flusso delle particelle.

Il primo dispositivo con cui il microcontrollore principale comunica mediante una porta seriale hardware è il GPS; il GPS adoperato è iQ Lassen (Fig. 1.19) della Trimble [20] provvisto di due porte seriali per la comunicazione: la prima utilizza il

protocollo TSIP[5] (Trimble Standard Interface Protocol) con un baud rate di 9600 con controllo di parità, mentre la seconda porta che utilizza il protocollo NMEA 0183[6] (National Marine Electronics Association) con un baud rate di 4800 senza controllo di parità.

Figura 1.19:GPS iQ Lassen

Il ricevitore GPS per essere operativo in minor tempo (si risparmiano circa dieci secondi), è dotato di una batteria di backup per conservare in memoria il setting del ricevitore, le informazioni temporali e la posizione dei satelliti corrispondenti all'ultimo utilizzo.

Per avere una maggiore flessibilità, nella progettazione del DAQ si è previsto il collegamento di entrambe le porte seriali con la porta seriale utilizzata nel microcontrollore. Solo in fase di assemblaggio del DAQ si decide la porta da utilizzare e la si collega alla porta seriale del microcontrollore attraverso resistenza da 0 Ω (Fig. 1.20: riquadro blu). Il GPS oltre alle informazioni temporali e di posizione permette la sincronizzazione tra i diversi dispositivi tramite il PPS (Pulse per Second), che è un segnale con frequenza di 1 Hz (Fig. 1.20: riquadro rosso) con un errore di circa 3 ns.

[5] TSIP è un protocollo proprietario della Trimble, i suoi pacchetti sono trasmessi in formato binario.

[6] NMEA è uno standard industriale gestito dalla National Marine Electronics Association e utilizzato generalmente nella industria marittima e i pacchetti sono codificati in ASCII.

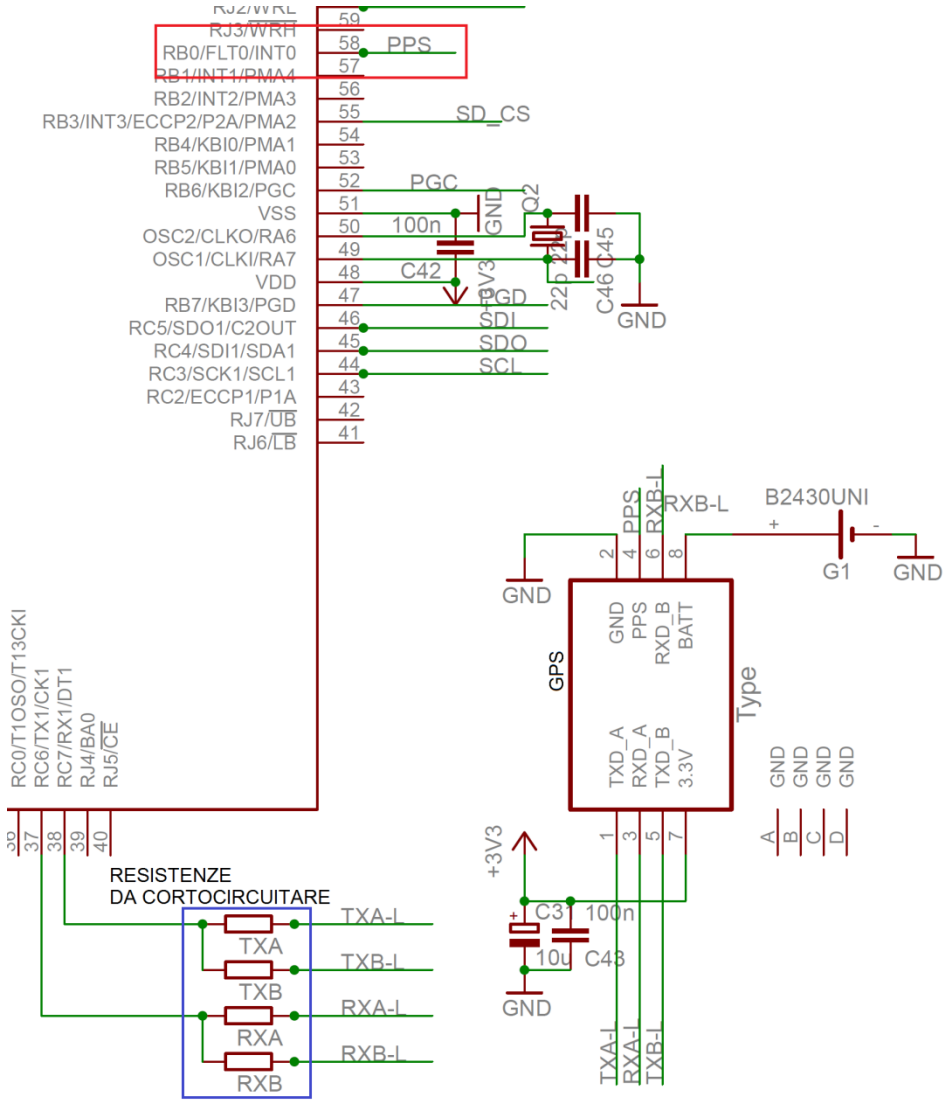


Figura 1.20:Connessione tra microcontrollore e GPS e ingresso PPS

Dopo aver ottenuto le informazioni dal GPS il microcontrollore principale comunica con quello secondario per ottenere le informazioni sul rollio e sulla temperatura del DAQ e del rivelatore; la comunicazione tra i due microcontrollori avviene tramite una porta seriale software[7] con baudrate di 9600 baud (Fig 1.23: riquadro viola). È inoltre prevista un'ulteriore linea di comunicazione di interrupt per

[7] La porta seriale software è molto utilizzata nei microcontrollori perché questi hanno un numero limitato e non sempre sufficiente di porte hardware. Le porte seriali software sono implementate utilizzando i normali I/O (GPIO) del microcontrollori e il firmware per la gestione degli interrupt.

trasmettere al microcontrollore principale la disponibilità delle che le informazioni dei sensori.

Il microcontrollore secondario interagisce con dei sensori di temperatura mediante il protocollo 1-wire[8]. I sensori di temperatura utilizzati sono i DS18B20 (Fig. 1.21: il sensore e in Fig. 1.23: lo schema, riquadro rosso) della Maxim IC [21], questo lavora in un range di temperatura tra -55°C e 125°C con un'approssimazione di ±0.5°C e codifica le informazioni in 12 bit.

Figura 1.21Sensore DS18B20

Sono stati utilizzati tre sensori diversi di temperatura per monitorare la temperatura interna e esterna del crate del DAQ (il DAQ è inserito in un contenitore (crate) che verrà montato sulla navicella del pallone) e la temperatura rilevata nel punto in cui saranno situati i rivelatori. Queste informazioni non sono strettamente legate alla statistica del flusso dei raggi cosmici ma servono solo per il controllo del funzionamento del DAQ.

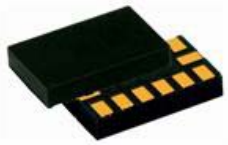


Figura 1.22:LSM303DLHC

Il microcontrollore secondario, oltre a comunicare con i sensori di temperatura, comunica anche con un accelerometro-magnetometro; mediante le informazioni di accelerazioni e del campo magnetico si ricavano in fase di analisi dati e informazioni sul rollio e sull'orientamento del rivelatore. Il sensore utilizzato è l'LSM303DLHC (Fig. 1.22: il sensore e in Fig. 1.23 lo schema, riquadro blu) dell'ST

[8] 1-wire è un protocollo di comunicazione seriale asincrono half-duplex sviluppato dalla Dallas Semiconductor

Microelectronics [22] che funziona sia come accelerometro a tre assi con range tra ±8.1 Gauss che come magnetometro a tre assi con range tra ±16 g. Per la comunicazione, esso utilizza il protocollo[9] I^{2}C con pacchetti di 20 bit; il sensore comunica con il microcontrollore con una linea dedicata di interrupt (Fig. 1.23: INT_ACC) e una linea dedicata che segnala la presenza di dati (Fig. 1.23: READYACC)

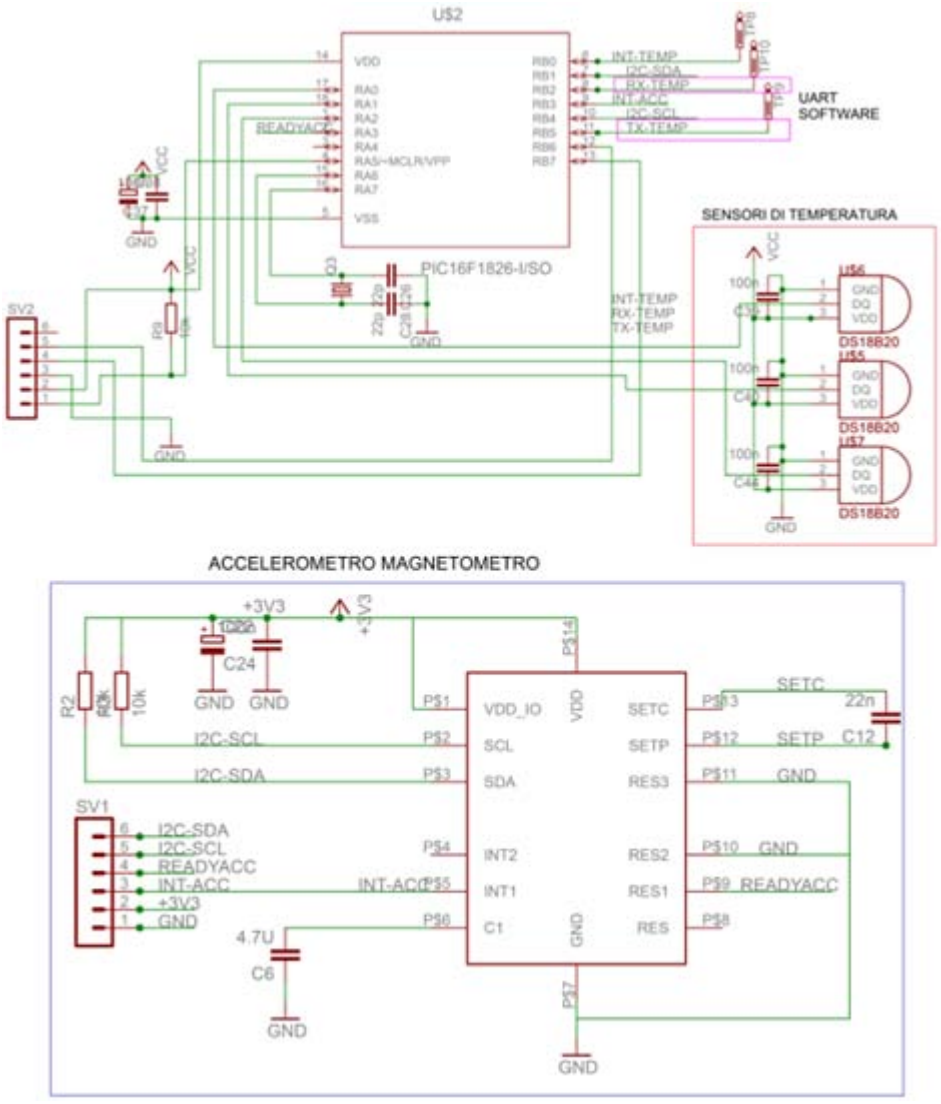


Figura 1.23Collegamenti microcontrollore secondario e sensori di temperatura

[9] I^{2}C (Inter Integrated Circuit) è un protocollo di comunicazione seriale sincrono che utilizza una linea di clock (SCL) e una linea dati hal-duplex (SDA). il protocollo si basa sulla comunicazione tra un master e slave

Il microcontrollore secondario, dopo aver interrogato tutti i sensori e ricevuto le relative informazioni le trasmette, in seguito ad una richiesta di interrupt, al microcontrollore principale, tramite la porta seriale.

Il microcontrollore principale, ricevuti i dati, li accoda a quelli del flusso dei raggi cosmici e del GPS e li invia al sottosistema di telemetria del pallone, tramite la porta seriale (Fig. 1.24: riquadro blu) o a ad un computer per eseguire l'analisi dati real-time, tramite la porta USB (Fig. 1.24: riquadro rosso). Quando è utilizzata la porta USB, quella seriale viene disabilitata dal software poiché che il DAQ non è montato sul pallone. Per effettuare la comunicazione seriale con il sotto sistema di telemetria è inserito un adattatore MAX3232 (Fig. 1.24: riquadro blu).

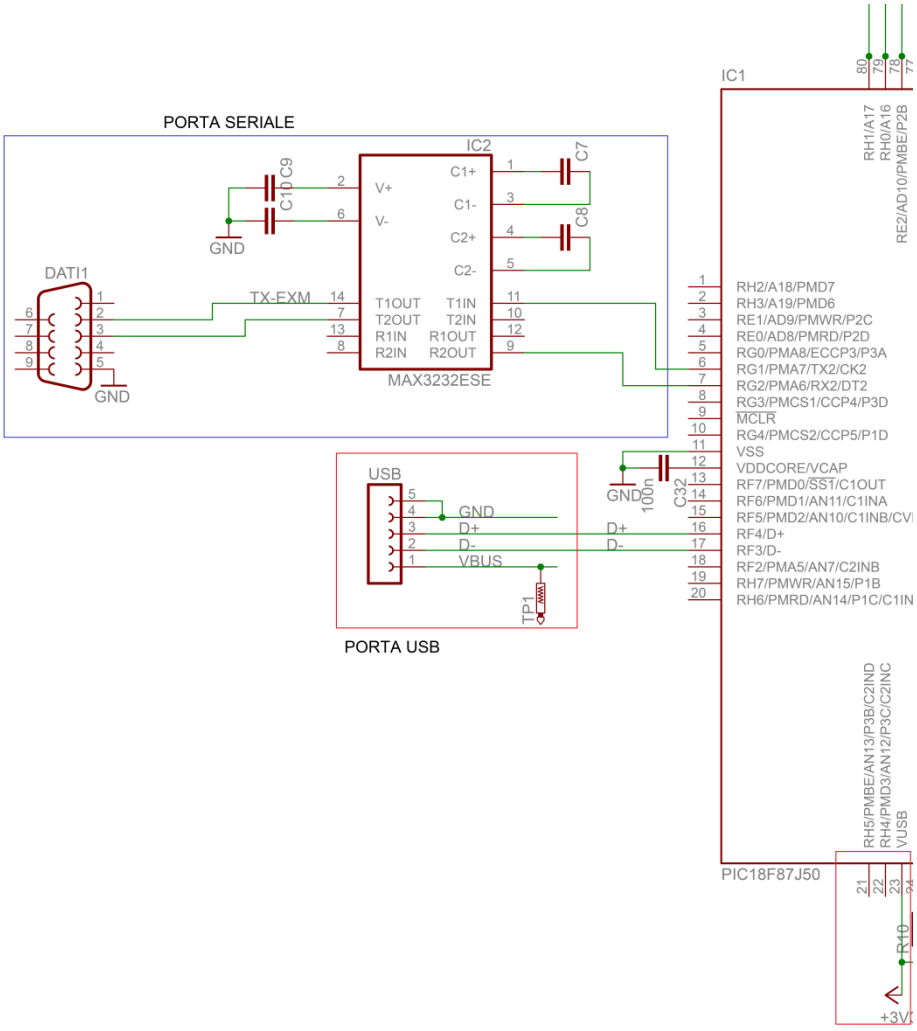


Figura 1.24:Connessione del microcontrollore con porta seriale e USB

Dopo aver inviato le informazioni tramite porta seriale o USB il microcontrollore effettua il salvataggio dei dati localmente su una memoria micro SD CARD. La comunicazione con la memoria SD può avvenire in due diversi modi: utilizzando il bus SD[10] o tramite il protocollo SPI (Serial Peripheral Interface)[11]. La principale differenza delle due modalità, oltre ad una differente mappatura dei piedini del componente, è sul numero di linee utilizzate per la trasmissione dei dati. Il primo protocollo per il trasferimento sfrutta il metodo wide-bus che permette di utilizzare quattro linee per scrivere o leggere e una linea per inviare i comandi mentre il protocollo SPI utilizza una linea per la scrittura e un'altra per la lettura.

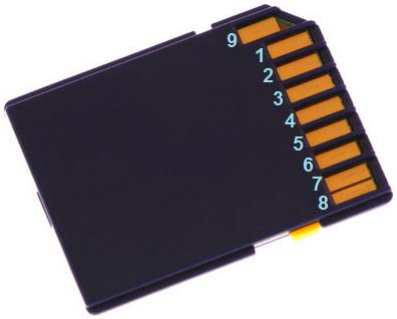


Figura 1.25:Piedinatura SD CARD

Tabella 1:Piedinatura dei diversi protocolli

SPI			SD wide-bus		
Pin	**Name**	**Descrizione**	**Pin**	**Name**	**Descrizione**
1	CS	Card Select	1	DAT3	Data 3
2	DI	Data In	2	CMD	Command, Response
3	VSS	Ground	3	VSS	Ground
4	VDD	Power	4	VDD	Power
5	CLK	Clock	5	CLK	Clock
6	VSS	Ground	6	VSS	Ground

[10] il bus SD è un sistema di comunicazione veloce, parallelo, che utilizza al massimo quattro linee di trasmissione e rappresenta l'evoluzione del sistema di trasferimento utilizzato nelle MMC Card.

[11] l'SPI è uno standard sviluppato dalla Motorola nel quale la trasmissione avviene tra un dispositivo master e uno slave. Il bus SPI è un protocollo seriale sincrono (per la presenza del clock) e full duplex (per la doppia linea di trasmissione dati). La frequenza di trasferimento dipende dal limite massimo che può gestire il dispositivo slave e dal numero di dispositivi connessi al bus (in quanto ogni dispositivo introduce una capacità parassita).

7	DO	Data Out
8	NC	NC
9	NC	NC

7	DAT0	Data 0
8	DAT1	Data 1.
9	DAT2	Data 2

Lo modalità di comunicazione scelta per il DAQ è l'SPI, una modalità implementata a livello hardware che ha il vantaggio di semplificare la gestione della comunicazione e renderla più stabile. Per il corretto utilizzo dell'SPI occorre che la linea dati in uscita (SDO) sia collegata alla tensione di alimentazione attraverso una resistenza di pull-up di 4.7 kΩ (Fig. 1.26: resistenza R1).

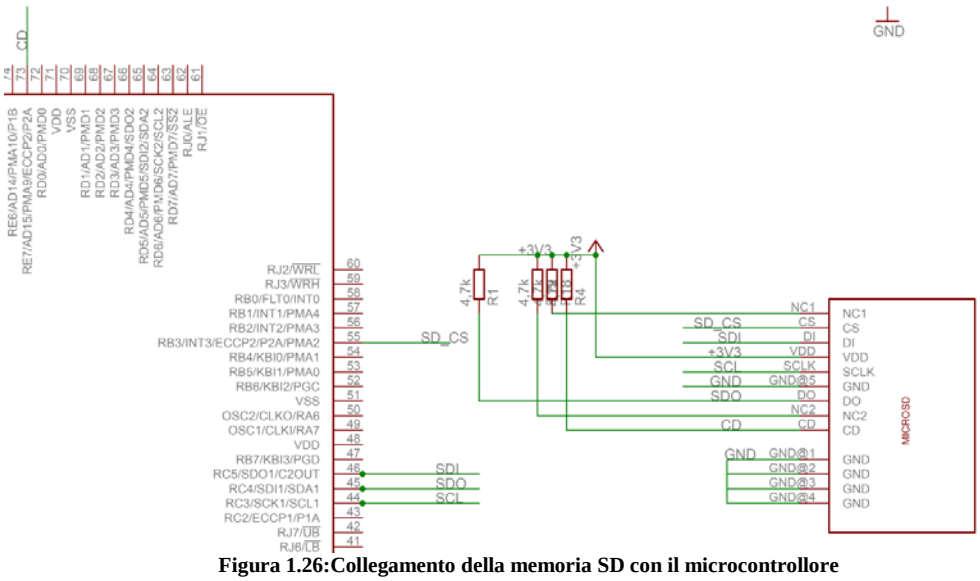


Figura 1.26:Collegamento della memoria SD con il microcontrollore

Il microcontrollore, completato il processo di scrittura delle informazioni su una memoria SD, verifica che non ci siano comandi remoti per la variazione della finestra di conteggio (sottosistema di telemetria del pallone); nel caso li trasmetta all'FPGA (Fig 1.18: riquadro rosso) e aspetta i nuovi dati dall'FPGA.

Per effettuare le misure ad un'altitudine elevata, il DAQ deve essere ospitato a bordo di un pallone atmosferico dell'ASI. Poiché il numero dei lanci che l'ASI compie in un anno sono limitati, si è scelto di rendere il sistema totalmente ridondante assicurandosi che le informazioni sul flusso dei raggi cosmici vengano salvate e inviate tramite porte seriali al sottosistema di telemetria e che la catena di acquisizione (FPGA, microcontrollori, GPS e sensori) non si interrompi incidentalmente in nessun punto; questo comporta la presenza nel DAQ di una catena

di acquisizione master e una slave, la catena master è l'unica che ha gli ingressi e le uscite di trigger (Fig. 1.27).

Per avere una ridondanza completa tra le catene, si effettua una connessione fully-meshed[12] con un bus di tre linee tra le FPGA e i microcontrollori principali (Fig. 1.27). La decisione di utilizzare un collegamento fully-meshed scaturisce dal voler evitare l'impiego di un dispositivo di controllo tipo "watchdog" che costituirebbe l'anello debole del sistema. Questa soluzione ci permette di risparmiare potenza e tempo in caso di malfunzionamento scambiando i singoli dispositivi e non l'intera catena. Per cui, nel caso di malfunzionamento dell'FPGA Master la catena master non viene disattivata e passato il controllo alla catena Slave, ma il microcontrollore master inizia a comunicare con l'FPGA Slave in pochi microsecondi evitando la perdita di informazioni sul flusso dei raggi cosmici (per scambiare le catene di acquisizione occorrono 10 secondi perché si deve attivare la comunicazione con il sottosistema dei telemetria avendo a disposizione un'unica porta seriale).

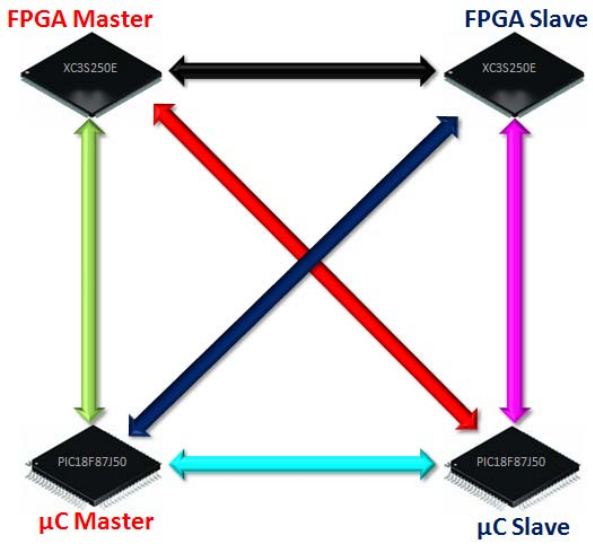


Figura 1.27:Connessione fully-meshed

In Figura 1.27 è riportato uno schema a blocchi della connessione e nella Figura 1.28 le connessioni fisiche tra i diversi dispositivi, in particolare le

[12] ciò vuol dire che ogni componente ha una linea di collegamento con ogni altro componente del sistema

connessioni di colore verde e viola (Fig. 1.27 e Fig. 1.28) sono le connessioni dirette tra i dispositivi della catena Master e Slave; le connessioni blu e rossa (Fig. 1.27 e Fig. 1.28) quelle per far comunicare i dispositivi tra le due catene o scambiare i dispositivi tra le due catene; invece le connessioni nera e celeste (Fig. 1.27 e Fig. 1.28) sono le connessioni di collegamento tra le FPGA e microcontrollori. Queste ultime connessioni sono molto importanti perché sono utilizzate in un primo momento per stabilire la funzionalità del dispositivo, tramite dei meccanismi di *TIMER*.

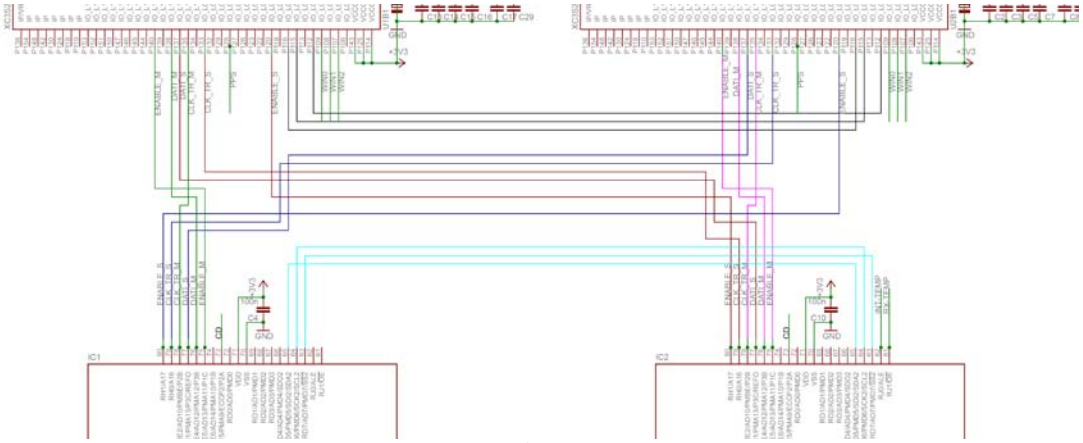


Figura 1.28:Connessione fisica tra i diversi dispositivi

Dopo aver scelto i dispositivi facenti parte del DAQ e deciso l'algoritmo da utilizzare per calcolare il flusso dei raggi cosmici, si è effettuato un test del suo funzionamento, che vedremo nel dettaglio nel prossimo capitolo.

1.4 Dispositivi utilizzati nel DAQ

In questo paragrafo si vuole spiegare il motivo delle scelte effettuate sia riguardo i componenti che nella progettazione dell'architettura del DAQ.

La prima scelta è stata quella di utilizzare sia un FPGA che un microcontrollore. L'utilizzo dell'FPGA è obbligatorio per poter elaborare i segnali provenienti dai rivelatori mentre l'esigenza di introdurre il microcontrollore è nata nel momento in cui ci si è resi conto che la gestione dell'intero sistema da parte della sola FPGA risultava troppo complessa. La grande quantità ci celle occupate nell'FPGA peggiorava le prestazioni temporali rendendo difficile la gestione dei dati,

per cui per migliorare le prestazioni si è demandata la gestione dei sensori e della comunicazione con il sottosistema di telemetria al microcontrollore. Con questa configurazione il microcontrollore, nel tempo di un secondo, che rappresenta la finestra di conteggio più bassa, deve comunicare con l'FPGA, con il GPS, con i 4 sensori e scrivere sulla memoria SD CARD; dopo aver stimato e verificato che la finestra di conteggio non era sufficiente per eseguire tutte le operazioni assegnate al microcontrollore con un certo margine di tempo, si è deciso di introdurre un secondo microcontrollore dedicato alla gestione dei sensori. La comunicazione tra i due microcontrollori avviene tramite un'interfaccia seriale software.

Secondo quanto detto nei precedenti paragrafi il DAQ, per effettuare le misure ad alta quota deve essere collocato su un pallone aerostatico dell'ASI, per cui le temperature raggiunte durante il volo, molto basse, potrebbero stressarne ulteriormente l'elettronica del DAQ e non far funzionare correttamente i dispositivi. Per evitare ciò, oltre a coibentare il crate del DAQ, sono stati scelti dei dispositivi di classe industriale che lavorano a regime con ottime prestazioni nel range di temperatura da -40°C a 120°C. Tenendo conto che il DAQ è assemblato in laboratorio, la scelta dei componenti è dovuta cadere su quelli con package QFP (quad flat package), adatti per il montaggio superficiale.

Nella scelta dell'FPGA, in un primo momento si era optata per una della famiglia SPARTAN 3AN della Xilinx a causa del vantaggio di avere internamente una PROM in cui memorizzare il firmware. Purtroppo non erano disponibili modelli con package QFP (Fig. 1.29: figura a sinistra) con una memoria sufficiente per la nostra applicazione ma solo package BGA (Ball Grid Array, Fig. 1.29: figura a destra), per cui la nostra scelta è caduta sull'FPGA XC3S250E (Fig. 1.29) con package PQFP (Plastic Quad Flat Package). L'FPGA XC3S250E [24-26] implementa gli standard LVDS 1.8 V e LVDS 2.5 V e non lo standard LVDS 3.3 V utilizzato dai rivelatori. Lo standard LVDS 3.3V risulta essere compatibile e utilizzabile con lo standard LVDS 2.5V solo se utilizzato come input[13].

[13] Per maggiori informazioni si faccia riferimento all'application note Xilinx; Interfacing LVPECL 3.3 V Drivers with Xilinx 2.5 V Differential [38]

Figura 1.29:SPARTAN 3E con package QFP (destra) e BGA (sinistra)

Il microcontrollore principale deve avere la possibilità di implementare delle porte seriali per la comunicazione con il GPS e con il sottosistema di telemetria, una porta USB per l'analisi dati per quando il DAQ non è montato sul pallone e una porta SPI per la gestione della memoria SD. Uno dei microcontrollori che ingloba tutte queste caratteristiche è il PIC18F4550 della Microchip con 44 pin, di cui è disponibile un sistema di sviluppo con il quale sono state eseguite le prime prove con la porta USB e il salvataggio dei dati su memoria SD. Da queste prime prove si è notato che il firmware che implementa queste due funzioni occupava quasi l'80% (26 KB su 32 KB disponibili) della memoria del microcontrollore per cui, dovendo implementare ulteriori funzioni (porte seriali, comunicazione con FPGA e processing dei dati), si è deciso di passare a un microcontrollore con una Program Memory (memoria per la memorizzazione del firmware) maggiore; si è scelto quindi il PIC18F87J50 (Fig. 1.30) della Microchip con 80 pin e package TQFP (Thin QFP).

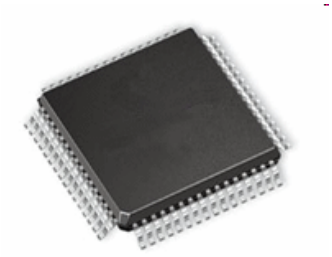


Figura 1.30:PIC18F87J50

La scelta del microcontrollore secondario è stata effettuata considerando che deve comunicare con i sensori di temperatura e con l'accelerometro e il magnetometro. Il protocollo di comunicazione dei primi non richiede un particolare microcontrollore in quanto implementato tramite software; invece per comunicare con il secondo sensore occorre che il microcontrollore implementi il protocollo I^2C

perciò si è scelto il PIC16F1827 (o PIC16F1847) della Microchip con 20 pin e package TQFP .

Le prime prove di comunicazione e di firmware dei diversi dispositivi sono state effettuate mediate dei sistemi di sviluppo. Per il microcontrollore si è utilizzato il sistema di sviluppo della Microchip PICDEM PIC18 Explorer [29] (Fig 1.31: destra) su cui si è montato il microcontrollore scelto mediante il modulo Plug-In (PIM); per l'FPGA si è utilizzata la demoboard Spartan-3E FPGA Starter Kit Board [23] (Fig1.31: sinistra) su cui è montata una FPGA della stessa famiglia di quella da noi scelta sebbene con una memoria di capacità superiore.

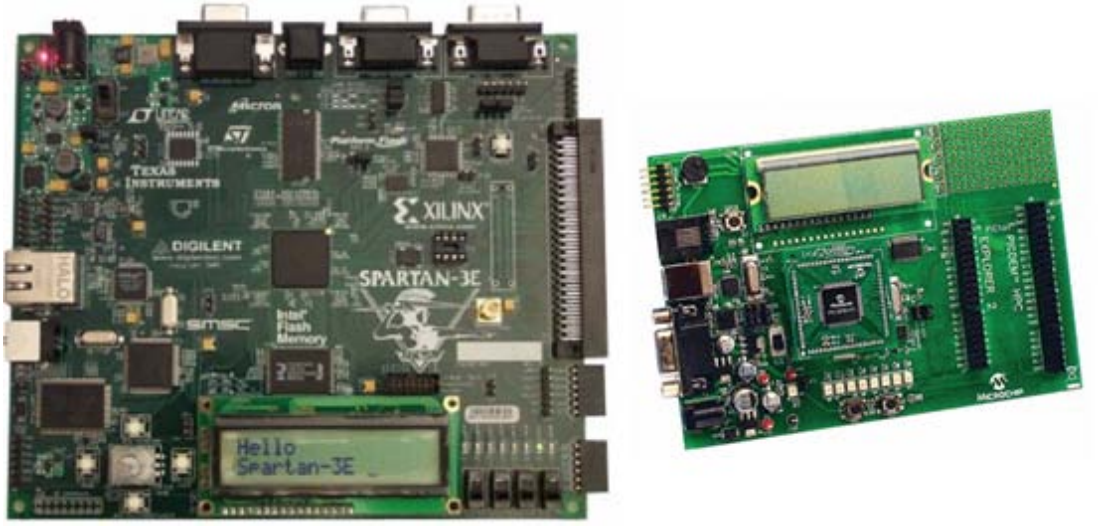


Figura 1.31:Sistema di sviluppo: A sinistra Spartan-3E FPGA Starter Kit Board a destra Microchip PICDEM PIC18 Explorer

Utilizzando questi due sistemi sviluppo si è realizzato un prototipo di DAQ (di cui si parlerà ampiamente nel prossimo capitolo) che è stato utilizzato per effettuare un primo test di misura del flusso dei raggi cosmici, e la verifica della validità del firmware.

1.5 Principio di funzionamento: elaborazione dati

Come si è visto precedentemente, i rilevatori producono un segnale di 60 ns in corrispondenza del loro attraversamento di una particella ionizzante di opportuna energia. L'FPGA riceve il segnale proveniente dal rilevatore ed esegue una prima elaborazione, che comporta la determinazione del tipo di coincidenza degli

scintillatori; ciò vuol dire stabilire quali e quanti scintillatori rilevano il passaggio di una particella in una opportuna finestra temporale di acquisizione. La valutazione del tipo di coincidenza è uno dei principali compiti che deve eseguire il DAQ.

Per capire cosa vuol dire valutare il tipo di coincidenza tra gli scintillatori, consideriamo un rilevatore costituito da una struttura telescopica di quattro scintillatori (Fig. 1.32); per maggior chiarezza etichettiamo i quattro scintillatori della struttura telescopica dall'alto verso il basso con le lettere *x*, *y*, *z* e *w* (Fig. 1.32). Valutare il tipo di coincidenza tra gli scintillatori vuol dire verificare quali fra tutte le possibili combinazioni si sono verificate nella finestra temporale di acquisizione. Nel caso di un rivelatore composto da 4 scintillatori significa determinare quali delle quindici combinazioni si sono verificate; le quindici possibili combinazioni di quattro scintillatori sono: quattro singole (*x*, *y*, *z* e *w*), sei doppie (*xy*, *xz*, *xw*, *yz*, *yw*, *zw*), quattro triple (*xyz*, *xzw*, *xyw*, *yzw*) e una quadrupla (*xyzw*).

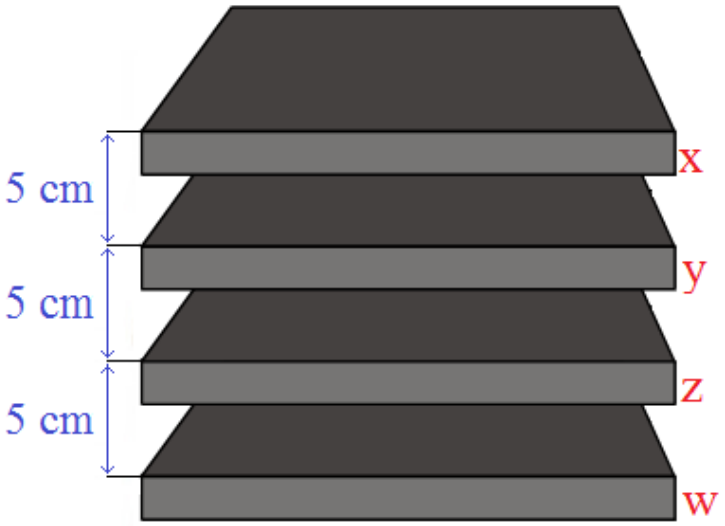


Figura 1.32: Disposizione degli scintillatori

La finestra temporale di ascolto è utilizzata per determinare il tipo di coincidenza che si verifica. La durata della finestra è di 100 ns, ed è determinata dalla distanza di circa 5 cm (Fig 1.32), tra due scintillatori consecutivi, dal segnale di durata 60 ns che lo scintillatore invia al passaggio di una particella e che la velocità di questi è prossima a quella della luce.

La finestra temporale di conteggio determina la frequenza di invio delle informazioni da FPGA a microcontrollore mediante la porta seriale o la porta USB. la durata di questa finestra è impostabile da remoto e per default è di 1 s.

Le informazioni che l'FPGA invia al microcontrollore, in corrispondenza di ciascuna finestra temporale di conteggio, sono il numero di volte che si verificano le diverse combinazioni. Il conteggio delle volte in cui si verificano le diverse combinazioni è effettuato mediante contatori. Ogni qualvolta uno degli scintillatori invia all'FPGA il segnale del passaggio di una particella ionizzante, questa apre una nuova finestra temporale di ascolto (Fig 1.33) se non ne esiste un'altra attiva. Alla fine ogni finestra temporale di ascolto aperta l'FPGA valuta il tipo di coincidenza e incrementa i valori delle combinazioni verificate nella finestra di acquisizione considerata.

Conoscendo quali scintillatori hanno rilevato il passaggio della particella ionizzante, si possono valutare le combinazioni che si verificano nella finestra temporale di acquisizione. Nel caso in cui la particella ionizzante è rilevata da due scintillatori (Fig 1.33:(1) la particella è rilevata da *x* e *y*) oltre alla combinazione di doppia si rilevano anche due singole, inoltre in questo modo si ottiene una ridondanza dei dati, come da Fig 1.33:(1). Dopo aver valutato la coincidenza, l'FPGA deve incrementare i valori di *x*, *y* e della doppia *xy* (Fig 1.33: (2),(3)).

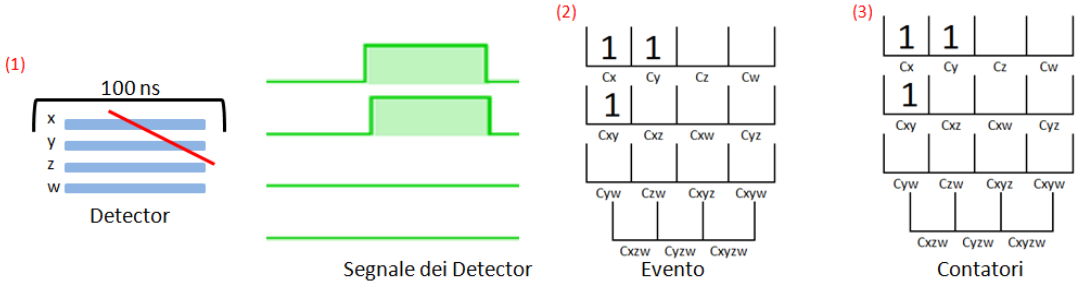


Figura 1.33:Evento di una doppia

Quando si verifica che la particella ionizzante è rilevata da tutti gli scintillatori (Fig. 1.34: (1), (2),(3)) vuol dire che si verificano tutte le possibile combinazioni per cui devono essere incrementati tutti i contatori.

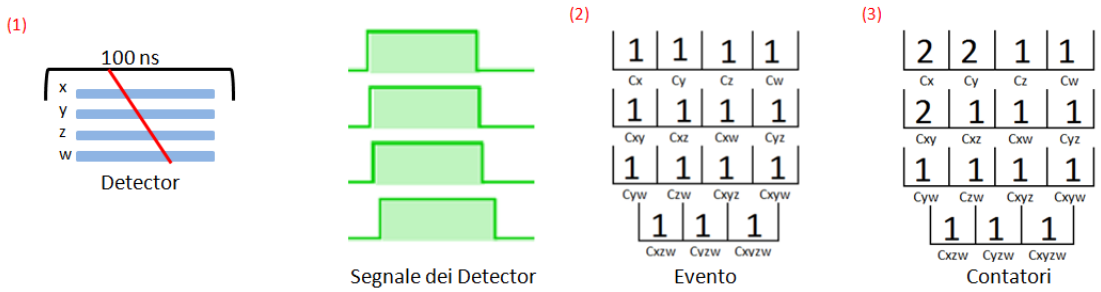


Figura 1.34:Evento di una quadrupla

In Fig. 1.35 è riportata la valutazione del tipo di coincidenza effettuata dall'FPGA quando la particella ionizzante è rilevata da tre scintillatori, infatti in questo caso si deve incrementare solo il valore dei tre contatori di singola (*x*, *y* e *z*), dei tre contatori di doppia (*xy*, *xz* e *yz*) e del contatore di tripla (*xyz*).

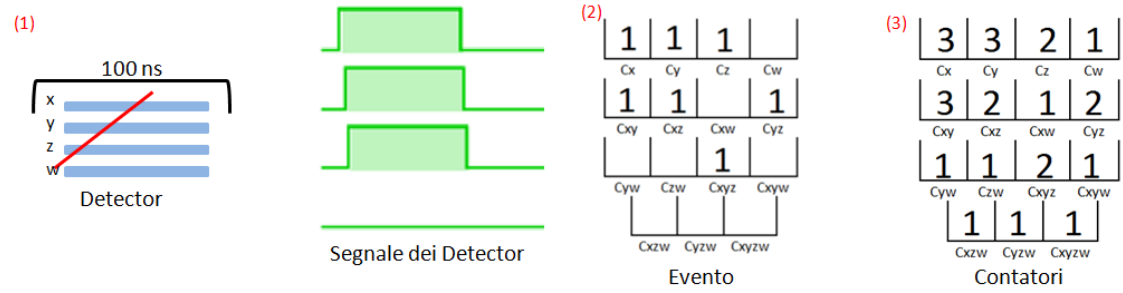


Figura 1.35:Evento di tripla

Al termine della durata della finestra di conteggio, l'FPGA trasferisce i dati in una coda in attesa della risposta del microcontrollore e ed apre immediatamente una nuova finestra temporale per un nuovo conteggio di particelle.

La finestra temporale di conteggio, come già detto nel precedente paragrafo, è programmabile da remoto, mediante l'invio di particolari informazioni tramite la porta seriale collegata al sottosistema di telemetria o porta la USB; le finestre impostabili mediante diversi comandi sono di 1 s, 5 s, 10 s, 30 s, 60 s, 5 minuti, 10 minuti.

Le informazioni inviate dall'FPGA al microcontrollore in un primo momento vengono memorizzate in binario e poi convertite in esadecimale, per minimizzare il tempo di trasmissione mediante porta seriale al sottosistema di telemetria ed eseguire un salvataggio su una memoria SD. Prima di essere trasmesse, alle informazioni dei conteggi vengono accodate le informazioni del magnetometro e dell'accelerometro

(Fig. 1.36: *Bx*, *By*, *Bz* (magnetometro) e *Ax*, *Ay*, *Az* (accelerometro)), le informazioni di latitudine (GPS), longitudine (GPS), data e ora (ggmm hhmmss) del momento in cui si effettua la misura, le informazioni provenienti dai sensori di temperatura, l'informazione di altitudine (alt) e il numero di satelliti (ns) rilevati dal satellite (Fig. 1.36:(4))

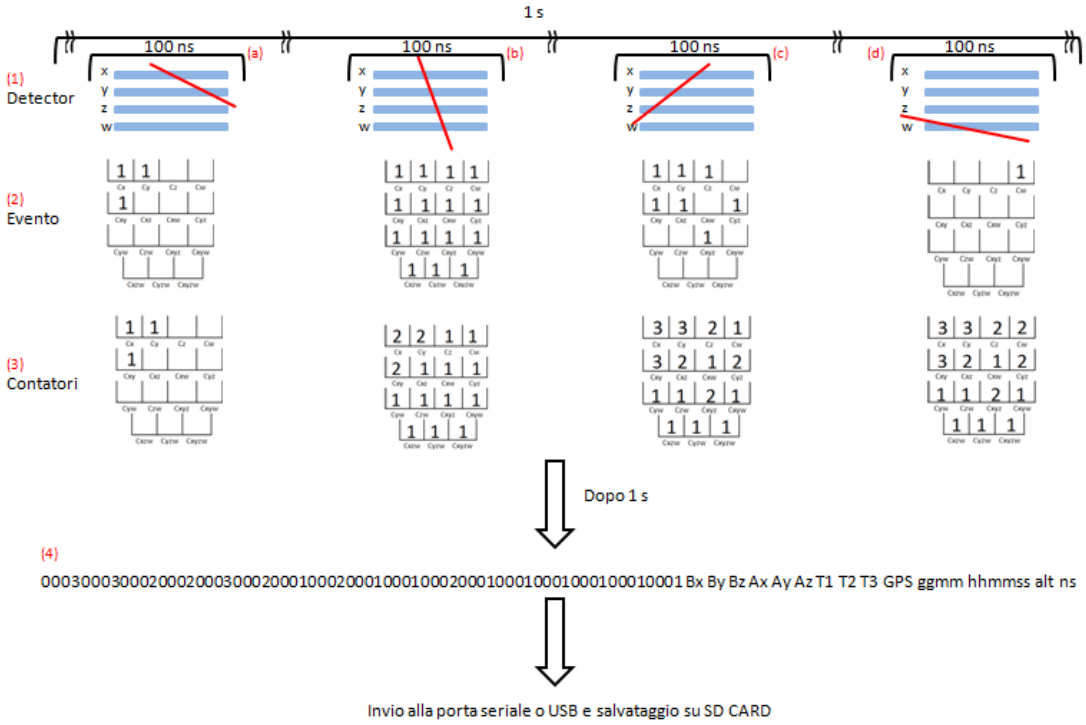


Figura 1.36:Funzionamento DAQ

CAPITOLO 2

PROTOTIPO DEL DAQ

2.1 Introduzione

Per verificare la funzionalità di ogni componente, e degli algoritmi di coincidenza si è sviluppato un prototipo del DAQ (la cui architettura è stata trattata nel precedente capitolo) utilizzando degli opportuni, sistemi di sviluppo.

Inoltre visto che i test dovevano essere effettuati in luoghi in cui la temperatura del DAQ è pari alla temperatura ambientale e la posizione del rilevatore è controllabile direttamente, si è deciso di non inserire il microcontrollore secondario nel prototipo.

Un'ulteriore modifica, rispetto al DAQ descritto nel precedente capitolo, consiste nella struttura del rilevatore, che sarà composto da quattro scintillatori di dimensioni 15 x 15 x 1 cm nella struttura telescopica e distanziati tra di loro di 5 cm (Fig.2.1: Distanza tra due strati blu). Tra i rivelatori sono stati interposti strati di ferro (Fig.2.1: Strato rosso) e strati di poliestirene (Fig.2.1: Strato viola) con funzione di assorbitore.

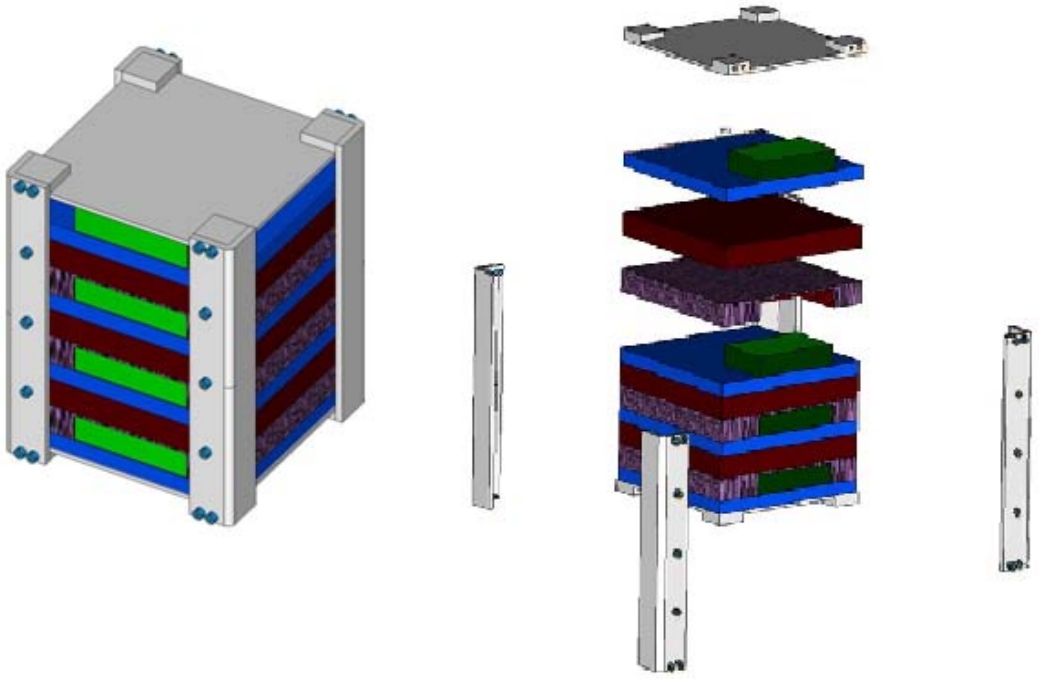


Figura 2.1:A destra il rilevatore assemblato, a sinistra il rilevatore esploso

Per alimentare i diversi sistemi di sviluppo, il rilevatore e il GPS, si è realizzato un modulo di alimentazione (Fig.2.2) che, utilizzando dei regolatori di tensione con una tensione d'ingresso pari a 12 V ottenuti da un generatore esterno, producano le tensioni stabilizzate necessarie per alimentare tutti i dispositivi. In particolare sono stati utilizzati due regolatori di tensione, uno per ottenere i 3.3 V che alimentano il rilevatore e il GPS, e il secondo per i 5 V che alimenta il sistema di sviluppo dell'FPGA; il sistema di sviluppo del microcontrollore è alimentato tramite i 12 V forniti dal generatore esterno.

