UNIVERSITÀ DEL SALENTO
Facoltà di Ingegneria
Corso di Laurea Triennale in Ingegneria dell'Informazione

TESI DI LAUREA

SVILUPPO DEL FIRMWARE DI GESTIONE DI UN MODULO ELETTRONICO PER IL TIME STAMPING

Relatore:
Chiar.mo Prof. **Marco Panareo**

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

Laureando:
Gianluigi Chiarello

A.A. 2008/2009

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

INDICE

CAPITOLO 4

Introduzione

In molteplici campi dell'ingegneria e della fisica è emersa spesso la necessità di dare un riferimento temporale univoco ad eventi che hanno luogo in diversi punti situati sulla superficie terrestre. Per la sincronizzazione dei diversi apparati di rilevazione degli eventi si sono esperiti vari approcci come l'utilizzo di protocolli per la sincronizzazione all'interno della rete internet. Tuttavia tali soluzioni consentono sincronizzazioni entro intervalli di tempo non definibili a priori e pertanto inefficaci qualora le precisioni richieste siano dell'ordine del centinaio di ns. Per risolvere il problema della sincronizzazione e nello stesso tempo effettuare il time-stamping di eventi si può utilizzare come riferimento temporale il Global Positioning System (GPS).

Il sistema GPS fa uso di orologi atomici ad alta precisione che forniscono indicazioni temporali estremamente accurate. Il ricevitore GPS riceve ogni secondo un segnale chiamato pulse per second (PPS) sincronizzato con il sistema di riferimento per la misura del tempo accettato in tutto il mondo, l'UTC (Universal Coordinated Time). Tramite questo segnale si può effettuare il time-stamping accurato al secondo ma, nel campo delle applicazioni ingegneristiche e fisiche, nell'intervallo di un secondo si possono manifestare più eventi, per cui si ha il bisogno di sincronizzare un oscillatore al quarzo con il ricevitore GPS con in modo da ottenere dei riferimenti temporali molto precisi nell'ordine del centinaio di ns.

L'obbiettivo della tesi è la progettazione, la realizzazione e la programmazione di un modulo che consenta di estrarre informazioni temporali dal GPS e di eseguirne la correzione per raggiungere l'accuratezza desiderata. Il modulo è costituito da una PIC che estrae i dati, inviati serialmente dal ricevitore GPS, da una FPGA che permette di calcolare la frazione di secondo e quindi di raggiungere l'accuratezza desiderata

attraverso un oscillatore al quarzo a 50 MHz e infine da un sistema embedded che ha il compito di combinare le informazioni provenienti dalla PIC e dalla FPGA. Una volta costruito il modulo si sono programmati i dispositivi sviluppando gli algoritmi, prima del sistema embedded nel linguaggio C e successivamente quello della PIC in MicroBasicPro e infine della FPGA tramite il linguaggio VHDL.

Il modulo sviluppato è uno dei dispositivi che utilizzati nel ambito dell'esperimento EEE, che ha come obbiettivo la rivelazione dei raggi cosmici di alta energia. Quando un raggio cosmico incontra l'atmosfera terrestre, interagisce con essa generando particelle secondarie, queste particelle a loro volta interagiscono o decadono creandone delle altre, il risultato è la generazione di uno sciame. Per studiare i raggi cosmici di energia elevata vengono usati opportuni rilevatori distribuiti sul territorio nazionale i quali rilevano l'evento associato all'arrivo del raggio cosmico. Per accertarsi che gli eventi acquisiti siano associati allo stesso raggio cosmico, tutti i rilevatori devono essere opportunamente sincronizzati tra di loro.

Capitolo 1

TIME STAMPING

1.1 Il problema del time stamping

L’evoluzione dei sistemi elettronici, della ricerca ingegneristica e fisica ha fatto emergere il problema di avere un riferimento temporale preciso. In particolar modo: nei sistemi elettronici, per avere una comunicazione sicura e autenticata, si sono utilizzati i meccanismi della crittografia e firma digitale che hanno bisogno del riferimento temporale per indicare la data dell’emissione dello stesso documento; nel campo della ricerca ingegneristica e fisica per stabilire il riferimento temporale di eventi o per la sincronizzazione di apparecchiature elettroniche.

Per riferimento temporale si intende il time stamping, cioè una sequenza di caratteri che rappresentano una data e/o un orario per accertare l'effettivo avvenimento di un certo evento.

Il problema sostanziale della tecnica del time stamping è avere un riferimento univoco, ciò è importante specialmente per la sincronizzazione delle apparecchiature elettroniche. Si parla di riferimento univoco quando due eventi avvenuti simultaneamente in due posti lontani tra di loro hanno lo stesso time stamping. Per far si che ciò avvenga si è dato luogo a diversi meccanismi tipo il protocollo (TSP) che effettua il time stamping attraverso internet: utilizzato per le firme digitali. Detto protocollo però non può essere utilizzato per la ricerca ingegneristica o fisica in quanto può subire ritardi variabili dovuti alla congestione della rete per cui è necessario utilizzare un'altra tecnica quale l’utilizzo del GPS come riferimento.

1.2 Il time stamping con il GPS

Il sistema GPS (Global Positioning System), consente di determinare la posizione in tempo reale, con incertezze di pochi metri, di un oggetto posto sulla superficie terrestre. Oltre alle informazioni delle coordinate geografiche, come latitudine e longitudine, il sistema fornisce anche la possibilità di datare tramite un opportuno segnale trasmesso, detto 1PPS , con indeterminazione dell'ordine dei nanosecondi, un evento rispetto alla scala temporale utilizzata dal GPS (scala UTC)[1]. Se il ricevitore è mantenuto in una posizione fissa, il segnale 1PPS può essere utilizzato come sistema di riferimento per il tempo. Il risultato finale è che tutti gli eventi locali, correttamente riferiti a quell'impulso 1PPS, possono essere riferiti alla scala UTC.

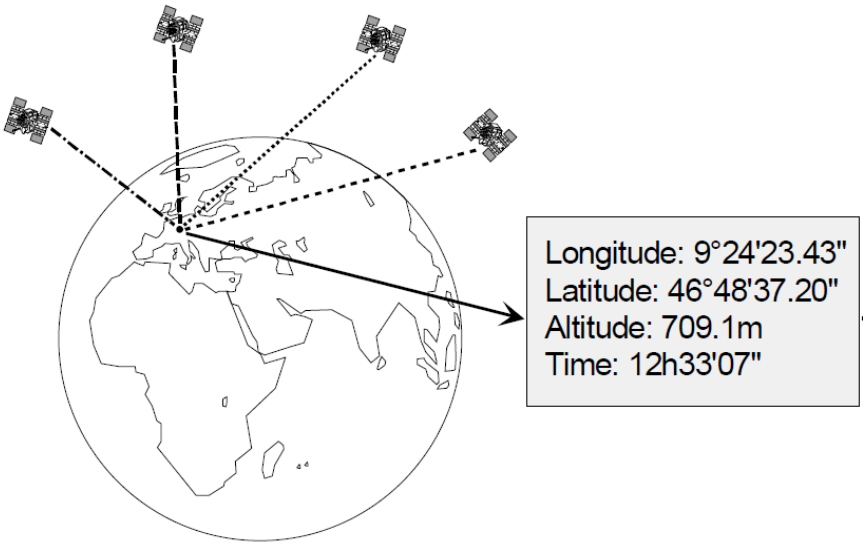


Fig.1. 1: Funzionamento GPS

Utilizzando il dato proveniente dal GPS quale riferimento temporale e poi eseguendo una opportuna correzione della misura per compensare l'errore (del quale si parlerà successivamente) intrinseco del GPS (tale errore

[1] UTC è il fuso orario di riferimento da cui sono calcolati tutti gli altri fusi orari del mondo. Esso è derivato (e coincide a meno di piccole approssimazioni) dal *Greenwich Mean Time* (GMT).

è dovuto alla combinazione dell'effetto dopler, multipath e ritardi d'origine ionosferica ed atmosferica) si può effettuare il time stamping di un evento.

Il sistema GPS pertanto, è un accurato sistema di sincronizzazione di apparecchiature sia a livello locale, ma sopratutto a livello internazionale. Infatti, mentre a livello locale la sincronizzazione è semplicemente raggiungibile anche con altri mezzi, ciò non succede a livello internazionale per eventuali ritardi dovuti alla rete utilizzata. Questo problema è risolto tramite l'utilizzo del GPS.

Il GPS è usato nelle reti telefoniche e dati per mantenere le loro base station in perfetta sincronizzazione, permettendo la condivisione dello spettro di frequenze in maniera efficiente. Varie sono le ricerche e gli esperimenti già eseguiti, in cui si è utilizzato il GPS per effettuare il time stamping, tra questi c'è l'esperimento EEE (Extreme Energy Events) per il quale viene realizzato l'interfacciamento del modulo di time stamping discusso in questo lavoro.