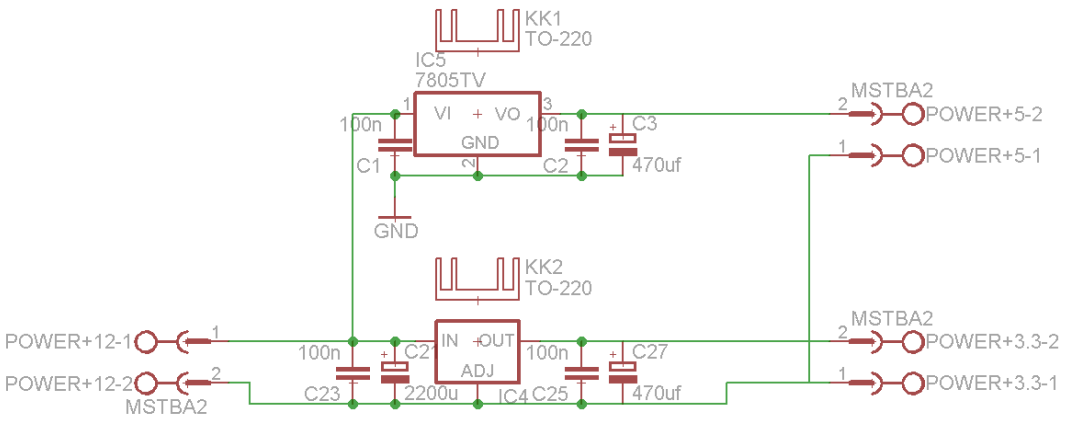


Figura 2.2Modulo di alimentazione

Per testare il dispositivo e il suo funzionamento sono stati eseguiti diversi test in laboratorio: una campagna di misure sul Gran Sasso e nei Laboratori Nazionali del Gran Sasso (LNGS) per effettuare misure a diverse altitudini. I risultati delle misure effettuate sono riportati a fine capitolo.

2.2 Algoritmo di coincidenza

Per effettuare la misura del flusso delle particelle ionizzanti si deve calcolare quanti e quali eventi sono occorsi in una finestra temporale di conteggio, gestibile da remoto. Nel caso di un rivelatore formato da quattro scintillatori (Fig. 2.1), per tenere traccia di tutti gli eventi, occorre utilizzare quindici contatori, uno per ogni possibile combinazione.

Il modo più semplice per sapere quali coincidenze ed eventi si sono verificati tra gli scintillatori di cui è composto il rilevatore, in una finestra di ascolto predefinita, è quello di implementare una rete combinatoria a più livelli di porte AND. Lo svantaggio di questa soluzione è dovuto al numero elevato di porte utilizzato e al ritardo da loro introdotto, che provoca una sincronia non perfetta dei segnali di trigger dai contatori. Per risolvere il problema di sincronia e nello stesso tempo semplificare il circuito, si è ideato un algoritmo che utilizza una matrice che considera tutte le possibili combinazioni, in sostituzione della rete combinatoria.

La matrice utilizzata dall'algoritmo è implementata assegnando ad ogni contatore una colonna e ad ogni riga un evento verificabile; per accedere alla matrice si assegna ad ogni scintillatore un peso e, tramite un' equazione, si attribuisce ad ogni possibile evento un valore che permette di accedere alla riga a cui si riferisce l'evento. Ad ogni evento, le celle della matrice andranno a sommarsi al valore dei contatori. Il contenuto delle celle della matrice sarà uguale a 0 quando l'evento verificato non ha interessato la combinazione, altrimenti, conterrà 1 se la combinazione è verificata per l'evento corrispondente al numero di riga della cella presa in considerazione.

Nell'algoritmo sviluppato si è deciso di assegnare ad ogni scintillatore un peso corrispondente ad una potenza di due, in particolare al rivelatore *x* (Fig. 1.32) si

è assegnato il valore 1, ad *y* (Fig. 1.32) il valore 2, a *z* (Fig. 1.32) il valore 4 e a *w* (Fig. 1.32) il valore 8: in questo modo la loro combinazione darà luogo al codice binario dell'evento, per cui l'equazione per calcolare il peso sarà 8**Tw*+4**Tz*+2**Ty*+*Tx*, in cui le variabili *Tx*, *Ty*, *Tz* e *Tw* assumono il valore uno se il corrispondente scintillatore ha rilevato il passaggio di radiazione. Dopo aver assegnato alle righe della matrice un evento e alle colonne un contatore, si procede popolando la matrice. Ad esempio, nel caso in cui gli scintillatori che rilevano la particella ionizzante sono i primi tre (Fig 1.35), il peso dell'evento dato dall'equazione 8*0+4*1+2*1+1*1 è uguale a 7, quindi le celle della riga 7, che sono popolate con 1, sono quelle che si riferiscono all'evento *xyx* e a tutti i suoi possibili sottoeventi, cioè *yz*, *xz*, *xy*, *z*, *y* e *x* , come si può notare dalla tabella sottostante (Tabella 2).

Tabella 2: Riga per l'evento xyz

Segn. Riv				P	Incremento Contatori														
Tw	Tz	Ty	Tx		Cx	Cy	Cz	Cw	Cxy	Cxz	Cxw	Cyz	Cyw	Czw	Cxyz	Cxyw	Cxzw	Cyzw	Cxyzw
0	1	1	1	7	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0

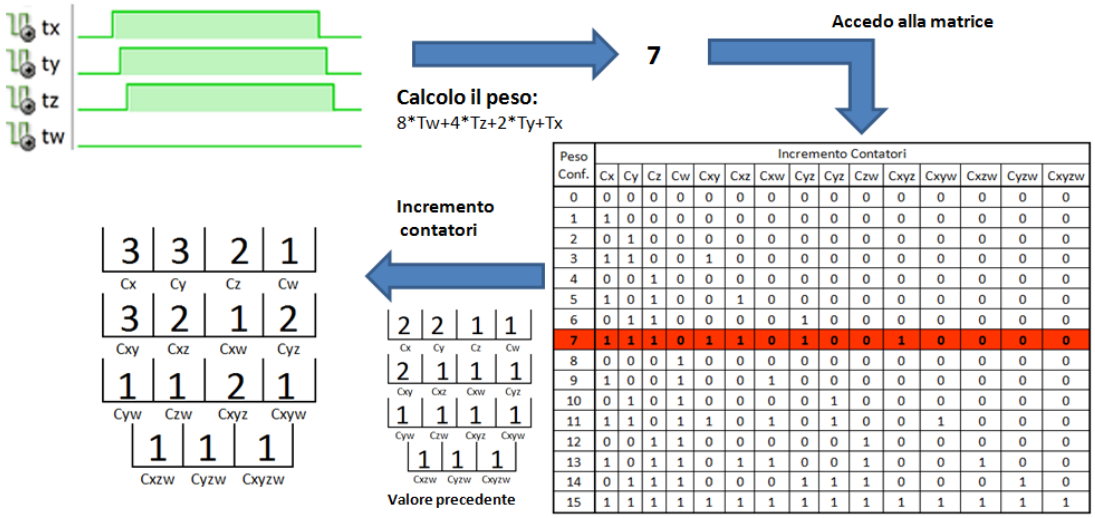
Ripetendo lo stesso ragionamento per tutti i possibili eventi, si ottiene la seguente tabella (Tabella 3).

Tabella 3:Matrice utilizzata per la codifica dell'evento

Segn. Riv				P	Incremento Contatori														
Tw	Tz	Ty	Tx		Cx	Cy	Cz	Cw	Cxy	Cxz	Cxw	Cyz	Cyw	Czw	Cxyz	Cxyw	Cxzw	Cyzw	Cxyzw
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	3	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	5	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	0	6	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
0	1	1	1	7	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0
1	0	0	0	8	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	9	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	10	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0
1	0	1	1	11	1	1	0	1	1	0	1	0	1	0	0	1	0	0	0
1	1	0	0	12	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0
1	1	0	1	13	1	0	1	1	0	1	1	0	0	1	0	0	1	0	0

1	1	1	0	14	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0
1	1	1	1	15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Tramite questa matrice, dopo ogni evento, e cioè alla chiusura della finestra d'ascolto, l'FPGA valuta l'evento occorso, calcola il peso accedendo alla matrice e incrementa ogni contatore sommando il valore della cella ad esso associata nella riga selezionata (in Fig. 2.3 viene riportato il comportamento dell'algoritmo nel caso dell'esempio precedente).



Peso Conf.	Incremento Contatori													
	Cx	Cy	Cz	Cw	Cxy	Cxz	Cxw	Cyz	Cyz	Czw	Cxyz	Cxyw	Cxzw	Cyzw	Cxyzw
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
6	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
7	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
8	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0
10	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0
11	1	1	0	1	1	0	1	0	1	0	0	1	0	0	0
12	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0
13	1	0	1	1	0	1	1	0	0	1	0	0	1	0	0
14	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figura 2.3: Comportamento dell'algoritmo

2.3 Implementazione dell'algoritmo su FPGA

L'algoritmo di coincidenza è implementato usando diversi processi paralleli, in particolare se ne utilizzano quattro per la gestione dei segnali provenienti dagli scintillatori, uno per ogni scintillatore, e un processo per valutare il tipo di evento. Oltre ai processi appena descritti, che effettuano una prima elaborazione sui segnali degli scintillatori, ce ne sono altri che consentono il corretto funzionamento dell'algoritmo, come quello per il trasferimento dei dati o per la temporizzazione interna. In Figura 2.4 sono riportati i processi che compongono il firmware dell'FPGA e i principali segnali interni.

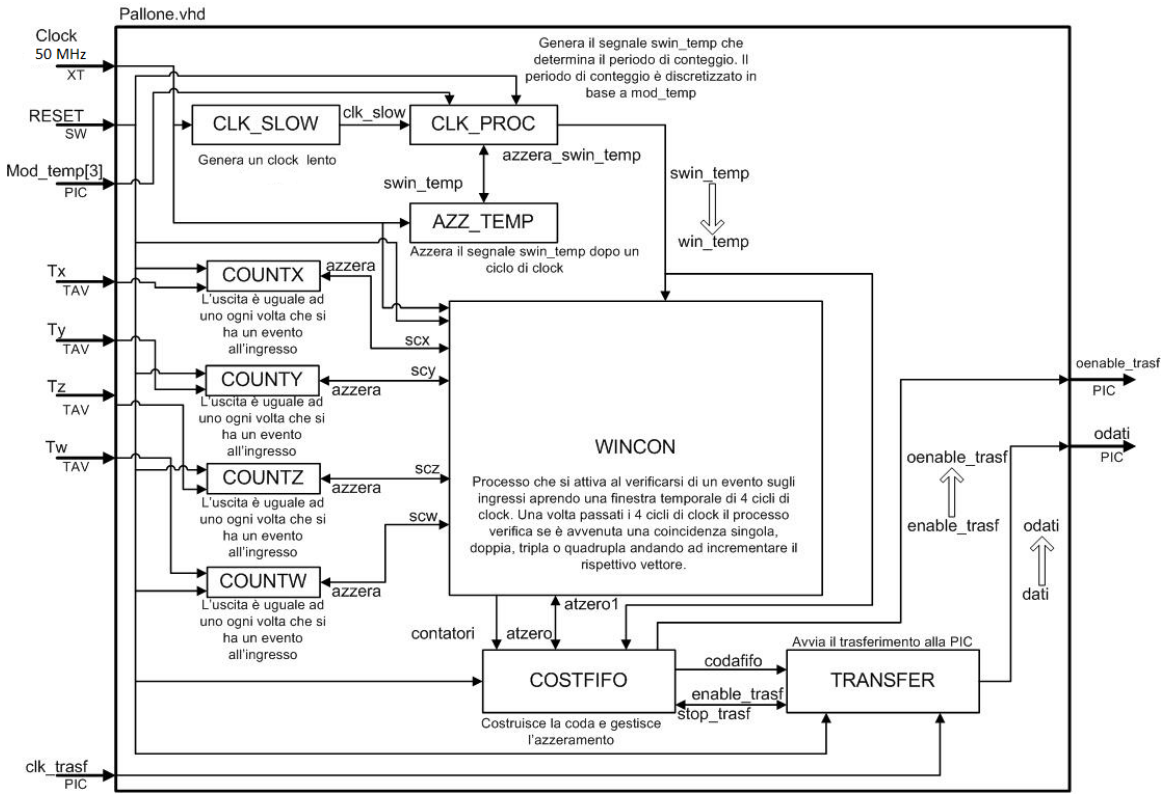


Figura 2.4: Processi eseguiti dall'FPGA

I processi per la gestione della temporizzazione interna sono *CLK_SLOW* e *CLK_PROC* (Fig. 2.4). Il primo processo è un divisore di frequenza, che dal clock esterno di 50 MHz (Fig. 2.5) , genera un clock lento a 1 Hz (Fig.2.6) utilizzato per la temporizzazione e la gestione della finestra di conteggio tramite il processo *CLK_PROC*.

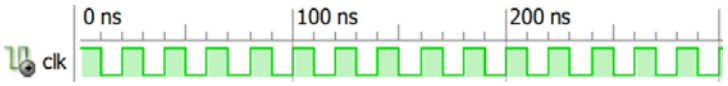


Figura 2.5: Clock a 50 MHz

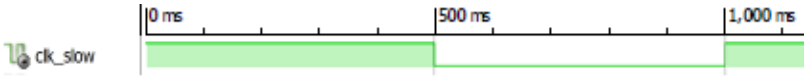


Figura 2.6: Clock a 1 Hz

Come si può notare nella Fig. 2.7., dove è riportato il flow chart del processo *CLK_SLOW*, per implementare il divisore di frequenza, si è realizzato un contatore di cicli di clock che commuta il segnale del clock lento dopo 500 ms corrispondente a 2500000 cicli di clock.

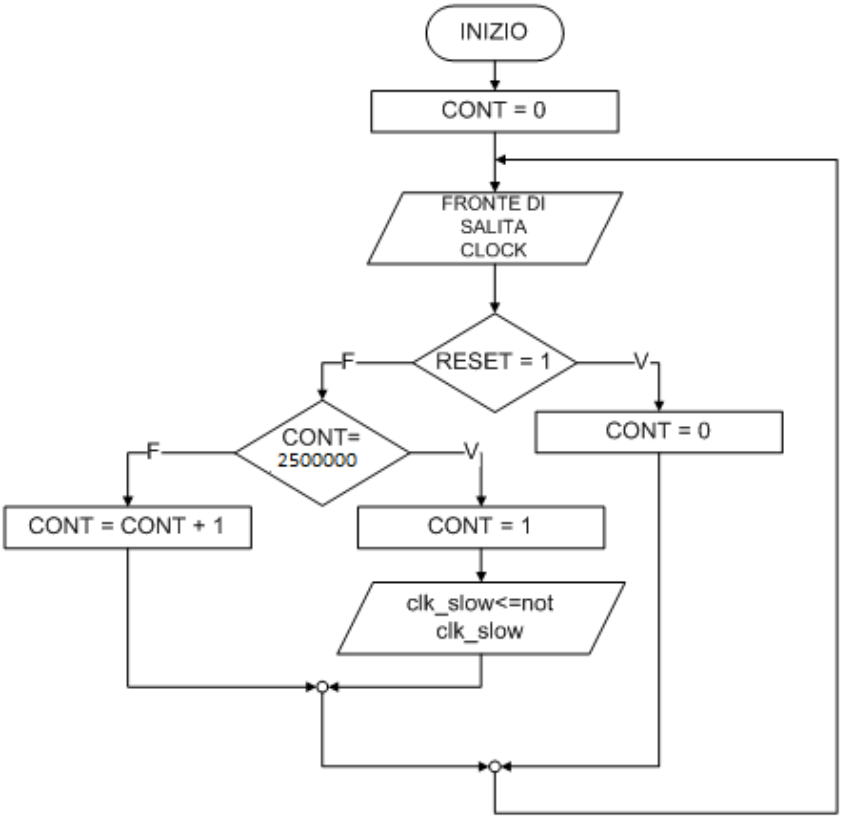


Figura 2.7:Flow chart del processo CLK_SLOW

Il secondo processo ossia il *CLK_PROC* (Fig. 2.4) è utilizzato nella gestione della finestra di conteggio per determinarne, in particolare, l'inizio e la fine e portare il segnale *win_temp* (Fig. 2.8) a livello logico alto, al termine della finestra. Tale segnale è utilizzato come segnale di enable da un altro processo per il trasferimento del contenuto dei contatori in una coda, predisponendo in questo modo una nuova finestra di conteggio e quindi l'acquisizione del flusso dei raggi cosmici senza dover aspettare il trasferimento dei dati al microcontrollore.

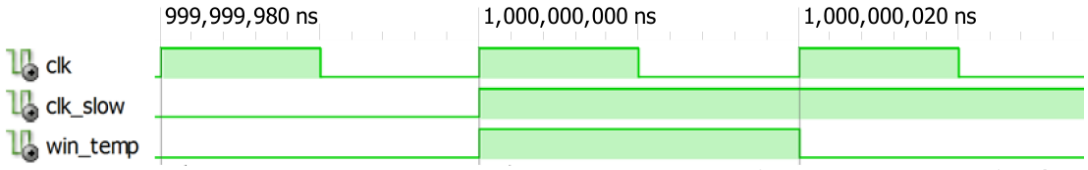


Figura 2.8: Simulazione della commutazione del segnale win_temp al termine della finestra d'acquisizione (default è 1s)

La Figura 2.9 riporta il flow chart del processo e anche in questo caso, per effettuare gestione e controllo della finestra di conteggio, si è implementato un contatore per contare i fronti di salita del clock lento generato dal processo

CLK_SLOW. Quando il contatore raggiunge un valore uguale a quello comunicato da remoto, il segnale *swin_temp* è posto a livello logico alto, per un ciclo di clock; questo segnale verrà utilizzato come enable da un altro processo. Il valore della finestra di conteggio, come già detto, è gestito dal microcontrollore e viene settato tramite il bus *mod_temp* (formato da tre linee) con informazioni provenienti dalla porta seriale o USB utilizzata per il trasferimento dati. Ad ogni valore che il bus di tre linee può assumere, si è associato una durata della finestra di conteggio come da tabella (Tabella 4):

Tabella 4: Comandi di gestione finestra di conteggio

mod_temp	**Durata della finestra di conteggio**
"010"	5 s
"011"	10 s
"100"	30 s
"101"	60 s
"110"	300 s
"111"	600 s
"000" e "001"	1 s

Per default la durata della finestra di conteggio è impostata a un secondo, per cui ad ogni fronte di salita del clock lento si ha la sua fine. Ad ogni variazione del clock lento, il firmware controlla il bus *mod_temp* per verificare se il suo valore è stato modificato e, se la nuova finestra d'integrazione è stata superata, si inizia la procedura di trasferimento dei dati verso il microcontrollore.

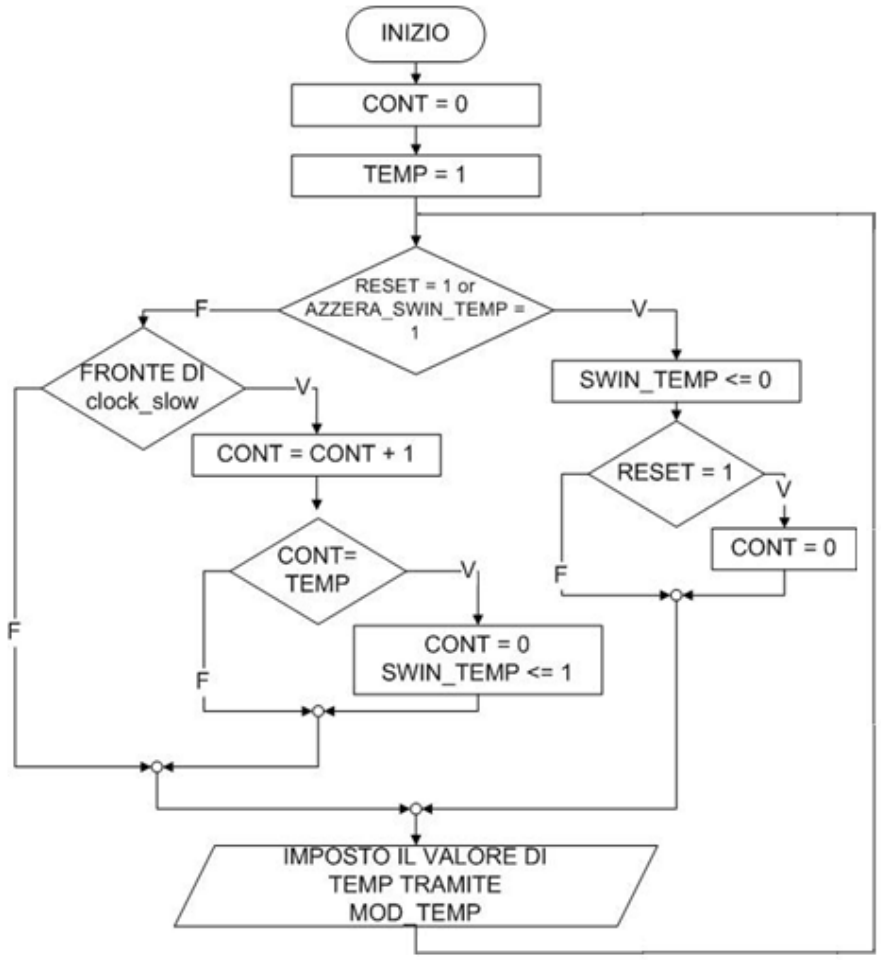


Figura 2.9: Flow chart del processo CLK_PROC

Il primo processo che si occupa dell'elaborazione dati è *WINCON* (Fig. 2.4), il cui compito è quello di gestire la finestra di ascolto di 100 ns, tramite un contatore di quattro cicli di clock che inizia a contare solo quando uno degli scintillatori rileva il passaggio della particella ionizzante, se non c'è un'altro conteggio avviato. Alla fine della finestra d'ascolto, e cioè dopo i quattro cicli di clock, il processo effettua la valutazione dell'evento calcolandone il peso, come riferito nel paragrafo 2.2, tramite accesso alla matrice e l'incremento dei contatori. Esso valuta il tipo di evento attraverso i segnali generati dai processi *COUNTX*, *COUNTY*, *COUNTZ* e *COUNTW*. Ognuno di questi processi è associato ad ogni scintillatore del rilevatore ed ha il compito di mantenere il segnale a livello logico alto dal momento in cui lo scintillatore rileva il passaggio della particella, fino a quando la finestra d'ascolto aperta non ha termine. In questo modo si evita di considerare il caso in cui lo stesso

scintillatore rilevi contemporaneamente due eventi nella stessa finestra d'ascolto (Fig. 2.10). La finestra d'ascolto è *wintalk*, *Tx* è il segnale associato al primo scintillatore del rilevatore e *scx* è il segnale generato dal processo *COUNTX*, utilizzato dal processo *WINCON* per la valutazione dell'evento. Il secondo evento di *Tx* non è stato volutamente preso in considerazione, in quanto la probabilità che si verifichi il doppio evento è molto bassa e perciò era inutile complicare il livello computazionale del firmware.

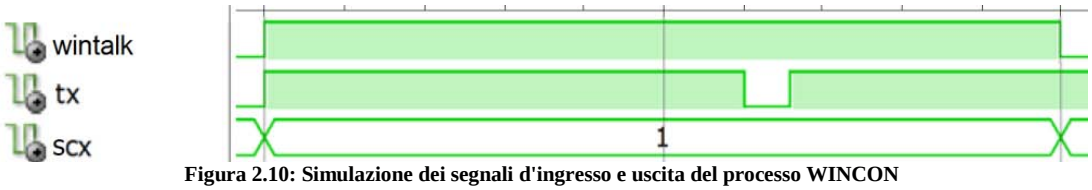


Figura 2.10: Simulazione dei segnali d'ingresso e uscita del processo WINCON

Osservando il flow-chart del processo *WINCON*, riportato in Figura 2.11, si può notare che il contatore dei quattro cicli di clock è in funzione solo se uno degli scintillatori rileva una particella. Nel processo si è volutamente considerato il caso in cui ci fosse una finestra d'ascolto aperta alla chiusura della finestra di conteggio per valutare il tipo di evento, ciò viene attuato tramite il controllo sullo stato logico di *win_temp* e di un flag del processo.

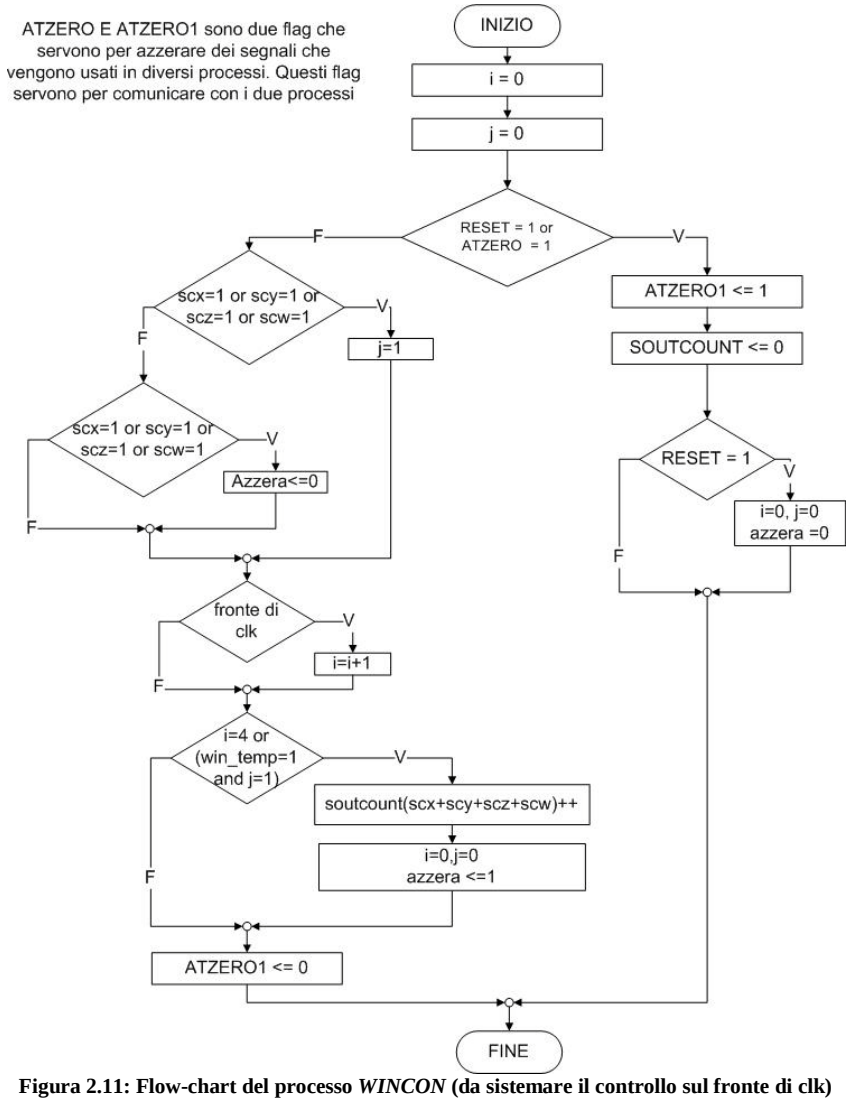


Figura 2.11: Flow-chart del processo *WINCON* (da sistemare il controllo sul fronte di clk)

La figura sottostante riporta il risultato della simulazione dei processi descritti finora: a 100 ns lo scintillatore *x* rileva il passaggio di una particella, il segnale (*tx)* prodotto dal front-end dello scintillatore viene intercettato dal processo *COUNTX* che porta a livello logico alto il segnale di holding *scx* (Fig. 2.12). Nello stesso istante il processo *WINCON* avvia una nuova finestra d'ascolto (Fig 2.12: *wintalk*), nei successivi 100 ns, intervallo di tempo in cui la finestra d'ascolto è attiva, gli scintillatori *y*, *z* e *w* rilevano il passaggio della particella e quindi i rispettivi processi *COUNTY*, *COUNTZ*, *COUNTW* portano i corrispettivi segnali di holding a livello logico alto. I segnali provenienti dai front-end degli scintillatori dopo 60 ns ritornano a livello logico basso mentre i segnali di holding mantengono il loro stato fino alla

chiusura della finestra d'ascolto. Alla fine di questa finestra, cioè a 200 ns, il processo *WINCON* valuta l'evento calcolando il suo peso (*index_tri*), il processo successivamente accede alla matrice e incrementa i contatori, riportati in Figura 2.12 con il nome *cx*,*cy*,*cz*,*cw*,*cxy*,ecc.

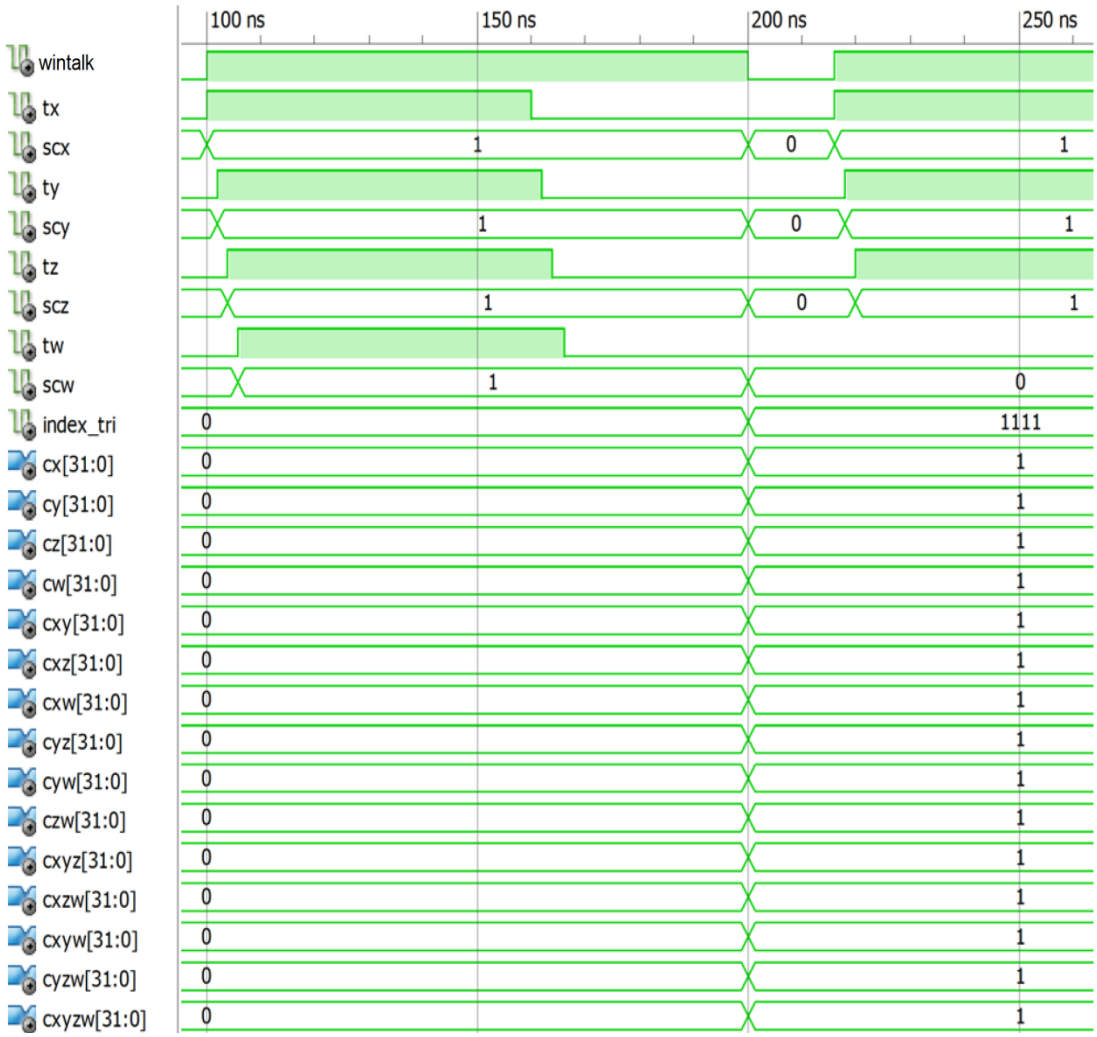


Figura 2.12: Simulazione dell'incremento dei contatori

Come anticipato precedentemente, alla fine della finestra di conteggio, gestita dal processo *CLK_PROC*, viene avviato il trasferimento dati al microcontrollore. Il processo CLK_PROC pone a livello logico alto il segnale *win_temp* che avvisa il processo *COSTFIFO* che la finestra di conteggio è terminata e fa copiare il contenuto di tutti i contatori nella coda utilizzata per il trasferimento dei dati. Dopo quest'operazione il processo:

- avvisa, tramite un flag, il processo *WINCON* affinché azzeri tutti i contatori e inizia una nuova finestra di conteggio;
- nello stesso tempo informa, tramite il segnale *enable_trasf,* il microcontrollore che un set dati è disponibile per il trasferimento (il protocollo ideato per la comunicazione tra le due identità sarà approfondito successivamente).

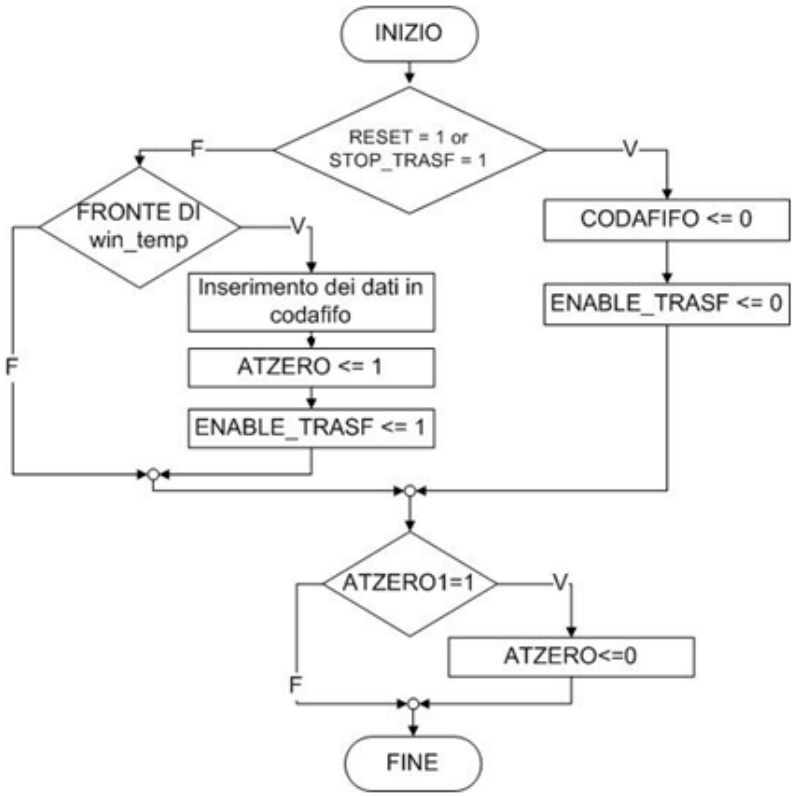


Figura 2.13: Flow-chart del processo COSTFIFO

La simulazione effettuata per verificare il funzionamento del processo *COSTFIFO*, riportata in Figura 2.13, ci indica che, trascorso il tempo di 1 s, allo scadere della finestra di conteggio, il processo *CLK_PROC* pone il segnale *win_temp* a livello logico alto. Nell'istante in cui questo segnale ritorna a livello logico basso, si azzerano i contatori favorendo l'inizio di una nuova finestra di conteggio e ,parallelamente il trasferimento dati (nella Figura 2.14, che descrive questa sequenza, si può notare che dopo 200 ns inizia un nuovo evento). La decisione di iniziare la nuova fase di conteggio sul fronte di discesa del segnale *win_temp*, e non sul fronte

di salita, è scaturita dal voler considerare il caso in cui si avesse una finestra di ascolto aperta alla chiusura della finestra di conteggio. Prima di azzerare i contatori il contenuto viene trasferito nella coda di trasferimento (Fig 2.14: *codafifo*) e segnale *enable_trasf* (Fig 2.14) portato a livello logico alto per inviare i dati della coda al microcontrollore.

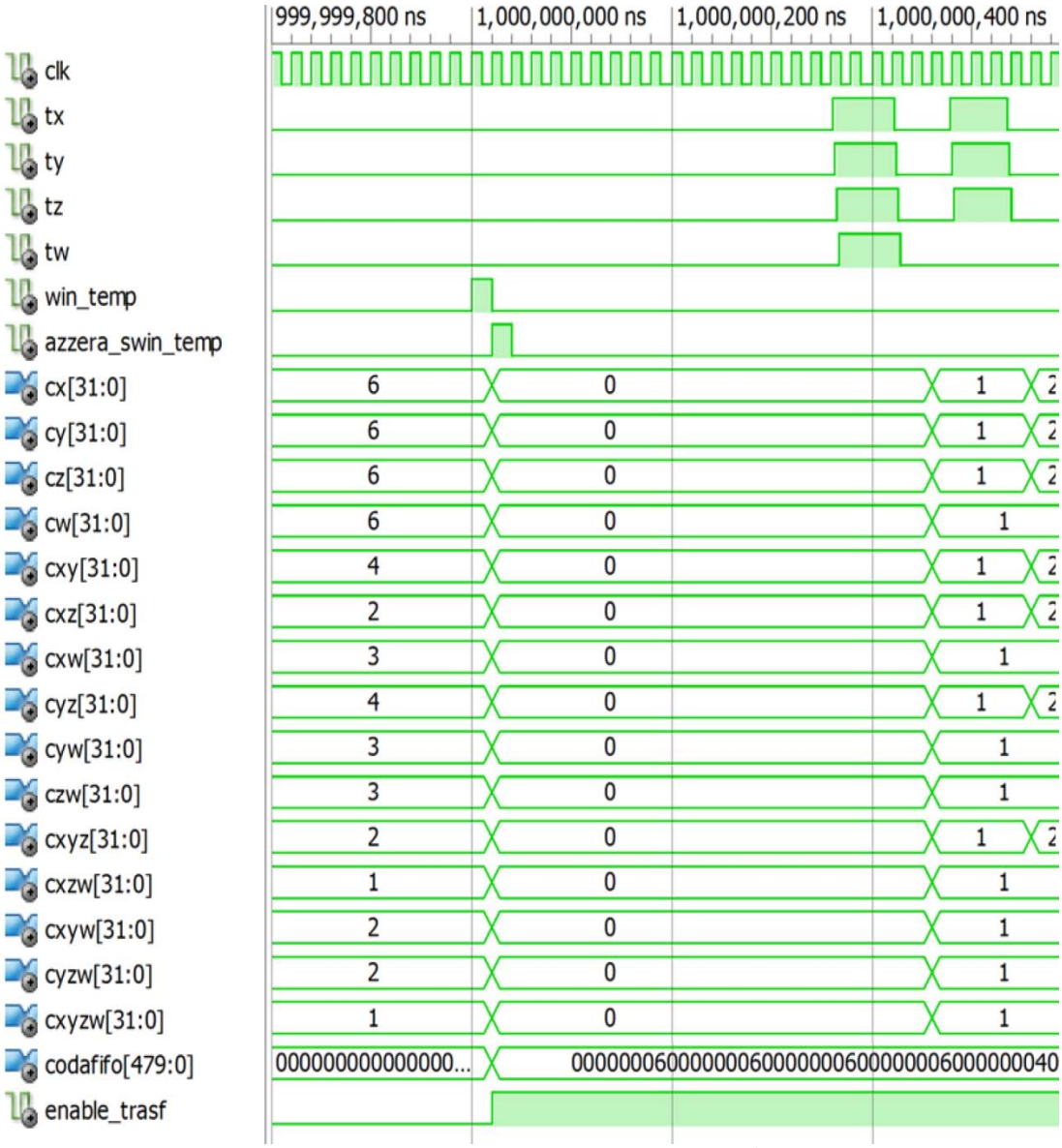


Figura 2.14: Simulazione dell'andamento dei segnali alla chiusura della finestra di conteggio

Oltre ai processi descritti finora, essenziali per la rivelazione del flusso dei raggi cosmici, sono previsti altri processi per la creazione di segnali della durata di 60 ns sincroni tra di loro alla rilevazione di una coincidenza singola, doppia (*xy*),

tripla (*xyz*) e quadrupla (Fig. 2.15). Questi segnali sono generati solo al termine della finestra di ascolto e sono stati previsti per essere utilizzati come segnali di sincronismo per altri sistemi.

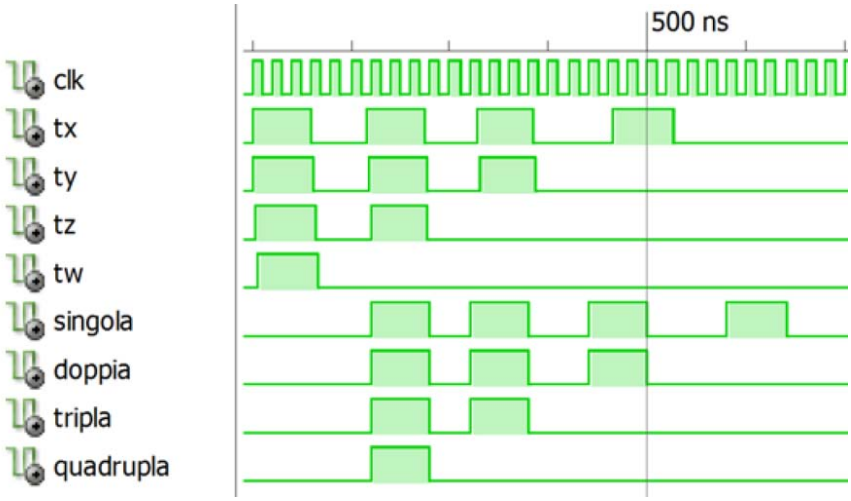


Figura 2.15: Simulazione dei segnali di trigger

Le simulazioni dei processi effettuate e riportate in questo paragrafo, visualizzate nelle figure, sono realizzate con Isim incluso nel pacchetto ISE della Xilinx. Questo sistema ci ha permesso di eseguire e visualizzare le simulazioni dopo ognuno dei quatto distinti passaggi necessari a convertire il firmware scritto in VHDL nella matrice di bit, indicante le connessioni presenti nelle celle dell'FPGA. Le simulazioni sopra riportate sono ottenute subito dopo la sintesi del codice VHDL; questa fase consiste nella trasformazione del codice VHDL in una netlist di componenti circuitali di base (porte logiche NAND e NOR e Flip Flop) in cui il simulatore non considera i ritardi intrinseci dei componenti. Le successive fasi di conversione sono il Translate, il Mapping e il Place&Route. Nella prima di queste fasi avviene la trasformazione della netlist creata dalla sintesi nei componenti che costituiscono le celle che compongono l'FPGA, tuttavia in questa fase ancora non sono considerati i ritardi intrinseci della logica. La fase successiva, ossia il Mapping, inserisce i componenti creati nella fase di Translate nelle celle dell'FPGA e provvede a raggruppare le celle con la stessa funzionalità in una sola: ciò serve perché le varie FPGA hanno architetture diverse. L'ultima fase, il Place&Route, è suddivisa in due sottofasi: il Place che provvede a posizionare le celle create nella fase di Mapping

all'interno dell'FPGA; e la sottofase di Route in cui il tool decide le connessioni tra le diverse celle. Per comprendere qual è il ritardo di risposta dell'FPGA allo stimolo occorre eseguire una simulazione Post-Place&Route. Attraverso questa simulazione, si rileva che il ritardo della risposta dell'FPGA allo stimolo proveniente da uno degli scintillatori è di circa 3 ns (Fig. 2.16).

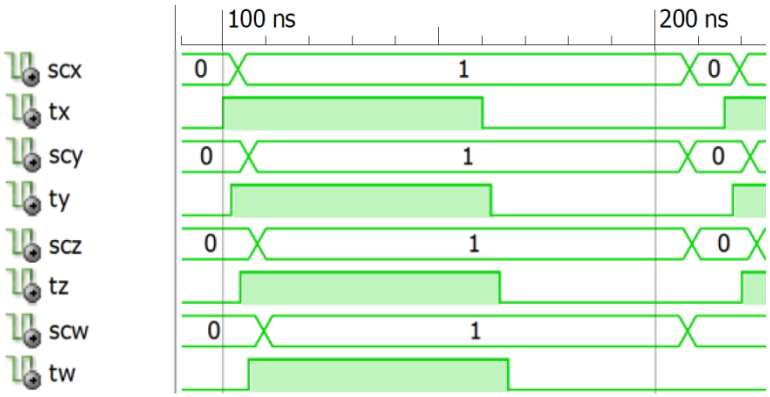


Figura 2.16: Simulazione Post-Place&Route tramite la quale si è valutato il ritardo di 3ns

Per poter testare il funzionamento del firmware ideato per l'FPGA si è progettata e realizzata una scheda di espansione per il sistema di sviluppo della Xilinx, in quanto la scheda di sviluppo non ha un numero sufficiente di I/O, a meno che non si utilizzano gli I/O accessibili tramite il connettore *Hirose FX2* a cui si collega la scheda di espansione. In Figura 2.17 è riportato il PCB della scheda di sviluppo, realizzato tramite CadSoft Eagle, e la scheda realizzata e raffigurata in Figura 2.18.

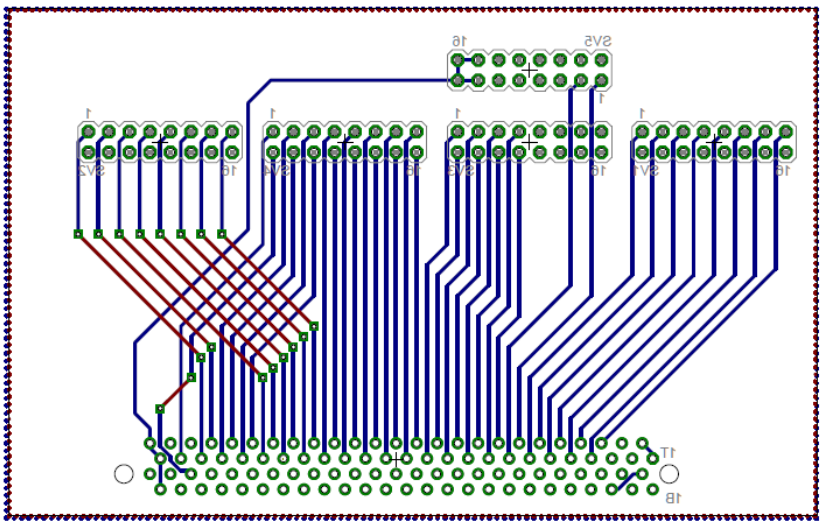


Figura 2.17: PCB della scheda di espansione

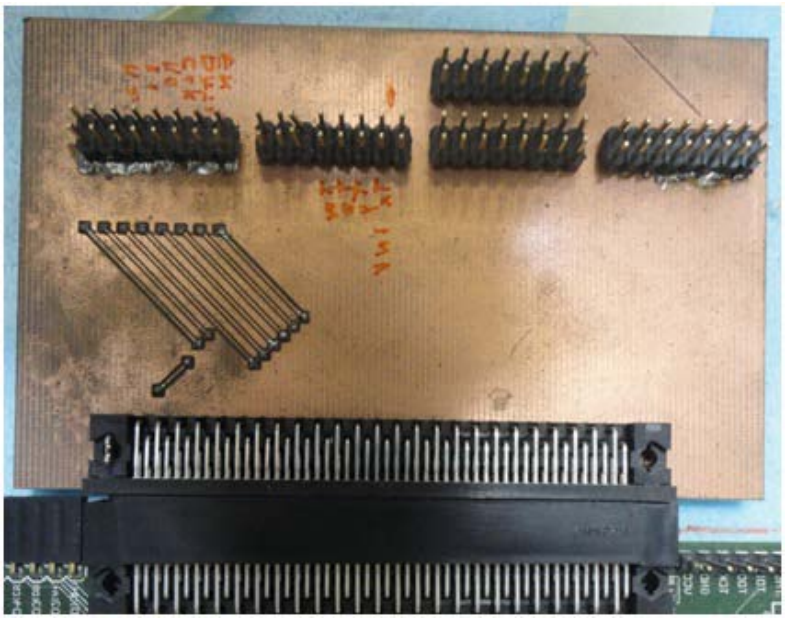


Figura 2.18: Scheda di espansione

2.4 Il microcontrollore

Il microcontrollore, nel prototipo del DAQ, deve gestire l'interfacciamento con il sottosistema di telemetria o con un computer, eseguire l'acquisizione dei dati dall'FPGA, il salvataggio dei dati in locale su SD CARD e l'interrogazione del GPS: tutte queste operazioni devono essere effettuate in un tempo inferiore a quello della durata della più piccola finestra di conteggio, cioè 1 s, per cui, nella realizzazione del firmware si sono valutati e verificati tutti i tempi delle singole operazioni che il microcontrollore deve allo scopo di identificare la combinazione migliore che ottimizzi il tempo di lavoro.

Prima di iniziare ad acquisire i dati dall'FPGA, il microcontrollore effettua alcune operazioni preliminari. Nel firmware implementato si controlla inizialmente se la memoria locale è presente, per evitare un tentativo di salvataggio a vuoto e quindi la perdita di tempo. Se l'esito del controllo è positivo, e cioè se la memoria è presente e funzionante, il firmware crea un file per il salvataggio delle informazioni, a cui viene assegnato un nome ricavato dalle informazioni provenienti dal GPS (in particolare dalla data e dall' orario in cui inizia l'acquisizione). Nel caso di informazioni del GPS non coerenti (numero di satelliti non sufficiente) il nome è

ottenuto interrogando un file salvato su SD CARD che conserva il nome dell'ultimo creato, assegnandone uno nuovo dopo 900 eventi. Per stabilire se le informazioni trasmesse dal GPS sono coerenti, quando viene acceso, bisogna aspettare una decina di secondi affinché il sistema rilevi un numero sufficiente di satelliti. Dopo aver creato il file per il salvataggio, il microcontrollore attende il segnale di dati disponibili (*enable_trasf*) dall'FPGA e, una volta intercettato il segnale *enable_trasf* , genera il segnale clock ed inizia il trasferimento dati. Il segnale di clock è generato dal microcontrollore in quanto questo è un dispositivo molto più lento rispetto all'FPGA (per ottimizzare il tempo di trasferimento sono stati testati diversi firmware, come verrà descritto successivamente in questo capitolo). Al termine dell'invio da parte dell'FPGA di tutti i bit dei contatori, il microcontrollore converte ogni valore binario ricevuto nel corrispettivo valore esadecimale (questo per occupare meno spazio e ottimizzare il tempo di salvataggio dei dati); accoda alle informazioni ottenute dall'FPGA quelle provenienti dal GPS e salva l'insieme su SD CARD; successivamente invia le informazioni al sottosistema di telemetria tramite la porta seriale o ad un computer, se il DAQ comunica tramite USB. Utilizzando lo stesso canale di comunicazione con cui invia le informazioni sul flusso dei raggi cosmici al sistema di elaborazione, il microcontrollore riceve i comandi di gestione della durata della finestra di conteggio, di pausa e di richiesta di informazioni sullo stato del sistema. Questi comandi vengono mandati al microcontrollore mediante l'invio di un carattere numerico in particolare:

- 0 => imposta a "001" il bus *mod_temp* (1 s);
- 1 => imposta a "010" il bus *mod_temp* (5 s);
- 2 => imposta a "011" il bus *mod_temp* (10 s);
- 3 => imposta a "100" il bus *mod_temp* (30 s);
- 4 => imposta a "101" il bus *mod_temp* (60 s);
- 5 => imposta a "110" il bus *mod_temp* (300 s);
- 6 => imposta a "111" il bus *mod_temp* (600 s);
- 7 => informazioni sullo stato del sistema;
- 8 => PAUSE.

Un insieme di led (Fig. 2.20) presenti sulla scheda di sviluppo del microcontrollore permette di evidenziare l'avvenuta ricezione dei comandi impartiti; ogni led è acceso o spento a seconda del comando ricevuto dal microcontrollore. Nella figura 2.19 è riportato il flow-chart del firmware del microcontrollore.

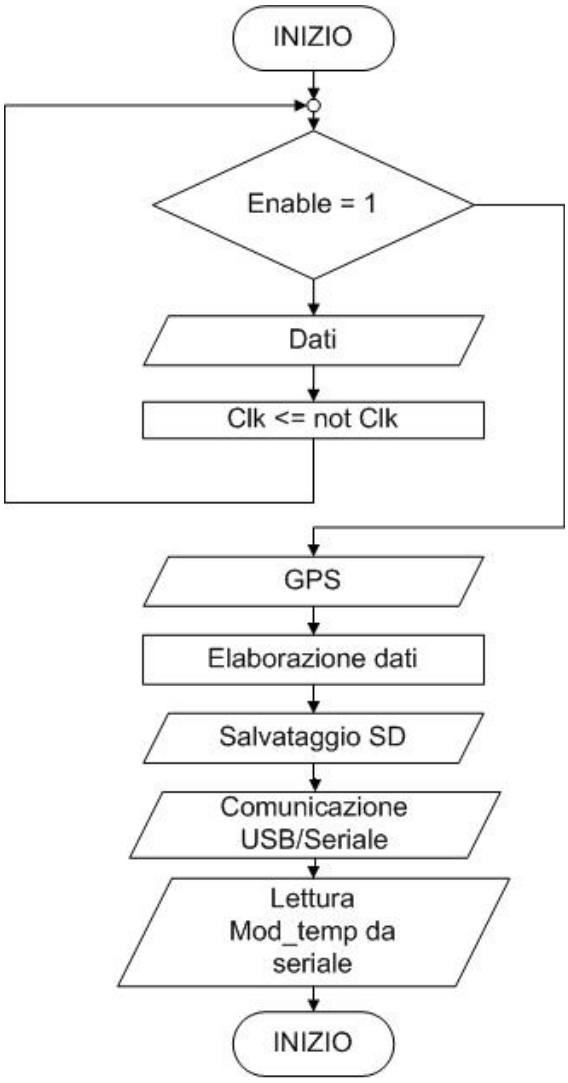


Figura 2.19: Flow-chart microcontrollore

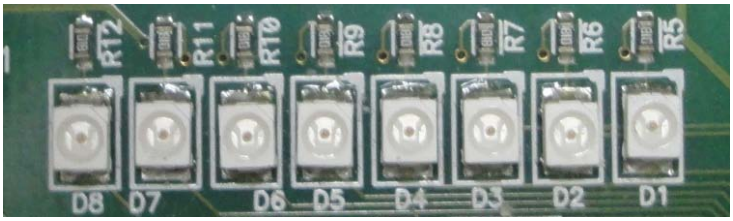


Figura 2.20: LED presenti sulla scheda di sviluppo

2.4.1 Acquisizione e elaborazione dati

Dopo la fase di inizializzazione, il microcontrollore inizia quella di comunicazione con l'FPGA per ottenere il primo set di misure sul flusso dei raggi cosmici, tramite un bus composto da tre canali unidirezionali, due da FPGA a microcontrollore e uno da microcontrollore a FPGA. Nel sistema di comunicazione ideato, il microcontrollore, essendo il dispositivo più lento dell'intero DAQ, determina la frequenza di trasferimento della comunicazione. Il microcontrollore abilita il clock di trasferimento (Fig. 2.21: *clk_trasf*) solo quando il segnale *enable_trasf* (Fig. 2.21) è impostato a livello alto dall'FPGA, che provvede a trasferire il bit d'informazione (Fig. 2.21: *dati*) ad ogni fronte di c*lk_trasf* . Dopo aver ricevuto il bit inviato, il microcontrollore inizia il nuovo periodo di clock. L'FPGA imposta il segnale *enable_trasf* a livello logico basso dopo aver inviato tutti i bit della coda in cui ha salvato le informazioni, segnalando in questo modo, al microcontrollore, che non ha al momento altri dati da inviare.

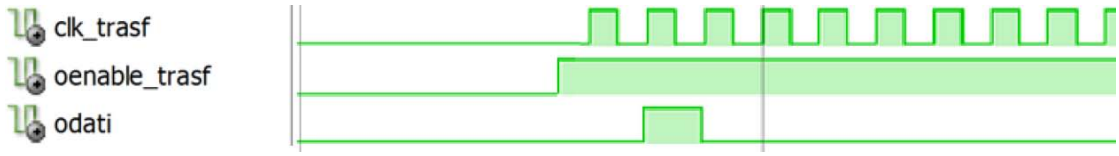


Figura 2.21:Sistema di comunicazione tra FPGA e microcontrollore

Il microcontrollore riceve le informazioni in binario e le converte in esadecimale per limitare l'occupazione di memoria e per ridurre il tempo di salvataggio su SD CARD: si è cercato di ottimizzare temporalmente quest'operazione testando diversi tipi di firmware di acquisizione.

```
FOR i=1 TO 480 'ciclo per la lettura dell'intera coda
   HIGH clk  'gestione clock di trasferimento
   LOW clk
   lett =PORTG.3 'lettura del bit
   DATO = DATO + (lett*shift) 'formazione del gruppo di bit
   shift = shift << 1
   'quando acquisisco 4 bit effettuo la conversione del dato
   IF (shift = 16 ) THEN
     ARRAYWRITE  DATOHEX,[HEX DATO] 'effettuo la conversione
     CONT[j]=DATOHEX 'accodo i dati ai precedenti
     j=j-1
     shift =1
     DATO=0
   ENDIF
NEXT
```

Figura 2.22: Primo firmware del sistema di comunicazione

Il primo tipo di firmware (Fig. 2.22), testato sul microcontrollore, acquisisce quattro bit in una variabile d'appoggio tramite l'utilizzo di una maschera e la converte in esadecimale, impiegando circa 6 ms (Fig. 2.23) .

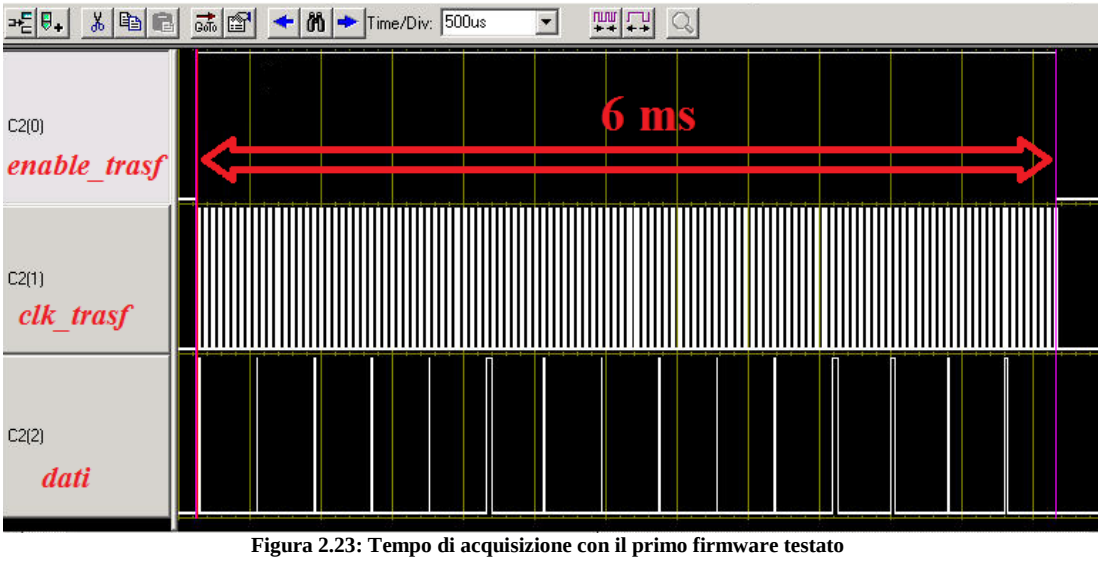


Figura 2.23: Tempo di acquisizione con il primo firmware testato

Il secondo firmware (Fig. 2.24), a differenza del primo, non utilizza la variabile d'appoggio ma il bit è scritto tramite una maschera direttamente nella posizione corretta della variabile *Long*[1], la cui lunghezza, 32 bit, è pari a quella dei contatori: la conversione in esadecimale avverrà solo nel momento del salvataggio su SD CARD e nell'invio delle informazioni al sottosistema di telemetria

```
'ciclo di lettura dell'intera coda
FOR i=1 TO 480
   HIGH clk  'clock di trasferimento
   LOW clk
   lett =PORTG.3 'lettura del bit
   DATO[j] = DATO[j] + (lett*shift)    'creazione del dato in decimale
   shift = shift >> 1     'maschera di shift
   IF (shift = 0 ) THEN    'azzeramento maschera
     j=j+1
     shift =2147483648
     DATO[j]=0
   ENDIF
NEXT
```

Figura 2.24: Secondo firmware del sistema di comunicazione

[1] Long è un tipo di dato che il linguaggio di programmazione del microcontrollore, il PICBASIC, implementa solo per i microcontrollori della famiglia PIC18F corrispondente a 32 bit.

Con il secondo tipo di firmware il tempo d'acquisizione delle informazioni è aumentato a 8ms (Fig. 2.25).

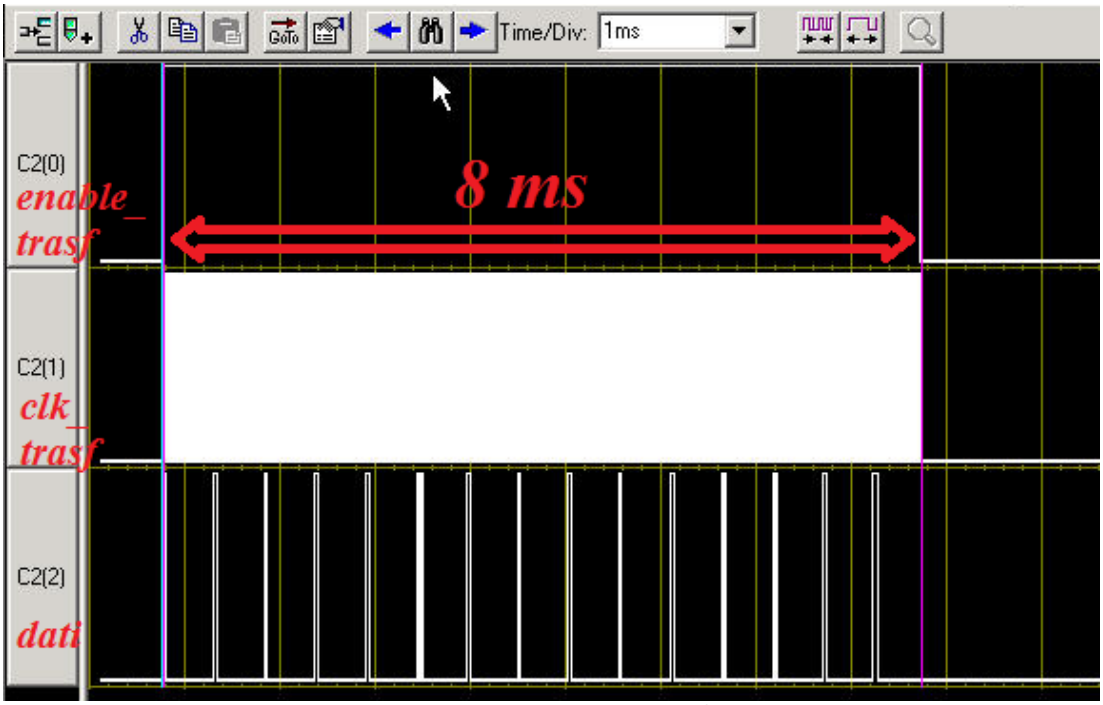


Figura 2.25: Tempo di acquisizione con il secondo firmware testato

```
'ciclo d'acquisizione dei 15 contatori
FOR i=0 TO 14
   'ciclo per la lettura dei 32 bit
   FOR j=0 TO 31
       HIGH clk 'clock
       LOW clk
       'lettura del bit e inserimento nella corretta posizione
       APP.0[j]=PORTB.1
   NEXT j
   DATO[i] = APP
NEXT i
```

Figura 2.26: Terzo firmware testato

Il terzo firmware (Fig. 2.26) testato utilizza due cicli di acquisizione nidificati in modo da poter utilizzare quindici variabili long, una per ogni contatore. I bit d'informazione ricevuti sono collocati nella corretta posizione delle variabili tramite l'indice del secondo ciclo. Il tempo d'acquisizione di questo firmware è pari a 3,2 ms (Fig. 2.27) e la frequenza di trasferimento tra microcontrollore e FPGA è di circa 150 kHz.

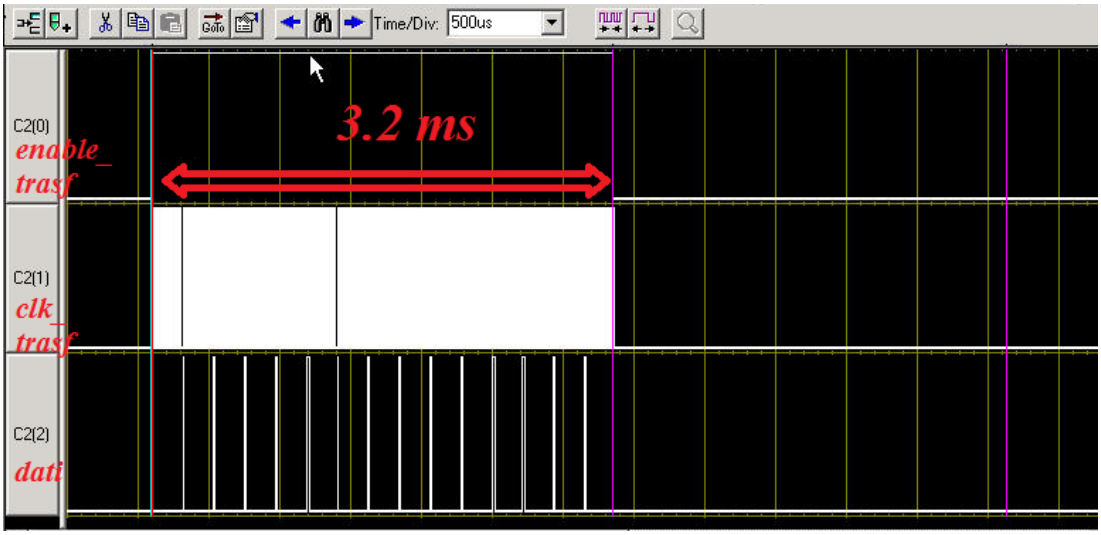


Figura 2.27: Tempo di acquisizione del terzo firmware testato

L'ultimo testato è una parte del firmware del DAQ perché ottimizza al meglio i tempi d'acquisizione.

2.4.2 Il GPS

Il GPS utilizzato è l'iQLassen (Fig 1.19) della Trimble, che lavora con tensione di alimentazione di 3,3 V. Nel prototipo, il dispositivo è montato sulla stessa basetta in cui è presente il modulo di alimentazione e la batteria di backup. Questo dispositivo ha due porte seriali per comunicare, una utilizza il protocollo TSIP mentre la seconda il protocollo NMEA.

Il protocollo NMEA 0183, gestito dalla National Marine Electronics Association, fornisce le informazioni direttamente in ASCII, consentendo, di conseguenza, di concatenare le informazioni provenienti dal GPS alle informazioni sul flusso dei raggi cosmici, senza eseguire alcuna elaborazione sui dati. Il NMEA è un protocollo a pacchetti, ogni pacchetto inizia con i caratteri $GP ed è seguito da un prefisso che identifica il contenuto del pacchetto. Comprendendo le informazioni del pacchetto con il relativo checksum di controllo e la sequenza CRLF che ne determina la fine, ogni pacchetto ha un massimo di 82 caratteri. I principali pacchetti del protocollo sono:

- *$GPGGA*: fornisce informazioni sull'ora (UTC), la posizione (latitudine e longitudine) del ricevitore, l'altitudine, il numero di satelliti;
- *$GPGLL*: contiene latitudine, longitudine e l'ora;
- *$GPRMC*: contiene l'ora, la data, la posizione e il rate dei dati provenienti dal ricevitore;
- *$GPGSA*: contiene il numero di satelliti visti e le informazioni operative del ricevitore;
- *$GPGSV*: contiene il numero di satelliti visti, e la posizione, la sequenza identificativa e l'SNR (Signal to Noise Ratio) dei quattro satelliti che sente meglio;
- *$GPVTG*: contiene informazioni sulla COG (Course Over Ground) e sulla SOG (Speed Over Ground). Distanza ottimizzata e distanza rispetto al fondo.

Le informazioni che caratterizzano il flusso dei raggi sono:

- data e ora in cui avviene l'acquisizione dei dati,
- latitudine, longitudine e altitudine del DAQ.

Per ottenere tutte queste informazioni occorre utilizzare i pacchetti *GGA* e *RMC*. Il GPS utilizzato per default riceve solo i pacchetti *GGA* e *VTG* quindi per ricevere anche il pacchetto RCM si sono dovute modificare e successivamente le impostazioni del ricevitore, salvandole in maniera definitiva sulla memoria non volatile.

Il GPS è dotato di una batteria di backup che ci permette di conservare nella memoria volatile le informazioni su data, ora e posizione dei satelliti anche quando è spento, in modo che questo possa essere operativo nel più breve tempo possibile quando viene riacceso (risparmiando circa dieci secondi).

Come si nota dal flow-chart (Fig. 2.19), nel firmware l'interrogazione del GPS è effettuata subito dopo aver completato il trasferimento dai dati dall'FPGA.

2.4.3 SD CARD

Il firmware del microcontrollore, dopo aver elaborato i dati e ricevuto le informazioni del GPS tramite porta seriale, esegue il salvataggio dei dati in locale su un file allocato su una SD CARD [31], che cambia ogni 900 eventi registrati. La decisione di creare diversi file per il salvataggio degli eventi è necessaria per ottimizzare la catalogazione delle misure effettuate e per non far gestire al microcontrollore dei file di dimensioni molto grandi.

Per gestire e comunicare con l'SD CARD , il microcontrollore può usare diversi protocolli, in questo caso si è scelto di usare il protocollo SPI. La gestione dell'SD CARD, tramite questo protocollo, si basa sulla manipolazione di file di testo. L'SPI è il protocollo che consente ad un dispositivo master di comunicare con uno o più slave in modo sincrono grazie ad una linea di clock, controllata dal master e che controlla anche il rate di trasmissione dei dati. Il rate naturalmente ha un limite superiore dettato dal massimo rate sopportabile dal dispositivo slave. La comunicazione del protocollo è di tipo seriale, infatti si basa sulla gestione di registri a scorrimento (shift register) e full duplex, avendo a disposizione una linea di trasmissione e una di ricezione. Per questo lavoro il dispositivo master è il microcontrollore, quello slave è l'SD CARD e per implementare il collegamento SPI si ha bisogno di 6 linee di comunicazione:

- Due di alimentazione (3,3V e GND)
- Una per il clock (*SCL*);
- Una linea dati da microcontrollore alla SD CARD (*SDO*);
- Una linea dati da SD CARD a microcontrollore (*SDI*);
- Una per selezionare il dispositivo slave (*CS*);

Per gestire l'SD CARD, tramite il microcontrollore con il protocollo SPI, esistono apposite librerie che, in fase di programmazione, sono state modificate per renderle del tutto compatibili con il microcontrollore scelto per questo circuito. Il PIC18F87J50 usato nel DAQ richiede opportune definizioni di hardware e dei

registri affinché poassa gestire due porte SPI. Le interfacce seriali, utilizzate dai microcontrollori per comunicare, si servono del modulo *MSSP* (Master *Synchronous Serial Port*), il quale a sua volta implementa o il protocollo SPI o il protocollo I2C. Il microcontrollore utilizzato ha due *MSSP* indipendenti. Per gestire l'SPI ogni modulo MSSP utilizza quattro registri: *SSPXCON1*, *SSPXSTAT*, *SSPXBUF* e *SSPXSR* (X assume valore 1 o 2 in base al MSSP utilizzato). I primi due registri sono dei registri di controllo e di stato, il terzo registro è un registro buffer per i dati e il quarto è lo shif register. Il registro *SSPXSTAT* permette di decidere se inviare i dati sul fronte di salita o sul fronte di discesa (*CKE*), quando effettuare il campionamento tramite il bit *SMP* e contiene inoltre un bit che indica lo stato del buffer (*BF*). Il registo *SSPXCON1* è quello che determina l'operazione dell'SPI, in particolare ha un bit (*WCOL*) di verifica della collissione; un bit (*SSPOV*) che indica che il buffer è stato sovrascritto prima della lettura e deve essere azzerato da firmware; un bit per l'abilitazione del SPI (*SSPEN*); un bit (*CKP*) per determinare se leggere sullo stato alto o su quello basso del clock e se il microcontrollore è un dispositivo slave o master e la sua velocità di funzionamento (*SSPM3-SSPM0*).

Oltre ai registri appena descritti, che riguardano direttamente il modulo MSSP, ci sono anche i registri per la gestione di interrupt. Nella figura sottostante (Fig. 2.28) sono riportati i registri direttamente o indirettamente coinvolti nella gestione del SPI.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
PIR1	PMPIF	ADIF	RC1IF	TX1IF	SSP1IF	CCP1IF	TMR2IF	TMR1IF
PIE1	PMPIE	ADIE	RC1IE	TX1IE	SSP1IE	CCP1IE	TMR2IE	TMR1IE
IPR1	PMPIP	ADIP	RC1IP	TX1IP	SSP1IP	CCP1IP	TMR2IP	TMR1IP
PIR3	SSP2IF	BCL2IF	RC2IF	TX2IF	TMR4IF	CCP5IF	CCP4IF	CCP3IF
PIE3	SSP2IE	BCL2IE	RC2IE	TX2IE	TMR4IE	CCP5IE	CCP4IE	CCP3IE
IPR3	SSP2IP	BCL2IP	RC2IP	TX2IP	TMR4IP	CCP5IP	CCP4IP	CCP3IP
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
TRISD	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0
TRISF	TRISF7	TRISF6	TRISF5	TRISF4	TRISF3	TRISF2	—	—
SSP1BUF	MSSP1 Receive Buffer/Transmit Register							
SSPxCON1	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
SSPxSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
SSP2BUF	MSSP2 Receive Buffer/Transmit Register							
ODCON3[1]	—	—	—	—	—	—	SPI2OD	SPI1OD

Figura 2.28: Registri utilizzati nel SPI

La libreria utilizzata rispetto a quella fornita della casa produttrice, come accennato precedentemente, è stata modificata agendo sui registri *SSPEN*, *WCOL*, *BF* e *SPPIF* (bit di gestione dell'interrupt) per renderla compatibile con il microcontrollore utilizzato. La libreria utilizzata permette inoltre di impostare la frequenza del clock del SPI; questo parametro è molto importante perché il suo valore è condizionato dalla massima frequenza con cui può funzionare il dispositivo slave, l'SD CARD, ed è settato a 1/64 della frequenza del clock (nel nostro caso è di circa 750 kHz).

La libreria richiede anche di definire a quali porte sono collegate le linee del SPI, cioè la linea *SDI*, *SDO*, *SCL* e la linea *CS*. Oltre a queste linee si devono anche definire, se presenti, la linea *CD* (Card Detection) e la linea *WP* (Write Protection) che non dipendono dall'SD CARD, ma dal socket di SD CARD montato sul PCB, in quanto queste linee sono dei contatti hardware e sono attive basse, cioè assumono il valore zero se l'SD CARD è inserita con la protezione della scrittura attiva. Il connettore (Fig. 2.29) da noi utilizzato ha entrambi i contatti WP e CD. La Figura 2.30 riporta sia i collegamenti con il microcontrollore che le definizioni dei collegamenti usati nel firmware.

```
SD_WE       VAR   PORTA.3     ' SD CARD protetta
SDI         VAR   PORTC.4     ' data in
SCL         VAR   PORTC.3     ' SPI clock
```

```
SD_CS       VAR   PORTA.4     ' SD CARD chip select
SD_CD       VAR   PORTA.0     ' SD CARD detect
SDO         VAR   PORTC.5     ' SPI data out
```

Figura 2.29: Socket SD CARD

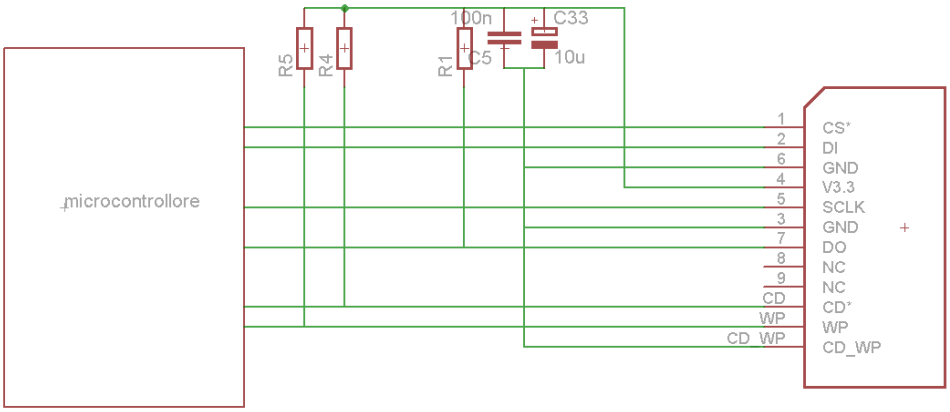


Figura 2. 30: Collegamenti tra microcontrollore e SD CARD

Le librerie disponibili sono due: una che lavora con SD CARD con file system FAT16; la seconda lavora anche con SD CARD HC con file system FAT16/FAT32. Entrambe le librerie possono essere utilizzate solo con i microcontrollori della famiglia PIC18. Per la realizzazione del DAQ si è deciso di utilizzare la libreria più performante, cioè quella che può scrivere anche sulle SD CARD HC.

Come si è detto, per scrivere e leggere i file da SD CARD si deve operare come se fossero dei veri file di testo. Perciò la libreria mette a disposizione diverse subroutine che vengono richiamate dal codice principale; ogni subroutine restituisce un codice di errore (*FAT_error*). Quando il codice è uguale a 0 l'operazione è completata in maniera corretta. Le principali subroutine implementate, sono:

- *FSInit*: Inizializza l'SD CARD impostando le linee di I/O, una delle prime operazioni da eseguire per poterla utilizzare;

- *FSExit*: Subroutine utilizzata quando si vuole rimuovere l'SD CARD, in questa subroutine non è implementato il codice di errore;
- *FSfopen*: Utilizzata per aprire un file esistente o crearne uno nuovo. Le modalità di apertura del file sono tre e vengono definite tramite la variabile *FAT_mode*:
    - *r*: in lettura (il file viene aperto e può essere solo letto);
    - *w*: in scrittura (se il file esiste viene aperto e inizia a scrivere dall'inizio del file, altrimenti lo crea);
    - *a*: in append (il file viene aperto e inizia a scrivere in coda al file).
- Il nome del file da aprire è impostato dalla variabile *FAT_FileName* nel formato 8.3 (8 caratteri per il nome più 3 caratteri per l'estensione);
- *FSfclose*: Chiude il file precedente aperto;
- *FSfread*: Subroutine utilizzata per leggere il file. Per far ciò si deve definire, inizialmente, il numero di byte da leggere tramite la variabile *FAT_count*. I byte letti sono assegnati all'array *FAT_src*;
- *FSfwrite*: Subroutine utilizzata per scrivere il file. Anche in questo caso si deve definire, inizialmente, quanti byte si vogliono leggere tramite la variabile *FAT_count*. I byte da scrivere sono assegnati all'array *FAT_dest*.;
- *FSremove*: Sub routine che rimuove il file puntato da *FAT_FileName;*
- *FINDfirst* e *FINDnext*: Trova i file nell' SD CARD;
- *FSFspacefree*: Restituisce lo spazio libero in kByte.

I file creati e utilizzati per il salvataggio delle informazioni devono essere salvati nella directory principale dell'SD CARD, in quanto la libreria non riconosce

le sottodirectory e, nel caso si utilizzasse la libreria FAT16, il numero massimo di file salvabile è 512.

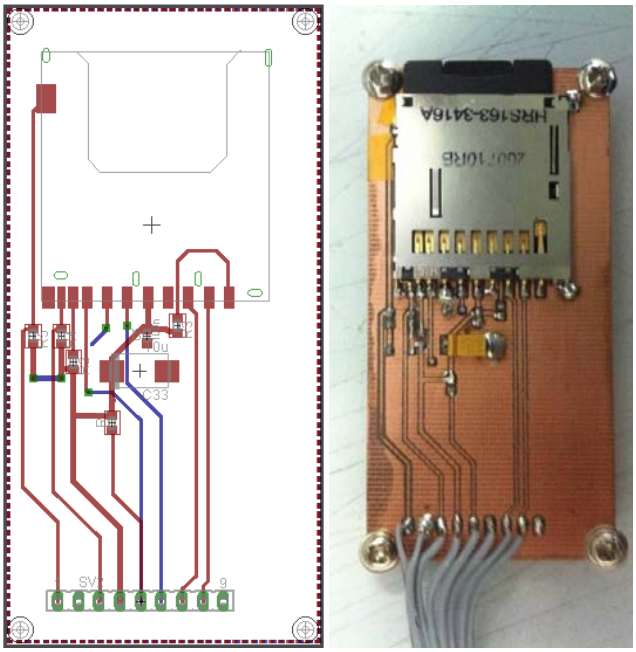


Figura 2.31: PCB e adattatore dell'SD CARD

Il sistema di sviluppo del microcontrollore non ha uno slot per ospitare l'SD CARD perciò si è realizzato un adattatore (Fig. 2.31) che la collega al sistema di sviluppo mediante un cavo flat. Nell'adattatore si sono previsti anche i controlli *CD* e *WP*: il primo è usato nel firmware in modo che il DAQ funzioni anche senza SD CARD, se questo controllo non fosse presente il microcontrollore si bloccherebbe sull'inizializzazione del SD CARD con un errore rintracciabile mediante la variabile *FAT_error*; il secondo controllo è solo un controllo software in quanto, se la SD CARD è protetta da scrittura, il microcontrollore riesce a scrivere ugualmente. In questo caso si è deciso che il DAQ continui a salvare i dati e l'utente, a richiesta, può avere informazioni sullo stato del sistema (opzione 7 del menu del controllo della finestra di conteggio).

È stata eseguita una stima dei tempi necessari per compiere le varie operazioni per decidere come ottimizzare il salvataggio delle informazioni, in particolare se è conveniente scrivere tutte le informazioni in un blocco o in diversi blocchi e quando e come aprire il file.

Il tempo di inizializzazione dell'SD CARD pari a 32 ms (Fig. 2.32) non influisce sul salvataggio delle informazioni, perché viene eseguito all'inizio dell'esecuzione del firmware e prima di iniziare l'acquisizione delle informazioni dall'FPGA.

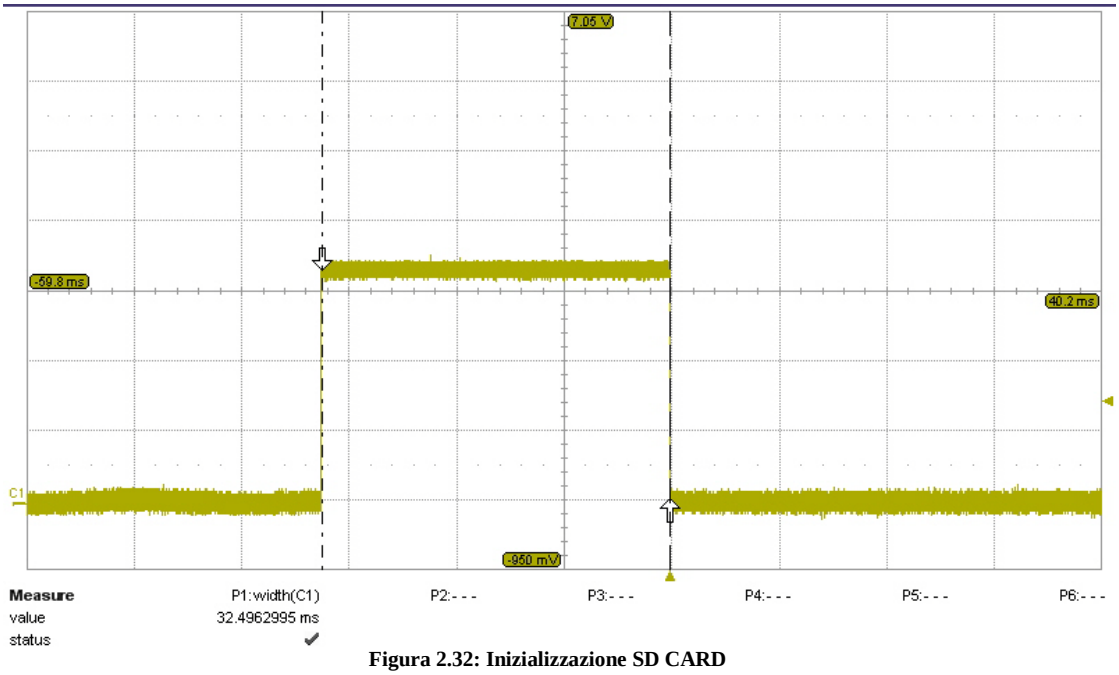


Figura 2.32: Inizializzazione SD CARD

Le prestazioni temporali rilevate nell'apertura dei file, in particolare in scrittura (*FAT_mode =w*) e in append *(FAT_mode =a*), hanno evidenziato un impiego maggiore di tempo pari a 296 ms per l'apertura in scrittura (Fig 2.33), in quanto questa deve creare il file all'interno dell'SD CARD invece, l'apertura in append (Fig. 2.34), consistente nel puntare il file e posizionarsi alla fine, impiega solo 37 ms.

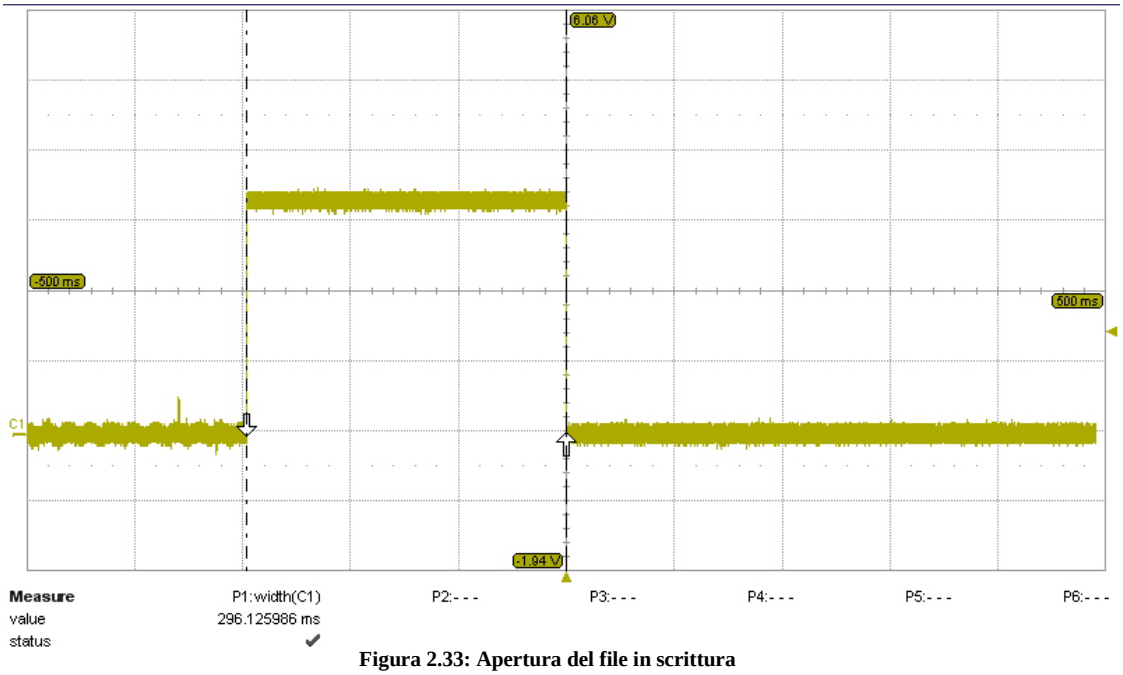


Figura 2.33: Apertura del file in scrittura

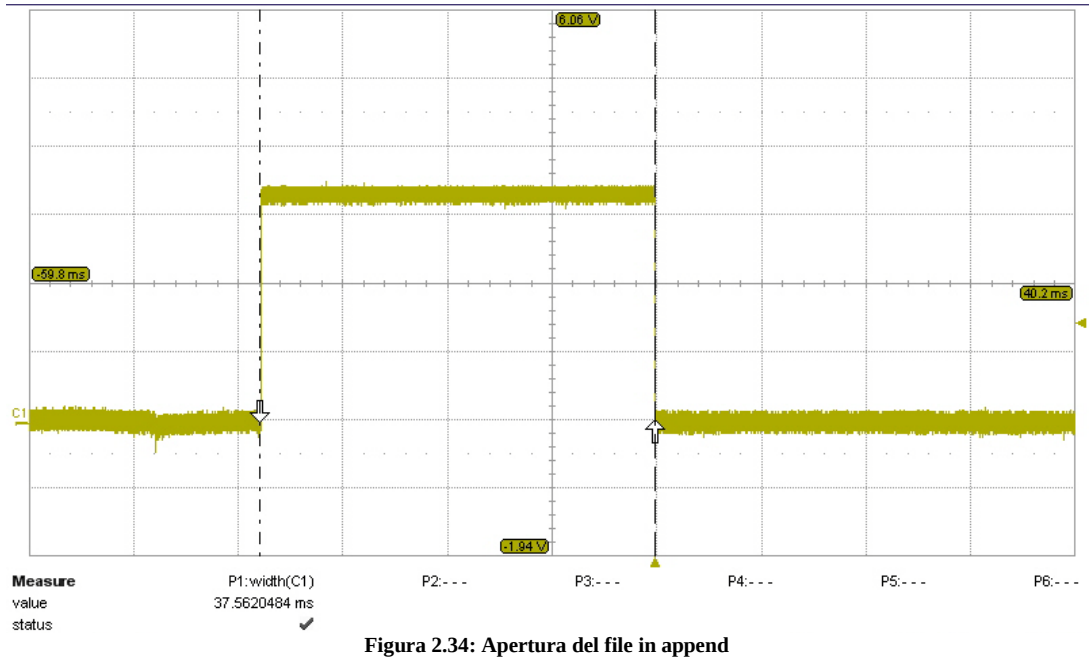


Figura 2.34: Apertura del file in append

Dopo l'apertura, si è valutato anche il tempo di scrittura confrontando in particolare il tempo di scrittura di un singolo carattere e il tempo di scrittura di un blocco, 166 caratteri, che corrisponde alla stessa lunghezza del blocco di informazioni da scrivere sull' SD CARD a chiusura di ogni finestra di conteggio. Questo confronto è servito per capire se è più conveniente scrivere tutte le

informazioni in una sola volta (un blocco di 166 caratteri) o scriverle in diverse blocchi. Dalle prove effettuate, si nota che per scrivere un unico carattere occorre 9.1 ms (Fig 2.35), invece per scrivere 166 caratteri occorre 13 ms (Fig 2.36): questo vuol dire che il tempo maggiore per la scrittura si perde per accedere nella giusta posizione all'interno dei file.

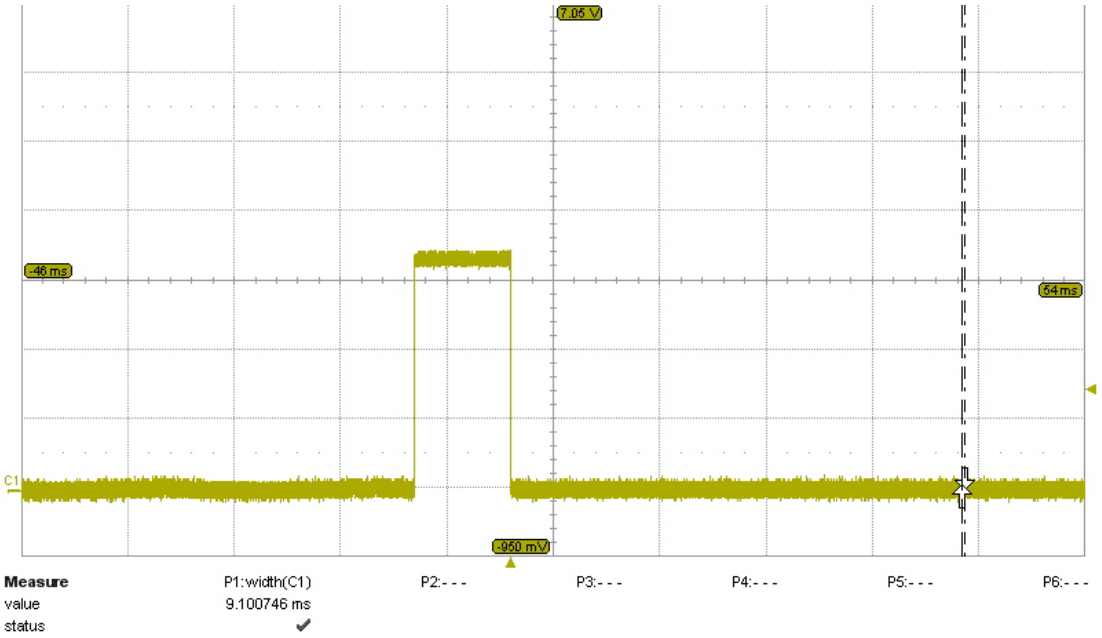


Figura 2.35: Tempo di scrittura di un carattere

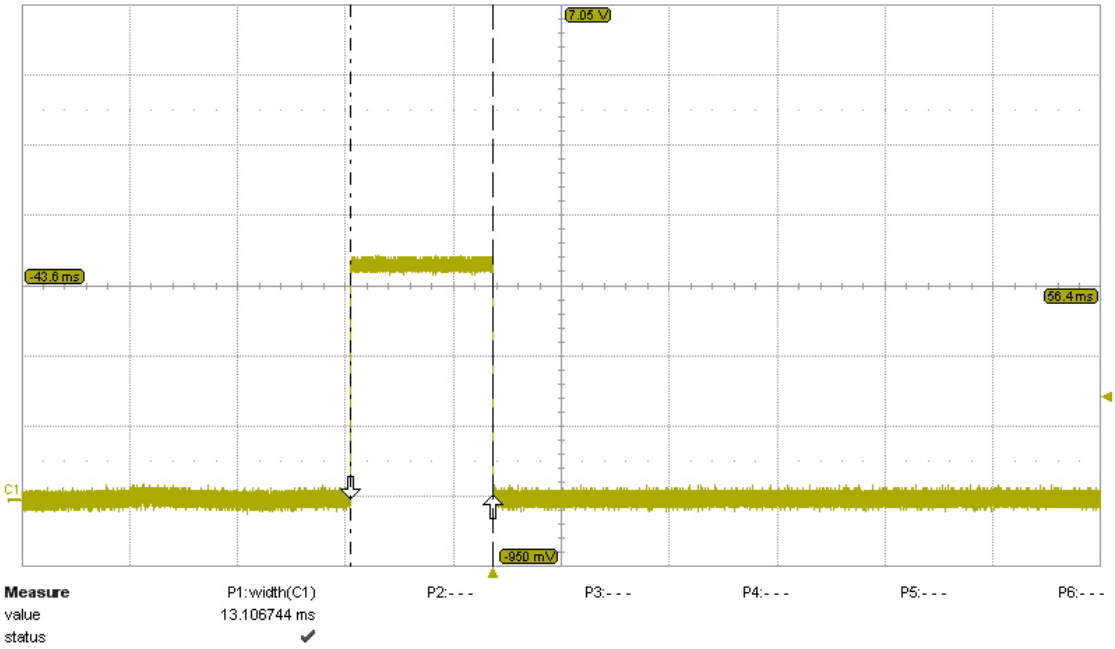


Figura 2.36: Tempo di scrittura di 166 caratteri

L'ultima misura effettuata è il tempo di chiusura del file (Fig. 2.37) che è pari a circa 34 ms.

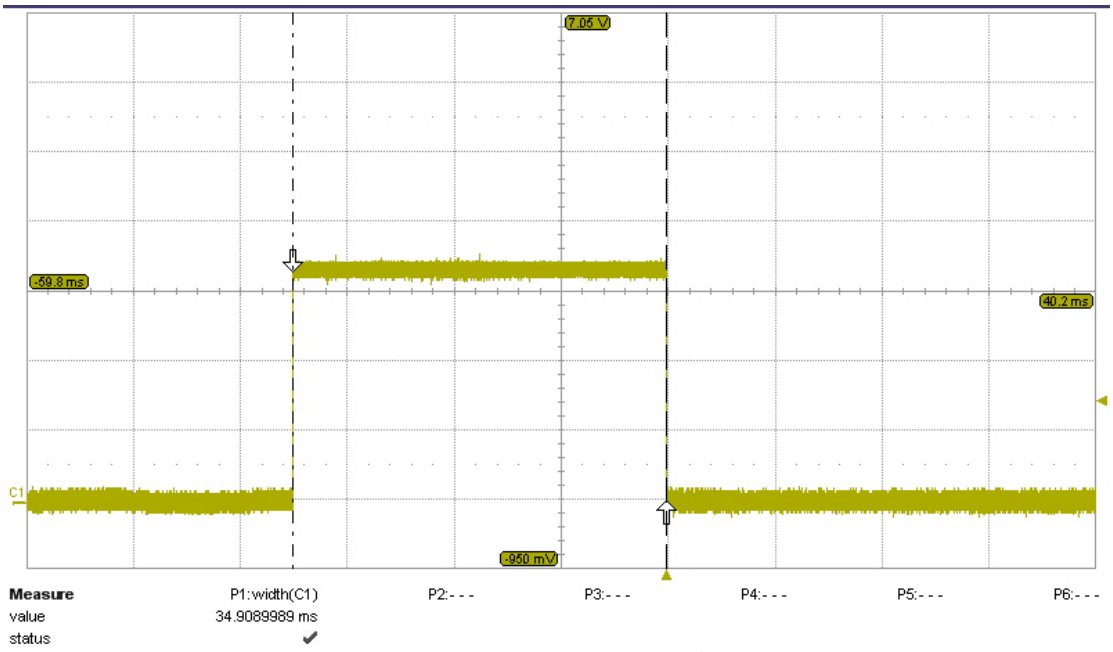


Figura 2.37: Tempo di chiusura del file

Dopo aver effettuato tutte le rilevazioni, la scelta che ci permette di effettuare il salvataggio delle informazione nel minor tempo possibile e nello stesso tempo di avere l'affidabilità del salvataggio, è quella di scrivere tutte le informazioni in un blocco di 166 caratteri, e per ogni scrittura aprire e chiudere il file in append. Quest'ultima è dovuta alla necessità di avere una maggior affidabilità dei dati infatti, nel caso di problemi al DAQ, chiudendo ogni volta il file non c'è il rischio di perdere l'intero set di misure. Con la combinazione scelta per effettuare il salvataggio dei dati occorre circa 85.47 ms, tempo più che sufficiente per la gestione della finestra di integrazione (per default di 1 s).