1.3 L'esperimento EEE e il telescopio a MRPC

L'esperimento EEE (Extreme Energy Events) ha come obiettivo lo studio della radiazione cosmica ad alta energia, in altre parole capire come, l'origine dei raggi cosmici primari (protoni e nuclei) che costituiscono il residuo del Big Bang che, da milioni e milioni di anni viaggiano dalle zone più remote dello spazio verso il nostro pianeta [1]. Quando un raggio cosmico incontra l'atmosfera terrestre, interagisce con i nuclei di cui essa è composta generando particelle secondarie. Le particelle secondarie sono tanto più numerose ed energetiche, quanto più è energetico il raggio cosmico primario. Il loro tempo di vita è molto breve, ed esse decadono in altre particelle, tra cui i muoni, che costituiscono la maggior parte della componente elettricamente carica dei raggi cosmici, a livello del mare. I raggi cosmici ad

elevata energia (maggiore di 10^{18}eV) generano sciami di particelle estesi anche alcune decine di chilometri quadrati.

Poiché eventi con tali energie sono rari, l'esperimento EEE prevede l'uso di una serie di telescopi situati su diversi edifici scolastici sul territorio nazionale. Il componente principale dei telescopi è un rilevatore MPRC (Multigap Reistive Plate Chamber). Il suo principio di funzionamento è analogo a quello di un condensatore piano nella cui intercapedine è presente un gas. Una particella dotata di carica elettrica, nell'attraversare il gas, ionizza le particelle che incontra sul suo percorso. Le cariche elettriche generate dal gas all'interno del condensatore si muovono verso le armature del condensatore, e muovendovi danno origine ad un impulso elettromagnetico [1]. È questo segnale elettrico, abbastanza elevato, che viene usato per rivelare il passaggio della particella nel rilevatore. L'ampiezza del segnale elettrico dipende dal gas presente nell'intercapedine e dalla differenza di potenziale applicata alle armature(circa 8kV). Lo scopo principale del rivelatore MRPC è quello di identificare con grande precisione la posizione e l'istante di tempo in cui avviene il passaggio della particella al suo interno [1], pertanto:

- le superficie delle armature metalliche del condensatore a contatto con il gas sono rivestite di materiale dielettrico, reso resistivo: vetro o plastica;
- le armature metalliche, dette elettrodi di raccolta dei segnali, sono segmentate in striscioline (strip) o rettangolini (pad);
- l'intercapedine contenente il gas tra le armature è suddivisa in tante sotto-intercapedini (gas gap), di piccolissimo spessore, tramite sottili piani intermedi di vetro.