2.4.4 USB (Universal Serial Bus)

Il DAQ comunica con il sistema che effettua l'analisi dati in real time tramite porta seriale se è collegato al sottosistema di telemetria o porta USB se non allocato sul pallone. Entrambe le porte, nel DAQ, sono gestite dal microcontrollore anche se in modo completamente differente perché il framework della porta USB è molto più complesso rispetto alla porta seriale.

Per gestire la porta USB [32], oltre ad avere l'hardware predisposto cioè il *SIE*[2] *(Serial Interface Engine,* Fig. 2.38*)* che supporta sia la modalità *full-speed* che *low-speed*, il microcontrollore utilizza una serie di informazioni generali che descrivono la tipologia e il funzionamento del dispositivo. Queste informazioni sono contenute in un file chiamato *descrittore* che viene richiamato dal firmware del microcontrollore. In questo paragrafo si descriverà brevemente l'architettura dell'USB, i registri e l'hardware del microcontrollore utilizzato per il controllo dell'USB, premessa necessaria per spiegare in modo esaustiva il firmware del microcontrollore del DAQ.

L'architettura USB ha due componenti fondamentali:

- l'USB Host, unico nel sistema di comunicazione generalmente corrispondente ad un computer;
- l'USB Device, tipi di dispositivi collegati all'Host, suddivisi in diverse classi[3] in base alle loro funzionalità.

Il microcontrollore è un USB Device della classe *HID* (i dispositivi di questa classe hanno la possibilità di generare interrupt). Il componente principale dell'architettura è l'Host che controlla la comunicazione dei diversi device mediante un protocollo a *token*. L'Host riconosce il Device con cui comunica attraverso il PID (Product Id) e il VID (Vendor ID), entrambe contenute nel *descrittore*.

Ogni Device, e quindi anche il microcontrollore, è composto da un insieme di *Endpoints* che permettono l'invio e la ricezione di informazioni tramite una *pipe* e ne determinano il tipo di trasferimento. Ogni *Endpoint* è identificato tramite un ID e per cui sullo stesso Device si possono eseguire diverse funzioni utilizzando diversi *Endpoint*. Esiste inoltre un *Endpoint* di controllo, cioè l'*Endpoint0* che viene utilizzato per la configurazione del Device. L'Host comunica con il relativo *Endpoint*

[2] Il SIE effettua la serializzazione e deserializzazione del flusso dei dati, codifica e decodifca i dati secondo la logica NRZI, genera il CRC e rileva il PID.

[3] Le principali classi sono HID (Human interface Device), MSD (Mass Storage Device) e CDC (Communication Device Class).

attraverso la tripletta di informazioni PID,VID e ID *Endpoint*, fornita dal Device. Per ogni Endpoint deve essere specificato il tipo di trasferimento utilizzato, lo standard USB ne definisce quattro tipi:

- Trasferimento Isochronous: utilizzato per trasferire grandi flussi di dati e mantenere basso il tempo di volo[4] delle informazioni trascurando l'integrità dei dati.
- Trasferimento Bulk: utilizzato per trasferire grandi flussi di dati e garantire l'integrità dei dati ma non il tempo di volo.
- Trasferimento Interrupt: per piccoli flussi di dati con tempo di volo basso e una buona integrità dei dati. In questo tipo di trasferimento l'host esegue un polling sull'interrupt *Endpoints* dei Device attivi in cerca della periferica pronta per inviare i dati.
- Trasferimento Control: questo tipo di trasferimento avviene durante la fase di setup del device e i pacchetti sono inviati in maniera casuale e con priorità massima.

Normalmente per i device della classe HID si usa il trasferimento Interrupt, che è quello utilizzato nel firmware perché più affidabile a livello temporale.

Durante la fase di setup della comunicazione USB, l'Host ottiene le informazioni necessarie ad instaurare la comunicazione dal *Device* attraverso l'*Endpoint0*, queste informazioni sono riportate nel *descrittore.* Il *descrittore* contiene diverse informazioni e normalmente è suddiviso in cinque diverse sezioni:

- *Device*: questa sezione è unica per ogni Device e contiene informazioni come il PID, VID e la lunghezza dei pacchetti;
- *Configuration*: ogni device può avere diverse sezioni *Configuration* che racchiudono la modalità di funzionamento del device e la potenza richiesta da ogni interfaccia;

[4] Tempo di volo: tempo di transito del pacchetto dal mittente al destinatario, più il tempo di ricezione ACK, se previsto.

- *Interface*: questa sezione specifica il numero di Endpoints per ogni interfaccia;
- *Endpoint*: stabilisce le caratteristiche di ciascun Endpoint di un interfaccia come la grandezza dei pacchetti, il tempo di polling di interrogazione e il tipo;
- *String*: sezione opzionale in cui vengono riportate le informazioni del dispositivi convertite in stringa;

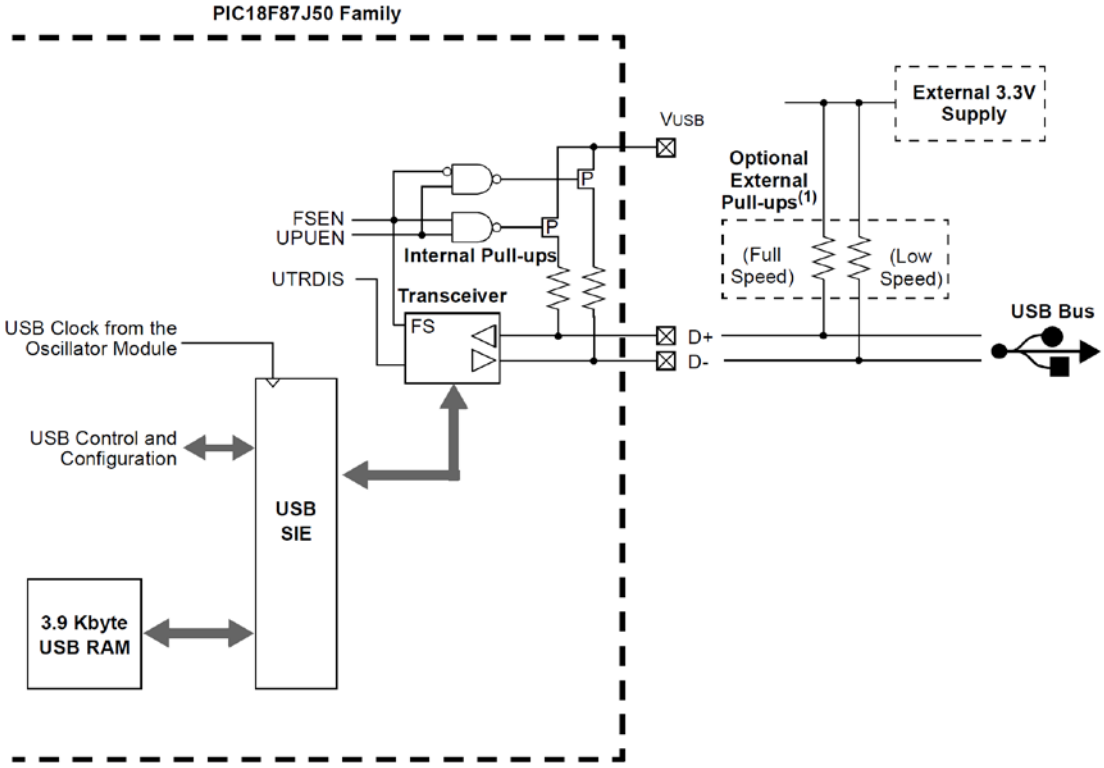


Figura 2.38: Schema a blocchi del modulo USB del microcontrollore (Le resistenze di pull-up esterne (1) devono essere previste se quelle interne sono disabilitate *UPUEN=0)*

Il microcontrollore utilizza il modulo *SIE* per la gestione della porta USB e il suo stato è determinato dal contenuto dei registri impostati secondo le direttive ricevute dal descrittore. I registri utilizzati per gestire il *SIE* sono:

- USB Control register (*UCON*) che controlla il comportamento del SIE durante il trasferimento dati. In particolare controlla il buffer Ping-Pong (*PPBRST*, che verrà spiegato successivamente), l'abilitazione delle periferiche (*USBEN*), il controllo della modalità sospensione

(*SUSPND*) e la disabilitazione del trasferimento dei pacchetti (*PKTDIS*);

- USB Configuration register (*UCFG*) che permette la configurazione del transceiver *UTRDIS*), di selezionare la modalità full speed o low speed (*FSEN*), l'abilitazione delle resistenza di pull-up (*UPUEN*)[5] e l'uso del buffer Ping-Pong (*PPB1:PPB0*);
- USB Transfer Status register (*USTAT)* che riporta lo stato delle transizioni all'interno del *SIE* e contiene il numero di *Endpoints* (*ENDP3:ENDP0*) e il puntatore del buffer Ping-Poing (*PPBI*);
- USB Device Address register (*UADDR*) che contiene l'indirizzo USB che la periferica decodifica quando è attiva. Questo indirizzo viene assegnato alla periferica durante il processo di enumerazione[6];
- Frame Number registers (*UFRMH:UFRML*), registro di sola lettura, utilizzato dal microcontrollori per alcune modalità dei trasferimento;
- Endpoint Enable registers 0 through 15 (*UEPn*) che permette il controllo dell'*Endpoint* ad esso associato; in particolare permette l'abilitazione dell'input dell'Endpoint (*EPINEN*) e l'abilitazione dell'output (*EPOUTEN*).

Gli *Endpoint* sono controllati dai registri organizzati in una struttura conosciuta come *Buffer Descriptor Table* (*BDT*) che permette di gestirli in modo flessibile. La *BDT* è composta dal *Buffer Descriptor* (*BD*) che controlla la memoria RAM riservata all'USB attraverso l'utilizzo di quattro registri (*BDnSTAT*: *BD Status*

[5] La resistenza di pull-up è abilitata su una delle due linee dati dell'USB in base alla modalità selezionata: Full-speed (*FSEN* = 1), low speed (*FSEN* = 0). Se la modalità è full speed, la resistenza è inserita sulla linea D+, se la modalità è low speed, la resistenza è inserita sulla linea D-.

[6] Il processo di enumerazione è il processo di gestione, configurazione e cambiamento di stato del Device da parte dell'Host. In questo processo l'Host abilita la porta USB dopo la connessione del Device, assegna l'indirizzo al Device e legge il descrittore del Device per ottenere le informazioni di configurazione.

register, *BDnCNT*: *BD Byte Count register*, *BDnADRL*: *BD Address Low register*, *BDnADRH*: *BD Address High register).*

Ad ogni *Endpoint* è associato un buffer ping-pong, meccanismo di trasmissione dati che consiste nell'usare due buffer, completamente indipendenti, che accedono allo stesso spazio di memoria e massimizzano il throughput: un buffer per la scrittura in memoria e l'altro per la lettura. Quando questo buffer ha due *BD* attivi, uno è utilizzato per il trasferimento Even e l'altro per il trasferimento Odd. In questo modo, il microcontrollore può elaborare un *BD* mentre il *SIE* elabora l'altro *BD*. Il modulo USB supporta quattro modalità di funzionamento del buffer:

- buffer ping-pong non abilitato;
- buffer ping-pong funzionante solo per l'*Endpoint0;*
- buffer ping-pong funzionante per tutti gli *Endpoint;*
- buffer ping-pong funzionante per tutti gli *Endpoint*, eccetto l'*Endpoint0;*

Il registro *UCFG* imposta la modalità di funzionamento del buffer mentre lo stato è salvato nel registro *USTAT* (bit *PPBI*).

Durante il normale funzionamento, il modulo USB può generare diversi interrupt e questi, se abilitati[7], possono essere utilizzati nel firmware per un maggior controllo del *SIE*. Tutti gli interrupt generati dal SIE vengono identificati dal microcontrollore attraverso un unico flag nel registro *PIR2* (bit *USBIF*). Il modulo *SIE* ha una struttura logica degli interrupt composta da due livelli: il primo, Top Level (Fig. 2.39), identifica gli interrupt di stato e il secondo (Fig. 8.39) gli interrupt causati da errori del modulo, quest'ultimo fa riferimento al flag *UEIR* nel Top Level.

Ogni livello ha due registri: Uno di abilitazione degli interrupt e l’altro di controllo dello stato dell'interrupt. Il top level utilizza il registro *UIE* per abilitare gli interrupt e il registro *UIR* per controllare lo stato, il secondo livello utilizza il registro *UEIE* per l'abilitazione e il registro di stato è *UEIR*.

[7] Per abilitare gli interrupt della porta USB occorre agire sul bit *UBIE* del registro *PIE2*

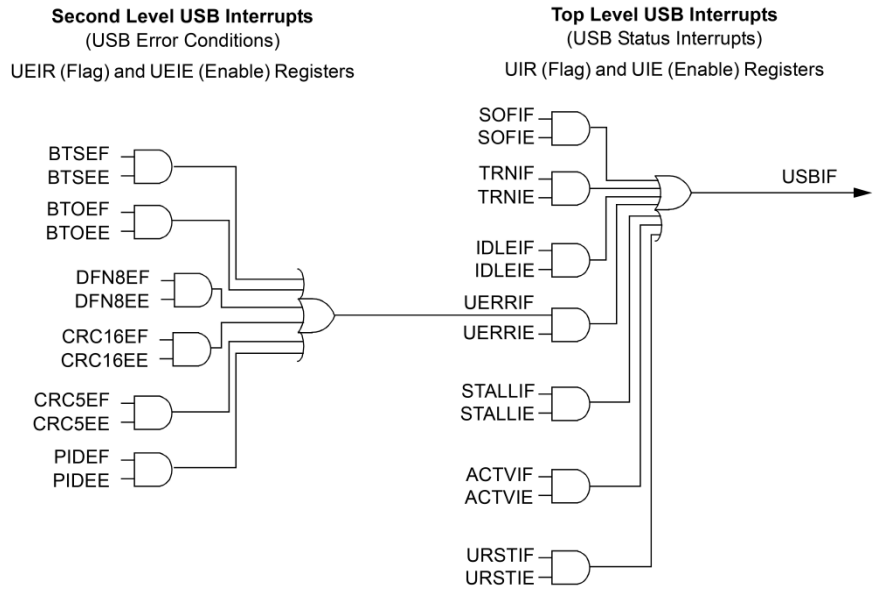


Figura 2.39:Livelli logici interrupt USB

Il top level permette di abilitare e controllare gli interrupt:

- di inizio del frame Token (*SOIF* e *SOFIE*, il primo bit indica il bit di stato del registro *UIR* e il secondo il bit di abilitazione del registro *UIE*);
- di stallo handshake (*STALLIF* e *STALLIE*);
- di rilevazione dello stato inattivo per più di 3 ms (*IDLEIF* e *IDLEIE*);
- di transizione completa (*TRNIF* e *TRNIE*);
- di rilevazione di attività del bus (*ACTVIF* e *ACTVIE*);
- di controllo di errore rilevato dal secondo livello (*UERRIF* e *UERRIE*);
- il reset degli interrupt (*URSTIF* e *URSTIE*).

Il secondo livello controlla gli errori del modulo USB, in particolare:

- gli errori di bit stuff (*BTSEF* e *BTSEE*);
- l'errore di Time-out del bus turnaround (*BTOEF* e *BTOEE*);
- l'errore di dimensione del campo dato (*DFN8EF* e *DNF8EE*);

- l'errore di CRC16 fallito (*CRC16EF* e *CRC16EE*);
- l'errore di CRC host fallito (*CRC5EF* e *CRC5EE*);
- l'errore di check fallito sul PID (*PIDEF* e *PIDEE*).

Tutti questi interrupt possono essere utilizzati all'interno del firmware, se abilitati, per un maggior controllo sul comportamento del modulo *SIE*.

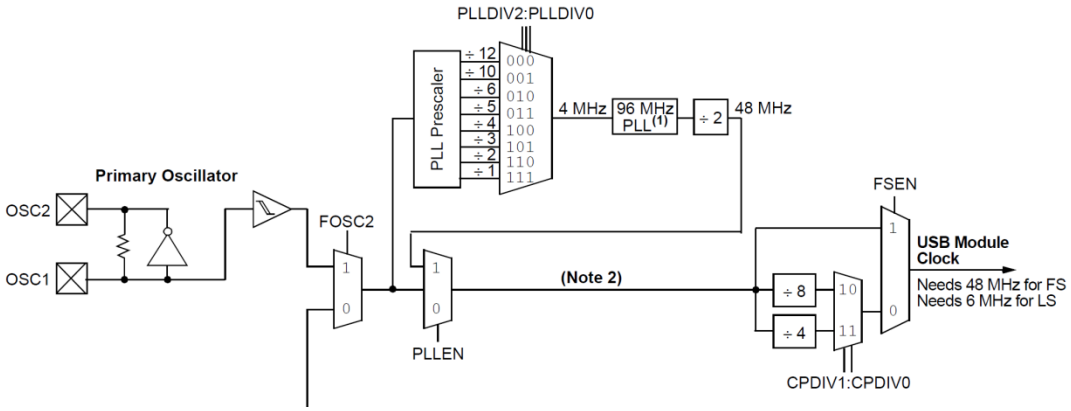


Figura 2.40:Gestione clock e PLL

Per un corretto sincronismo, il modulo *SIE* deve avere un clock stabile, con una frequenza pari a 48 MHz per la modalità full-speed e a 6 MHz per la modalità low-speed. Il microcontrollore ha un unico modulo *SIE* per cui la sorgente del clock deve essere unica e la sua frequenza di funzionamento deve variare in base alla modalità scelta. Per avere un clock stabile, questo deve essere generato internamente, in questo modo il core del microcontrollore e il modulo *SIE* possono lavorare a frequenze diverse. Per poter generare un clock indipendente e stabile rispetto al clock esterno si utilizza un PLL (Phase Locked Look, Fig. 2.40) presente nel microcontrollore PIC18F8750, che fornisce una frequenza di 96 MHz partendo da una frequenza di 4 MHz. La presenza del PLL permette di implementare la porta USB anche nei microcontrollori dotati di oscillatori a bassa frequenza. Per ottenere la frequenza di 4 MHz e quindi poter utilizzare oscillatori esterni con una frequenza maggiore, il microcontrollore adopera otto prescaling (divisori di frequenza, Fig. 2.41). Qualora per il core del microprocessore si voglia utilizzare una frequenza diversa rispetto a quella del modulo USB, si adopera un altro divisore in modo da ridurre la frequenza del PLL di un fattore 2, 3 o 6 (Fig. 2.41). Il PLL può essere solo

utilizzato con gli oscillatori *HSPLL*[8], *ECPLL*[9], *INTOSCPLL*[10] e per abilitarlo si pone a 1 il bit *PLLEN* del registro *OSCTUNE*. Una volta configurato il PLL non genera immediatamente la frequenza di 96 MHz, infatti inizialmente il microcontrollore opera alla frequenza dell'oscillatore esterno e, solo dopo circa 2 ms dalla sua abilitazione, il microcontrollore varia la frequenza di lavoro.

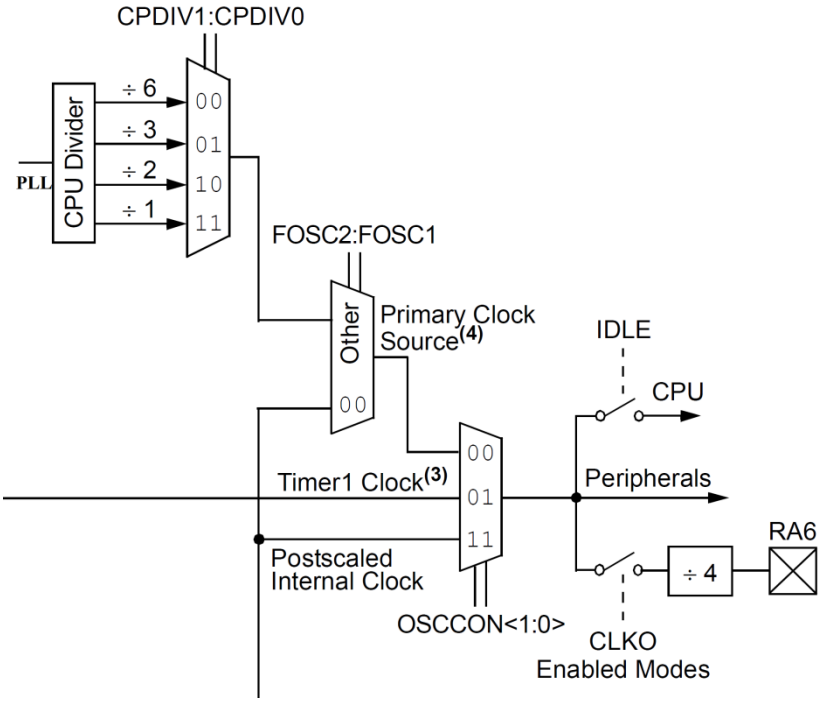


Figura 2.41: Post scaler e clock per il core del microcontrollore

Il collegamento tra il microcontrollore e il connettore USB avviene tramite una connessione differenziale (Fig. 2.42) indicata con i nome D+ e D-. La comunicazione avviene mediante pacchetti codificati con NRZI (Non Return to Zero Inverted, Fig. 2.43). Questa codifica è auto-sincronizzante: 0 rappresenta un cambiamento di livello; 1 indica l'assenza del cambiamento di livello. La codifica

[8] HSPLL (High Speed PLL): Oscillatore High Speed con PLL; l'oscillatore è connesso tra RA6 e RA7.

[9] ECPLL (External Clock PLL): Clock esterno con PLL; il segnale di clock è applicato sul pin RA7 invece sul pin RA6 è presente il clock esterno scalato di un fattore quattro.

[10] INTOSCPLL: Oscillatore interno con PLL; l'oscillatore interno provvede a generare il clock in ingresso al PLL e per i divisori di frequenza.

può perdere la sincronizzazione solo nel caso di una lunga sequenza di 1, quindi, per evitare che questo possa accadere, si inserisce uno 0 ogni 6 bit. Lo zero inserito è detto *stuffing bit.*

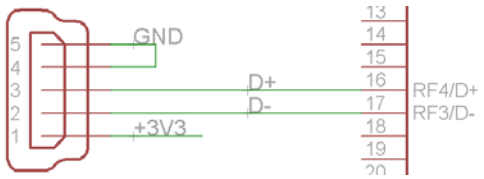


Figura 2.42:Collegamento tra microcontrollore e porta USB

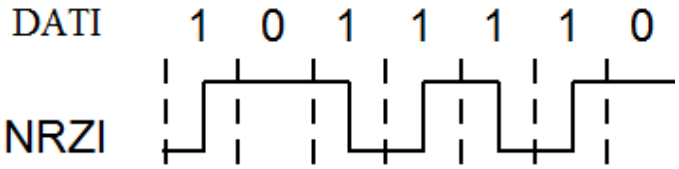


Figura 2.43:Codifica NRZI

Per utilizzare la porta USB e il modulo SIE si deve abilitare il transceiver e fornirne la potenza necessaria per il funzionamento. Per abilitarlo si deve agire sul registro *UCFG* (bit *UTRDIS*), la potenza necessaria è ottenuta dal pin V_{USB}, collegato a una tensione che può variare tra 3 V e 3.6 V, normalmente impostata a 3.3 V che corrisponde alla tensione di alimentazione.

Dopo aver brevemente descritto l'hardware e il software necessario al microcontrollore per la gestione della porta USB, si cercherà di spiegare il modo in cui questi sono stati adattati per il firmware del microcontrollore. In Figura 2.44 è riportata la parte del file descrittore contenente le impostazioni del *SIE*, del transceiver e degli Endpoint.

```
#define     USB_EP0_BUFF_SIZE       64    ; 8, 16, 32, or 64
#define     USB_MAX_NUM_INT         1     ; For tracking Alternate Setting
#define     USB_MAX_EP_NUMBER       1     ; UEP1
#define     NUM_CONFIGURATIONS      1
#define     NUM_INTERFACES          1

#define UCFG_VAL  USB_PULLUP_ENABLE|USB_INTERNAL_TRANSCEIVER|
                  USB_FULL_SPEED|USB_PING_PONG__NO_PING_PONG
#define USB_POLLING   ;#define USB_INTERRUPT

#define     HID_INTF_ID               0x00
#define     HID_EP                    1
#define     HID_INT_OUT_EP_SIZE       64
#define     HID_INT_IN_EP_SIZE        64
#define     HID_NUM_OF_DSC            1
```

Figura 2.44:Parte del descrittore

Il primo define, USB_*EP0_BUFF_SIZE*, stabilisce la grandezza in byte del buffer per l'*Endpoint0* e la lunghezza dei pacchetti che il microcontrollore può scambiare, il massimo valore è 64 byte. Il campo *USB_MAX_NUM_INT* determina la dimensione dell'array che tiene traccia delle impostazioni di altre interfacce. Il campo *MAX_EP_NUMBER* indica il numero di endpoint utilizzati nel firmware, escluso l'*Endpoint0* che viene inizializzato automaticamente dal gestore di memoria. Con l'impostazione utilizzata nel firmware, oltre all'*Endpoint0,* viene utilizzato anche l'*Endpoint1*. Perciò verranno inizializzate quattro *BD*, due per l'*Endpoint0* e due per l'*Endpoint1*. Gli ultimi due campi della prima parte determinano il numero di configurazioni (*NUM_CONFIGURATIONS*) e interfacce (*NUM_INTERFACES*) che l'host possiede. Il campo *UCFG_VAL* permette di definire i bit del registro UCFG, in particolare la configurazione utilizzata abilita le resistenze di pull-up su una delle linee dati (*USB_PULLUP_ENABLE*), in questo caso sulla linea D+ per la modalità full-speed; il transceiver (*USB_INTERNAL_TRANSCEIVER*) con la modalità full-speed (*USB_FULL_SPEED*). Per attivare la modalità di comunicazione low-speed si deve utilizzare l'impostazione *USB_LOW_SPEED*. L'ultimo parametro di configurazione disabilita il buffer ping-pong (*USB_PING_PONG__NO_PING_PONG*).

Il define *USB_POLLING* stabilisce la modalità di gestione del microcontrollore da parte dell'host, in questo caso l'host interroga i registri e il buffer dell'interfaccia USB a intervalli di tempo costanti. Nel tipo interrupt (*USB_INTERRUPT*), l'Host interroga l'interfaccia USB del microcontrollore solo

quando questa abilita uno degli interrupt visti in precedenza. Il microcontrollore non controlla tutti gli interrupt, ma solo il flag *USBIF*. La prima modalità di gestione risulta essere la peggiore a livello di impiego del microcontrollore in quanto lo occupa ciclicamente, ma è sicuramente la più efficiente per la semplicità di gestione e, dalle misure effettuate, per il minor tempo di scrittura sul buffer USB. Infatti in modalità polling, il tempo di scrittura sul buffer è 42 µs (Fig. 2.45), invece in modalità interrupt è circa 220 µs (Fig. 2.46); per questo motivo nel firmware del DAQ si utilizza la modalità polling.

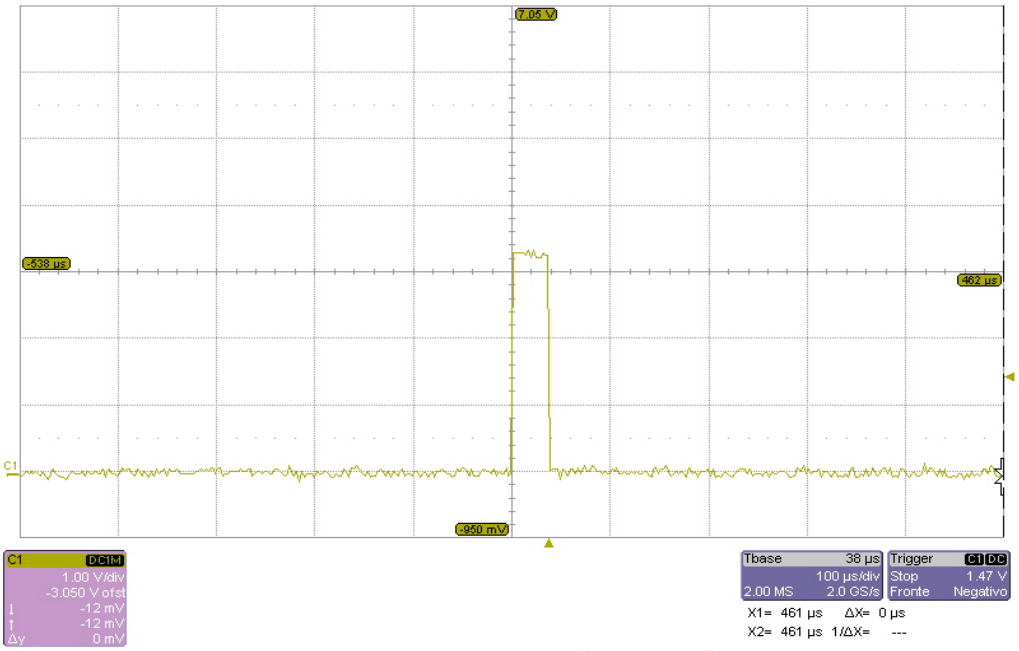


Figura 2.45: Tempo di scrittura sul buffer in modalità polling

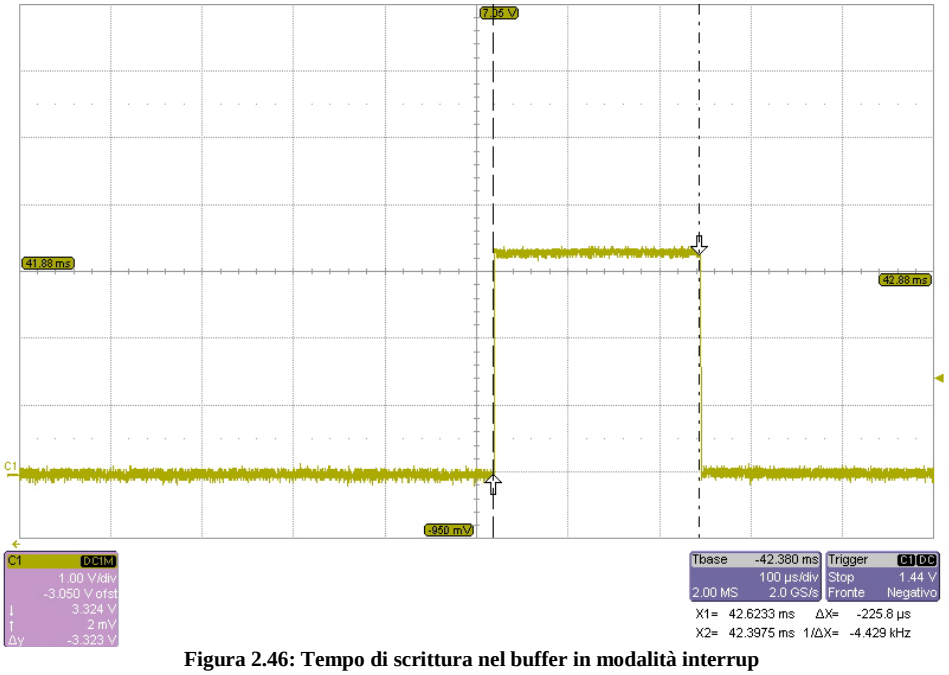


Figura 2.46: Tempo di scrittura nel buffer in modalità interrup

L'ultimo blocco di define riportato in Figura 2.44, identifica la classe di appartenenza del device, in questo caso la classe HID. La prima define, *HID_INTF_ID*, rappresenta l'identificativo dell'interfaccia HID e determina se il comando ricevuto dal device appartiene ai comandi della classe HID. La successiva define, *HID_EP*, individua quale dei registri di controllo degli *Endpoints* (*UEPn*) è utilizzato, nel nostro caso è il registro *UEP1*. Le ultime tre define identificano le dimensioni dei buffer di input ed output associati all'*Enpoint* (*HID_INT_OUT_EP_SIZE* e *HID_INT_IN_EP_SIZE*) e il numero di descrittori associati al dispositivo HID. Il file descrittore è stato completato in tutte le sue parti, utilizzando come modello quello creato dal programma EasyHID USB Wizard della Mecanique per il microcontrollore PIC18F4550.

Nel firmware del microcontrollore deve essere incluso il file descrittore e abilitato il PLL ponendo a 1 il bit *PLLEN* del registro *OSCTUNE* (*OSCTUNE.6 = 1*). Per poter utilizzare il modulo USB, esso deve essere prima inizializzato tramite i comandi USBINIT e USBSERVICE, e successivamente tramite la routine di scritttura e lettura (Fig. 2.47) del buffer USB. In questa routine, i valori letti e da scrivere provengono dall'array *USBBuffer* e il numero di caratteri da scrivere o da leggere sono indicati nella variabile *USBBufferCount*.

```
' ************************************************************
' * 'Lettura                                                 *
' ************************************************************
DoUSBIn:
   USBBufferCount = USBBufferSizeRX
   USBSERVICE
   USBIN 1, USBBuffer, USBBufferCount, DoUSBIn
   RETURN

' ************************************************************
' * Scrittura                                                *
' ************************************************************
DoUSBOut:
   USBBufferCount = USBBufferSizeTX
   USBSERVICE
   USBOUT 1, USBBuffer, USBBufferCount, DoUSBOut
   RETURN
```

Figura 2.47: Routine di lettura e scrittura del buffer

Per risolvere il problema relativo alle trasmissione delle informazioni tramite porta seriale (il DAQ è alloggiato sul pallone) o porta USB (il DAQ è usato per effettuare misure a terra), il firmware del microcontrollore interroga il registro *UCON* e in particolare il bit *USBEN* (Fig. 2.48), che indica se il Device è connesso. Se il Device è connesso, i dati sono inviati tramite USB altrimenti sono trasmessi attraverso la porta seriale.

```
IF (UCON.3 = 1) THEN
    .....
    GOSUB DoUSBOut          'USB
    .....
ELSE
    .....
    HSEROUT [STR fat_src\167,10]      'Seriale
    .....
END IF
```

Figura 2.48:Controllo del registro per identificare la periferica con la quale comunicare

Premettendo che la dimensione massima di un pacchetto trasferito tramite USB è di 64 byte e che la stringa da trasmettere, contenente tutte le informazioni necessarie per l'analisi dei dati, è di 168 byte, è stata effettuata un' attenta analisi temporale per scegliere la soluzione migliore. Nella prima misura effettuata, riguardante il trasferimento di 64 byte, il tempo impiegato risulta essere circa di 316 µs (Fig. 2.49).

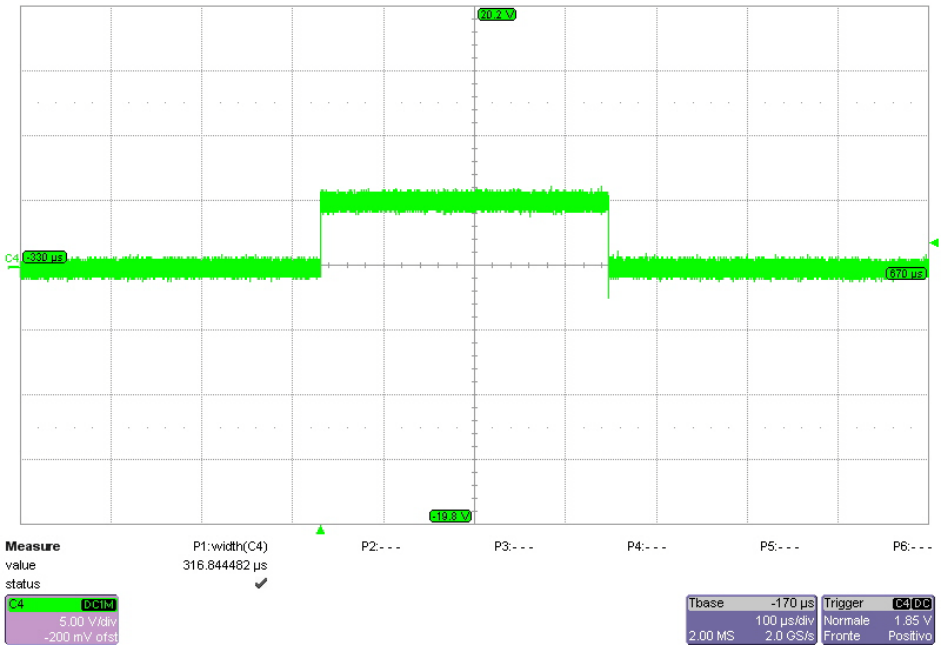


Figura 2.49: Trasmissione di 64 byte

Nella misura successiva, il tempo impiegato per trasmettere 168 caratteri è di circa 12 ms (Fig. 2.50).

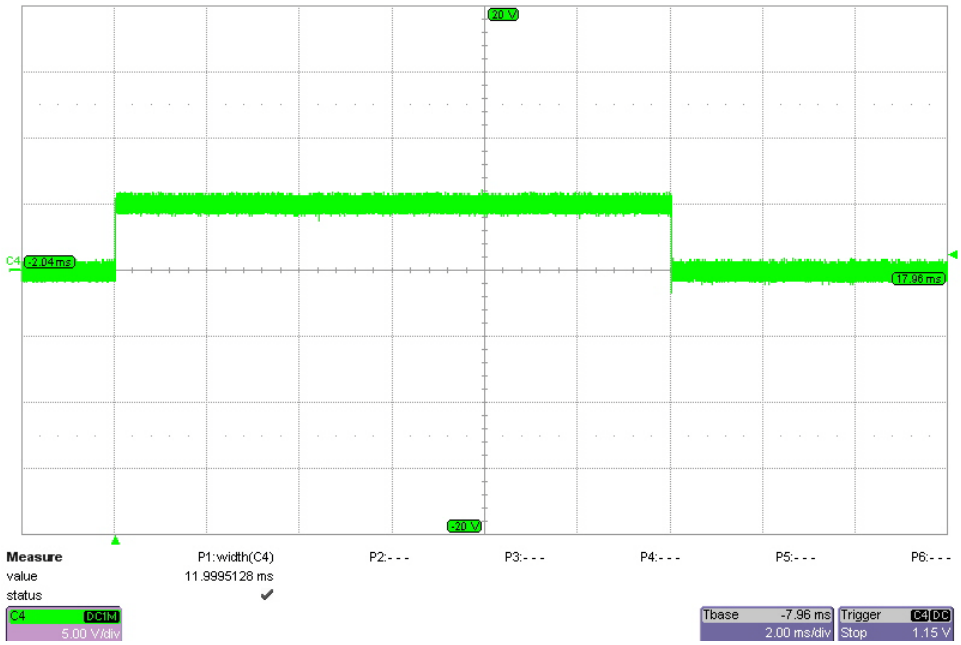


Figura 2.50: Trasmissione di 168 byte

Questo tempo deriva dalla scrittura consecutiva per tre volte del buffer USB e, risultando troppo alto, si è valutato il tempo di scrittura dei singoli blocchi . Per scrivere sul buffer USB il primo blocco, occorrono 35.7 µs (Fig. 2.51); per il secondo

blocco circa 4 ms (Fig. 2.51); e per il terzo blocco circa 8 ms (Fig. 2.51). Per tanto l'aumento significativo del tempo di trasmissione dei diversi blocchi è dovuto al tempo di gestione del buffer (invio dati e svuotamento) durante la comunicazione con l'Host.

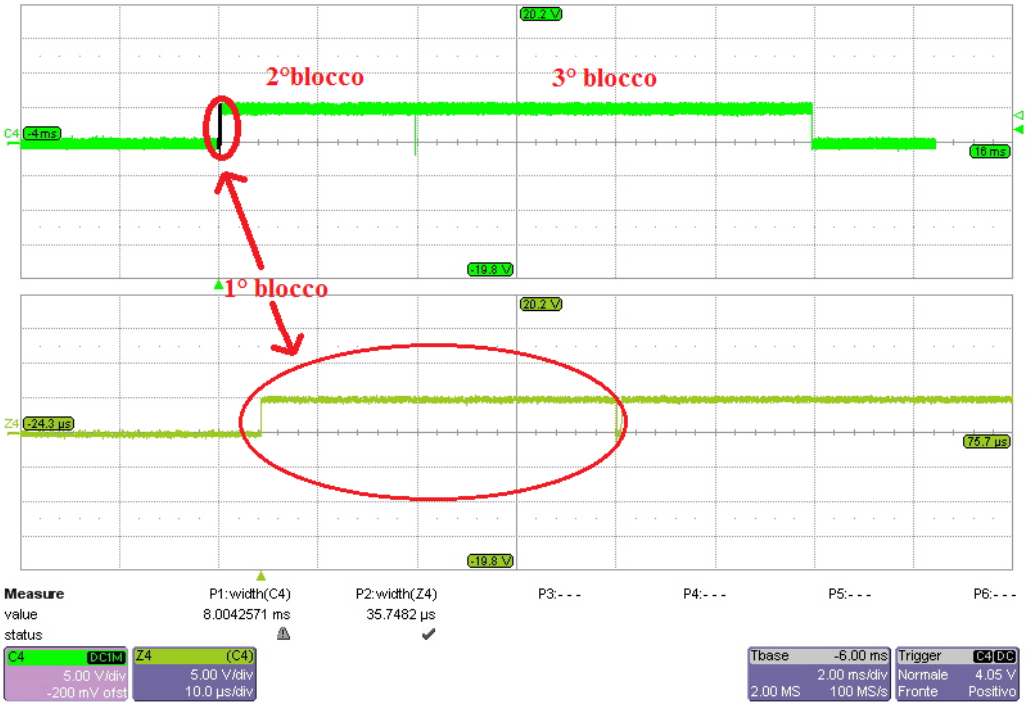


Figura 2.51: Tempo di scrittura dei diversi blocchi

2.4.5 Analisi temporale dell'invio delle informazioni

Il tempo di invio delle informazioni dal microcontrollore al sistema di telemetria deve essere inferiore a 1 s che rappresenta la minima finestra di conteggio, ciò è necessario per non perdere informazioni sul flusso dei raggi cosmici.

Il tempo d'invio dei dati dall'FPGA al microcontrollore è di 3.2 ms (Fig. 2.27), mentre per ottenere le informazioni dal GPS, il tempo impiegato è di circa 1 ms. Il tempo di gestione del SD CARD è suddiviso in tre parti:

- il tempo di apertura del file, 37.5 ms (Fig. 2.34);
- il tempo di scrittura dei 168 caratteri nel file, 13 ms (Fig. 2.36);
- il tempo di chiusura del file, 35 ms (Fig. 2.37).

Quindi per la gestione dell' SD CARD occorrono circa 85.5 ms, per cui, il tempo totale di elaborazione dei dati è di circa 89.7 ms, escludendo dal conteggio il tempo di trasmissione delle informazioni dal microcontrollore al sottosistema di telemetria attraverso la porta seriale o l'invio ad un computer tramite la porta USB. L'invio delle informazioni attraverso la porta USB richiede circa 12 ms (Fig. 2.50), perciò il tempo totale impiegato nella trasmissione ed elaborazione delle informazioni è di circa 101.7 ms, molto inferiore a 1 s della finestra di conteggio. Usando la porta seriale il tempo di invio dei dati è di 172.6 ms (Fig. 2.52) ed il totale sale a 262.3 ms.

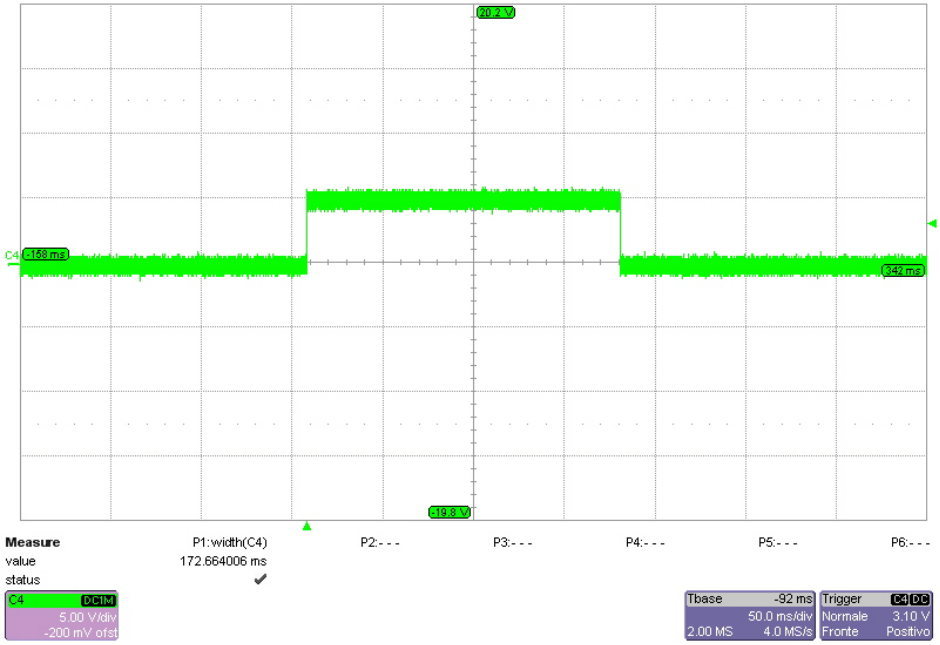


Figura 2.52: Tempo di trasmissione di 168 caratteri attraverso porta seriale con un rate di 9600 baud

2.5 Programma di analisi dati

Per effettuare l'analisi dati in real-time si utilizza un programma realizzato con LabView, che, interfacciandosi con il sistema di telemetria del pallone mediante porta seriale, ottiene le informazioni sul flusso dei raggi cosmici e invia le informazioni sulla gestione da remoto della finestra di conteggio. Una volta ottenute le informazioni dal DAQ, il programma estrae le informazioni sul flusso dei raggi cosmici, aggiorna il calcolo della media e visualizza gli istogrammi del flusso dei

raggi cosmici. Inoltre il programma permette di creare il file di log delle informazioni acquisite.

La Figura 2.53 riporta l'interfaccia grafica del programma. Nella prima sezione si impostano i parametri della porta seriale (Fig 2.53: Impostazione porta seriale), che devono essere definiti precedentemente all'inizio dell'acquisizione dati. Accanto a questa sezione c'è quella di invio dati al DAQ (Fig. 2.53: Invio dati porta seriale), cioè dei comandi per la gestione della finestra di conteggio e per il controllo del suo stato (per sapere l'associazione tra comando e tempo di conteggio si faccia riferimento ai comandi spiegati illustrati del precedente paragrafo). Nella sezione dati acquisiti (Fig. 2.53: Dati acquisiti dalla porta seriale) è visualizzata la stringa con tutte le informazioni (flusso dei raggi cosmici e informazioni del GPS) ricevute, mentre al di sotto di questa sezione, ossia in quella dei Valori contatori (Fig. 2.53), sono visualizzati i valori dei contatori, in esadecimale, estratti dalla stringa acquisita. Questi valori vengono convertiti in decimale e inseriti nella matrice (Fig 2.53: Matrice dei valori dei contatori) in cui sono anche presenti tutti quelli precedentemente acquisiti. Di tutti i valori presenti nella matrice si calcola la media (Fig 2.53: Media dei contatori) e i diversi valori ottenuti vengono raffigurati negli istogrammi (Fig 2.53: Istogrammi dei valori dei contatori). In alto a destra la Figura 2.53 riporta la sezione per la creazione del file di Log degli eventi acquisiti. Il nome del file sarà composto dalla data ricevuta dal DAQ e da una radice inserita dall'utente. Il programma inizierà a salvare una volta spuntata la casella LOG (Fig. 2.53: File di Log).