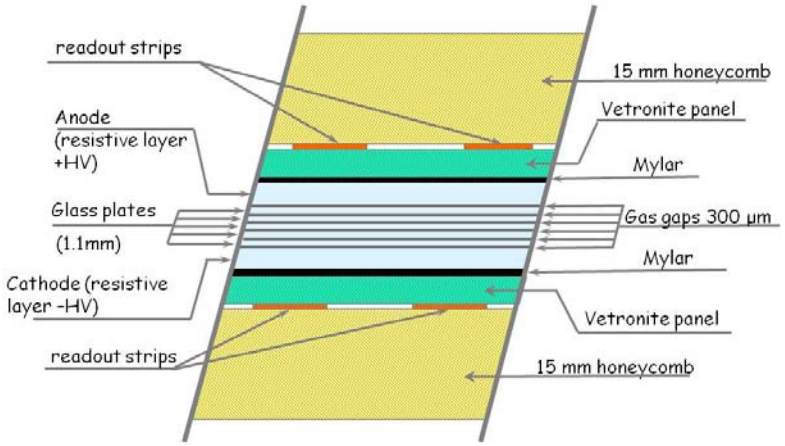


Fig.1.2: Sezione del rilevatore MRPC

Il telescopio dell'esperimento EEE è costituito da tre piani di rilevatori MPRC. Ogni piano è capace di misurare con grande precisione il punto d'impatto della particella cosmica incidente e il suo tempo di attraversamento. L'area sensibile di ogni piano è di (1.6 × 0.82) m^2. Il sistema di lettura (readout) dei segnali avviene tramite elettrodi metallici suddivisi in strisce (strip) longitudinali, ciascuna lunga 1.6m e larga 34mm. Ogni strip sarà connessa, a ciascuna delle sue estremità, con una catena elettronica di lettura e di acquisizione del segnale costituita da: un sistema di front end, per l'amplificazione e la discriminazione dei segnali forniti dagli elettrodi di readout dei rivelatori MRPC; da un sistema di conversione, per la digitalizzazione delle informazioni acquisite e da un sistema di trigger, per la selezione delle particelle, che genera un segnale (trigger), quando almeno una striscia di ogni singolo piano è attraversata da una particella. Il segnale di trigger si ottiene effettuando un OR logico fra tutti i segnali provenienti dalle strisce di ogni singolo piano e, successivamente, ponendo gli OR di ciascun piano in AND logico. La catena elettronica è connessa con un calcolatore tramite un'opportuna interfaccia. Tutti i telescopi, dei quali si conosce l'esatta posizione geografica, essendo temporalmente sincronizzati via satellite tramite un sistema GPS, sono messi in coincidenza durante la fase di analisi dei dati. Questo consente di rivelare eventi cosmici di energie

elevate che si manifestano attraverso sciami cosmici di grande apertura angolare, ognuno dei quali costituito da un notevole numero di muoni, provenienti da un punto comune nell'alta atmosfera terrestre.

Il rivelatore MRPC deve garantire un'ottima precisione per misurare i tempi di volo (precisione del nano secondo) delle particelle cosmiche e le loro traiettorie, necessarie per stabilire il punto in cui si è verificata l'interazione che ha prodotto la serie di eventi registrati dai rivelatori.

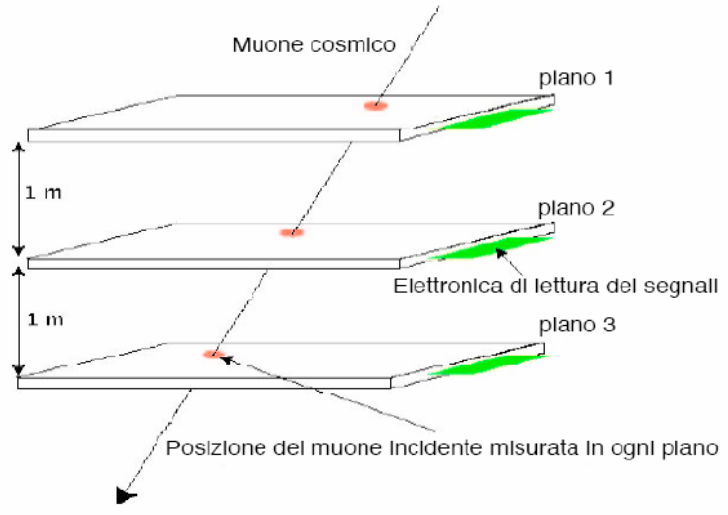


Fig.1.3: Funzionamento telescopio MRPC

Tramite la misura dei tre punti d'impatto (uno per piano), è possibile ricostruire la traiettoria della particella che ha attraversato il telescopio. È inoltre possibile conoscere il verso di attraversamento della particella, grazie alla misura del suo tempo di volo tra un piano e l'altro.

Capitolo 2

Global Positioning System (GPS)

2.1 Cenni storici

Il GPS, o più precisamente il sistema NAVSTAR-GPS (NAVigation System for Timing and Ranging –Global Positioning System), è nato negli Stati Uniti negli anni Settanta come sistema militare per rispondere all'esigenza di seguire il percorso di mezzi militari e consentire eventuali operazioni di supporto e di salvataggio.

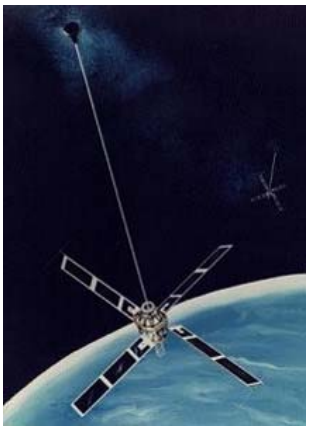


Fig.2.1: Satellite dell'esperimento Transit

Il GPS è l'evoluzione di due altri sistemi che prendono vita nei primi anni '60 il primo esperimento è il TRANSIT (Fig.2.1), frutto della collaborazione tra il dipartimento della difesa (DOD), la NASA e il dipartimento dei trasporti, interessati entrambi allo sviluppo di un sistema satellitare per l'individuazione della posizione in tre dimensioni [3]. I contributi maggiori di questo esperimento furono dovuti alla copertura globale, all'utilizzo in qualsiasi condizione climatiche e ad una ottima accuratezza tuttavia, per le sue limitazioni (per la lentezza dell'invio della posizione), alcune organizzazioni tra cui la Navy svilupparono sistemi alternativi al TRANSIT. Il secondo esperimento portato avanti negli stessi anni dal Naval Reasearch Laboratory (NRL) è il TIMATION, nel quale si ritrovano le prime applicazioni di orologi atomici a bordo dei satelliti per il calcolo della

posizione [3]. Successivamente seguirono diversi sistemi come il System 621B dell' Air Force che usava orbite ellittiche con diverse inclinazioni e una modulazione PNR (pseudorandom noise) per la comunicazione, successivamente utilizzata nel GPS.

Nel 1969, l'ufficio del segretario della difesa (OSD) stabilì il programma Defense Navigation Satellite System (DNSS) per l'unione in un unico sistema satellitare di ogni ricerca fatta dalle diverse agenzie militari [3]. Venne creato il Navigation Satellite Executive Steering Group per determinare l'attuabilità del DNSS, da ciò nacque il sistema NAVSTAR GPS, sviluppato nel GPS Joint Program Office (JPO) , California. I primi 11 satelliti sperimentali, successivamente sostituiti da quelli operativi a partire dal 1989, vennero lanciati dal 1978 al 1985. Alla fine del 1993 fu dichiarato l'inizio dell'operatività del sistema con una costellazione (Fig.2.3) di 28 satelliti del tipo BLK IIA (Fig.2.2) e BLK IIR di cui 4 di riserva [2] [3].

Fig.2.2 satellite BLK IIA

2.2 Funzionamento del GPS

2.2.1 I segmenti del sistema

Il sistema GPS è un sistema satellitare basato su una costellazione (Fig.2.3) di 28 satelliti orbitanti ad una quota di circa 20,180 Km con un periodo orbitale di 11h 58m 02s. Ogni satellite compie due orbite complete in poco meno di un giorno solare, in modo che, per un punto qualsiasi della

Terra, tutta la costellazione si ripresenta quotidianamente con un anticipo di un po' meno di quattro minuti rispetto al giorno precedente. I parametri orbitali adottati fanno sì che, in ogni istante e in ogni luogo, nell'ipotesi di assenza di ostacoli, siano visibili almeno quattro satelliti. I satelliti sono divisi su 6 orbite distanti fra loro di un angolo di 60° e formanti un angolo di 55° rispetto al piano equatoriale [2]. Ogni satellite consente la misura della posizione del ricevitore nelle tre coordinate spaziali (latitudine, longitudine e altitudine) e del tempo UTC (Universal Coordinated Time) con una copertura globale e continua e altre grandezze qualora richieste.

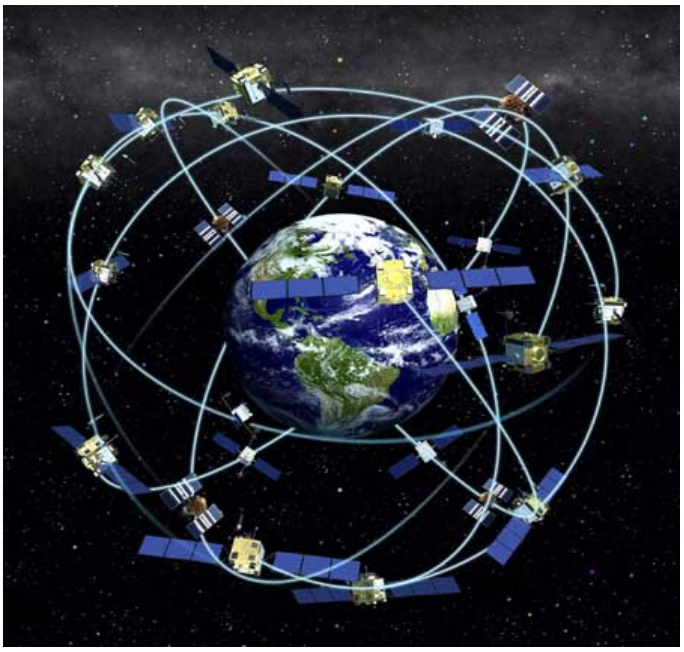


Fig.2.3: Costellazione dei satelliti

Il sistema è costituito da tre segmenti:

- Il **segmento SPAZIALE**, formato dalla costellazione satellitare GPS (appena descritta) orbitante intorno alla Terra. Ogni satellite pesa circa 800 kg ed è alimentati con pannelli solari [3]. Ciascun satellite si muove sulla sua orbita non geostazionaria con una velocità circa di 4 Km/s passando sullo stesso punto 2 volte in un giorno. La costellazione è stata disegnata in modo da garantire che qualsiasi utente usufruisca della presenza

sopra l'orizzonte di almeno 4 satelliti che inviano i dati necessari al posizionamento sulla terra, questo vuol dire che ogni satellite risulta in vista 5 ore sulle 12 del passaggio. Ciascun satellite ha a bordo quattro orologi atomici (due al cesio e due al rubidio) aventi stabilità media pari a una parte su 10^{12}; ciò comporta che essi perdono un secondo ogni 317000 anni circa. Questi orologi servono per la generazione dei segnali in trasmissione, infatti, danno luogo ad un oscillatore con una frequenza base di 10,23 MHz, da cui è possibile ricavare tutte le frequenze. A bordo di ogni satellite sono presenti motori per le correzioni orbitali e dei sistemi giroscopici per la stabilizzazione.

- Il **segmento di CONTROLLO** verifica lo stato di funzionamento dei satelliti e aggiorna le relative orbite ed il funzionamento dei loro orologi [3]. Il controllo, è effettuato da diverse stazioni GPS permanenti poste lungo l'equatore in modo tale che, il segnale di un qualunque satellite sia ricevuto da almeno una di queste stazioni. Le stazioni(Fig.2.4) hanno diversi compiti e sono suddivise:

    - cinque Monitor Station (MS) per il controllo dei satelliti, situate a Colorado Springs (Colorado,Stati Uniti), Isole Hawaii e Isola Kwajalein (Oceano Pacifico), Isola di Ascensiòn (Oceano Atlantico) ed Isola Diego Garcia (Oceano Indiano);

    - tre Ground Antenna (GA) per la trasmissione in banda verso i satelliti dei comandi di controllo e delle informazioni da inserire ne messaggi; esse sono col-locate con le Monitor Station delle isole di Ascensiòn, Diego Garcia e Kwajalein;

- una Master Control Station (MCS) situata a Colorado Springs (Colorado,Stati Uniti), responsabile del lavoro svolto da tutto il control segment; elabora le informazioni pervenute da tutti i satelliti attraverso le 5 stazioni di monitoraggio, mette a punto tutte le correzioni necessarie per ogni satellite e comanda la trasmissione delle stesse attraverso le 3 stazioni di controllo. La MSC per poter effettuare le correzioni con grande precisione degli orologi a bordo dei satelliti è dotata di una serie di orologi atomici estremamente precisi ai quali vengono riferiti tutti gli altri orologi, sia a terra che a bordo dei satelliti. Tempo e frequenza a bordo dei satelliti sono forniti da tre o quattro orologi atomici (sia al Cesio che al Rubidio) sincronizzati giornalmente dal centro di controllo terrestre per compensare le derive e gli errori, che si manifestano col trascorrere del tempo.

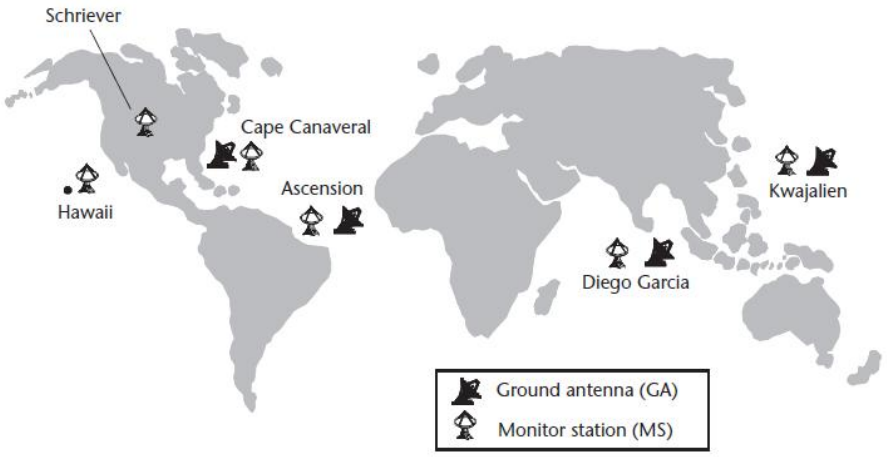


Fig.2.4: Posizione MS e GA

- Il **segmento UTENTE**, è rappresentato dagli utenti (civili e militari) del servizio GPS; un ricevitore GPS demodula i segnali emessi dai satelliti GPS allo scopo di stimare, in tempo reale, la propria posizione.