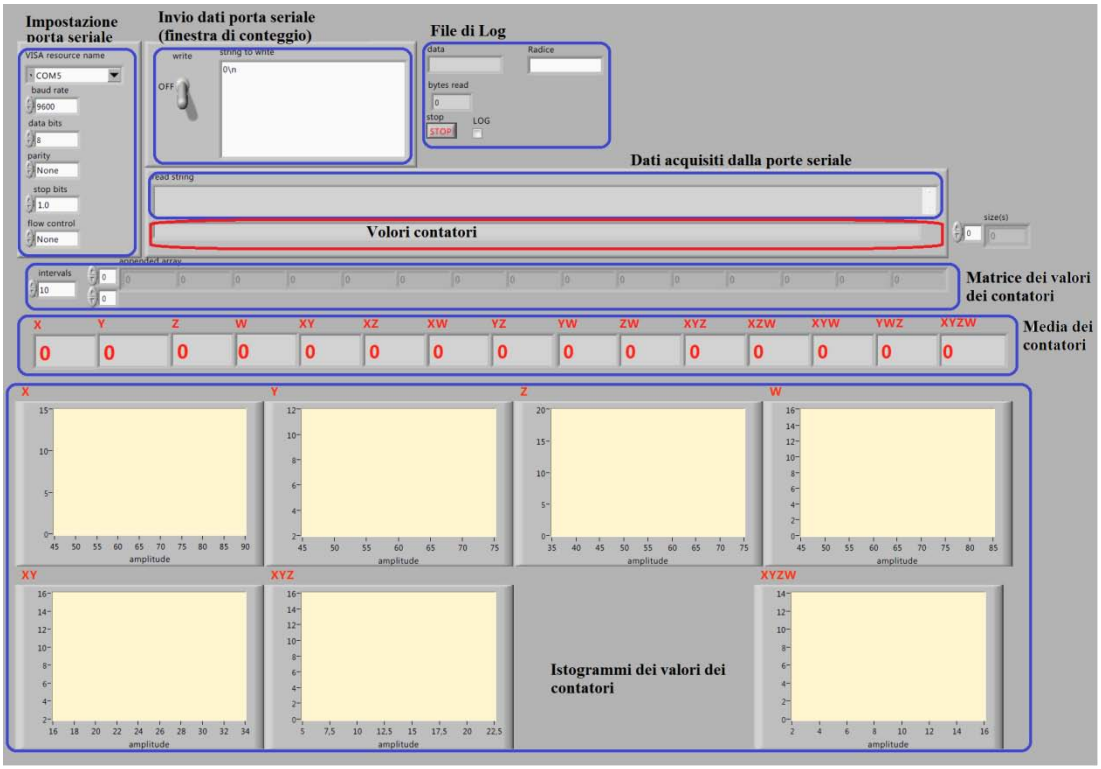


Figura 2.53: Interfaccia grafica programma LabView

Come tutti i programmi realizzati in LabView, all'interfaccia grafica è associato uno schema a blocchi in cui vengono controllati gli elementi dell'interfaccia grafica e in cui se ne assegna il compito.

Il primo blocco riguarda la porta seriale, le informazioni inserite nelle caselle di testo dell'interfaccia grafica (Fig. 2.53: Impostazioni porta seriale) sono utilizzate dal blocco *VISA serial* (Fig. 2.54) di LabView per inizializzare la porta seriale. Questo blocco genera alcune informazioni necessarie ai blocchi di scrittura e lettura del buffer della porta seriale.

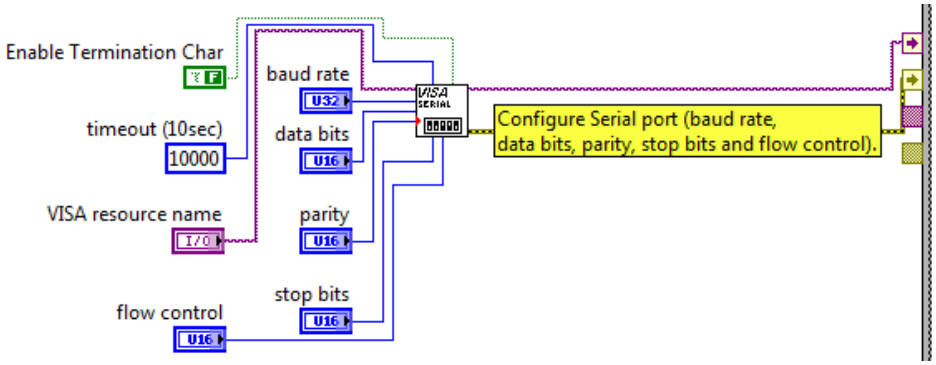


Figura 2.54: Impostazioni porta seriale

Il blocco di lettura della porta seriale (Fig. 2.55) è costituto da un ciclo while (Fig. 2.55: Ciclo While) che termina solo quando non riceve i byte della porta (Fig. 2.55: Fine ciclo). Prima di effettuare il controllo di fine ciclo e l'aggiornamento del numero di byte ricevuti, si aspettano 100 ms (Fig 2.55: Delay). La lettura del buffer della porta seriale avviene tramite il blocco VISA Read (Fig 2.55: Read) e i dati letti (Fig 2.55: Linea Viola) sono inviati al blocco per l'elaborazione.

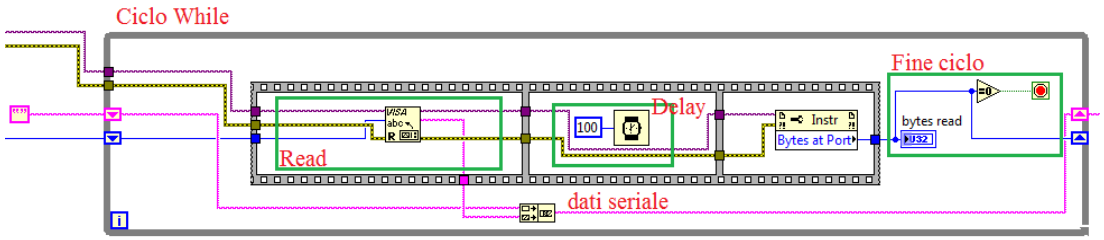


Figura 2.55: Lettura informazioni da porta seriale

La stringa dei dati acquisiti dalla porta seriale viene visualizzata nella casella di testo (Fig. 2.53: Dati acquisiti dalla porta seriale) e successivamente elaborata in un ciclo For (Fig. 2.56: ciclo For) per estrarre il valore esadecimale dei 15 contatori che, una volta convertiti in decimale, sono accodati ai precedenti valori nella matrice dei valori dei contatori (Fig. 2.53).

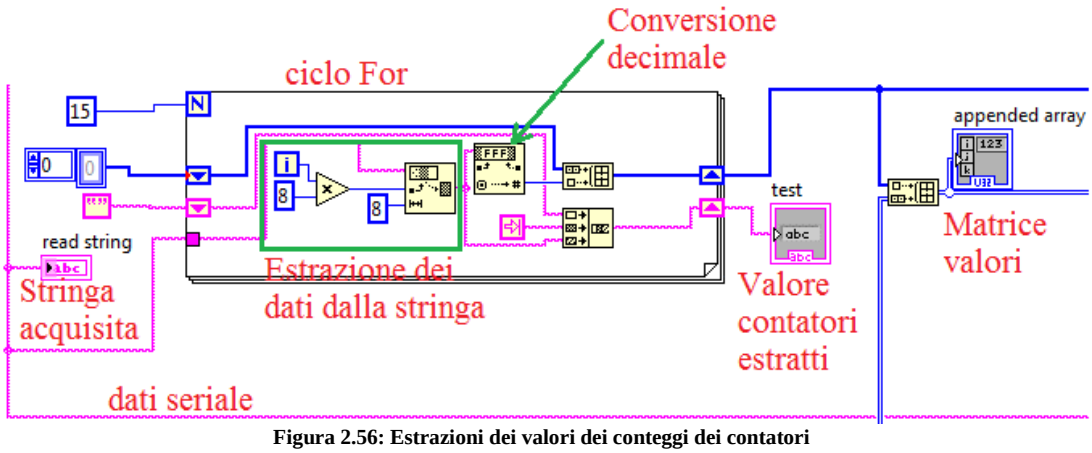


Figura 2.56: Estrazioni dei valori dei conteggi dei contatori

I dati inseriti nella matrice dei valori (Fig 2.56:appended array) sono utilizzati per calcolare la media dei contatori e visualizzarne gli istogrammi di alcuni. Per far ciò, dai dati inseriti nella matrice dei valori si estrae il valore del contatore (Fig. 2.57:

Estrae il valore), si calcola la media utilizzando il blocco Mean e si visualizza l'istogramma utilizzando il subVi[11] *Histogram.*

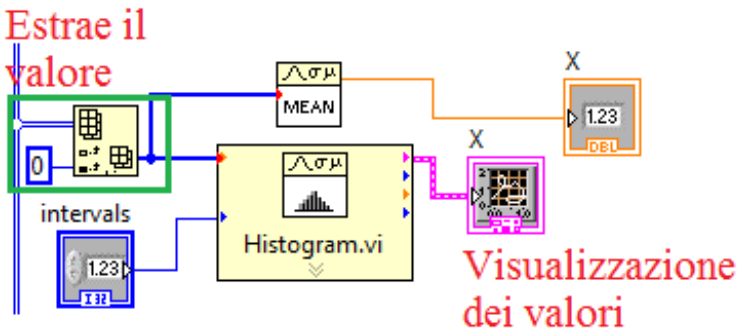


Figura 2.57: Analisi dati

Il programma realizzato in LabView comunica tramite la porta seriale con il DAQ inviando le informazioni sulla definizione della finestra di conteggio (Fig. 2.53: Invio dati porta seriale). Per far ciò si utilizza il blocco *VISA Write* (Fig. 2.58: Visa Write) che invia le informazioni solo se si pone su ON lo switch Write (Fig. 2.58: Switch).

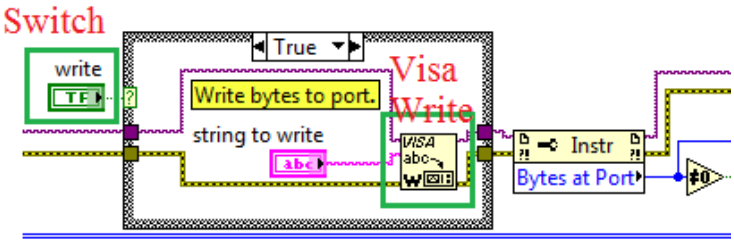


Figura 2.58: Scrittura porta seriale

I dati acquisiti dalla porta seriale vengono anche scritti in un file di Log, il cui nome (Fig. 2.59: Nome file) è composto dalla data di inizio dell'acquisizione e da una radice inserita dall'utente (Fig 2.53: File di Log).

[11] Programma di LabView utilizzato da un altro programma.

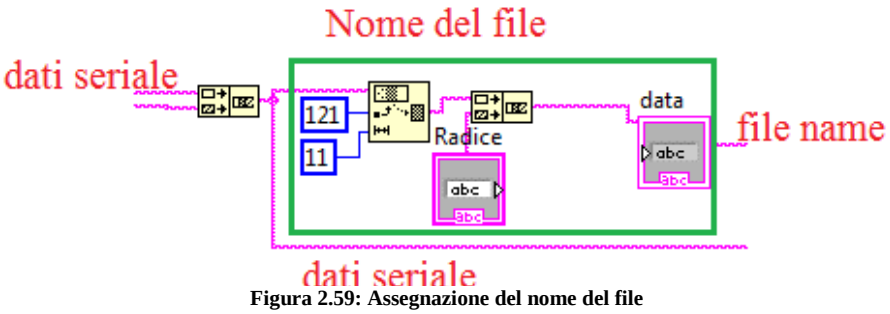


Figura 2.59: Assegnazione del nome del file

Il nome del file (Fig. 2.59: file name) e i dati provenienti dalla porta seriale (Fig. 2.59: dati seriale) sono utilizzati da due subVi adoparati per la creazione del file e per la scrittura sul file (Fig. 2.60: Log Save e Data Log).

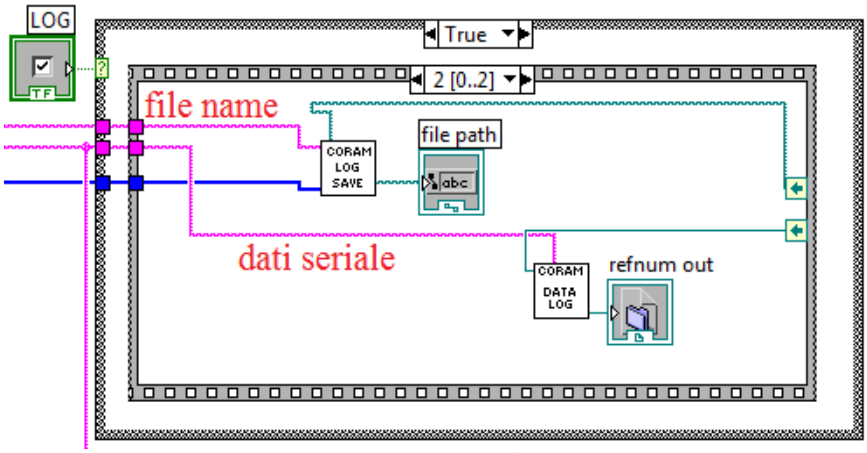


Figura 2.60: Creazione del file di Log

Per la creazione del file si deve spuntare la check box LOG (Fig. 2.60: Log).

Il programma LabView, descritto in questo paragrafo, è stato utilizzato per eseguire una prima analisi dati durante il test di misura effettuato presso il Gran Sasso e i risultati di queste misure saranno presentate brevemente nel prossimo paragrafo.

2.6 Analisi dati del test di misura sul Gran Sasso

Il test di misura [15-17] è effettuato per verificare la dipendenza del flusso dei raggi cosmici con l'altitudine, cioè ripetere una parte delle misure effettuate nel 1939 da Bruno Rossi e J. Barton Hoag da Chicago a Mount Evans. Per ripetere questa

esperienza sono state eseguite delle misure a Lecce e in luoghi intorno al Gran Sasso ad altitudini diverse come riportato in Tabella 5.

Tabella 5: Luoghi e altitudini delle misure

Luogo	**Altitudine (m s.l.m.)**	**Prof. Atm. (g/cm^2)**
Lecce	35	1032
Brecciarola	65	1028
Navelli	684	955
LNGS[12] esterni	990	920
Rocca di Cambio	1270	906
Rocca di Mezzo	1366	889
Fonte Cerreto	1120	879
Monte Cristo	1453	870
Campo Imperatore	2140	799

Una prima analisi consiste nel valutare il rate di coincidenza doppia, tripla e quadrupla al variare dell'altitudine. Come si può notare dalla Figura 2.61, il conteggio della coincidenza diminuisce all'aumentare della profondità atmosferica, quindi al diminuire dell'altitudine, in accordo con il Pfotzer plot (Fig. 1.2). In questo tipo di analisi non si è tenuto conto dei conteggi di singola perché questo dato è influenzato dal rumore di ogni dispositivo e dalle differenti sorgenti di radiazioni ambientali che dipendono dal luogo in cui si effettua la misura.

[12] LNGS: Laboratori Nazionali del Gran Sasso dell'INFN.

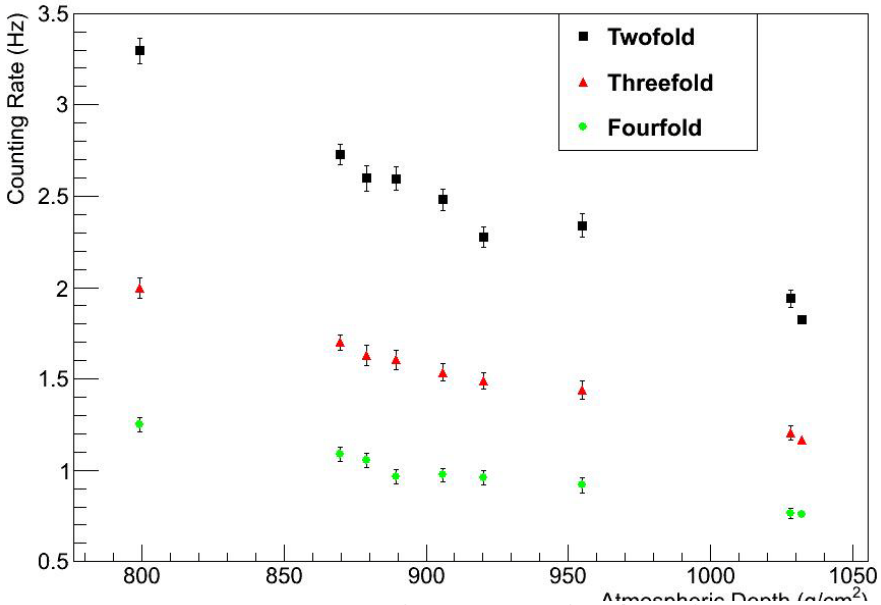


Figura 2.61: Coincidenze in funzione della profondità atmosferica

Un'ulteriore misura per verificare la dipendenza del flusso dei raggi cosmici con l'altitudine è stata effettuata durante l'ascesa Fonte Cerreto (1120 m) a Campo Imperatore (2140 m) in funivia. In Figura 2.62, si nota la crescita del conteggio di singola con il tempo, cioè ad altitudini sempre maggiori. Tale andamento non è molto chiaro perché si deve considerare che la salita dura circa dieci muniti e le fluttuazioni statistiche dovute alla bassa accettanza del rilevatore sono ampie.

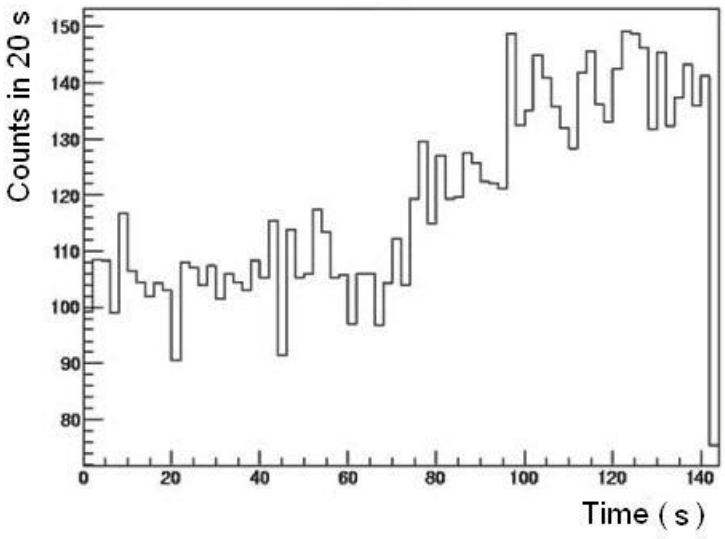


Figura 2.62: Coincidenza di singola durante la salita in funivia da Fonte Cerreto (1120 m) a Campo Imperatore (2140 m)

Per valutare il rumore elettronico del rilevatore e la radioattività ambientale sono state effettuate alcune misure nei LNGS e durante l'attraversamento della galleria necessaria a raggiungerli. Si può valutare il rumore proprio del rilevatore perché la roccia soprastante ai laboratori e alla galleria riduce di un fattore paria a circa 10^6 il flusso dei raggi cosmici. In teoria, in queste condizioni il DAQ non dovrebbe misurare alcun flusso. Come si può notare in Figura 2.63, all'interno della galleria del Gran Sasso (Fig 2.63: T2, T4 e T6) i conteggi di singola sono fortemente diminuiti rispetto a quelli misurati al di fuori (Fig 2.63: T1, T3 e T7) perché i raggi cosmici sono assorbiti dalla roccia soprastante e i pochi conteggi riscontrabili sono dovuti al rumore elettrico del rilevatore e alla radioattività ambientale. Inoltre si nota che all'interno del laboratorio (Fig. 2.63: T5) il conteggio di singolo diminuisce ulteriormente, probabilmente per una migliore pulizia ambientale. Si è riscontrato lo stesso trend per le coincidenze doppie, triple e quadruple.

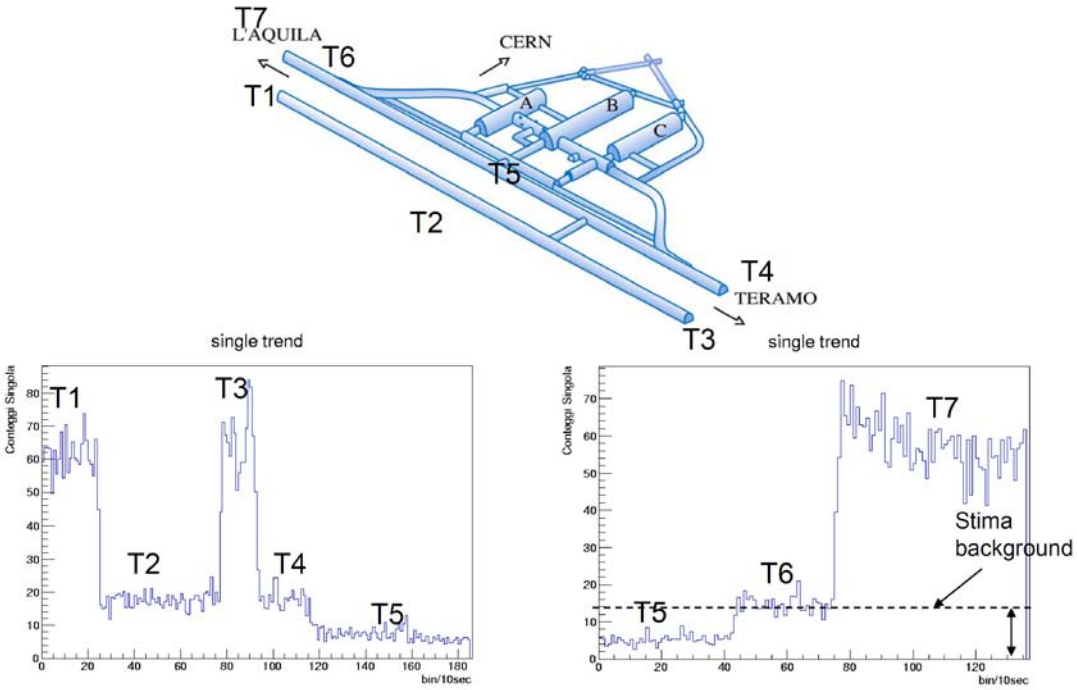


Figura 2.63: Conteggio di singola in underground

CAPITOLO 3

IL DAQ

3.1 Introduzione

In questo capitolo analizziamo il firmware di gestione del DAQ (FPGA e microcontrollore secondario, non previsto nel prototipo) esaminato in dettaglio nel Capitolo 1, evidenziando le modifiche effettuate rispetto a quello utilizzato nel prototipo, descritto nel Capitolo 2, infine sarà illustrata l'implementazione dell'intero DAQ sul circuito stampato.

La versione finale del firmware del DAQ è stata modificata rispetto a quella del prototipo in quanto il numero degli scintillatori che formano il rilevatore nel caso finale è raddoppiato, da 4 a 8 e, di conseguenza, per poter tenere traccia di tutte le loro possibili combinazioni, si è aumentato corrispondentemente il numero di contatori a 32 bit, portando da 15 a 255. Con questi numeri, l'algoritmo utilizzato per il firmware dell'FPGA del prototipo diventa inadeguato in quanto non mappabile sulla FPGA prevista perché, adattando il firmware del prototipo al rilevatore con otto scintillatori si ha un utilizzo di area del 245% (Fig. 3.1), incremento dovuto all'aumento del numero dei contatori, della matrice delle coincidenze e della coda di trasferimento. Per risolvere questo problema si è ideato un differente algoritmo per le coincidenze che tratteremo.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	11452	4656	245%
Number of Slice Flip Flops	16551	9312	177%
Number of 4 input LUTs	13427	9312	144%
Number of bonded IOBs	29	158	18%
Number of GCLKs	4	24	16%

Figura 3.1:Utilizzo del FPGA con l'algoritmo del prototipo adattato

Sul firmware del microcontrollore principale, rispetto a quello del prototipo, non sono state eseguite modifiche significative all'algoritmo di trasferimento delle informazioni sul flusso dei raggi cosmici; è stata solo integrata la comunicazione seriale con il microcontrollore secondario. Di quest'ultimo, si analizzerà il firmware con particolare attenzione alla comunicazione con il microcontrollore principale e i sensori (termometri e accelerometro-magnetometro) e infine si descriverà come calcolare le informazioni sul rollio e sull'orientazione del dispositivo secondo i valori ottenuti dall'accelerometro-magnetometro.

L'ultimo argomento trattato nel seguente capitolo, descriverà il layout del PCB e il suo popolamento oltre ad una attenta analisi sulla dispersione del calore del DAQ.

3.2 Algoritmo di coincidenza

L'algoritmo di coincidenza è stato modificato rispetto a quello utilizzato nel prototipo in quanto l'FPGA utilizzata, non permette di mappare i 255 contatori a 32 bit con la coda utilizzata per il trasferimento, infatti, per implementarli tutti occorrono circa 16000 bit mentre l'FPGA permette di salvarne "solo" 9312 (per salvare un bit occorre un elemento di memoria denominato Flip-Flop, l'FPGA utilizzata ne implementa 9312, Fig. 3.1). Per poter effettuare la statistica sul flusso delle particelle ionizzanti, senza escludere a priori alcune combinazioni (una delle prime ipotesi per risolvere il problema della non mappabilità era quella di eliminare alcune delle 255 combinazioni), si è deciso di salvare il valore dei contatori in una particolare memoria presente sulle FPGA della Xilinx. Ogni cella di memoria conterrà il valore di un contatore, per ogni operazione su questa memoria occorre un ciclo di clock, materialmente, non c'è il tempo utile per considerare i sottoeventi

dell'evento verificato nella finestra d'ascolto (Paragrafo 2.2 Algoritmo di coincidenza), rendendo così inutile l'utilizzo della matrice che determina l'incremento dei contatori. Nel prototipo del DAQ l'ordine dei contatori non influiva sulla statistica del flusso essendo sufficiente cambiare di posizione le colonne della matrice; invece nel DAQ finale per avere la statistica completa del flusso dei raggi cosmici si deve utilizzare una particolare memoria per il suo salvataggio e potendo accedervi una sola volta per ogni finestra d'ascolto occorre trovare una relazione diretta che identifica il contatore (cella di memoria) associato all'evento. Partendo dal presupposto che ad ogni scintillatore si è assegnato un peso, multiplo di due, sviluppando le diverse combinazioni in binario, si ottiene un valore corrispondente ad ogni singolo evento che sarà utilizzato come indirizzo di memoria.

Considerando, per esempio, il rilevatore composto da quattro scintillatori (Fig.1.32) e assegnando gli stessi pesi ad ogni scintillatore del Paragrafo 2.2, si ricava la posizione dei contatori in memoria (Tabella 6). Questa tecnica è stata utilizzata nella versione finale del DAQ con un rilevatore costituito da otto scintillatori.

Tabella 6: Posizione contatore

w	z	y	x	P	CONTATORI
8	4	2	1		
0	0	0	1	1	Cx
0	0	1	0	2	Cy
0	0	1	1	3	Cxy
0	1	0	0	4	Cz
0	1	0	1	5	Cxz
0	1	1	0	6	Cyz
0	1	1	1	7	Cxyz
1	0	0	0	8	Cw
1	0	0	1	9	Cxw
1	0	1	0	10	Cyz
1	0	1	1	11	Cxyw
1	1	0	0	12	Czw
1	1	0	1	13	Cxzw
1	1	1	0	14	Cyzw
1	1	1	1	15	Cxyzw

Una volta compreso come i contatori sono posizionati in memoria si può descrivere come avviene l'incremento della statistica sul flusso dei raggi cosmici, si faccia riferimento ad un rilevatore composto da quattro scintillatori e supponiamo che in una finestra d'ascolto la particella ionizzante sia rilevata da tre dei quattro scintillatori; l'algoritmo implementato sull'FPGA, terminata la finestra d'ascolto, calcola il peso dell'evento (in questo esempio 7) e accede alla memoria per incrementare il contatore relativo (Fig. 3.2).

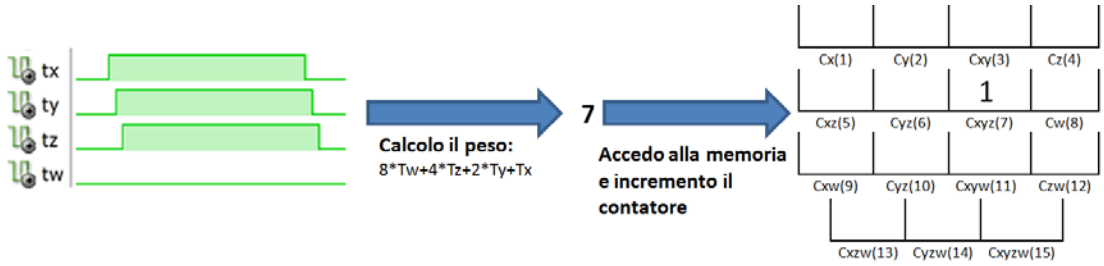


Figura 3.2:Comportamento dell'algoritmo

Nell'algoritmo del prototipo del DAQ, che risultava completamente ridondante, oltre alla cella 7 si sarebbe incrementato anche il valore delle celle 1, 2, 3, 4, 5, 6 per cui, l'algoritmo della versione finale, implementato sul DAQ è lo stesso di quello del prototipo a meno della ridondanza.

I risultati dell'algoritmo senza ridondanza (DAQ finale), in fase di analisi dati, sono elaborati attraverso semplici relazioni per fornire la statistica del flusso dei raggi cosmici completa di ridondanza. In Figura 3.3 è raffigurato il risultato dei due algoritmi e il metodo di passaggio da uno all'altro solo per alcuni contatori.

La ridondanza, presente nel prototipo del DAQ, permetteva di effettuare un controllo d'errore sui dati trasferiti; in quest'algoritmo invece, non essendoci un controllo intrinseco, di errore si è deciso di memorizzare anche i conteggi di singola degli otto scintillatori, cioè il numero di particelle rilevate dagli scintillatori in una finestra di conteggio. Tali dati sono successivamente trasmessi al microcontrollore, in coda ai valori salvati nella memoria.

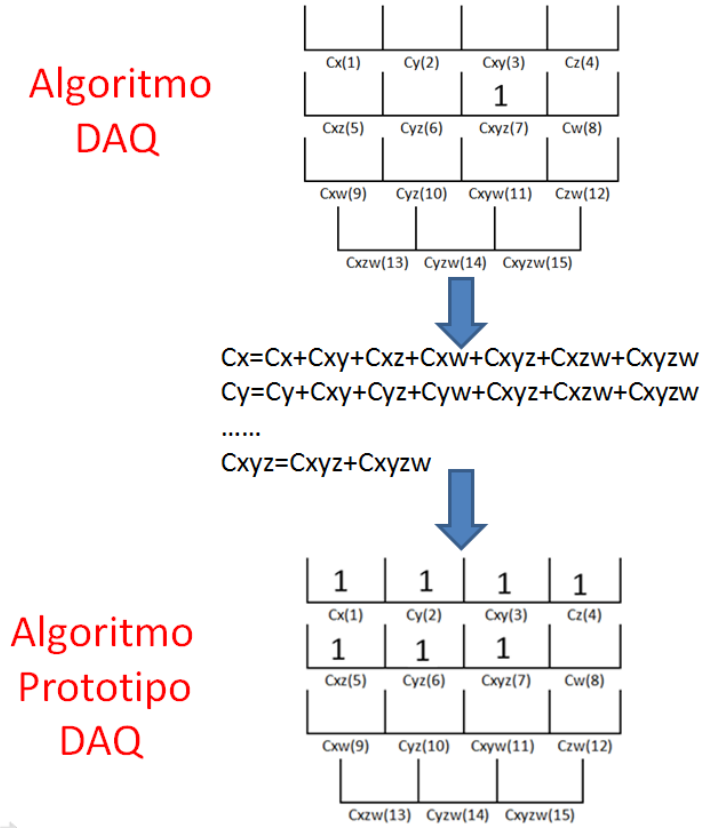


Figura 3.3:Confronto tra gli algoritmi

3.3 Firmware FPGA

Prima di descrivere il firmware utilizzato per il rilevatore con otto scintillatori e quindi nel DAQ finale, si descrivono degli elementi funzionali presenti su alcune famiglie di FPGA della Xilinx utilizzati nel firmware del DAQ. In particolare si descrive la Block RAM (Fig. 3.4), che consiste in una particolare memoria in cui salvare dei dati, in questo caso il valore dei diversi contatori, e il DCM (Digital Clock Manager, Fig. 3.4) che permette di modificare la frequenza del clock per alcuni blocchi.

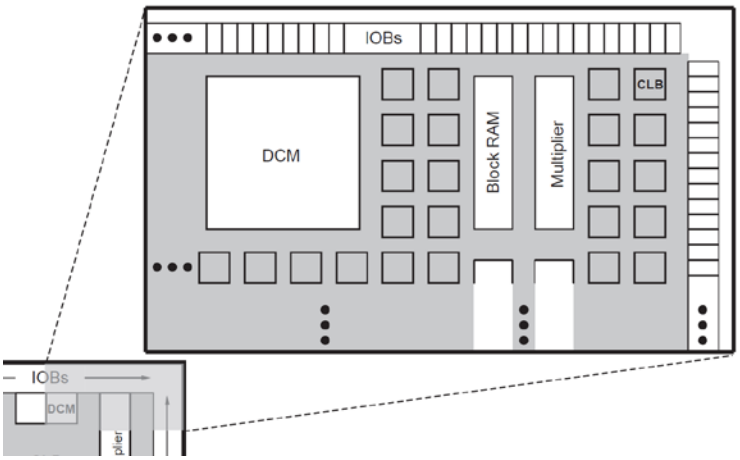


Figura 3.4:Architettura FPGA

3.3.1 Block RAM

L'elemento fondamentale delle FPGA sono le CBL (Configurable Logic Blocks), queste permettono l'implementazione dei circuiti sequenziali e combinatori. All'interno delle CBL sono presenti delle porte logiche e degli elementi di memoria, cioè i Flip-Flop. Nel momento in cui si mappa un codice VHDL nell'FPGA, normalmente il programma di sintesi assegna ai diversi Flip-Flop gli elementi di memoria (variabili, costanti, segnali.....) e, molto spesso, i Flip-Flop dell'FPGA non sono sufficienti per mappare la memoria richiesta dal codice e, per questo, si utilizzano le Block RAM.

La Block RAM [24-26] consiste in 18432 bits di RAM statica veloce di questi, 16 Kbits sono allocati per salvare dati e 2 Kbits sono utilizzati quali bit di parità dei dati memorizzati o per spazio aggiuntivo. Diversi blocchi di Block RAM, organizzati in colonne (Fig. 3.5) sono utilizzati dall'FPGA scelta e, in base alle caratteristiche di quest'ultima variano, in numero e capacità. Per l'FPGA utilizzata dal DAQ si hanno a disposizione 2 colonne composte da 6 blocchi con una capacità di memorizzazione aggiuntiva di 216 Kbits.

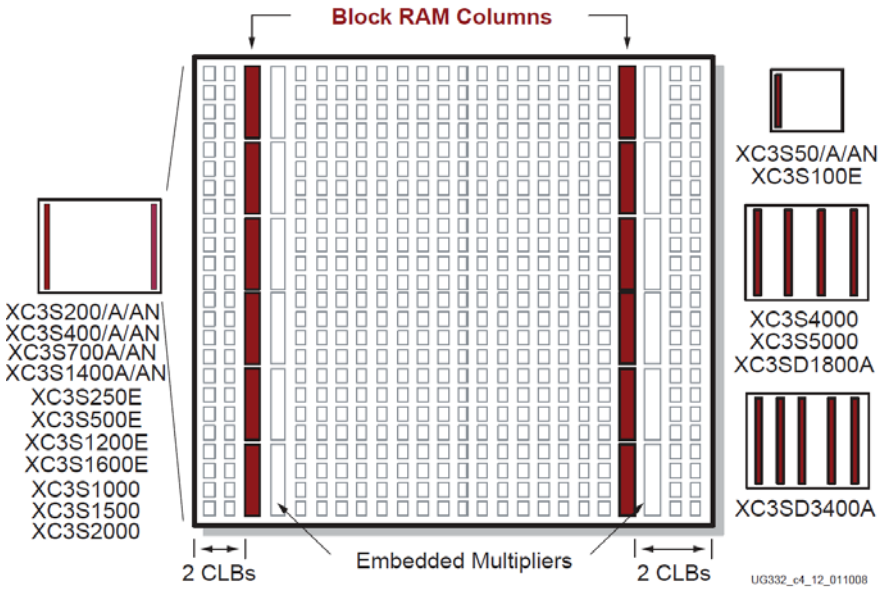


Figura 3. 5:Posizionamento delle Block RAM

Ogni Block RAM fisicamente è caratterizzata da due porte, denominate PORTA e PORTB, completamente indipendenti e che condividono la stessa memoria; entrambe le porte sono sincrone sia in lettura che in scrittura [39]. L'accesso alla memoria della Block RAM può avvenire utilizzando entrambe le porte (Dual-Port, Fig. 3.6: (a)) oppure utilizzandone una sola (Single-Port, Fig. 3.6: (b)). La configurazione dei blocchi della Block RAM può variarne le dimensioni condividendo tra di loro tutta o parte della memoria, la gestione dei Block RAM modificati avviene come un normale Block RAM.

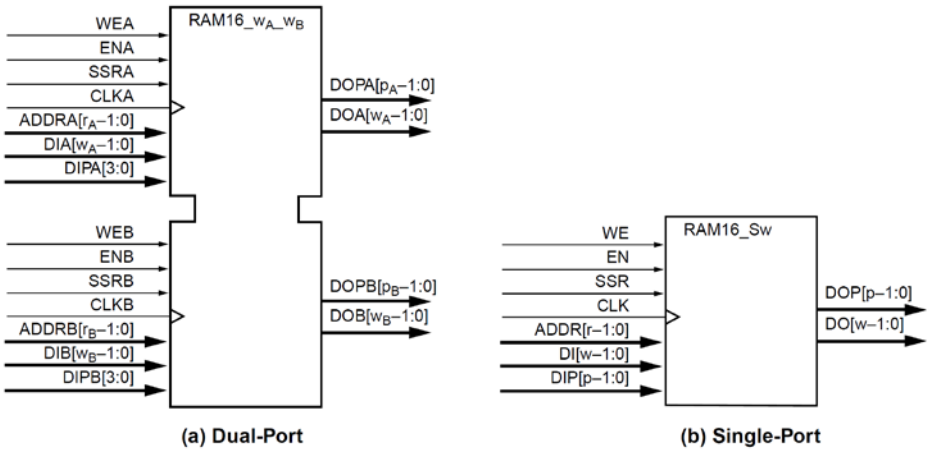


Figura 3. 6:Tipi Block RAM

Ogni porta ha due bus distinti di lettura e scrittura della memoria, ognuno dei quali utilizza un bus degli indirizzi e alcuni segnali di controllo. I segnali di gestione della Block RAM sono:

- *CLK* (Clock), segnale di clock utilizzato per la lettura e scrittura nella RAM;
- *EN* (Enable), permette di abilitare la lettura e scrittura nel Block RAM quando i segnali *WE*, *SSR* e *GSR* sono a livello logico basso;
- *WE* (Write Enable), abilita la scrittura in memoria all'indirizzo specificato da *ADDR*.
- *SSR* (Synchronous Set\Reset), imposta i latch d'uscita al valore specificato dall'attributo *SRVAL*, se la memoria è abilitata (*EN* a livello logico alto);
- *GSR* (Global Set/Reset), segnale di sistema utilizzato in fase di inizializzazione della Block RAM.
- *DIP* (Data Input Parity), bit di parità delle informazioni scritte in RAM;
- *DOP* (Data Output Parity), bit di parità delle informazioni lette dalla RAM;
- *DI* (Data Input), dati scritti nella RAM nella posizione indicata da *ADDR*;
- *DO* (Data Output), sati letti dalla RAM in posizione indicata da *ADDR*, il suo valore potrebbe essere modificato durante la fase di scrittura in accordo con l'impostazione scelta per l'attributo *WRITE_MODE* (che verrà trattato successivamente);
- *ADDR* (Address Bus), indirizzo di scrittura o lettura della RAM.

I segnali riportati sono riferiti al Block RAM con porta singola, per il Dual-Port si utilizzano gli stessi prefissi seguiti con il nome della porta (A o B).

Per definire la Block RAM nel firmware e configurarla ,si possono utilizzare o dei template di codice VHDL [27] all'interno di ISE o il tool Xilinx Core Generator che genera direttamente il codice VHDL della Block RAM con tutti i parametri

impostati. Per descrivere gli attributi utilizzati per la configurazione della Block RAM ci riferiamo ai parametri richiesti dal Core Generator.

Dopo aver selezionato il tipo di memoria (single-port o dual), il Core Generator richiede di specificare la dimensione della memoria (Fig. 3.7) e, da queste informazioni il tool determina le dimensioni dei bus dati di lettura, scrittura e di indirizzamento.

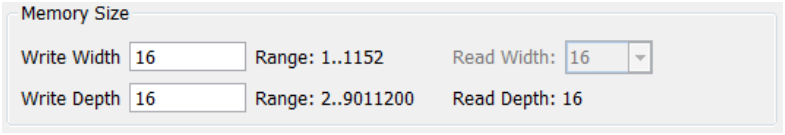


Figura 3.7:Dimensionamento Memoria

Un altro parametro richiesto dal tool è l'attributo *WRITE_MODE* (Fig. 3.8) che determina il comportamento della Block RAM e dei latch d'uscita in fase di scrittura. L'attributo può assumere tre differenti valori e sono:

Figura 3.8:Attributo WRITE_MODE

- *WRITE_FIRST*: questo è il modo di default in cui il dato scritto nella memoria è simultaneamente salvato nei latch d'uscita (Fig. 3.9 e Fig. 3.10). In questo modo la Block RAM risulta completamente trasparente. Nel caso si utilizzi le Block RAM in configurazione Dual-Port questa modalità causa l'invalidazione dei dati dell'altra porta.

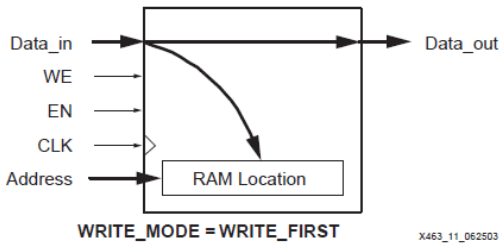


Figura 3.9:Flusso dei dati in WRITE_FIRST

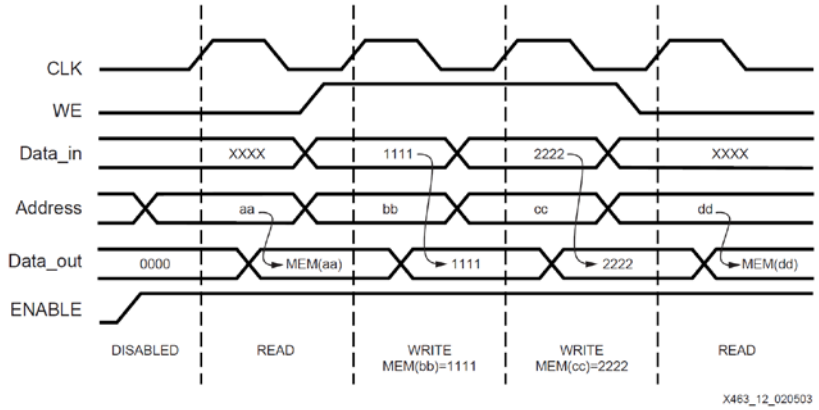


Figura 3.10:Forma d'onda nella modalità WRITE_FIRST

- *READ_FIRST*: in questa modalità viene prima letto il valore puntato da ADDR e salvato nei latch d'uscita e successivamente il valore in memoria viene sovrascritto con il valore contenuto nel bus dati (Fig. 3.1 e Fig. 3.12).

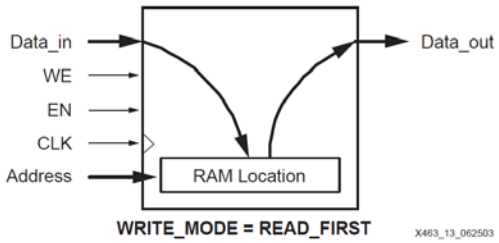


Figura 3.11:Flusso dei dati in READ_FIRST

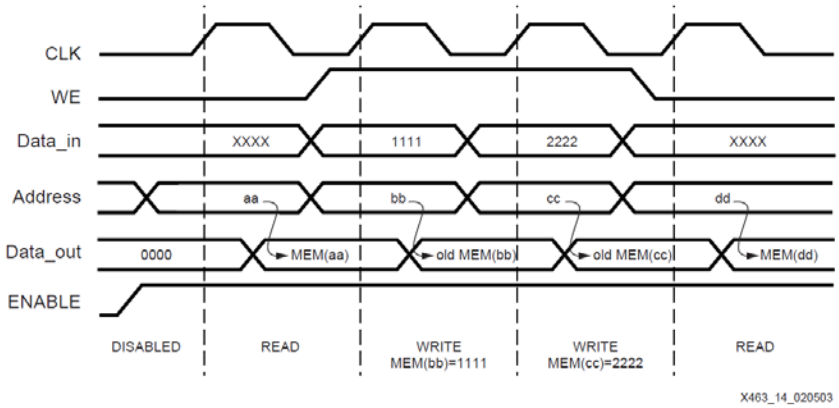


Figura 3.12:Forma d'onda nella modalità READ_FIRST

- *NO_CHANGE*: in cui il dato viene scritto in memoria senza effettuare nessun cambiamento sui latch d'uscita. Nel caso si utilizza la Block

RAM in configurazione Dual-Port questa modalità causa l'invalidazione dei dati dell'altra porta (Fig. 3.13 e Fig. 3.14).

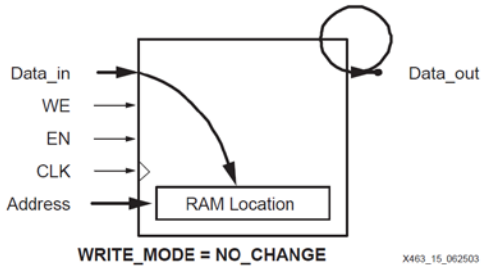


Figura 3.13:Flusso dei dati in NO_CHANGE

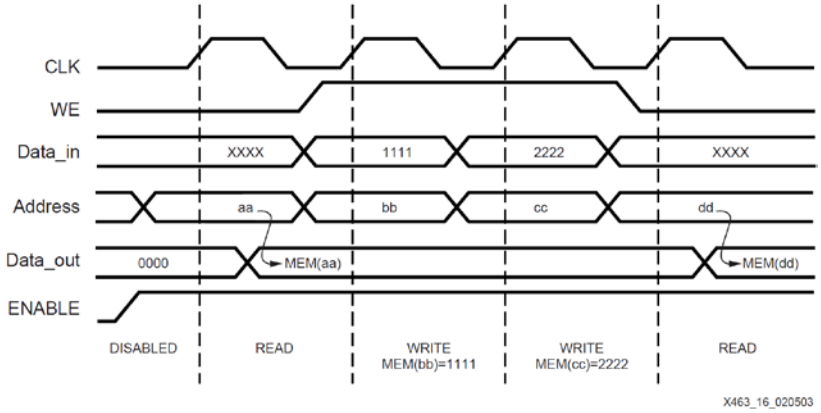


Figura 3.14:Forma d'onda nella modalità NO_CHANGE

Un ulteriore parametro richiesto dal Core Generator quando si utilizza la linea *RST*, è rappresentato dai valori che devono assumere i latch d'uscita (Fig. 3.15).

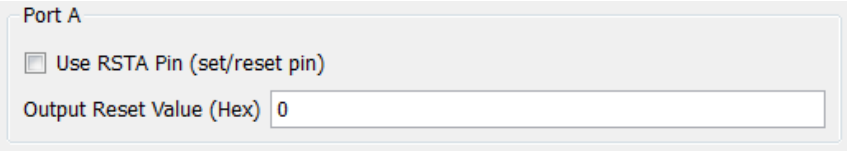


Figura 3.15:Configurazione di RST

Il Core Generator, inoltre chiede di specificare il file *coe* di inizializzazione della memoria (Fig. 3.16). Questo file oltre a rappresentare il valore delle diverse celle di memoria mediante la parola chiave *memory_initialization_vector* (Fig. 3.17) specifica anche mediante quale codice i valori sono scritti; ciò avviene mediante la parola chiave *memory_initialization_radix*, (Fig. 3.17 che essendo pari a 16 indica che i valori sono espressi in esadecimale).

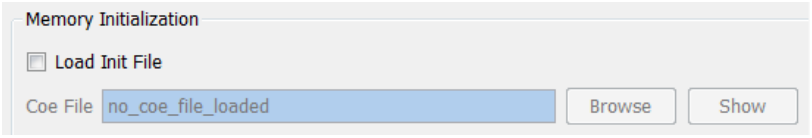


Figura 3.16:Opzione del file di inizializzazione della memoria

```
memory_initialization_radix=16;
memory_initialization_vector=00000000,00000000,........,00000000;
```

Figura 3.17:File coe di inizializzazione

Inoltre nella configurazione Dual-Port c'è la possibilità di impostare la gestione dei conflitti d'accesso tra le due porte.

Nel firmware del DAQ si utilizzeranno due blocchi di Block RAM per poter eseguire il salvataggio dei 255 contatori. Alla fine di ogni finestra d'ascolto il firmware calcola la posizione della memoria del relativo contatore, lo legge, ne incrementa il valore e lo riscrive in memoria nella stessa posizione.

3.3.2 Digital Clock Manager (DCM)

Al termine della finestra d'ascolto si deve eseguire l'aggiornamento del contatore associato all'evento verificato. Per far ciò si deve accedere alla Block RAM per leggere il valore attuale del contatore, incrementarlo e riscriverlo nella RAM; queste tre operazioni devono essere eseguite in un tempo inferiore alla finestra d'ascolto perché, nel caso di apertura di una nuova finestra d'ascolto, nell'istante in cui l'altra viene chiusa si può avere una perdita di dati. Sapendo che una finestra d'ascolto dura 4 cicli di clock e che, per ogni operazione sulla RAM occorre un ciclo clock, si potrebbe erroneamente dedurre che per eseguire l'aggiornamento del contatore siano sufficienti solo 3 cicli di clock, in realtà ne servono 4 in quanto, per ottenere il valore letto dalla memoria occorrono due cicli di clock e non uno perché l'aggiornamento dei latch d'uscita avviene sul fronte di discesa e non sul fronte di salita. Per non avere problemi di perdita dati si è deciso di interrogare la RAM con una frequenza doppia rispetto alla frequenza degli altri elementi. Per generare un clock a frequenza doppia rispetto al clock esterno si utilizza un Digital Clock Manger (DCM).

Il DCM [24-26, 40]è un blocco funzionale che permette di generare un nuovo segnale di clock con frequenza diversa e/o sfasato rispetto a quello fornito in ingresso, inoltre permette di eliminare lo skew[1] o migliorare il jitter[2].

In Figura 3.18 si può notare la posizione dei quattro DCM nell'FPGA, ciascuno ha delle connessioni dedicate con i pin di clock e di conseguenza con i buffer d'ingresso e con i multiplexer di gestione dei buffer.

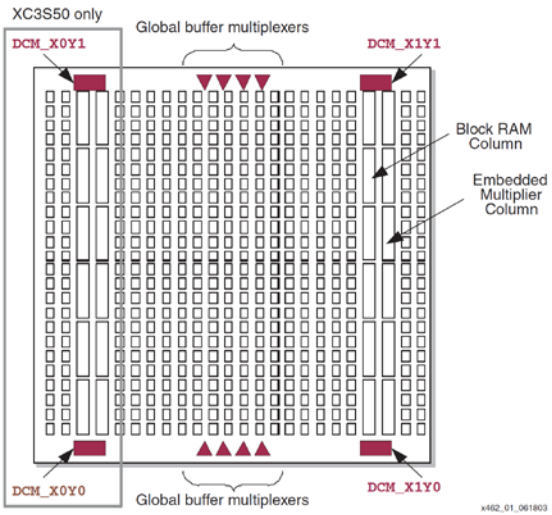


Figura 3.18:Posizione dei quattro DCM

Il DCM consiste di quattro unità funzionali distinte (Fig. 3.19) che operano insieme o in maniera indipendente, queste sono:

- *Delay-Locked Loop* (*DLL*), che permette di eliminare lo skew, compensando il ritardo introdotto dal routing degli elementi;
- *Digital Frequency Synthesizer* (*DFS*), che permette di modificare la frequenza del clock di un fattore dato dal rapporto tra *CLKFX_MULTIPLY* e *CLKFX_DIVIDE* entrambe definiti dall'utente;
- *Phase Shift* (*PS*), che permette di controllare la fase tra il clock d'ingresso e quello d'uscita. Lo sfasamento tra i due segnali è

[1] il Clock Skew è il ritardo, tipico dei sistemi sincroni, con cui lo stesso fronte del clock raggiunge i diversi dispositivi che costituiscono il sistema.

[2] il Clock jitter è dato dalla variazione della posizione nel tempo del fronte di salita rispetto al fronte ideale

determinato dai segnali *PSE*, *PSINCDEC* e *PSCLK* e da alcuni attributi;

- *Status Logic*, indica lo stato del DCM; il segnale *LOCKED* è posto a livello logico alto quando l'uscita del DCM è in fase con il *CLKIN*.

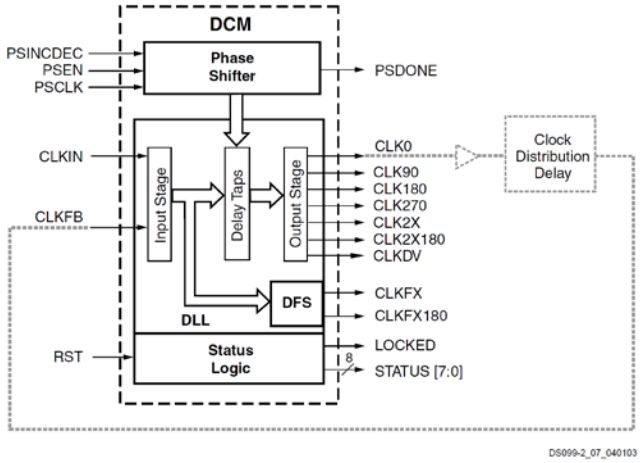


Figura 3.19:Blocchi funzionali DCM

In Figura 3.19 è riportata la primitiva del DCM. Si descriveranno solo gli ingressi [40] e le uscite che verranno utilizzati nel firmware; per avere una frequenza doppia rispetto alla frequenza d'ingresso si devono utilizzare:

- *CLKIN*, clock d'ingresso del DCM, la sua frequenza è limitata dagli attributi *DLL_FREQUENCY_MODE* e *DFS_FREQUENCY_MODE*;
- *CLKFB*, questo ingresso è utilizzato per avere una retroazione con l'uscita *CLK0* o *CLK2X*. La retroazione è utilizzata dal DCM per bilanciare il delay;
- *RST*, reset asincrono del DCM;
- *CLK0*, clock con la stessa frequenza di quello d'ingresso con sfasamento di 0°;
- *CLK2X*, clock con frequenza doppia rispetto a quello d'ingresso con sfasamento di 0°;
- *LOCKED*, il clock in uscita è stabile in fase e frequenza.

I parametri e gli attributi che si devono impostare per la gestione del DCM sono diversi e servono per controllare il periodo del clock d'ingresso, il clock di

feedback, la correzione del duty cycle, i parametri per il controllo della frequenza d'uscita e lo shift di fase. In particolare i parametri impostati nel firmware del DAQ sono:

- *DLL_FREQUENCY_MODE*, specifica il range di frequenza del clock d'ingresso e d'uscita e deve eesere impostato a LOW;
- *CLKIN_PERIOD*, specifica il periodo del clock d'ingresso in ns;
- *CLK_FEEDBACK*, definisce quale clock si utilizza come feedback, se il *CLK0* o *CLK2X*.

La creazione del DCM all'interno del firmware può avvenire tramite template o tramite il Core Generator (Fig. 3.20), in entrambi i casi si devono abilitare gli I/O descritti in precedenza.

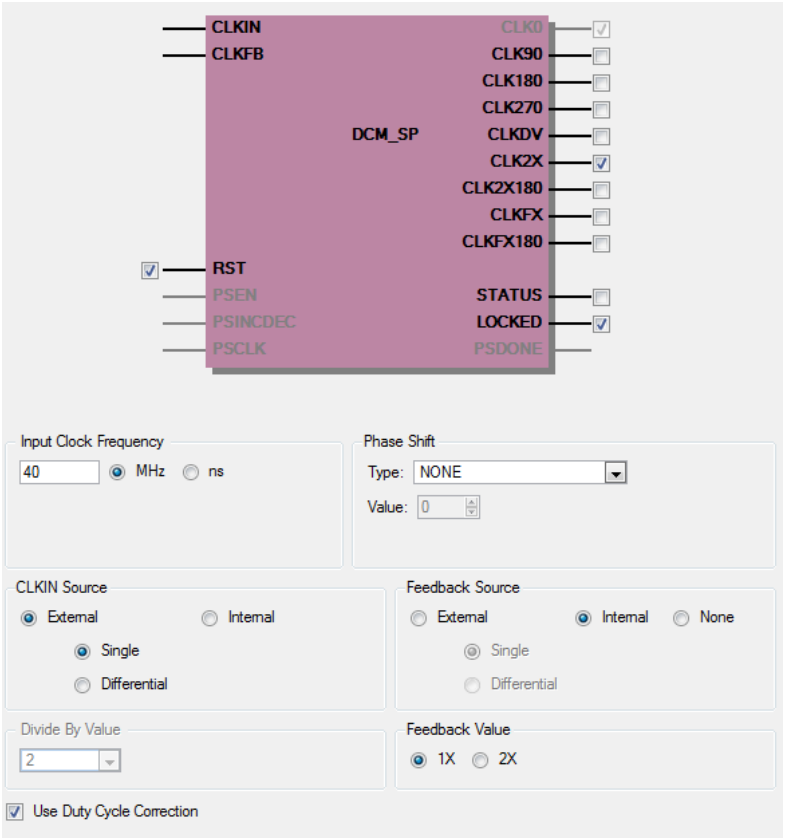


Figura 3.20:Impostazione del DCM tramite il Core Generator

3.3.3 Firmware

Il firmware del DAQ è simile al firmware del prototipo, implementato con i processi di holding dei quattro scintillatori aggiunti (*COUNTT*, *COUNTR*, *COUNTS*, *COUNTQ*) per mantenere costante il livello del segnale associato allo scintillatore per la durata della finestra d'ascolto. Il processo *WINCON*, come nel firmware del prototipo, gestisce l'incremento dei contatori utilizzati per il controllo d'errore, solo che, in questo caso, considerando gli eventi singoli non c'è bisogno della matrice delle coincidenze. Esso determina l'indirizzo di memoria associato al contatore dell'evento rilevato mediante i segnali prodotti dai processi di holding, e abilita il processo d'incremento (*INCDATE)*. Cambia, invece, la modalità di trasferimento dati rispetto al prototipo che, alla chiusura della finestra di conteggio costruiva la coda utilizzata per il trasferimento, riportando i valori dei contatori e il microcontrollore dettava i tempi di trasferimento. Nella versione finale del firmware non è possibile implementare la coda per il trasferimento in quanto non mappabile per cui, per il trasferimento dei dati, si utilizzano due Block RAM in modalità duale cioè, mentre una memoria è occupata per l'acquisizione dei dati sul flusso dei raggi cosmici, l'altra è utilizzata per il trasferimento dati e alla fine della finestra di conteggio le due memorie si scambiano di ruolo. Per identificare quale memoria è utilizzata per il trasferimento e quale per la raccolta dati si utilizza un opportuno flag. I valori dei contatori, memorizzati nella RAM utilizzata in quel momento per il trasferimento dati sono letti uno alla volta e trasferiti. In realtà il valore che deve essere trasferito successivamente è letto durante il trasferimento del valore attuale. Quando il trasferimento dei bit contenuti nel contatore è terminato il suo valore è sovrascritto su quello del contatore, successivo minimizzando, in questo modo, i tempi di morti; anche in questo caso il tempo di trasferimento è dettato dal microcontrollore

In Figura 3.21 si nota la relazione tra il clock d'ingresso (Fig. 3.21: *clk*) e quello a frequenza doppia, generato dal DCM (Fig. 3.21: *clk2x*). È opportuno sottolineare che tutti gli elementi sincroni dell'FPGA che lavorano alla frequenza di 40 MHz, cioè quella del clock esterno, non utilizzano quest'ultimo bensì quello generato dal DCM (*CLK0*), perché risulta essere più immune ai disturbi e ai ritardi.

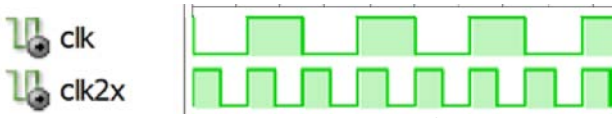


Figura 3.21: Clock d'ingresso (clk) del DCM e clock a frequenza doppia (clk2x)

Il processo WINCON gestisce la finestra d'ascolto e valuta l'evento che è occorso, andando ad abilitare l'incremento del contatore associato all'evento; in Figura 3.22 è riportato il flow-chart del processo in cui si può notare che la gestione della finestra d'ascolto è attiva solo se uno degli otto scintillatori rileva una particella, per cui alla fine della finestra d'ascolto incrementa i contatori di singola e calcola l'indirizzo della RAM associato all'evento (*add_ram*); il segnale *index* è utilizzato per abilitare i processi di trigger.

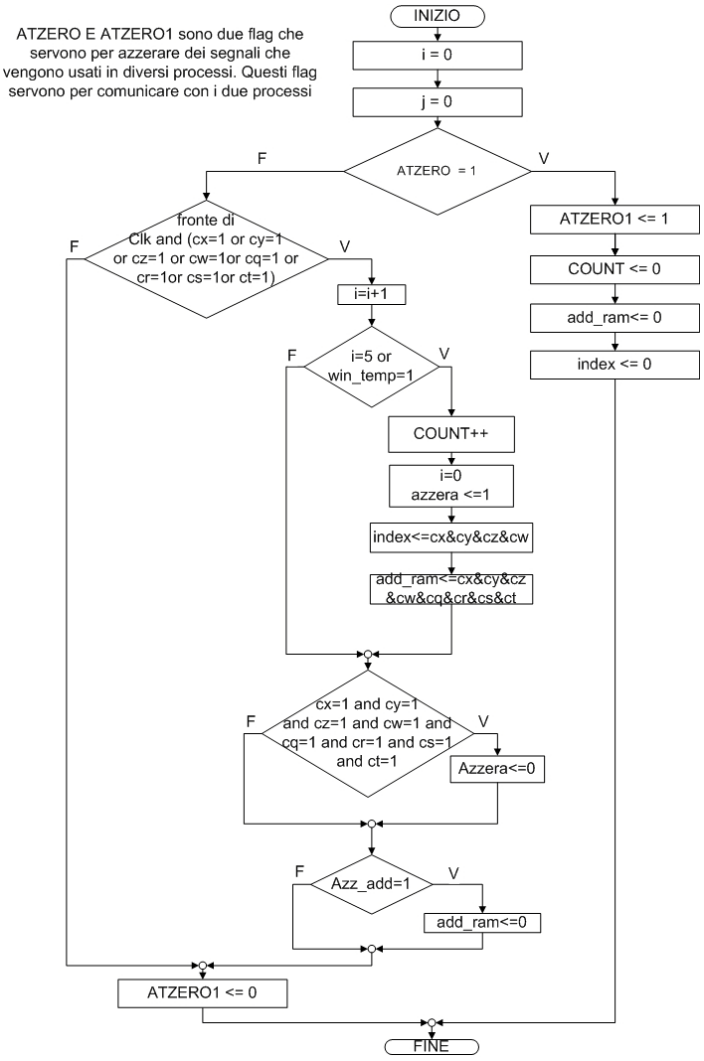


Figura 3.22:Flow-chart *WINCON*

In figura 3.23 sono riportati i segnali elaborati dai processi di holding e *WINCON* quando la particella è rilevata solo da due scintillatori (*x* e *y*). In questa simulazione i segnali provenienti dai scintillatori *x* e *y* sono nello standard LVDS, infatti i segnali per ogni rilevatore sono due (Fig. 3.23: *tx_p*, *tx_n* e *ty_p*, *ty_n*). Questi segnali, differenziali all'interno dell'FPGA, vengono trasformati in single-ended tramite un buffer, che è adoperato dai processi di holding per la creazione dei segnali (Fig. 3.23: *scx* e *scy*) utilizzati per valutare l'evento. Nell'istante in cui uno dei segnali di holding (Fig. 3.23: *scx*) assume il livello logico alto il processo *WINCON* avvia una nuova finestra d'ascolto (Fig 3.23: *wintalk*), nei successivi 100 ns lo scintillatore *y* rileva il passaggio della particella e, il processo *COUNTY*, pone il segnale di holding *scy* (Fig 3.23) a livello alto. Alla fine di questa finestra d'ascolto (Fig. 3.23: *win_talk*) il processo *WINCON* calcola l'indirizzo di memoria in cui è presente il contatore associato all'evento (Fig. 3.23: *add*); il calcolo dell'indirizzo avviene concatenando i segnali di holding (*scx&scy&scz&scw&scq&scr&scs&sct*), in questo caso pari a 192 (in binario 11000000). Lo stesso processo inoltre incrementa i contatori di singola utilizzati per il controllo d'errore (Fig. 3.23: *cx*, *cy*, *cz*, *cw*, *cq*, *cr*, *cs* e *ct*) e determina quali segnali di trigger attivare tramite il segnale *index_tri* (Fig. 3.23); in Figura 3.23 sono anche riportatiti i segnali d'uscita dei processi di trigger (Fig. 3.23: *singola*, *doppia*).

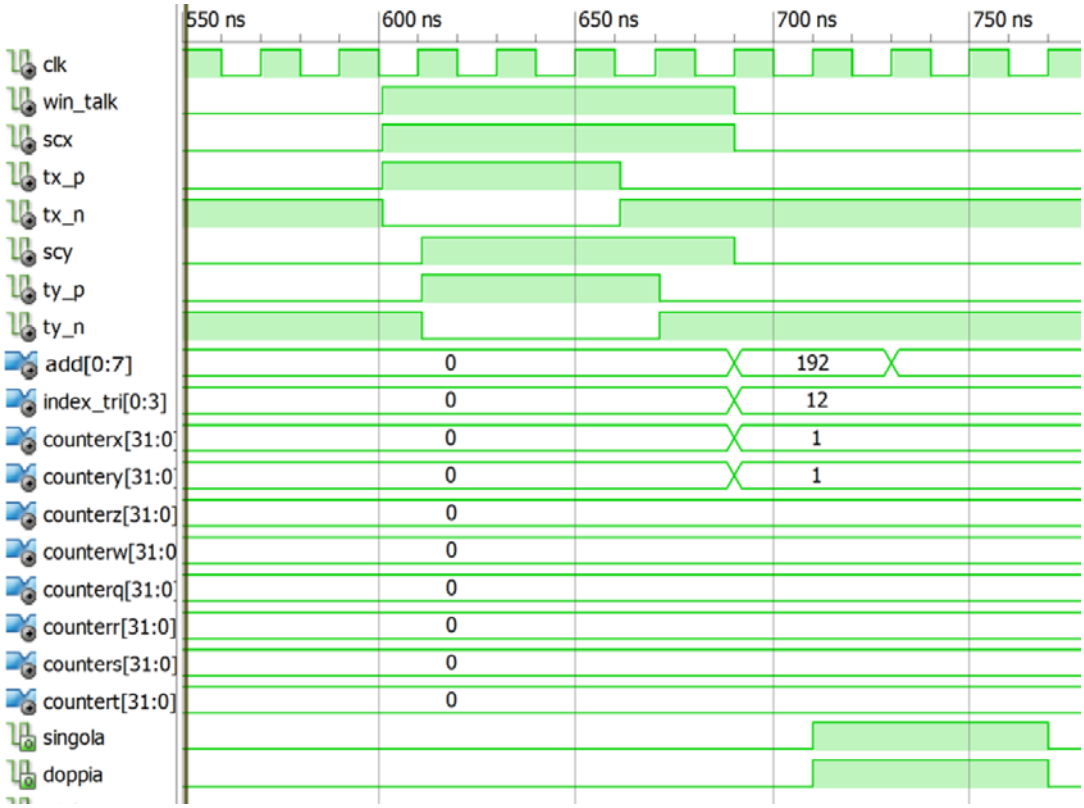


Figura 3.23:Simulazione del processo *WINCON*

L'indirizzo di memoria calcolato dal processo *WINCON* viene utilizzato dal processo di aggiornamento (*INC_DATE*), per puntare al contatore ed incrementarlo. Dal flow-chart del processo *INC_DATE*, riportato in Figura 3.24, si può notare che si utilizza il clock con frequenza doppia (*clk2x*) quando si deve modificare una cella di memoria e che ci sono due diversi blocchi di Block RAM (RAM0 e RAM1), ognuno dei quali ha la propria fase di scrittura e lettura. La scelta di quale memoria aggiornare dipende dal valore assunto dal segnale chooseRAM, 0 o 1 che, inizialmente vale 0 e, successivamente, varia al cambiamento di ogni finestra di conteggio. Questo segnale è utilizzato per individuare quale memoria si sta utilizzando per la statistica del flusso dei raggi cosmici e quale per il trasferimento dati; se il valore di chooseRAM è 0 si utilizza la RAM0 per la statistica e la RAM1 come coda per il trasferimento invece, se il valore è pari a 1 il ruolo delle due RAM è invertito.

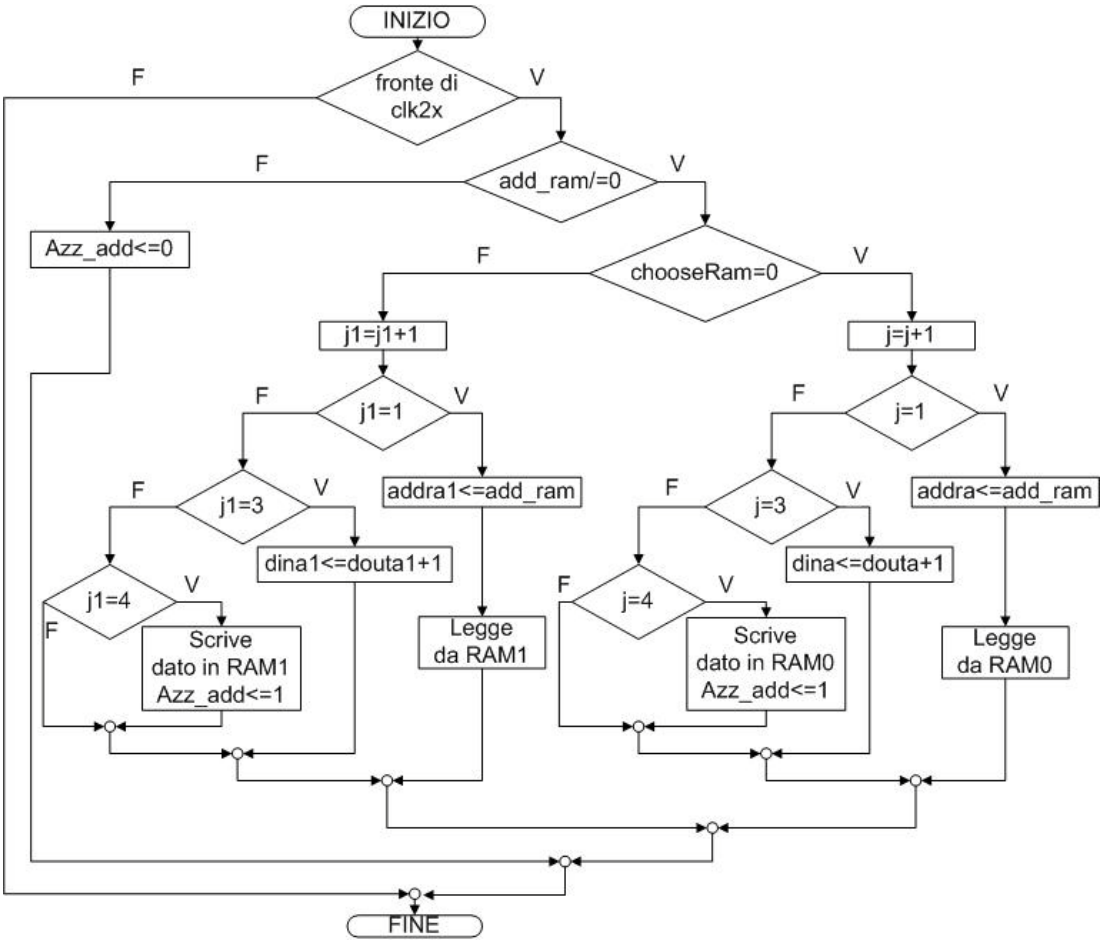


Figura 3.24:Flow-chart del processo INC_DATE

In Figura 3.25 è riportata la simulazione della scrittura in memoria, dopo un ciclo di clock (quello a frequenza doppia). Dal momento in cui il processo *WINCON* cambia il valore dell'indirizzo di memoria (Fig. 3.25: segnale *add* a 690 ns) il processo *INC_DATE* abilita la porta A del blocco della Block RAM (Fig. 3.25:*ena*) e trasferisce l'indirizzo calcolato nel bus degli indirizzi della memoria (Fig. 3.25:*addra*), dopodiché il processo legge il contenuto della memoria puntato da *addra* e lo pone in *douta* (in Figura 3.25 non si nota alcuna variazione perché il valore letto è 0). Nel ciclo di clock successivo si pone il bus dati in ingresso della memoria (Fig. 3.25: *dina*) al valore di *douta* incrementato di 1 e si abilita la scrittura in RAM ponendo a 1 il segnale *wea* (Fig. 3.25); dopo aver scritto in memoria il valore (Fig. 3.26), si disabilita la scrittura e la porta A della RAM, in Figura 3.25 si nota, inoltre, il cambiamento del segnale *douta* dovuto all'impostazione dell'attributo *WRITE_MODE* a *WRITE_FIRST*.

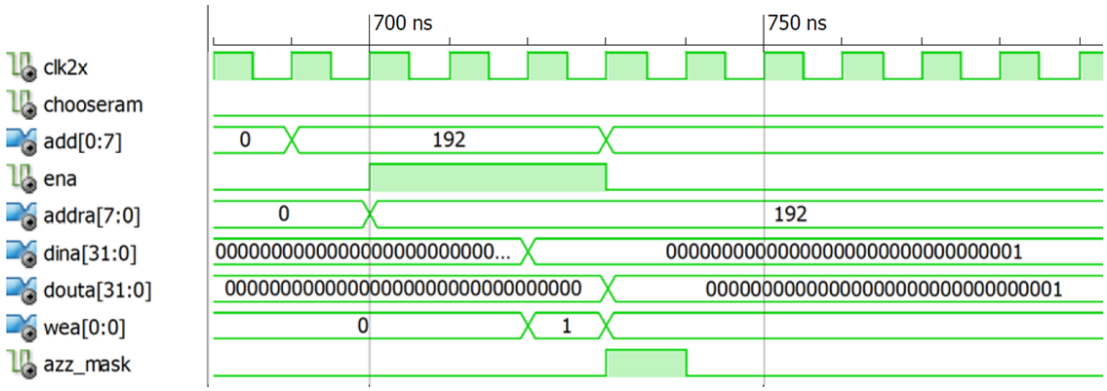


Figura 3.25:Simulazione del processo INC_DATE

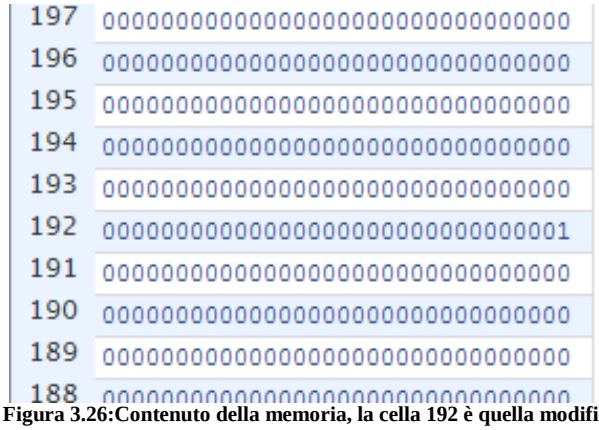


Figura 3.26:Contenuto della memoria, la cella 192 è quella modificata

Nella durata di una finestra di conteggio si sono verificati i sei eventi raffigurati in Figura 3.27, ciò comporta che nella memoria sia scritto un 2 nelle celle di memoria 1 e 192 e un 1 nelle celle di memoria 208 e 216, come evidenziato in Figura 3.28.

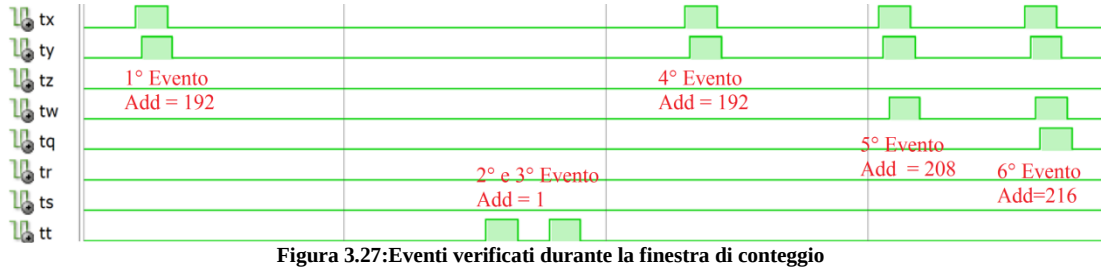
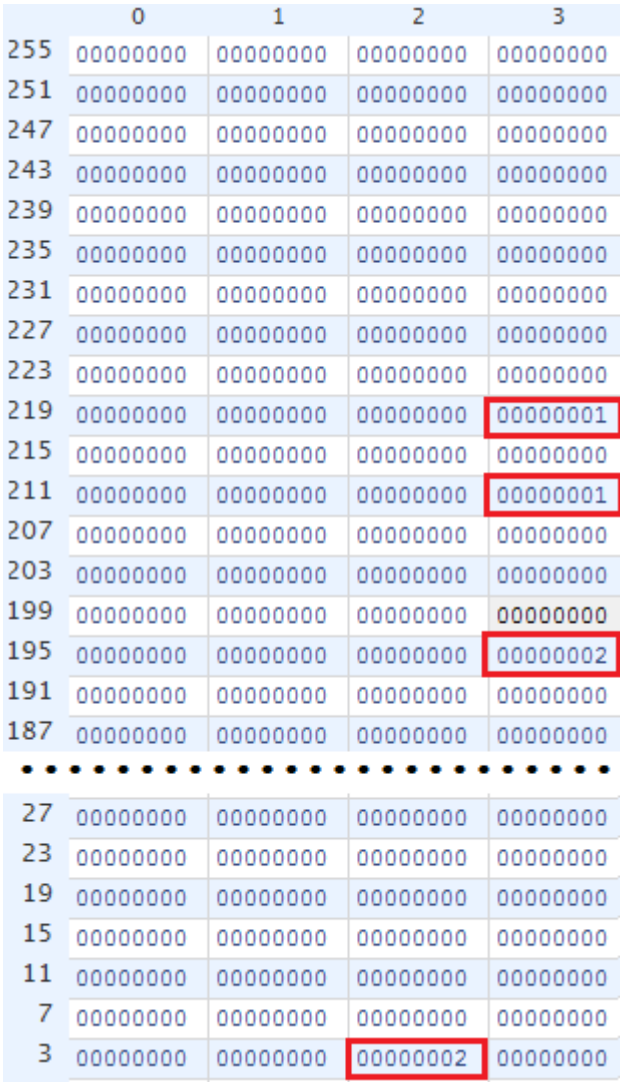


Figura 3.27:Eventi verificati durante la finestra di conteggio



	0	1	2	3
255	00000000	00000000	00000000	00000000
251	00000000	00000000	00000000	00000000
247	00000000	00000000	00000000	00000000
243	00000000	00000000	00000000	00000000
239	00000000	00000000	00000000	00000000
235	00000000	00000000	00000000	00000000
231	00000000	00000000	00000000	00000000
227	00000000	00000000	00000000	00000000
223	00000000	00000000	00000000	00000000
219	00000000	00000000	00000000	00000001
215	00000000	00000000	00000000	00000000
211	00000000	00000000	00000000	00000001
207	00000000	00000000	00000000	00000000
203	00000000	00000000	00000000	00000000
199	00000000	00000000	00000000	00000000
195	00000000	00000000	00000000	00000002
191	00000000	00000000	00000000	00000000
187	00000000	00000000	00000000	00000000
...				
27	00000000	00000000	00000000	00000000
23	00000000	00000000	00000000	00000000
19	00000000	00000000	00000000	00000000
15	00000000	00000000	00000000	00000000
11	00000000	00000000	00000000	00000000
7	00000000	00000000	00000000	00000000
3	00000000	00000000	00000002	00000000

Figura 3.28:Contenuto RAM alla fine della finestra di conteggio

Alla fine della finestra di conteggio, cioè sul fronte di discesa del segnale *WIN_TEMP*, il processo *NEWDATE* (Fig. 3.29) si attiva e inverte lo stato logico di *chooseRAM*, salva i contatori del controllo errore in una coda e avvisa il processo di trasferimento che ci sono dati disponibili mediante il segnale *READY_TRASF*. A differenza di quanto succedeva nel firmware del prototipo, in cui alla chiusura della finestra di conteggio si abilitava il trasferimento dati, nel firmware del DAQ finale prima di far ciò occorre effettuare la lettura del primo dato dalla memoria e porre a livello logico alto *READY_TRASF* e non *ENABLE_TRASF*.

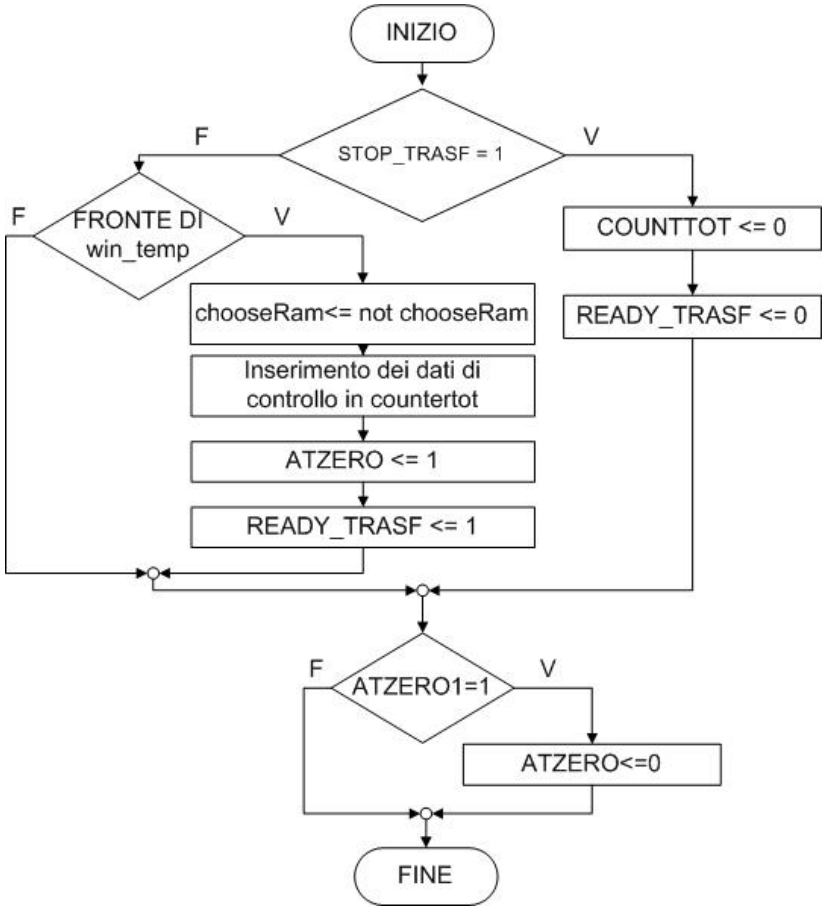


Figura 3.29: Flow-chart del processo NEWDATE

Dalla simulazione effettuata, riportata in Figura 3.30, il segnale *WIN_TEMP* non commuta a 1 s ma con un ritardo di 50 ns, ritardo dovuto al tempo necessario al DCM per fornire un'uscita stabile senza influenzare la statistica, in quanto il conteggio inizia solo quando il DCM è stabile. Sul fronte di discesa di *WIN_TEMP* (Fig. 3.30) il segnale *choose_RAM* si porta a livello logico alto, ciò vuol dire che la RAM0 è utilizzata per il trasferimento, quindi il segnale *READY_TRASF* viene posto ad 1 per leggere il primo valore dalla RAM, e successivamente il valore del segnale ritorna allo stato logico basso e i valori dei contatori di controllo (Fig 3.30: *cx, cy, cz, cw, cq, cr, cs* e *ct*) vengono allocati in una coda utilizzata per il trasferimento (Fig. 3.30: *countertot*).

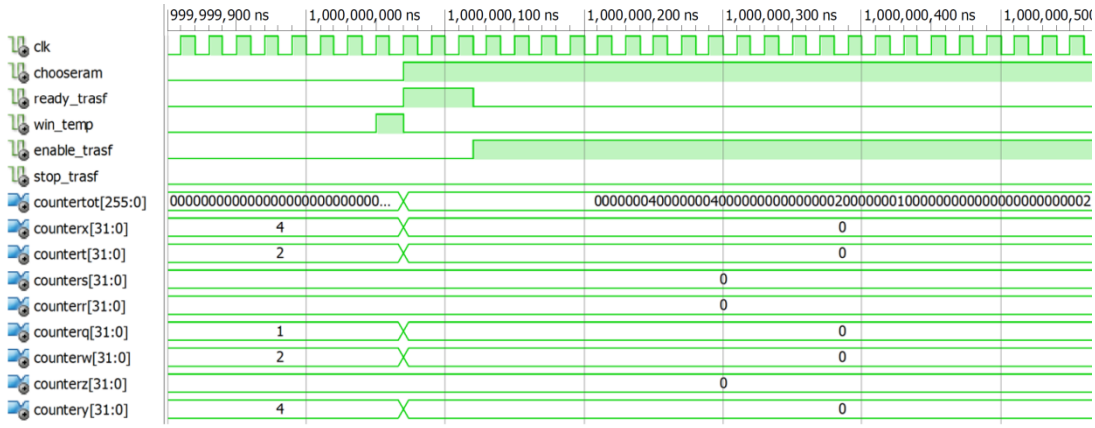


Figura 3.30:Simulazione di NEWDATE

Il processo *NEWDATE* pone a 1 il segnale *READY_TRASF* in modo che il processo *AGGIOR_DATE* legga il primo valore dalla memoria RAM e lo ponga in *DATA2;* questo segnale è utilizzato come appoggio per passare il dato al processo di trasferimento dati. Dal flow-chart (Fig. 3.31) si può notare che il processo lavora con il clock, generato dal DCM, a 80 MHz ed entra in esecuzione o quando il segnale *READY_TRASF* è a livello logico alto, questo succede solo quando si avvia una nuova finestra di conteggio, o quando il segnale *AGGIOR* è posto a livello logico alto e cioè solo quando è in corso il trasferimento dati e si deve accedere alla memoria per leggere il valore del contatore, che si trova nella celle adiacente a quella del contatore che si sta attualmente trasferendo. Dopo aver letto il valore, il contatore viene azzerato e reso disponibile per la successiva finestra di conteggio; se il valore letto corrisponde al primo della ram si pone a livello logico alto il segnale *ENABLE_TRASF*.

La simulazione effettuata per il processo *AGGIOR_DATE* è riportata in Figura 3.32, da dove si può notare che nel ciclo di clock successivo di *clk2x* da quando il segnale *READY_TRASF* è posto a livello logico alto, si abilita la RAM (Fig. 3.32: enb) e si pone a 1 l'indirizzo di memoria (Fig. 3.32: *addrb*), al successivo ciclo di clock il valore della cella puntata da *addrb* è posto in *doutb* (Fig. 3.32), che assume il valore 2, valore corrispondente a quello contenuto nella prima cella di memoria (Fig. 3.28), successivamente il valore letto è salvato in *DATA2* ed azzerata la cella di memoria(Fig. 3.32). L'operazione di scrittura a questo punto è conclusa e lo si capisce dal cambiamento di valore del bus d'uscita della memoria (Fig. 3.32:

doutb, il cambiamento è dovuto all'attributo *WRITE_MODE*). I processi di lettura e scrittura utilizzano due porte diverse della RAM perché VHDL non permette di utilizzare gli stessi segnali in processi diversi.

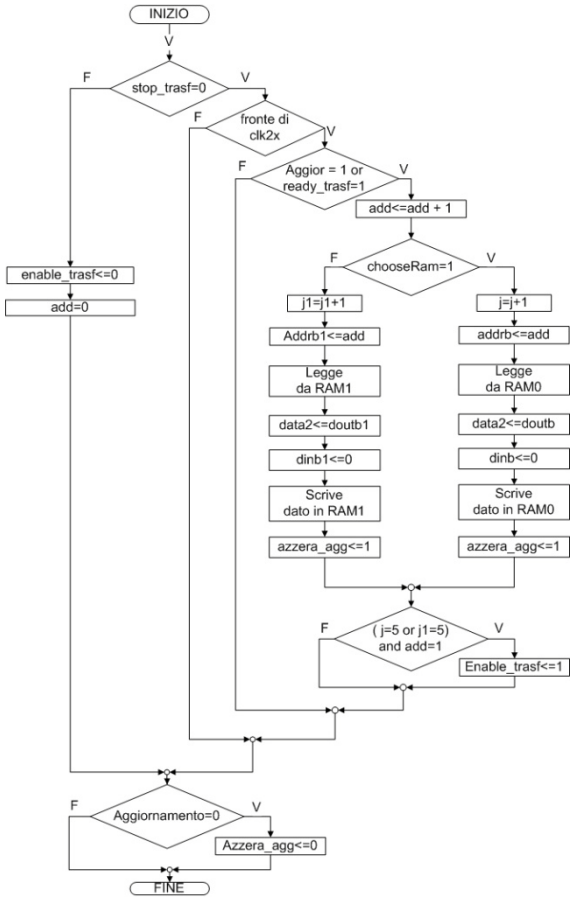


Figura 3.31Flow-chart AGGIOR_DATE

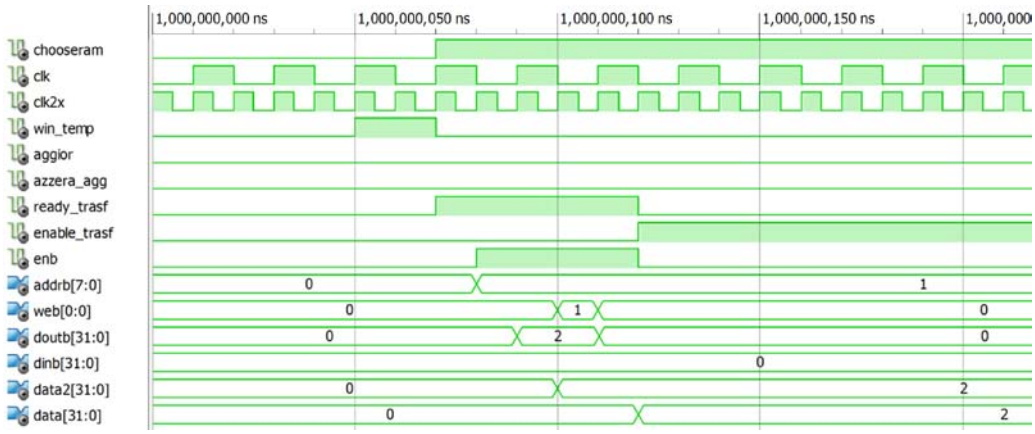


Figura 3.32: Simulazione Simulazione AGGIOR_DATE

Nella versione finale del firmware del DAQ il trasferimento dati verso il microcontrollore avviene in due fasi, una prima fase è dedicata al trasferimento dei dati contenuti nella memoria mentre nella seconda ha modo il trasferimento dati dei contatori di controllo contenuti nella coda *counttot*. Nella prima fase il processo *TRASF_RAM* (Fig. 3.33) attende il clock proveniente dal microcontrollore (come già avveniva nel prototipo del microcontrollore), invia i bit uno alla volta e durante l'invio del sedicesimo bit chiede al processo *AGGIOR_DATE* di assegnare al valore del segnale *DATA2* quello contenuto nel contatore successivo da inviare, mediante il segnale *AGGIOR*. Il processo dopo aver inviato il contenuto dei 255 contatori pone a uno il segnale PARITA per abilitare il processo di trasferimento della coda.

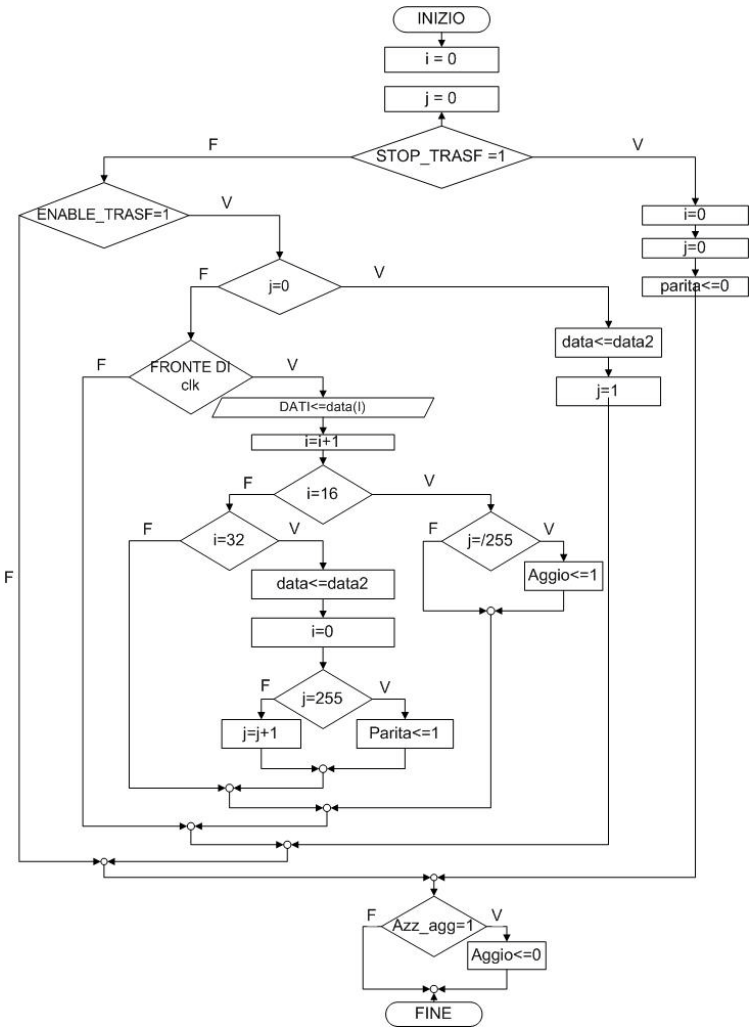


Figura 3.33:Flow-chart del processo TRASF_RAM

Nella simulazione del processo *TRASF_RAM* (Fig. 3.34), quando il segnale *AGGIOR* diventa 1, l'indirizzo di lettura della memoria (Fig. 3.34: *addrb*) viene incrementato e il valore di *DATA2* viene aggiornato con il nuovo valore, invece il segnale *DATA* è aggiornato solo dopo l'invio del trentaduesimo bit. In Figura 3.34 sono riportati i segnali *CLK_TRASF* e *ODATI* che sono quelli di comunicazione con il microcontrollore.

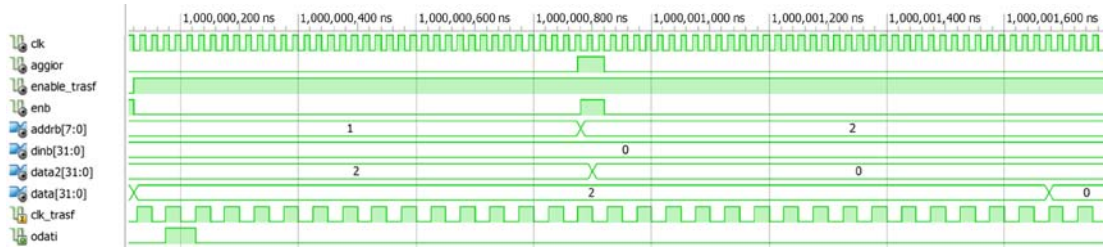


Figura 3.34:Simulazione del processo TRASF_RAM

Quando il processo *TRASF_RAM* pone a livello alto il segnale *PARITA* il processo *TRASF_CONT* (Fig. 3.35) trasmette tutti i bit contenuti nella coda *counttot,* successivamente, pone a livello logico alto il segnale *STOP_TRASF* e a livello logico basso il segnale *ENABLE_TRASF,* terminando così il trasferimento dati.

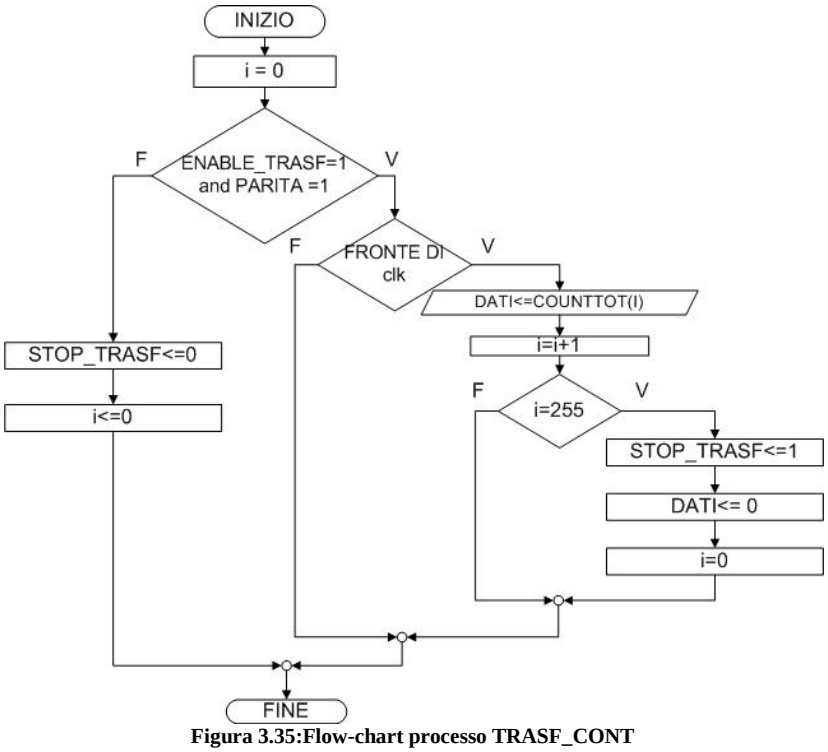


Figura 3.35:Flow-chart processo TRASF_CONT

3.4 Firmware del microcontrollore principale

Il firmware del microcontrollore principale si basa su quello sviluppato per il prototipo del DAQ, integrato con il codice per la comunicazione con il microcontrollore secondario e con la modifica per la scrittura di dati su SD CARD e il loro invio attraverso USB, per risolvere il problema relativo alla grande quantità di dati da gestire. Il microcontrollore del DAQ deve gestire 2484 byte di dati, così suddivisi

- 2104 byte provenienti dall'FPGA;
- 38 byte provenienti dal GPS;
- 342 byte provenienti dal microcontrollore secondario.

Sapendo che il massimo blocco di scrittura su SD CARD è di 512 byte e sulla porta USB di 64 byte, per scrivere tutti i dati occorre scrivere cinque volte su SD CARD e 39 volte su USB. Per far ciò, sul firmware del prototipo si ripete per cinque volte la scrittura su SD CARD e ogni blocco di scrittura è inviato attraverso la porta USB, suddividendolo in 8 sotto blocchi. Si è scelto di adattare il firmware in questo modo per renderlo più modulare e non specializzarlo al particolare caso.

Per controllare che il microcontrollore effettui tutte le operazioni in un tempo inferiore a 1 s (tempo minimo della finestra di default), sono state effettuate delle misure che riportiamo di seguito.

La prima misura è stata svolta per valutare il tempo di trasmissione dei dati tra FPGA e microcontrollore principale (Fig. 3.36), questo tempo è pari a circa 48.71 ms.

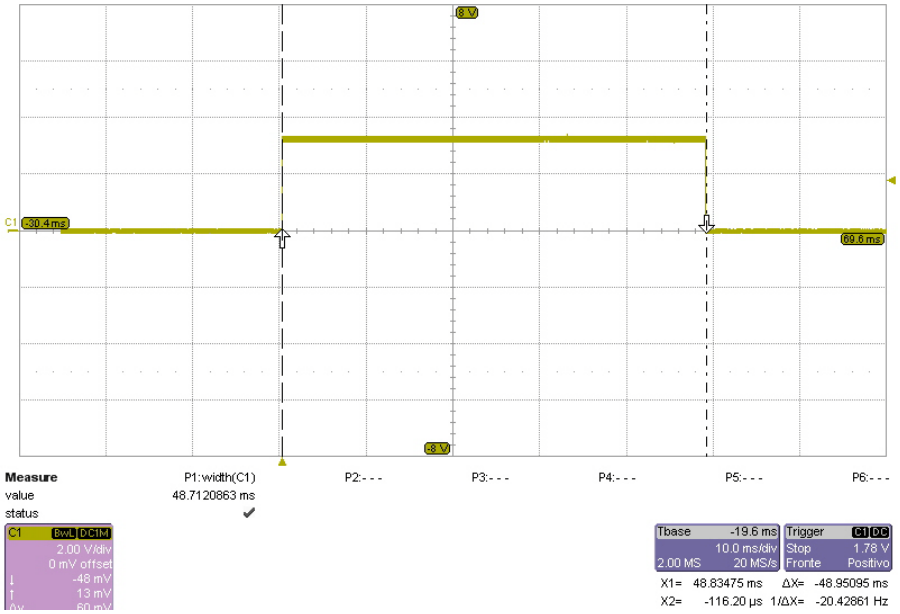


Figura 3.36:Tempo di trasmissione tra FPGA e microcontrollore principale

Nel firmware del DAQ si è introdotta la comunicazione con il microcontrollore secondario, che porta a livello logico alto un segnale, per pochi microsecondi consentendo l'avvio della procedura di interrupt nel microcontrollore secondario e l'invio dei dati tramite comunicazione seriale. Un fattore importante da determinare è il tempo di trasmissione di questi dati; dalla misura effettuata (Fig. 3.37) si nota che questo tempo e pari a 394.067 ms.

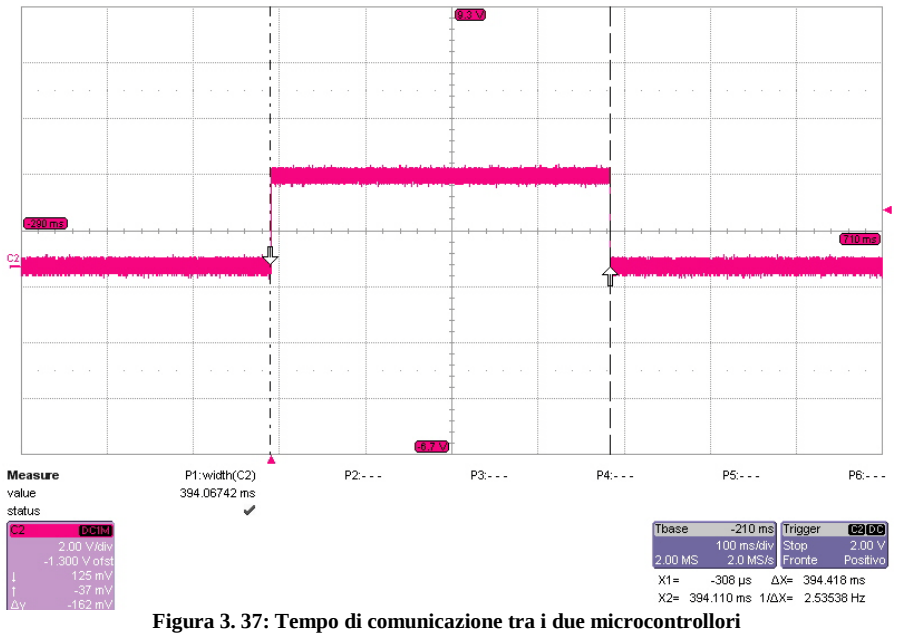


Figura 3. 37: Tempo di comunicazione tra i due microcontrollori

Si è inoltre valutato il tempo di scrittura su SD CARD, eseguendo due prove distinte, la prima che apre e chiude il file per tutti i 5 blocchi mentre la seconda lo apre e lo chiude una sola volta e poi scrive i 5 blocchi consecutivamente. Il tempo di scrittura nel primo caso (Fig. 3.38) è pari a 623.16 ms invece nel secondo (Fig. 3.39) è 421.52 ms, per cui quest'ultimo risulta essere quello migliore.

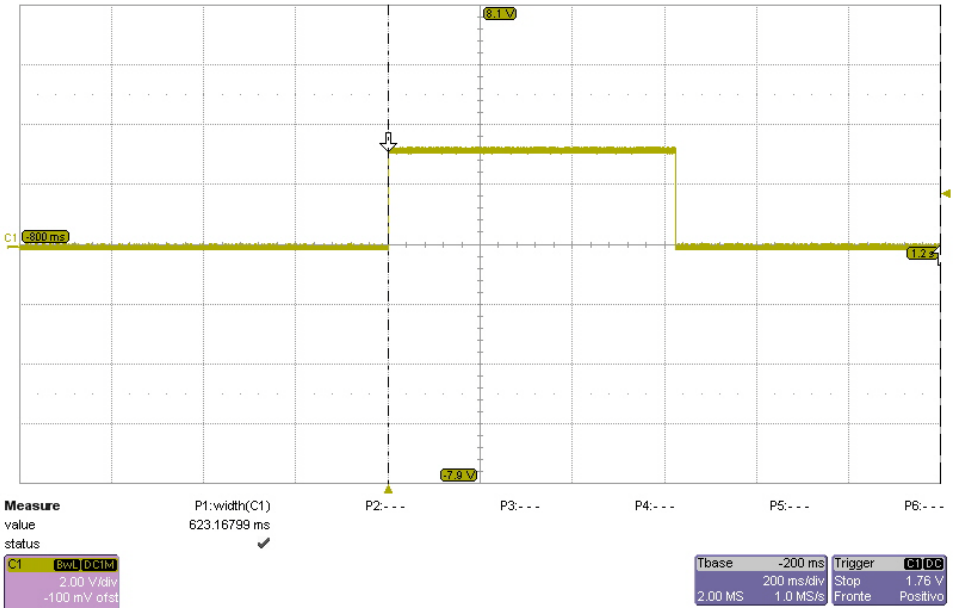


Figura 3.38:Tempo di scrittura su SD CARD nel primo caso

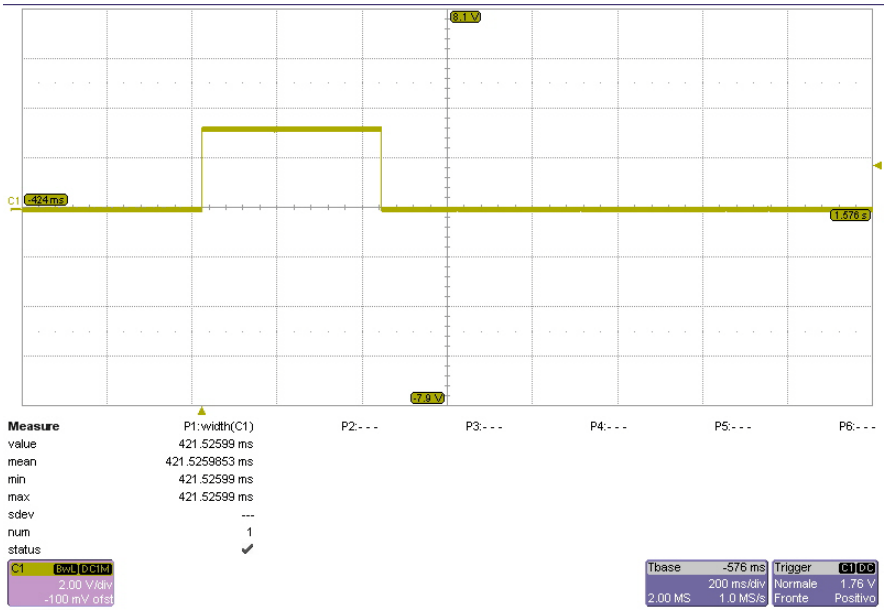


Figura 3.39:Tempo di scrittura su SD CARD nel secondo caso

Per ogni evento devono essere scritti 2484 byte e, dalle misure effettuate risulta che il tempo di scrittura su SD CARD aumenta in maniera lineare perché ogni volta che si scrive in modalità *append* si deve aprire il file e posizionarsi alla fine di esso, tale operazione richiede un tempo non trascurabile. Il file di scrittura viene sostituito ogni 900 eventi e il tempo impiegato per scrivere il novecentesimo evento risulta essere eccessivamente lungo, per cui si è deciso di sostituirlo ogni 150 eventi con un tempo massimo di scrittura pari a 1060 ms (Fig. 3.40). In Figura 3.40 è riportata la statistica temporale per la scrittura di 150 eventi, e in Figura 3.41 per 300 eventi con un massimo di 1350 ms.

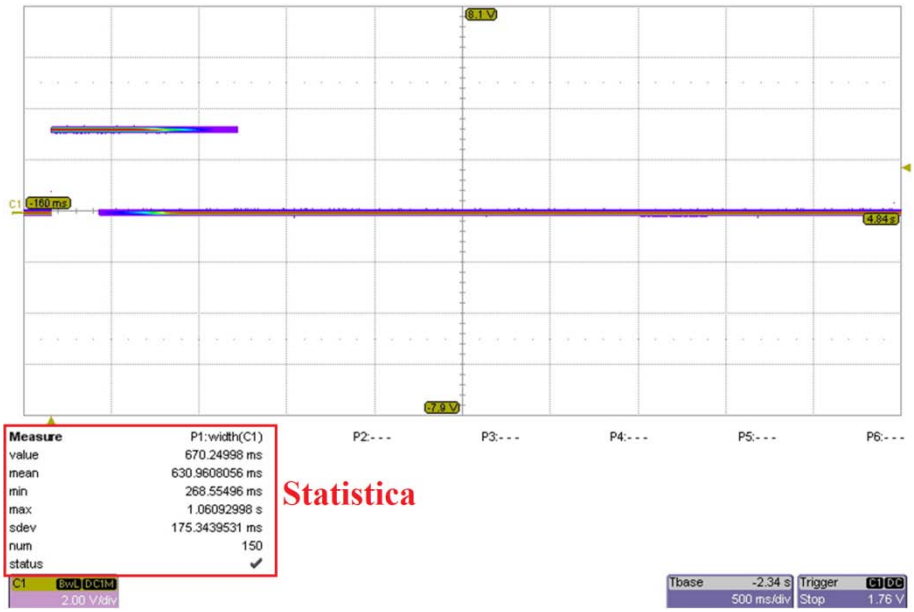


Figura 3.40:Statistica di scrittura su SD CARD per 150 eventi

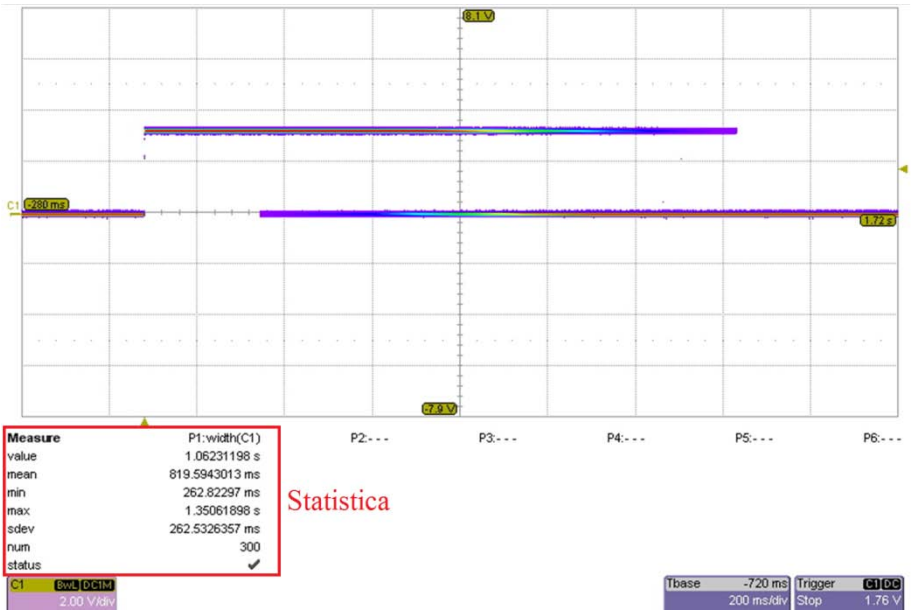


Figura 3.41:Statistica di scrittura su SD CARD per 300 eventi

La comunicazione mediante la porta USB deve avvenire interrogando 39 volte il bus della porta e, fra un interrogazione e l'altra, il microcontrollore deve comporre la stringa da inviare; il tempo impiegato per l'invio dei 39 blocchi è pari a 466.732 ms (Fig. 3.42).

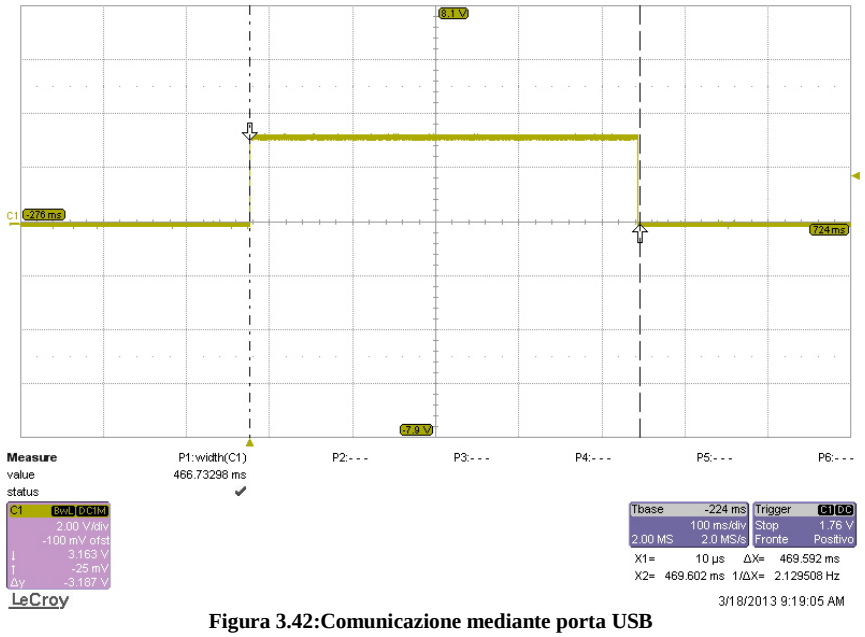


Figura 3.42:Comunicazione mediante porta USB

Le ultime misure effettuate riguardano il trasferimento seriale al sistema di telemetria del pallone e, dalle informazioni fornite dall'ASI il rate della

comunicazione seriale non può essere maggiore di 9600 baud. Con questo rate il tempo di comunicazione tra DAQ e sotto sistema di telemetria è pari a 2.3182 s (Fig. 3.43). Questo tempo non permette di implementare la finestra di conteggio da 1 s e siccome è un limite fisico che non può essere abbassato agendo sul firmware o sul software del DAQ si è portato la minima finestra di conteggio a 5 s.

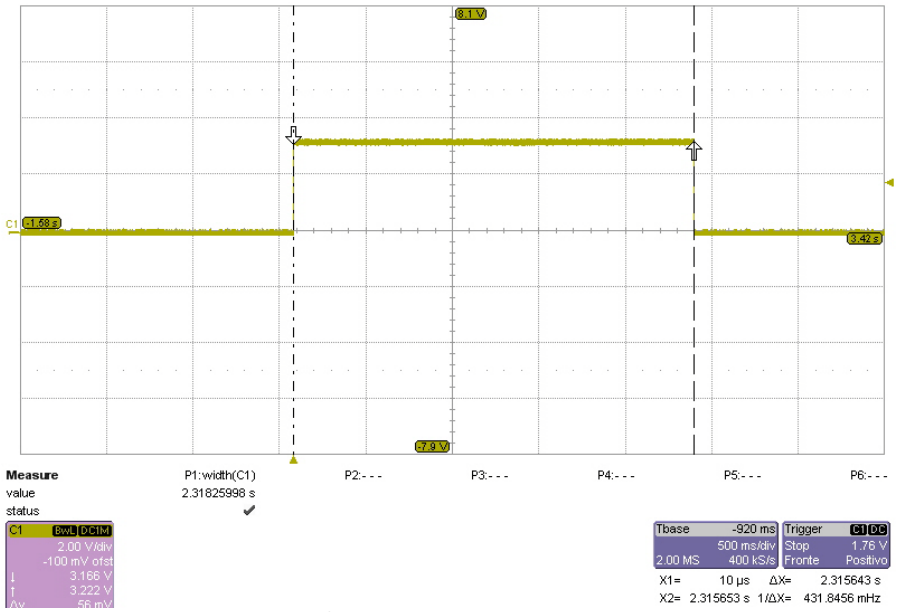


Figura 3.43:Trasferimento dati mediante seriale a 9600 baud

Si è anche valutato il tempo di trasferimento mediante la porta seriale con un rate di 115200 baud, che, per non essendo utilizzabile sul pallone potrebbe essere utile per possibili sviluppi futuri. Il tempo di trasmissione in questo caso è pari a circa 759.77 ms (Fig. 3.44).

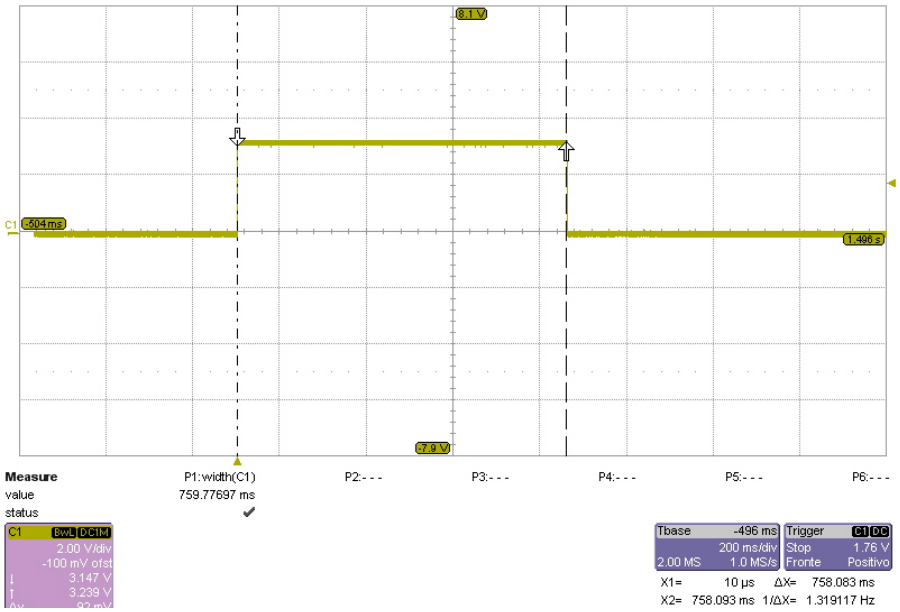


Figura 3.44: Tempo di trasmissione mediante porta seriale a 115200 baud

Pertanto il tempo di impiego del microcontrollore in una finestra di conteggio risulta, in media essere pari a:

- 1540.02 ms se si utilizza solo la porta USB;
- 3389.54 ms se si utilizza solo la porta seriale a 9600 baud;
- 1832.06 ms se si utilizza solo la porta seriale a 115200 baud;
- 3856.27 ms se si utilizza la porta seriale a 9600 baud e la porta USB;
- 2298.79 ms se si utilizza la porta seriale a 115200 baud e la porta USB.

In tutti i casi possibili il tempo di impiego del microcontrollore risulta essere inferiore alla finestra di conteggio minima (modificata) di 5 s (5000 ms). Gli ultimi due casi presi in considerazione sono fattibili ma inutili in quanto se si utilizza la porta USB vuol dire che il DAQ non è allocato sul pallone.

3.5 Firmware del microcontrollore secondario

Il microcontrollore secondario, PIC16F1847, è presente solo nel DAQ finale. Il suo compito è quello di acquisire informazioni sulla temperatura dai sensori, informazioni sull'accelerazione e sul campo magnetico. Mentre le prime

informazioni sono solo di controllo, quelle provenienti dall'accelerometro e del magnetometro servono per studiare gli effetti del campo magnetico sul flusso dei raggi cosmici (come l'effetto Est-Ovest [1-2]). La comunicazione con i sensori di temperatura avviene tramite il protocollo 1-wire, mentre con l'accelerometro-magnetometro avviene attraverso un bus I2C. Il microcontrollore, non avendo un quarzo esterno, per generare il clock necessario per il suo funzionamento abilita il PLL interno che genera un clock con una frequenza massima di 32 MHz. Dopo che il PLL fornisce un clock stabile, il microcontrollore invia le informazioni di inizializzazione all'accelerometro-magnetometro e inizia il ciclo di acquisizione dati dai sensori (Fig. 3.45: Programma). Durante la fase di acquisizione il microcontrollore interroga prima i tre sensori di temperatura (Fig. 3.45: Acquisizione della temperatura), poi l'accelerometro-magnetometro e, infine, calcola la media pesata, lo scostamento dei valori dell'accelerazione e del campo magnetico dall'ultimo invio di dati verso il microcontrollore principale. Il calcolo della media e lo scarto (Fig. 3.45) è stato introdotto perché in una finestra di conteggio di 5 s, i valori di accelerazione e campo magnetico possono avere grandi oscillazioni dovuti al beccheggio e alle oscillazioni della gondola del pallone per cui, facendo una media pesata e considerando il massimo e minimo scarto si può eseguire un analisi dati sul flusso dei raggi cosmici tenendo conto anche degli effetti dovuti al movimento del pallone.

I dati acquisiti dai sensori, la media e lo scarto sono inviati al microcontrollore principale mediante una comunicazione seriale, che viene attivata solo quando c'è una richiesta di interrupt esterno proveniente dal microcontrollore principale. Una volta che la richiesta di interrupt è stata eseguita vengono azzerati tutti i valori e il microcontrollore inizia nuovamente il ciclo di acquisizione (Fig. 3.45).

Di seguito verranno descritti i sensori di temperatura e l'accelerometro-magnetometro, in particolare per quest'ultimo si descriverà come ottenere il pitch, il roll e l'heading a partire dalla misura del campo magnetico e dall'accelerazione e come effettuare la calibrazione.

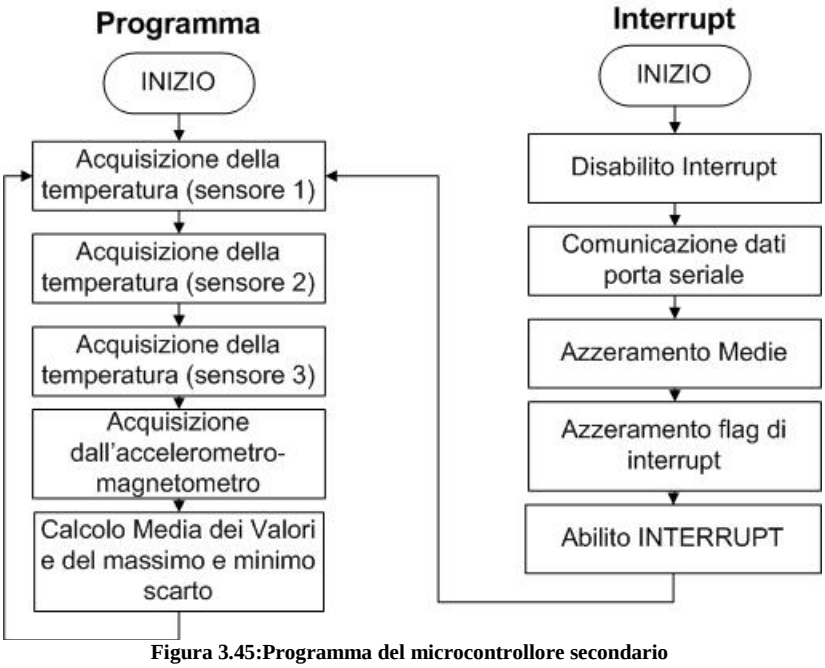


Figura 3.45:Programma del microcontrollore secondario

3.3.1 Sensore di temperatura

Il sensore di temperatura utilizzato è il DS18B20 [20] che come specificato nel Capitolo 1 è un sensore di tipo digitale che utilizza un protocollo proprietario denominato 1-wire. La comunicazione tra microcontrollore (unità master) e sensore (unità slave) comporta dei ritardi di elaborazione: il dispositivo master deve inviare un segnale di reset sulla linea dati che viene tenuta a livello logico basso per almeno 480 µs (per avvertire i dispositivi della presenza del master, Fig. 3.46), successivamente il microcontrollore imposta la linea come input e, un tempo compreso tra 15 µs e 60 µs, attende che il sensore invii l'impulso di presenza portando a livello logico basso la linea dati per un tempo tra 60 µs e 240 µs (Fig. 3.46).

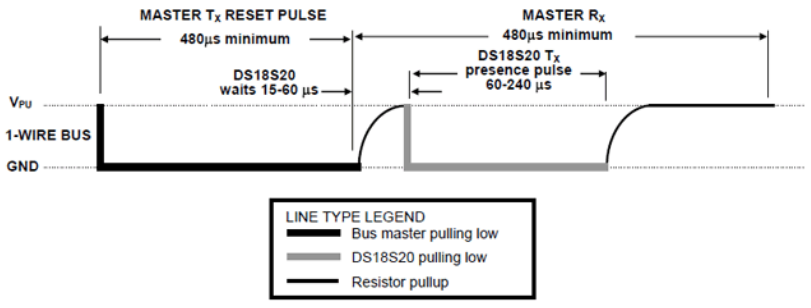


Figura 3.46:Tempo di interrogazione del sensore

La comunicazione tra master e slave avviene tramite una serie di Time Slots. Ogni slot in fase di scrittura e lettura va dai 60 ai 120 µs (Fig. 3.47) e tra due Time Slot consecutivi deve intercorrere un tempo di 1 µs (Fig. 3.47) in cui la linea viene tenuta a livello alto. Inoltre ogni slot, viene trasmesso dopo aver mantenuto la linea di comunicazione a livello logico basso per almeno 1 µs.

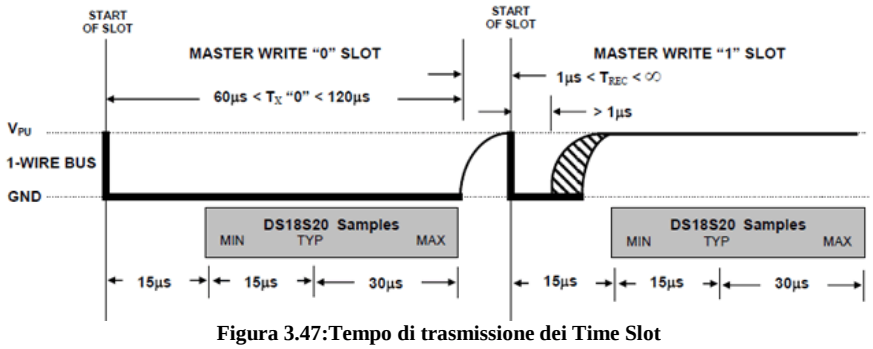


Figura 3.47:Tempo di trasmissione dei Time Slot

Sapendo che l'informazione della temperatura richiede 12 bit, per interrogare ogni sensore occorrono circa 800 µs.

In Figura 3.48 è riportato il codice del firmware del microcontrollore in cui si interroga il sensore (Fig. 3.48: *OWNIN*) e si effettua la conversione dei 12 bit in decimale (parte intera Fig. 3.48: *temp_int* e parte decimale Fig. 3.48: *temp_dec*)

```
OWOUT DQ1, 1, [$CC, $BE]
OWIN DQ1, 0, [temperature1.LOWBYTE, temperature1.HIGHBYTE, SKIP 4,
              count_remain, count_per_c]
 intero=0
IF  (temperature1.15) = 0 THEN
 segno="+"
ELSE
 segno="-"
 temperature1= ((NOT temperature1)+1)
ENDIF
temp_int = ((temperature1.HIGHBYTE << 4) + (temperature1.LOWBYTE >>4))
temp_dec = ((temperature1.LOWBYTE & %00001111)*625),10]
```

Figura 3.48:Comunicazione I2C

3.3.1 Accelerometro-Magnetometro

L'accelerometro-magnetometro utilizzato per rilevare il campo magnetico e l'accelerazione è LSM303DLHC [22]. Questo sensore ha tre assi sia per il campo magnetico che per l'accelerazione e un sensore di temperatura integrato.

Il sensore comunica con il microcontrollore mediante il protocollo di comunicazione I2C a 100 kHz o a 400 kHz. Inizialmente devono essere configurati

alcuni registri per abilitare e definire la sensibilità dell'accelerometro e del magnetometro. Per impostare questi registi occorre conoscere l'indirizzo che identifica il sensore, in particolare questo indirizzo è diversificato tra accelerometro e magnetometro e fra scrittura e lettura, come riportato in tabella (Tabella 7).

Tabella 7: Indirizzo sensore

Sensore	**Lettura**	**Scrittura**
Accelerometro	33h	32h
Magnetometro	3Dh	3Ch

Per poter utilizzare l'accelerometro si deve operare sui seguenti registri:

- *CTRL_REG1_A* (20h), impostato al valore 37h, permette di configurare il data rate dei dati e di abilitare i tre assi;
- *CTRL_REG2_A* (21h), impostato al valore 80h in modo da non eseguire nessun tipo di filtraggio sui dati.
- *CTRL_REG3_A* (22h), impostato al valore 00h per non generare nessun interrupt.
- *CTRL_REG4_A* (23h), il suo valore è 08h e permette di impostare il full-scale.

Per poter utilizzare il magnetometro si deve operare sui seguenti registri:

- *CRA_REG_M* (00h) impostato al valore 90h, definisce il data rate dei dati e abilita i sensore di temperatura.
- *CRB_REG_M* (01h), impostato al valore 80h per configurare il full scale del magnetometro.
- *MR_REG_M* (02h), impostato al valore 00h per avere una conversione continua.

I dati del campo magnetico e dell'accelerazione sono codificati in 12 bit e vengono letti dal microcontrollore e depositati in 12 registri, due per ogni asse, così suddivisi:

- *OUT_X_L_A* (28h), *OUT_X_H_A* (29h), accelerazione asse X;
- *OUT_Y_L_A* (2Ah), *OUT_Y_H_A* (2Bh), accelerazione asse Y;

- *OUT_Z_L_A* (2Ch), *OUT_Z_H_A* (2Dh), accelerazione asse Z;
- *OUT_X_H_M* (03), *OUT_X_LH_M* (04h),campo magnetico asse X;
- *OUT_Z_H_M* (05), *OUT_Z_L_M* (06h), campo magnetico asse Y;
- *OUT_Y_H_M* (07), *OUT_Y_L_M* (08h), campo magnetico asse Z;

Il microcontrollore, dopo aver letto queste informazioni, effettua la concatenazione dei due registri, e la conversione ed aggiorna il valore della media e dello scarto.

Per effettuare un analisi completa sul flusso dei raggi cosmici non servono i semplici valori di accelerazione e del campo magnetico ma quelli di pitch (ρ)[3], roll (γ)[3] e heading (ψ)[3] (Fig. 3.49), in particolare si utilizza l'accelerometro per calcolare i primi due e il magnetometro per calcolare l'heading.

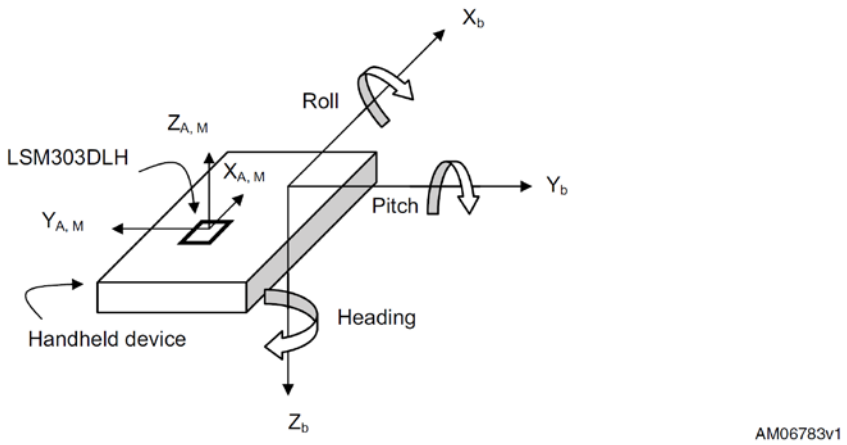


Figura 3. 49:Pitch, Roll e Heading

Per calcolare il pitch e roll si deve partire da una posizione arbitraria nello spazio (X_b', Y_b', Z_b') e attraverso alcune rotazioni il dispositivo ritorna solidale con il sistema di riferimento (X_b, Y_b, Z_b), Risolvendo i sistemi ottenuti dalle rotazione si ottiene che [12-Appendice A]:

$$\rho = \arcsin(-A_{x1}) \quad (A_{x1} \text{ è il valore misurato dal sensore calibrato})$$

[3] Pitch è l'angolo tra l'asse X_b e il piano orizzontale.

Roll è l'angolo tra l'asse Y_b e il piano orizzontale.

Heading è l'angolo tra l'asse X_b e la direzione del nord magnetico sul piano orizzontale.

$$\gamma = \arcsin(\frac{A_{y1}}{\rho})$$

Per calcolare l'Heading si deve ripetere lo stesso ragionamento ρ eseguito per il pitch e roll e si ottiene[42-Appendice A]:

$$M_{x2} = M_{x1}\cos\rho + M_{z1}\sin\rho$$

$$M_{y2} = M_{x1}\sin\rho\sin\gamma + M_{y1}\cos\gamma - M_{z1}\cos\rho\sin\gamma$$

$$M_{z2} = -M_{x1}\sin\rho\cos\gamma + M_{y1}\sin\gamma - M_{z1}\cos\rho\cos\gamma$$

$$\psi = \arctan\left(\frac{M_{y2}}{M_{x2}}\right) \text{ per } M_{x2} > 0 \; e \; M_{y2} \geq 0$$

$$= 180° + \arctan\left(\frac{M_{y2}}{M_{x2}}\right) \text{ per } M_{x2} < 0$$

M_{x1}, M_{y1}, M_{z1} sono i valori calibrati del sensore

Il sensore una volta montato sul PCB soffre intrinsecamente di tre problemi:

- Interferenza Hard-iron. È un campo magnetico normalmente generato da materiale ferromagnetico di cui è composto il PCB e il package del dispositivo che si sovrappone al campo magnetico terreste.
- Interferenza Soft-iron. È un campo magnetico normalmente generato all'interno del device o dalle piste del PCB in cui scorre la corrente.
- Errore di fattore di scala. È l'errore causato dalla differenza delle sensibilità dei sensori magnetici di ciascun asse;

Per eliminare questi tre effetti occorre applicare delle tecniche di calibrazione numeriche (metodo dei minimi quadrati).

Il sensore, inizialmente, ha una calibrazione di fabbrica e per averne una adatta alla particolare circostanza, occorre effettuare delle calibrazioni post-processor. Per eseguire la calibrazione dell'accelerometro occorre determinare i 12 parametri da ACC10 a ACC33 (Fig. 3.50) tramite delle misure eseguite nelle 6 diverse posizioni riportate in Figura 3.51.

$$\begin{bmatrix} A_{x1} \\ A_{y1} \\ A_{z1} \end{bmatrix} = [A\_m]_{3x3} \begin{bmatrix} 1/A\_SC_x & 0 & 0 \\ 0 & 1/A\_SC_y & 0 \\ 0 & 0 & 1/A\_SC_z \end{bmatrix} \cdot \begin{bmatrix} A_x - A\_OS_x \\ A_y - A\_OS_y \\ A_z - A\_OS_z \end{bmatrix}$$
$$= \begin{bmatrix} ACC_{11} & ACC_{12} & ACC_{13} \\ ACC_{21} & ACC_{22} & ACC_{23} \\ ACC_{31} & ACC_{32} & ACC_{33} \end{bmatrix} \cdot \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} + \begin{bmatrix} ACC_{10} \\ ACC_{20} \\ ACC_{30} \end{bmatrix}$$

Figura 3.50:Equazione di calibrazione dell'accelerometro

Stationary position	Accelerometer (signed integer)			Magnetic sensor (signed integer)		
	A_x	A_y	A_z	M_x	M_y	M_z
Z_b down	0	0	+1g	+ or -	+ or -	+
Z_b up	0	0	-1g	+ or -	+ or -	-
Y_b down	0	+1g	0	+ or -	+	+ or -
Y_b up	0	-1g	0	+ or -	-	+ or -
X_b down	+1g	0	0	+	+ or -	+ or -
X_b up	-1g	0	0	-	+ or -	+ or -

Figura 3.51:Tabella delle posizioni di calibrazione

Misurando i diversi valori delle accelerazioni per le 6 posizioni e successivamente risolvendo il sistema matriciale Y= w·X in cui:

- Y è la matrice della forza gravità terrestre normalizzata (questa matrice è già popolata);
- w è la matrice contenente i valori misurata dai sensori;
- X è la matrice che contiene i 12 parametri da determinare.

I parametri di calibrazione possono essere determinati mediante il metodo least square o dei minimi quadrati ottenendo:

$$X = [w^T \cdot w]^{-1} \cdot w^T \cdot Y$$

Per eseguire la calibrazione del magnetometro occorre determinare i 12 parametri da MR10 a MR33 (Fig. 3.52) tramite delle misure eseguite in 6 diverse posizioni riportate in Figura 3.51.

$$\begin{bmatrix} M_{x1} \\ M_{y1} \\ M_{z1} \end{bmatrix} = [M\_m]_{3x3} \begin{bmatrix} 1/M\_SC_x & 0 & 0 \\ 0 & 1/M\_SC_y & 0 \\ 0 & 0 & 1/M\_SC_z \end{bmatrix} \cdot [M\_si]_{3x3} \begin{bmatrix} M_x - M\_OS_x \\ M_y - M\_OS_y \\ M_z - M\_OS_z \end{bmatrix}$$
$$= \begin{bmatrix} MR_{11} & MR_{12} & MR_{13} \\ MR_{21} & MR_{22} & MR_{23} \\ MR_{31} & MR_{32} & MR_{33} \end{bmatrix} \cdot \begin{bmatrix} M_x - MR_{10} \\ M_y - MR_{20} \\ M_z - MR_{30} \end{bmatrix}$$

Figura 3.52:Equazione di calibrazione del magnetometro

Ripetendo lo stesso procedimento eseguito per l'accelerometro si ottiene

$$X = [H^T \cdot H]^{-1} \cdot H^T \cdot w$$

La calibrazione non può essere eseguita con il microcontrollore perché non può svolgere operazioni in virgola mobile per cui deve essere effettuata all'inizio, dopo il montaggio, ed usare le matrici trovate come offset durante l'analisi dati.

3.5 PCB

Dopo aver sviluppato il firmware del DAQ si è proceduto allo sbroglio del circuito mediante il software Eagle della CadSoft. Il DAQ è formato da due circuiti speculari che comunicano tra di loro e che ci permette di avere la ridondanza totale in caso di guasti o errori. Il PCB è formato da quattro strati, lo strato superiore (Fig. 3.53) contiene tutti i componenti e i loro collegamenti; nel secondo strato (Fig. 3.54) si ha il collegamento tra il circuito master e lo slave che occorrono per implementare i meccanismi di watchdog; il terzo strato, (Fig. 3.55) predisposto per l'alimentazione di 3.3 V, ha dei piani di alimentazione separati per il circuito master e lo slave mentre nel quarto strato (Fig. 3.56) sono ubicati alcuni componenti e tracce di collegamento.

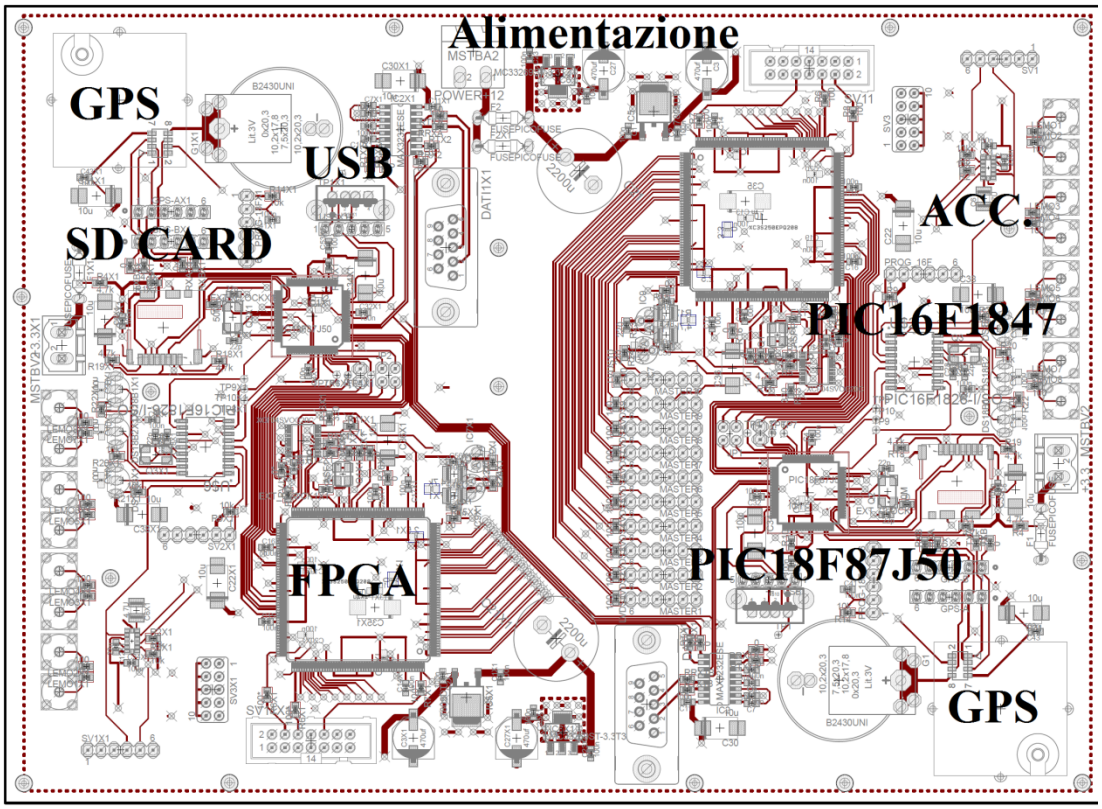


Figura 3.53:Primo strato

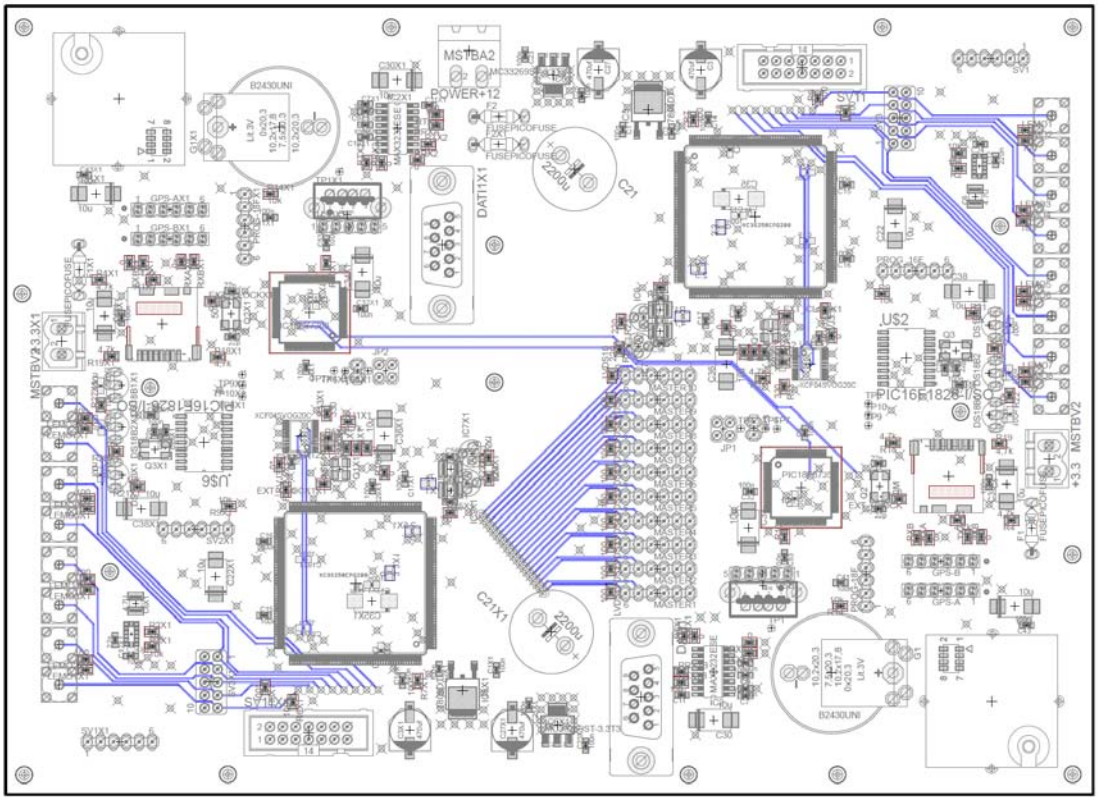


Figura 3.54:Secondo Strato

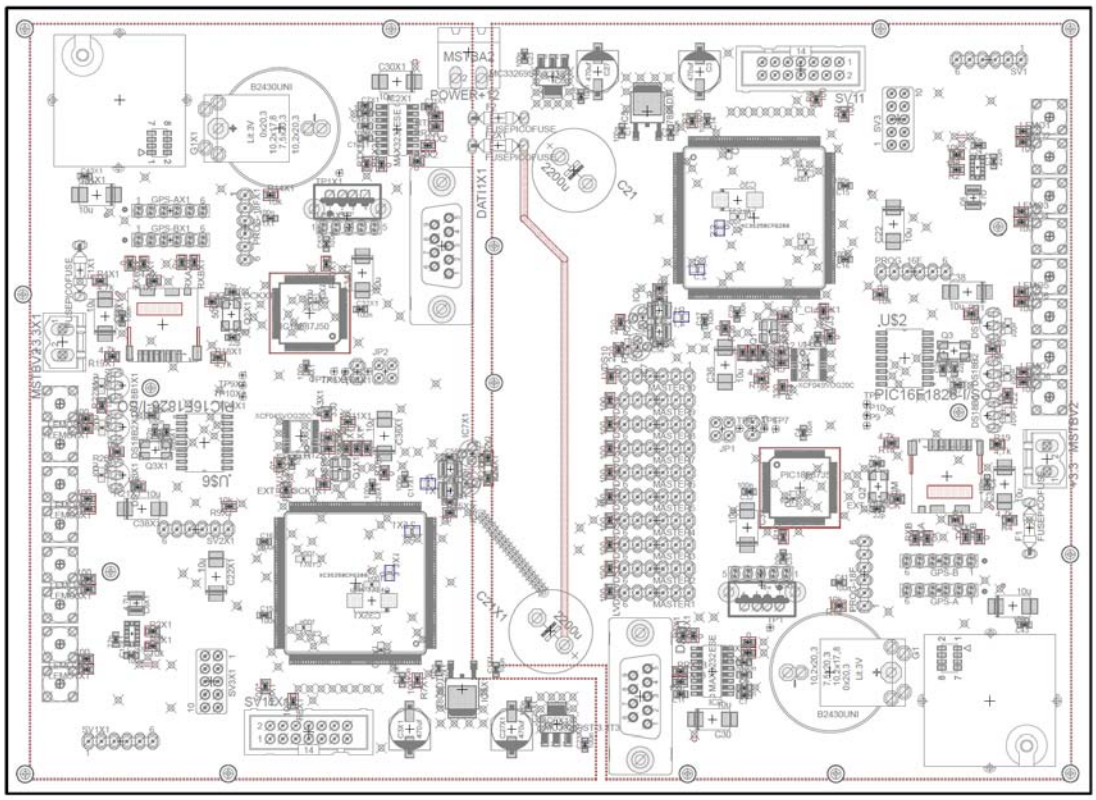


Figura 3.55:Terzo strato

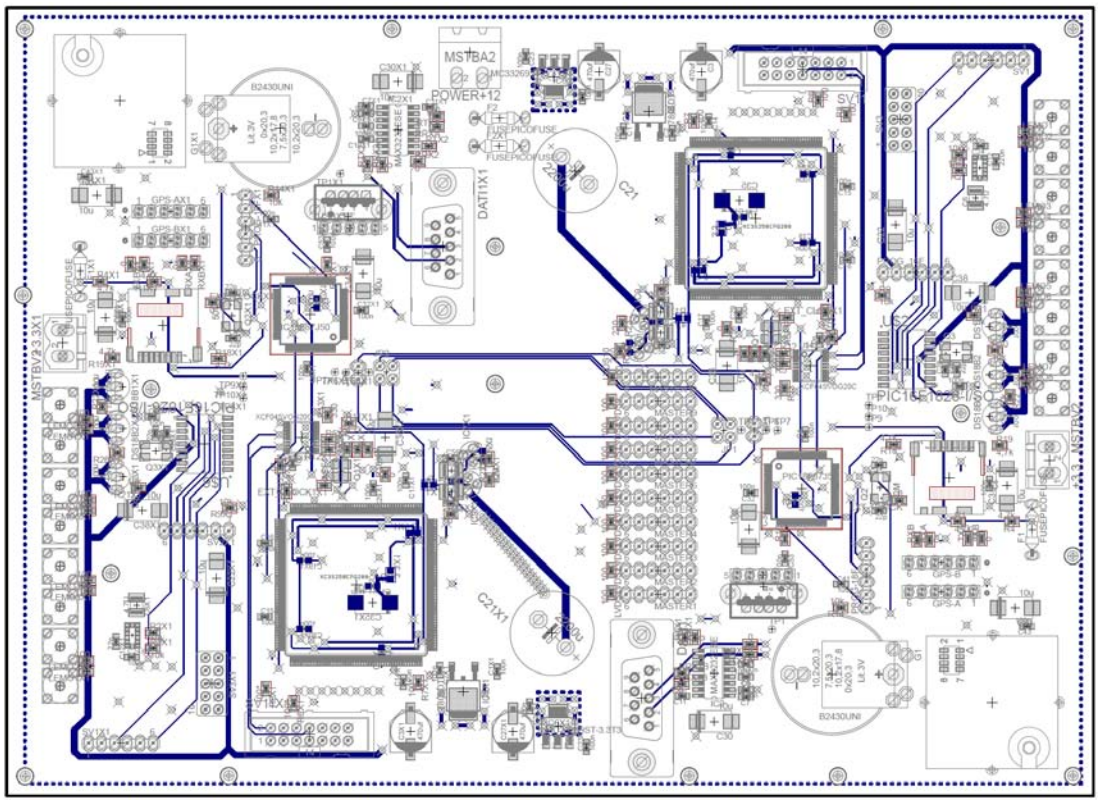


Figura 3.56:Quarto strato

Il circuito stampato ottenuto è raffigurato in Figura 3.57 il Top e in Figura 3.58 il Bottom.

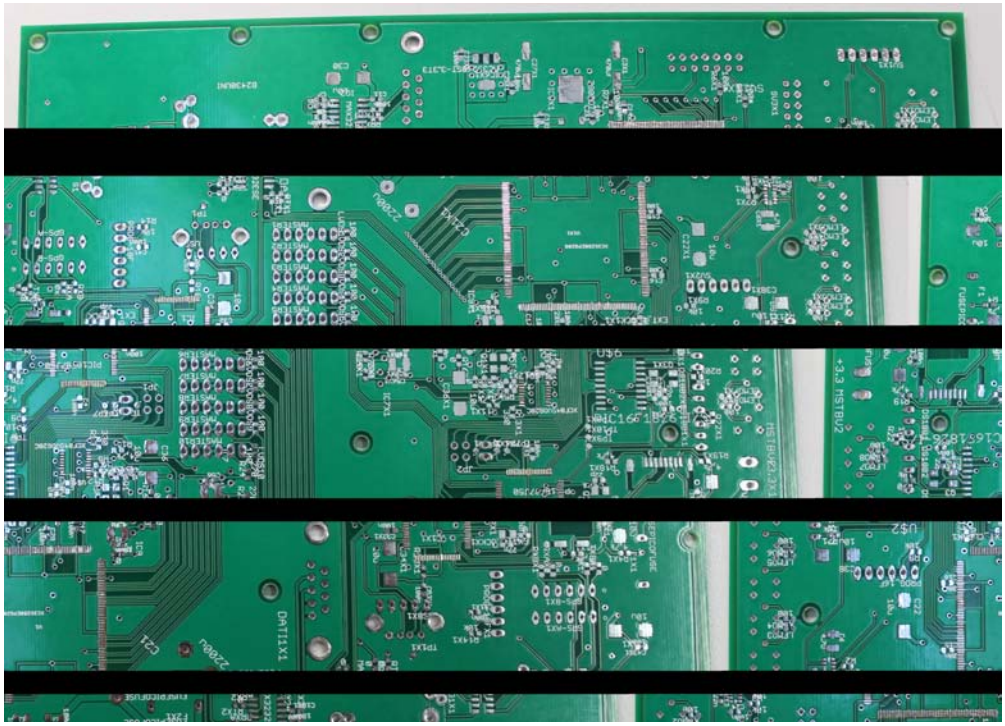


Figura 3.57: Top

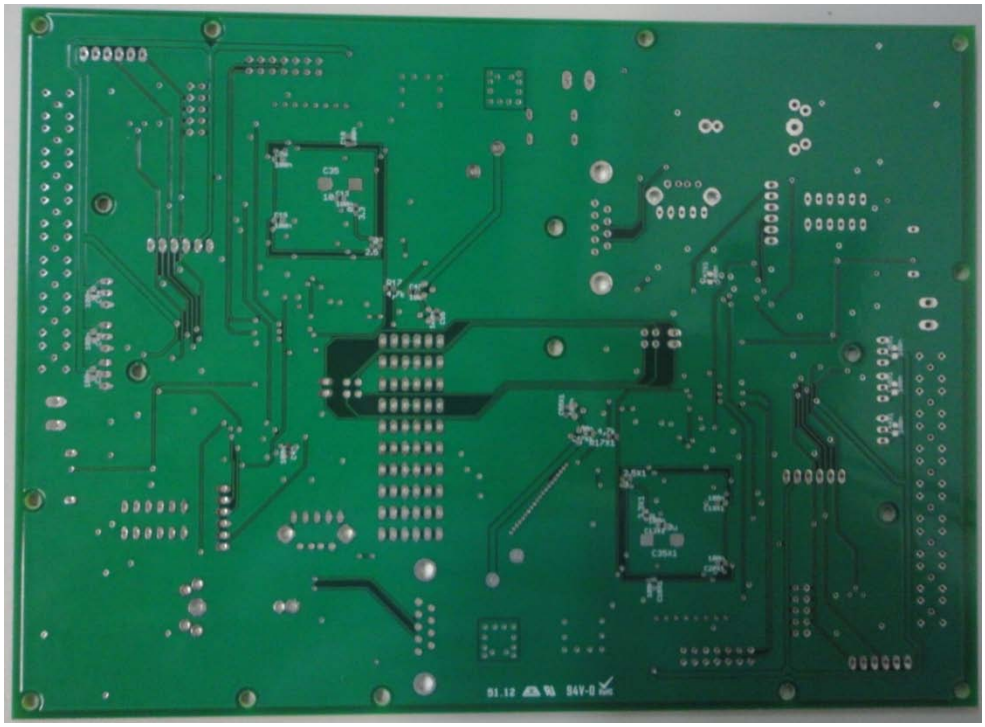


Figura 3.58: Bottom

In fase di popolamento del PCB si è testato il funzionamento dei singoli componenti e si è notato che, alimentando il microcontrollore secondario con una tensione di 5 V, il piano di alimentazione di 3.3 V manifestava una sovratensione di 1.4 V dovuta alla comunicazione tra microcontrollore e sensore che poteva provocare il danneggiamento dell'FPGA. L'FPGA può funzionare con una tensione massima pari a 3.6 V, perciò si è deciso si sostituire il regolatore di tensione da 5 V con un regolatore da 3.3 V, cambiamento che non comporta problemi al sistema in quanto tutti i componenti che inizialmente venivano alimentatati a 5 V possono tranquillamente lavorare a 3.3 V, evitando così la sovratensione sul piano di alimentazione.

Figura 3. 59: Circuito completo

Dopo aver popolato il circuito (Fig. 3.59) si è proceduto con il valutare, tramite termocamera, come i due regolatori di tensione disperdevano il calore sul PCB, dispersione che, per poter permettere ai vari componenti di funzionare in maniera adeguata anche a temperature ambientali basse, doveva essere il più uniforme possibile per consentire un auto riscaldamento del PCB durante la fase di volo.

In Figura 3.60 è raffigurato il comportamento del solo regolatore 3,3 V con un carico resistivo basso, invece in Figura 3.61 è riportata la dispersione di calore

durante un set di misure effettuato in laboratorio. Come si può notare sia l'FPGA che il microcontrollore principale operano ad una temperatura di circa 35°C che permette un funzionamento ottimale del dispositivo anche durante la fase di volo, la zona più calda del PCB è dove sono posizionati i regolatori da 3.3 V, 2.5 V e 1.2 V.

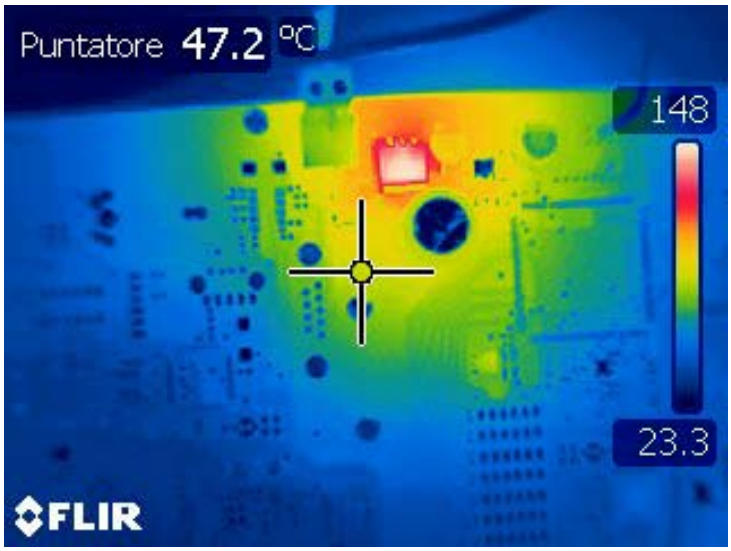


Figura 3.60:Dissipazione con il regolatore di tensione da 3.3 V

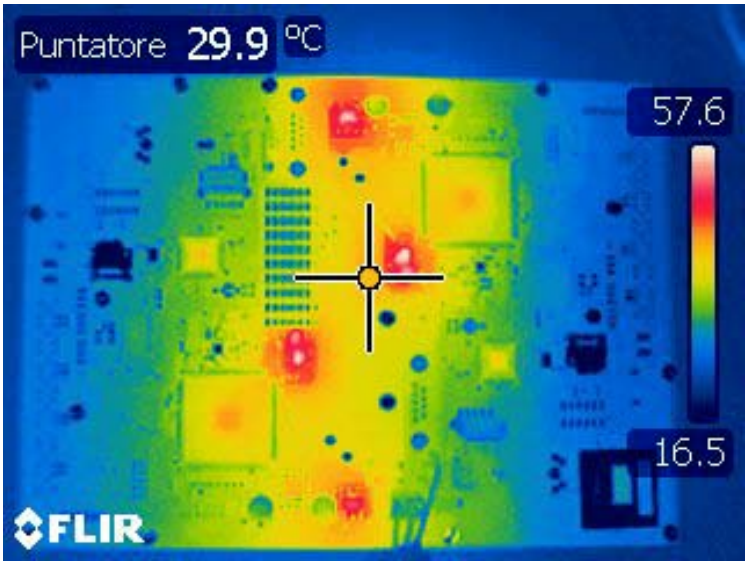


Figura 3.61:Dissipazione del circuito Master

CONCLUSIONE E SVILUPPI FUTURI

La prima versione del DAQ denominata prototipo, descritta nel Capitolo 2, utilizza un rilevatore composto da quattro scintillatori su cui si sono eseguiti i primi test del firmware dell'FPGA e del microcontrollore, oltre alle misure sul Gran Sasso e durante l'International Cosmic Day 2012 [43]. I risultati delle misure eseguite sul Gran Sasso sono stati presentati al workshop SCINEGHE 2012 e pubblicate su Nuclear Physics B. La seconda versione, quella definitiva del DAQ, cioè quella prevista per l'allocazione sul pallone aerostatico, descritto nel Capitolo 1 e 3, utilizza un rilevatore composto da otto scintillatori e i risultati prodotti da diversi test effettuati in laboratorio sono consistenti con le specifiche del progetto.

Nel DAQ l'FPGA verifica le coincidenze tra i vari scintillatori. Il microcontrollore principale, ricevuti i valori delle coincidenze, effettua la geolocalizzazione del dispositivo e il timestamping dell'evento tramite un GPS, riceve i dati dai sensori del microcontrollore secondario, salva su SD CARD e invia tutte le informazioni alla telemetria del pallone mediante porta seriale o ad un PC mediante porta USB. Il microcontrollore secondario, dopo aver elaborato le informazioni ricevute dai sensori, comunica con quello principale mediante una porta seriale software. Nel DAQ è prevista inoltre la ridondanza dei componenti in maniera tale che durante il volo il DAQ continui a inviare informazioni anche nel caso di guasto dei componenti.

Gli sviluppi futuri non riguardano l'elettronica del DAQ o il firmware ma il completamento del programma d'interfaccia in LabView, programma che permette di effettuare l'analisi dei dati del rilevatore con otto scintillatori ed elaborare le informazioni dell'accelerometro-magnetometro per calcolare il pitch, il roll e l'heading.

APPENDICE A

Firmware del prototipo del DAQ

In questa appendice sono riportate le parti principali del firmware del microcontrollore e dell'FPGA.

Nel firmware del microcontrollore si riporta l'abilitazione del PLL interno:

```
OSCTUNE.6 = 1     ' Enable PLL for 18F87J50 family
PAUSE 20
DEFINE OSC 48
ANCON1_alt.0=1
ANCON1_alt.3=1
```

Di seguito, la parte di codice in cui si controlla se l'SD CARD è presente e la si inizializza verificando, successivamente, l'esistenza del file per la creazione dei nomi dei file che, altrimenti, crea.

```
IF (SD_CD = 0) THEN ' Controlla presenza SD CARD
  GOSUB FSInit ' Inizializzza SD CARD
  IF (FAT_error != 0) THEN GOTO errore
  error=0
  sdcard_pre = 1
  trov=0
  GOSUB FINDfirst '' Cerca il file dei nomi "filefile.txt"
  WHILE (FAT_error=0 AND trov=0)
    IF (FAT_FileName[0] = "f" AND FAT_FileName[1] = "i" AND
        FAT_FileName[2] = "l" AND FAT_FileName[3] = "e" AND
        FAT_FileName[4] = "f" AND FAT_FileName[5] = "i" AND
        FAT_FileName[6] = "l" AND FAT_FileName[7] = "e") THEN
              trov = 1
    ENDIF
    GOSUB FINDnext
  WEND
  'Cerca Se il file non è stato trovato lo crea e scrive "A"
  IF (trov=0) THEN
    FAT_FileName[0] = "f"
    FAT_FileName[1] = "i"
    FAT_FileName[2] = "l"
    FAT_FileName[3] = "e"
    FAT_FileName[4] = "f"
```

```
    FAT_FileName[5] = "i"
    FAT_FileName[6] = "l"
    FAT_FileName[7] = "e"
    FAT_FileName[8] = "t"
    FAT_FileName[9] = "x"
    FAT_FileName[10] = "t"
    FAT_mode = "w"
    GOSUB FSfopen
    HSEROUT2 ["Open: ", DEC FAT_error, $d, $a]
    IF (FAT_error != 0) THEN GOTO errore
    error=0
    FAT_src[0] = "A"
    FAT_count = 1
    GOSUB FSfwrite
    HSEROUT2 [ "Write ", DEC FAT_error, $d, $a]
    IF (FAT_error != 0) THEN GOTO errore
    GOSUB FSfclose
    HSEROUT2 [ "Close ", DEC FAT_error, $d, $a]
    IF (FAT_error != 0) THEN GOTO errore
  ENDIF
ELSE
  sdcard_pre = 0
ENDIF
```

Nel codice seguente si riporta la parte del firmware in cui si ricevano le informazioni dal GPS per definire il nome del file destinato alla scrittura e lo si crea.

```
' Informazioni dal GPS
HSERIN [WAIT("$GPGGA,"),STR TIME\6,SKIP 4,STR LATITUDE\9,_
        SKIP 1,STR PCLAT1\1,SKIP 1,STR LONGITUDE\10,_
        SKIP 1,STR PCLON\1,SKIP 3 , STR SATELLITE\2,_
        SKIP 6,STR ALTITUDE\5]
HSERIN [WAIT("$GPRMC,"),SKIP 49,STR DATE\4]
IF (sdcard_pre = 1) THEN
    ' Controllo numero dei satelliti, se il numero è inferiore
    ' a 4 il nome del file è fornito dal microcontrollore
    ' altrimenti si utilizzano le informazioni dal GPS
    IF (satellite[1] < "4") THEN
      FAT_FileName[0] = "f"
      FAT_FileName[1] = "i"
      FAT_FileName[2] = "l"
      FAT_FileName[3] = "e"
      FAT_FileName[4] = "f"
      FAT_FileName[5] = "i"
      FAT_FileName[6] = "l"
      FAT_FileName[7] = "e"
      FAT_FileName[8] = "T"
      FAT_FileName[9] = "X"
      FAT_FileName[10] = "T"
      FAT_mode = "r"
      GOSUB FSfopen
      IF (FAT_error != 0) THEN GOTO errore
      error=0
```

```
            FAT_count = 1           ' Read 1 byte to buffer at a time
            GOSUB FSfread
            'Legge il nome del file dal "filefile.txt"
            name = FAT_dest[0]
            GOSUB FSfclose
            IF (FAT_error != 0) THEN GOTO errore
            FAT_mode = "w"
            GOSUB FSfopen
            IF (FAT_error != 0) THEN GOTO errore
            FAT_src[0] = name+1
            FAT_count = 1
            GOSUB FSfwrite
            GOSUB FSfclose
            IF (FAT_error != 0) THEN GOTO errore
            ' Crea il file con il nome letto dal file presente
            ' sull'SD CARD
            FAT_FileName[0] = name
            FAT_FileName[1] = " "
            FAT_FileName[2] = " "
            FAT_FileName[3] = " "
            FAT_FileName[4] = " "
            FAT_FileName[5] = " "
            FAT_FileName[6] = " "
            FAT_FileName[7] = " "
            FAT_FileName[8] = "T"
            FAT_FileName[9] = "X"
            FAT_FileName[10] = "T"
            IF (FAT_error != 0) THEN GOSUB errore
            FAT_mode = "w"
            GOSUB FSfopen
            IF (FAT_error != 0) THEN GOTO errore
            error=0
            GOSUB FSfclose
          ELSE
            ' nome del file dalle informazioni del GPS
            FAT_FileName[0] = TIME[0]
            FAT_FileName[1] = TIME[1]
            FAT_FileName[2] = TIME[2]
            FAT_FileName[3] = TIME[3]
            FAT_FileName[4] = TIME[4]
            FAT_FileName[5] = TIME[5]
            FAT_FileName[6] = date[0]
            FAT_FileName[7] = date[1]
            FAT_FileName[8] = "T"
            FAT_FileName[9] = "X"
            FAT_FileName[10] = "T"
          ENDIF
        ENDIF
```

Si riporta la parte del codice attraverso cui si le informazioni sulla finestra di conteggio da PC o dal sottosistema di telemetria.

```
'se la USB è presente legge il buffer USB altrimenti legge la
```

```
'porta seriale GPS
IF (pres=1) THEN
   GOSUB DoUSBIn
   com=USBBuffer[0]
ELSE
   HSERIN2 5,cont,[STR com\1]
   pres=0
ENDIF
```

Infine si riporta il codice in cui avviene la scrittura su SD CARD e/o il trasferimento dei dati mediante la porta USB.

```
FAT_count = 505
' se l'SD CARD è presente effettua la scrittura
IF (SD_CD=0 AND error=0) THEN
    IF (sdcard_pre = 0) THEN GOTO main
    FAT_mode = "a"            ' modalità di scrittura sul file
    GOSUB FSfopen       ' apre il file
    IF (FAT_error != 0) THEN GOTO errore
    error=0
    GOSUB FSfwrite
    IF (FAT_error != 0) THEN GOTO errore
    GOSUB FSfclose
    IF (FAT_error != 0) THEN GOTO errore
ENDIF
' invio dati alla porta Seriale
HSEROUT2 [STR fat_src\505]
GOSUB usbiniz 'inizializzazione della porta USB
IF (pres=1) THEN
' scrittura del buffer USB
    ARRAYWRITE USBBuffer,[$d, $a]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[HEX8 DATO[14],HEX8 DATO[13],_
                  HEX8 DATO[12],HEX8 DATO[11],HEX8 DATO[10],_
                  HEX8 DATO[9],HEX8 DATO[8],HEX8 DATO[7]]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[STR fat_src[64]\64]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[STR fat_src[128]\64]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[STR fat_src[192]\64]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[STR fat_src[256]\64]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[STR fat_src[320]\64]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[STR fat_src[384]\64]
    IF (ucon.1 = 0) THEN GOSUB DoUSBOut
    ARRAYWRITE USBBuffer,[STR fat_src[448]\56]
    IF (ucon.1 = 0) THEN  GOSUB DoUSBOut
ELSE
    IF (ucon.1 = 0) THEN USBINIT
```

```
ENDIF
```

Dopo alcune parti del firmware del microcontrollore si riportono alcune sezioni del firmware dell'FPGA con l'UCF[1]. Inizialmente il file UCF:

```
#clock con timing constraint
NET "clk" LOC = P82 | IOSTANDARD = LVCMOS33;
NET "clk" TNM_NET = clk;
NET "clk" PERIOD = 25.0ns HIGH 50%;
TIMESPEC TS_clk = PERIOD "clk" 25 ns HIGH 50%;
NET "clko" LOC = P47 | IOSTANDARD = LVTTL;
#gestione della finestra di conteggio
NET "mod_temp<0>" LOC = P109 | IOSTANDARD = LVTTL | PULLUP;
NET "mod_temp<1>" LOC = P108 | IOSTANDARD = LVTTL | PULLUP;
NET "mod_temp<2>" LOC = P107 | IOSTANDARD = LVTTL | PULLUP;
NET "clk_trasf" LOC =P135 | IOSTANDARD = LVTTL;
NET "clk_trasf" CLOCK_DEDICATED_ROUTE = FALSE;
PIN "U1/DCM_SP_INST.CLKIN" CLOCK_DEDICATED_ROUTE = FALSE;
#ingressi LVDS dei scintillatori
NET "Tx_p" LOC=P2|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tx_n" LOC=P3|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Ty_p" LOC=P8|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Ty_n" LOC=P9|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tz_p" LOC=P11|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tz_n" LOC=P12|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tw_p" LOC=P15|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tw_n" LOC=P16|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tq_p" LOC=P18|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tq_n" LOC=P19|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tr_p" LOC=P33|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Tr_n" LOC=P34|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Ts_p" LOC=P35|IOSTANDARD="LVDS_25"|SLEW = SLOW |DRIVE = 8;
NET "Ts_n" LOC = P36|IOSTANDARD="LVDS_25"|SLEW= SLOW |DRIVE = 8;
NET "Tt_p" LOC = P39|IOSTANDARD="LVDS_25"|SLEW= SLOW |DRIVE = 8;
NET "Tt_n" LOC = P40|IOSTANDARD="LVDS_25"|SLEW= SLOW |DRIVE = 8;
#uscite
NET "oenable_trasf" LOC = P140 | IOSTANDARD = LVTTL;
NET "odati" LOC = P138 | IOSTANDARD = LVTTL;
NET "singola" LOC = P160 | IOSTANDARD = LVTTL;
NET "doppia" LOC = P161 | IOSTANDARD = LVTTL;
NET "tripla" LOC = P162 | IOSTANDARD = LVTTL;
NET "quadrupla" LOC = P163 | IOSTANDARD = LVTTL;
```

Di seguito la parte del codice in cui viene dichiarata la matrice utilizzata per eseguire l'incremento dei contatori, alla fine della finestra di ascolto.

---

[1] UCF (User Constraint File) è il file che associa agli ingressi e uscite utilizzate nel firmware i pin dell'FPGA.

```
--matrice di incremeneto
type matrix is array (0 to 15, 0 to 14) of STD_LOGIC;
constant matr :matrix := ( ('0','0','0','0','0','0','0','0','0','0','0','0','0','0','0'),
                           ('1','0','0','0','0','0','0','0','0','0','0','0','0','0','0'),
                           ('0','1','0','0','0','0','0','0','0','0','0','0','0','0','0'),
                           ('1','1','0','0','1','0','0','0','0','0','0','0','0','0','0'),
                           ('0','0','1','0','0','0','0','0','0','0','0','0','0','0','0'),
                           ('1','0','1','0','0','1','0','0','0','0','0','0','0','0','0'),
                           ('0','1','1','0','0','0','0','1','0','0','0','0','0','0','0'),
                           ('1','1','1','0','1','1','0','1','0','0','1','0','0','0','0'),
                           ('0','0','0','1','0','0','0','0','0','0','0','0','0','0','0'),
                           ('1','0','0','1','0','0','1','0','0','0','0','0','0','0','0'),
                           ('0','1','0','1','0','0','0','0','1','0','0','0','0','0','0'),
                           ('1','1','0','1','1','0','1','0','1','0','0','1','0','0','0'),
                           ('0','0','1','1','0','0','0','0','0','1','0','0','0','0','0'),
                           ('1','0','1','1','0','1','1','0','0','1','0','0','1','0','0'),
                           ('0','1','1','1','0','0','0','1','1','1','0','0','0','1','0'),
                           ('1','1','1','1','1','1','1','1','1','1','1','1','1','1','1'));
```

Il codice sottostante riporta l'incremento dei contatori alla fine della finestra d'ascolto attraverso l'utilizzo della matrice descritta precedentemente.

```
--ad ogli ciclo di clock e quando uno dei scintillatori
--a rilevato una particella
if ((clk'event  and clk = '1')
  and (scx = 1 or scy = 1 or scz = 1 or scw = 1)) then
   i:=i+1;
   --alla fine della finestra di conteggio eseguo l'incremento
   if ( i = 5 or win_temp = '1') then
      index:=scx+2*scy+4*scz+8*scw;
      cx <= cx + matr(index,0);
      cy <= cy + matr(index,1);
      cz <= cz + matr(index,2);
      cw <= cw + matr(index,3);
      cxy <= cxy + matr(index,4);
      cxz <= cxz + matr(index,5);
      cxw <= cxw + matr(index,6);
      cyz <= cyz + matr(index,7);
      cyw <= cyw + matr(index,8);
      czw <= czw + matr(index,9);
      cxyz <= cxyz + matr(index,10);
      cxyw <= cxyw + matr(index,11);
      cxzw <= cxzw + matr(index,12);
      cyzw <= cyzw + matr(index,13);
      cxyzw <= cxyzw + matr(index,14);
      i:=0;
      index_tri<=index;
      azzera <= '1';
   end if;
end if;
```

Infine si riportano i processi utilizzati per il trasferimento dei dati, il primo *costfifo* permette la creazione della coda, il secondo *transfer* esegue il trasferimento dei dati verso il microcontrollore.

```
-- alla fine della finestra di conteggio si crea la coda di
-- trasferimento e pongo a '1' il segnale di enable
costfifo: process (win_temp,stop_trasf,atzero1)
    begin
      if (stop_trasf = '1') then
         codafifo <= (others =>'0');
         enable_trasf <= '0';
      elsif (win_temp'event  and win_temp = '0')  then
         --creazione della coda
         codafifo<= cx & cy & cz & cw & cxy & cxz & cxw & cyz
                   & cyw & czw & cxyz & cxzw & cxyw & cyzw & cxyzw;
         atzero <='1';
         enable_trasf <='1';
      end if;
      if (atzero1 = '1') then
         atzero <= '0';
      end if;
   end process;

--processo di trasferimento
transfer: process (clk_trasf,enable_trasf)
      variable i : integer := 0;
      variable dati: std_logic := '0';
   begin
     if (enable_trasf = '1') then
         --aspetto il fronte del clock del microcontrollore
         if(clk_trasf'event  and clk_trasf = '1')  then
            dati := codafifo(i);
            i:=i+1;
            --alla fine della coda
            if (i = 479) then
               stop_trasf <= '1';
               dati := '0';
               i:=0;
            end if;
         end if;
      else
         stop_trasf <= '0';
      end if;
      odati <= dati;
   end process;
```

APPENDICE B

Firmware del DAQ

In questa appendice sono riportate alcune parti del firmware del DAQ, in particolare del microcontrollore principale e di quello secondario e dell'FPGA. Del firmware del microcontrollore principale si riporta il codice inerente la comunicazione con il microcontrollore secondario, in quanto è l'unica parte aggiunta rispetto al firmware del microcontrollore del prototipo.

```
HIGH int
LOW int
SERIN2 PORTJ.2,84,1,program,[WAIT("="),STR sensor\342]
```

Del firmware del microcontrollore secondario si riportano diverse parti; di seguito il codice che permette l'abilitazione del quarzo interno e la configurazione delle porte:

```
OSCCON.0=0 ' abilito quarzo interno
OSCCON.1=0
OSCCON.3=0
OSCCON.4=1
OSCCON.5=1
OSCCON.6=1
OSCCON.7=1
PAUSE 10
WPUB = 255
CPSCON0.7=0
APFCON1.0=1 ' Definisce la funzione della porta
APFCON0.7=1
```

e successivamente il codice che permette la definizione delle impostazioni dell'accelerometro-magnetometro:

```
'Definisco i parametri dall'accelerometro-magnetometro
I2CWRITE SDA,SCL,$32,$20,[$37],bogus
```

```
I2CWRITE SDA,SCL,$32,$21,[$80]
I2CWRITE SDA,SCL,$32,$23,[$08]
I2CWRITE SDA,SCL,$3C,$00,[$90]
I2CWRITE SDA,SCL,$3C,$02,[$00]
```

Il codice che permette, al microcontrollore secondario, di comunicare con i sensori di temperatura è il seguente:

```
'Interrogo il sensore di temperatura
lett:
   OWOUT  DQ,  1,  [$CC,  $44]            '  Start  temperature
conversion
waitloop:
   OWIN DQ, 4, [count_remain] ' Check for still busy converting
   IF count_remain = 0 THEN waitloop
   OWOUT DQ, 1, [$CC, $BE]          ' Read the temperature
   OWIN DQ, 0, [temperature.LOWBYTE,temperature.HIGHBYTE,SKIP_
                4, count_remain, count_per_c]
```

Di seguito quello che permette di comunicare con l'accelerometro-magnetometro leggendo i diversi valori dei 6 assi; se ne riporta solo la parte riguardante la lettura del solo *asse x* e il calcolo della media pesata.

```
'Leggo dati dall'accelerometro asse x
I2CREAD SDA,SCL,$33,$28,[a],bogus
I2CREAD SDA,SCL,$33,$29,[b],bogus
ACCX = ((b<<8) + a)
.............
'calcolo la media pesato dell'accelerazione per l'asse x
dummy = (m_accx * cont)
m_accx = DIV32 (cont+1)
m_accx = m_accx + (accx/(cont+1))
```

Infine, si riporta la procedura di interrupt tramite la quale avviene la comunicazione con il microcontrollore principale,

```
 'interrupt di comunicazione
 DISABLE  'disabilito interrupt
 comm_serial:
    'invio dati tramite seriale
    HSEROUT ["=",segno,DEC ((temperature.HIGHBYTE << 4) +_
 (temperature.LOWBYTE >>4)),",",DEC2((temperature.LOWBYTE_
 &%00001111)*625),segno,DEC ((temperature1.HIGHBYTE << 4)+_
 (temperature1.LOWBYTE >>4)),",",DEC2((temperature1.LOWBYTE_
 &%00001111)*625),HEX4 m_accx,HEX4 m_accy,HEX4 m_accz,_
```

```
HEX4 m_magx,HEX4 m_magy,HEX4 m_magz,HEX4 max_accx,_
HEX4 max_accy,HEX4 max_accz,HEX4 max_magx,HEX4 max_magy,_
HEX4 max_magz,HEX4 min_accx,HEX4 min_accy,HEX4 min_accz,_
HEX4 min_magx,HEX4 min_magy, HEX4 min_magz,_
DEC temperature2]
   INTCON.1 = 0 'azzero interrupt
   RESUME 'riprendo il programma
   ENABLE 'abilito interrupt
```

Per il firmware dell'FPGA si riporta inizialmente il file UCF

```
NET "clk" LOC = P82 | IOSTANDARD = LVCMOS33;
NET "clk" TNM_NET = clk;
NET "clk" PERIOD = 25.0ns HIGH 50%;
TIMESPEC TS_clk = PERIOD "clk" 25 ns HIGH 50%;
NET "clko" LOC = P47 | IOSTANDARD = LVTTL;
## switch per modtemp
NET "mod_temp<0>" LOC = P109 | IOSTANDARD = LVTTL | PULLUP;
NET "mod_temp<1>" LOC = P108 | IOSTANDARD = LVTTL | PULLUP;
NET "mod_temp<2>" LOC = P107 | IOSTANDARD = LVTTL | PULLUP;
NET "clk_trasf" LOC =P135 | IOSTANDARD = LVTTL;
NET "clk_trasf" CLOCK_DEDICATED_ROUTE = FALSE;
PIN "U1/DCM_SP_INST.CLKIN" CLOCK_DEDICATED_ROUTE = FALSE;
NET "Tx_p" LOC=P2|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tx_n" LOC=P3|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Ty_p" LOC=P8|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Ty_n" LOC=P9|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tz_p" LOC=P11|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tz_n" LOC=P12|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tw_p" LOC=P15|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tw_n" LOC=P16|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tq_p" LOC=P18|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tq_n" LOC=P19|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tr_p" LOC=P33|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tr_n" LOC=P34|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Ts_p" LOC=P35|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Ts_n" LOC=P36|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tt_p" LOC=P39|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "Tt_n" LOC=P40|IOSTANDARD="LVDS_25"|SLEW=SLOW|DRIVE=8;
NET "oenable_trasf" LOC=P140|IOSTANDARD=LVTTL;
NET "odati" LOC=P138|IOSTANDARD=LVTTL;
NET "singola" LOC=P160|IOSTANDARD=LVTTL;
NET "doppia" LOC=P161|IOSTANDARD=LVTTL;
NET "tripla" LOC=P162|IOSTANDARD=LVTTL;
NET "quadrupla" LOC=P163|IOSTANDARD=LVTTL;
```

Di seguito e riportato il codice per l'inizializzazione del DCM e della prima RAM.

```
--dcm
U1 : DCM2 port map(
             CLKIN_IN  => clk,
             CLKIN_IBUFG_OUT =>open ,
             CLK0_OUT => clk_in,
             CLK2X_OUT => clk2x,
             LOCKED_OUT => open);
 -- RAM0
 U0 : counter port map (
       clka => clk2x,
       rsta => '0',
       ena => ena,
       wea => wea,
       addra => addra,
       dina => dina,
       douta => douta,
       clkb => clk4x,
       rstb => '0',
       enb => enb,
       web => web,
       addrb => addrb,
       dinb => dinb,
       doutb => doutb);
```

Successivamente la parte di codice che interviene nel processo *WINCON* per la gestione della finestra d'ascolto, la valutazione dell'evento e il successivo incremento dei contatori di controllo della parità.

```
--ad ogni ciclo di clock e se uno degli scintilaltori
--ha rilevato la particella
if ((clk_in'event  and clk_in = '1') and
   (cx='1' or cy='1' or cz='1' or
   cw='1' or cq='1' or cr='1' or cs='1' or ct='1')) then
   i:=i+1;
   --alla fine della finestra diascolto
   if ( i = 5 or win_temp = '1') then
      add<=cx&cy&cz&cw&cq&cr&cs&ct; --indirizzo RAM
      counterx<=counterx + cx;   --Contatore error
      countery<=countery + cy;
      counterz<=counterz + cz;
      counterw<=counterw + cw;
      counterq<=counterq + cq;
      counterr<=counterr + cr;
      counters<=counters + cs;
      countert<=countert + ct;
      i:=0;
      index_tri(0 to 3)<=cx&cy&cz&cw;
      azzera <= '1';
   end if;
end if;
```

L'incremento del contatore dopo che il processo *WINCON* ha valutato l'evento avviene con il seguente codice.

```
--alla fine della finestra d'ascolto dopo
--la valutazione dell'evento
if (clk4x='1' and clk4x'event) then
  if (mask /= "00000000") then
     addr:=mask;
     if (chooseRam = '0') then
          j:=j+1;
          if (j=1) then
              ena<='1';  --leggo valore da RAM
              addra<=addr;
          elsif (j=3) then
              wea<="1"; --abilito scrittura
              addra<=addr;
              dina<=douta + '1'; --incremento valore letto
          elsif (j=4) then
              wea <="0"; --disabilito RAM
              ena <= '0';
              azz_mask<='1';
              j:=0;
          end if;
     else
          j1:=j1+1;
          if (j1=1) then
              ena1<='1';
              addra1<=addr;
          elsif (j1=3) then
              wea1<="1";
              addra1<=addr;
              dina1<=douta1 + '1';
          elsif (j1=4) then
              wea1 <="0";
              ena1 <= '0';
              j1:=0;
              azz_mask<='1';
          end if;
     end if;
```

Al termine della finestra di conteggio l'FPGA deve preparare i dati per trasferirli al microcontrollore principale e, con il processo *NEWDATA*, esegue lo scambio tra le due RAM e la creazione della coda per i contatori del controllo di parità.

```
newdata: process (win_temp,enable_trasf,stop_trasf,atzero1)
    begin
      if (stop_trasf  = '1') then
         ready_trasf <='0';
         countertot <=(others=>'0');
      elsif (enable_trasf = '1') then
         ready_trasf <='0';
      elsif (win_temp'event  and win_temp = '0')  then
         --scambio ram e creo coda per contatori di parita
         chooseRam <= not chooseRam;
         countertot <=counterx&countery&counterz&counterw
                     &counterq&counterr&counters&countert;
         atzero <='1';
         ready_trasf <='1';
      end if;
      if (atzero1 = '1') then
         atzero <= '0';
      end if;
   end process;
```

Di seguito sono riportati i processi di trasferimento dati verso il microcontrollore. Il primo processo permette il trasferimento dei dati dei contatori di controllo contenuti nella coda *counttot*, mentre il secondo processo permette il trasferimento dei dati contenuti nella memoria RAM.

```
--trasferimento delle coda per il controllo di parita
transferParit: process (clk_trasf,countertot,enable_trasf,parita)
      variable j : integer:= 0;
   begin
        if (enable_trasf = '1' and parita = '1') then
            if(clk_trasf'event  and clk_trasf = '1')  then
                  --trasferimento dati
                  datip<=countertot(j);
                  j:=j+1;
                  --fine code
                  if (j=255) then
                     stop_trasf<='1';
                     j:=0;
                  end if;
           end if;
      else
         j:=0;
         stop_trasf<='0';
      end if;
   end process;
   odati <= dati or datip;
```

```
 if (stop_trasf = '1') then
    i:=0;
    j:=0;
    dati <= '0';
    parita <= '0';
 elsif (enable_trasf = '1') then
    if (j=0) then
     data<=data2;
     j:=1;
    elsif (clk_trasf='1' and clk_trasf'event) then
       dati <= data(i); --trasferisco dati
       i:=i+1;
       if (i=16) then
          if (j/=255) then
             --leggo nuovo valore dalla RAM
             --utilizzando il progesso aggior
             aggiornamento <='1';
          end if;
       elsif(i=32) then
          data<=data2;
          i:=0;
          if (j=255) then
             --abilito trasferimento dati errore
             parita <='1';
          else
             j:=j+1;
          end if;
       end if;
   end if;
 end if;
if (azzera_agg='1') then
    aggiornamento <='0';
end if;
```

GLOSSARIO

APD	Avalanche Photo-Diodes
ASI	Associazione Spaziale Italiana;
BD	Buffer Descriptor
BDT	Buffer Descriptor Table
BGA	Ball Grid Array
BRAM	Block RAM
CBL	Configurable Logic Block
CD	Card Detection
DAQ	Data AcQuisition system
DCM	Digital Clock Manager
DFS	Digital Frequency Synthesizer
DLL	Delay-Locked Loop
ECL	Emitter-Coupled Logic
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
GPS	Global Positioning System
HID	Human interface Device
I2C	Inter Integrated Circuit
INFN	Istituto Nazionale Fisica Nucleare
LNGS	Laboratori Nazionali del Gran Sasso
LVDS	Low-Voltage Differential Signaling
NMEA	National Marine Electronics Association
NRZI	Non Return to Zero Inverted
PCB	Printed Circuit Board
PCM	Pulse Code Modulation
PID	Product Id
PLL	Phase Locked Loop
PQFP	Plastic Quad Flat Package
PROM	Programmable ROM

PS	Phase Shifter
QFP	Quad Flat Package
SCL	Serial CLock
SDA	Serial DAta line
SDI	Serial Data Input
SDO	Serial Data Output
SDA	Serial DAta line
SIE	Serial Interface Engine,
SPI	Serial Peripheral Interface
TQFP	Thin Quad Flat Package
TSIP	Trimble Standard Interface Protocol
VDI	Vendor Id
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
WLS	WaveLength-Shifting
WP	Write Protection

BIBLIOGRAFIA

[1] De Angelis Alessandro, L'enigma dei raggi cosmici, Springer;

[2] ASSIMETRIE anno 5 numero 10/9.10, i raggi cosmici, rivista trimestrale INFN;

[3] Pacini D., Nuovo Cimento, 3 (1912) 93;

[4] Hess V., Phys. Z., 13 (1913) 1084;

[5] Pfotzer G., Z. Phy., 102 (1936) 23;

[6] Raggi cosmici - http://it.wikipedia.org/wiki/Raggi_cosmici

[7] Sito CORAM - http://cosmicrays.le.infn.it;

[8] A.Akindinov, Development of scintillation detectors with MRS APD light readout for CBM Muon system and ECAL preshower;

[9] C. Cheshkov, G. Georgiev, E. Gouchtchine, L. Litov, I. Mandjoukov, V. Spassov, Application of avalanche photodiodes as a readout for scintillator tile-fiber systems, Nuclear Instruments and Methods in Physics Research A 440 (2000) 38-45;

[10] Akindinov et al., High-efficiency and low noise scintillation detector for ionizing particles START (Scintillation Tile with MRS APD Light ReadouT);

[11] Akindinov et al., Long-term operation of a multi-channel cosmic muon system based on scintillation counters with MRS APD light readout;

[12] Akindinov et al, Scintillation counter with MRS APD light readout;

[13] ASI -http://www.asi.it/it/news/cosmic_ray_a_scuola_con_victor_hess;

[14] M.R. Coluccia et al., CORAM (COsmic RAy Mission), XCVII Congresso Nazionale Società Italiana di Fisica, L'Aquila, Italy, 26 Settembre - 30 Settembre 2011;

[15] M.R. Coluccia et al., CORAM (COsmic RAy Mission): an outreach program 100 years after Pacini and Hess works, Proc. of the 9th Workshop on Science with the New Generation of High Energy Gamma-ray Experiments, 2012. (SciNeGHE 2012). Lecce, Italy, 20-22 Jun. 2012; to be published on Nucl. Phys. B (Proc. Suppl.);

[16] G. Chiarello et al., CORAM (COsmic RAy Mission): un esperimento di outreach 100 dopo i lavori di Pacini e Hess, XCVIII Congresso Nazionale Società Italiana di Fisica, Napoli, Italy, 17 Settembre - 21 Settembre 2012;

[17] G. Chiarello et al., The CORAM (COsmic Ray Mission) outreach program: cosmic ray measurements at high altitude, ComunicareFisica 2012, Torino, 8-12 Ottobre 2012;

[18] Datasheet Microchip PIC18F87J50 http://ww1.microchip.com/downloads/en/devicedoc/39775b.pdf;

[19] Datasheet Microchip PIC16F1847 http://ww1.microchip.com/downloads/en/DeviceDoc/41453B.pdf;

[20] Datasheet Maxim DS18B20 - http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf;

[21] Datasheet Trimble Lassen iQ - http://trl.trimble.com/docushare/dsweb/Get/Document-338501/LassenIQ_RM_1B_19220.pdf;

[22] Datasheet ST LSM303DLHC - http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00027543.pdf;

[23] Xilinx, Spartan-3E FPGA Starter Kit Board User Guide ug230 - http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf;

[24] Xilinx, Spartan-3E FPGA Family: Data Sheet ds312 - http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf;

[25] Xilinx, Spartan-3 Generation Configuration User Guide ug332 -

http://www.xilinx.com/support/documentation/user_guides/ug332.pdf;

[26] Xilinx, Spartan-3 Generation FPGA User Guide ug331 - www.xilinx.com/ support/documentation/user_guides/ug331.pdf;

[27] Xilinx, Spartan-3E Libraries Guide for HDL Designs ug617 - http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/spartan3e_hdl.pdf

[28] Xilinx, Using Xilinx ChipScope Pro ILA Core with Project Navigator to Debug FPGA Applications ug750 - http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/ug750.pdf

[29] Datasheet PIC18 Explorer Board - http://ww1.microchip.com/downloads/en/DeviceDoc/51721b.pdf;

[30] MicroEngineering Labs, Reference Manual PBP3 PicBasic Pro

[31] ELETTRONICA IN, CORSO SDCARD;

[32] ELETTRONICA IN, CORSO USB;

[33] SD CARD - http://en.wikipedia.org/wiki/Secure_Digital;

[34] USB - http://en.wikipedia.org/wiki/Universal_Serial_Bus;

[35] Xilinx, The 3.3V Configuration of Spartan-3 FPGAs xapp453 - http://www.xilinx.com/support/documentation/application_notes/xapp453.pdf;

[36] Xilinx, Platform Flash In-System Programmable Configuration PROMs ds123 - http://www.xilinx.com/support/documentation/data_sheets/ds123.pdf;

[37] Xilinx, Platform Flash PROM User Guide UG161 - http://www.xilinx.com/support/documentation/user_guides/ug161.pdf;

[38] Xilinx, Interfacing LVPECL 3.3V Drivers with Xilinx 2.5V Differential Receivers, xapp696 - http://www.xilinx.com/support/documentation/application_notes/xapp696.pdf;

[39] Xilinx, Using Block RAM in Spartan-3 Generation FPGAs xapp463 - http://www.xilinx.com/support/documentation/application_notes/xapp463.pdf;

[40] Xilinx, Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs,xapp462 - http://www.xilinx.com/support/documentation/application_notes/xapp462.pdf;

[41] coe file, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ cgn_r_coe_file_syntax.htm;

[42] ST, Using LSM303DLH for a tilt compensated electronic compass AN3192 - http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00269797.pdf;

[43] International Cosmic Day 2012 - http://ippog.web.cern.ch/resources/2012/international-cosmic-day-2012;

RINGRAZIAMENTI

Ringrazio Prof. Stefano D'Amico e il Prof. Marco Panareo per avermi permesso di svolgere la tesi presso il laboratorio di Elettronica dell'INFN del Dipartimento di Fisica e Matematica e per la loro costante disponibilità.

Ringrazio il Dott. Alessandro Corvaglia, che mi ha seguito e supportato con i suoi consigli durante tutto il periodo dello svolgimento della tesi e per avermi, aiutato e "sopportato" a superare le difficoltà incontrate durante la realizzazione del lavoro.

Ringrazio il Prof. De Mitri Ivan e Dott.ssa Coluccia Maria Rita che mi hanno permesso di aggregarmi al gruppo di lavoro di CORAM e per i loro consigli e informazioni durante la stesura di questo lavoro

Ringrazio il personale del Laboratorio di Elettronica dell'INFN, in particolare Carlo Pinto, Roberto Assiro, Fulvio Ricciardi, Pietro Creti, Aurora Pepino e Patrizio Primiceri per i suggerimenti dati nei momenti difficili.

Uno speciale ringraziamento a mia madre Maria Lucia, mio padre Pietro, a mia sorella Annachiara, nonna Mina e nonno Biagio che hanno creduto in me e mi hanno sostenuto e incoraggiato, affrontando sacrifici senza i quali non avrei mai potuto raggiungere questo traguardo e per i quali vi sarò sempre grato.

Un ringraziamento a zio Enzo e a zia Anna, i quali mi hanno sempre sostenuto e incoraggiato e mi hanno anche permesso di viaggiare con loro nel mondo.

Un ulteriore ringraziamento, non meno importante, a Renato, Aiko e all'ultima arrivata Minako per la loro costante presenza e aiuto in questi anni.

Inoltre devo rivolgere un particolare GRAZIE a coloro (mia madre, zia Anna, Renato e nonno) che hanno avuto la sfortuna di leggere in fase di stesura questo lavoro e mi hanno dato importanti consigli per migliorarlo.

Per altri motivi devo nuovamente ringraziare nonna Mina.

Un ultimo grazie a tutti gli amici (Alberto, Luvi, Giò, Matteo, Beppe, Paolo, Fra, Alberto, Andrea, Valeria, Vincenzo, Mirko, Mino, Alessandra, Federica, Giuseppe e anche a qualcuno che ho dimenticato di scrivere.....) che mi sono stati vicini in questi anni e con cui ho passato esperienze e momenti bellissimi indimenticabili .