2.2.2 Il segnale GPS

Tutti i satelliti trasmettono i segnali modulati su due diverse portanti nella banda L, entrambe multiple della frequenza fondamentale f_0=10,23MHz generata dall'utilizzo concorrente degli oscillatori al cesio ed al rubidio installati sui satelliti; indicando con L1 ed L2 le 2 portanti e con f_1 ed f_2 le due frequenze corrispondenti, si ha:

$f_1=154 * f_0=1575,42MHz$

$f_2=120 * f_0=1227,60MHz$

Le portanti L1 ed L2 sono modulate dai seguenti segnali:

- **P code (Precision):** una sequenza di bit con frequenza di 10,23 MHz che si ripete periodicamente dopo circa 38 settimane. Non viene mai trasmessa l'intera sequenza, essa è suddivisa in segmenti che durano una settimana e che vengono rigenerati ogni inizio settimana, assegnati ai satelliti e alle 5 stazioni, di controllo di cui uno non utilizzato [3]. Il codice P modula entrambe le portanti e consente di raggiungere la massima precisione nella procedura di posizionamento. Su questo codice si basa il PPS (Precise Positioning Service) che fornisce un elevato grado di precisione nel posizionamento assoluto con un accuratezza di 18m. Purtroppo questo servizio non è accessibile dagli utenti civili perché il codice P(Fig.2.5) è utilizzabile solo dai militari, in quanto criptato dal codice Y [3]. Per ricavare entrambi i codici si possono utilizzare dei ricevitori che implementano la tecnica del AS (Anti-Spoofing)

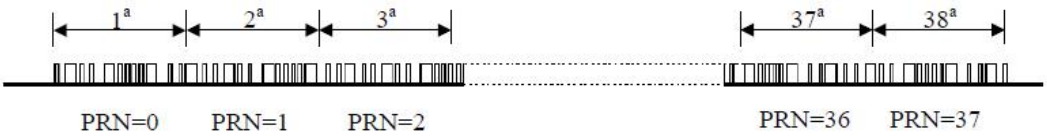


Fig.2.5: P code completa

- **C/A code (Clear Access or Corse Acquisition):** una sequenza di 1023 bit con frequenza 1,023 Mbps ed un periodo di ripetizione pari a 1 ms; modula soltanto la portante L1 e risulta di più facile ricezione in quanto più corto; è unico per ciascun satellite, utilizzato da tutti i ricevitori e consente una precisione molto bassa, infatti su di esso si basa la modalità del segnale trasmesso denominata SPS (Standard Positioning Service) accessibile a tutti gli utenti.

- **D code (Navigation Data):** contiene una gran quantità di informazioni che il segmento di controllo trasmette all'utenza sfruttando il segnale GPS [2].

I tre codici, come si è appena detto, sono sequenze di bit; il codice D trasporta messaggi informativi mentre i codici P e C/A sono costituiti da sequenze pseudo casuali di bit [2], tali sequenze risultano periodiche con periodo 1 ms per il codice C/A e 7 giorni per il codice P. Per questi motivi i codici P e C/A vengono definiti codici di tipo PRN, ogni satellite utilizza una propria sequenza PRN distintiva ed ortogonale alle altre; sono utilizzate 38 sequenze diverse numerate da 0 a 37. Si noti che ogni satellite utilizza le stesse frequenze trasmissive: lo schema d'accesso multiplo adottato è la tecnica CDMA (Code Division Multiple Access) che sfrutta l'ortogonalità dei codici sopra evidenziata [3].

Tutti i satelliti utilizzano le stesse frequenze per trasmettere simultaneamente i codici C/A, P e D, per far ciò si utilizza la modulazione BPSK(Fig.2.6) (Binary Phase Shift Keying). In particolare risulta che la

portante L1 è modulata dai codici C/A, P e D invece, la portante L2 è modulata soltanto dai codici P e D.

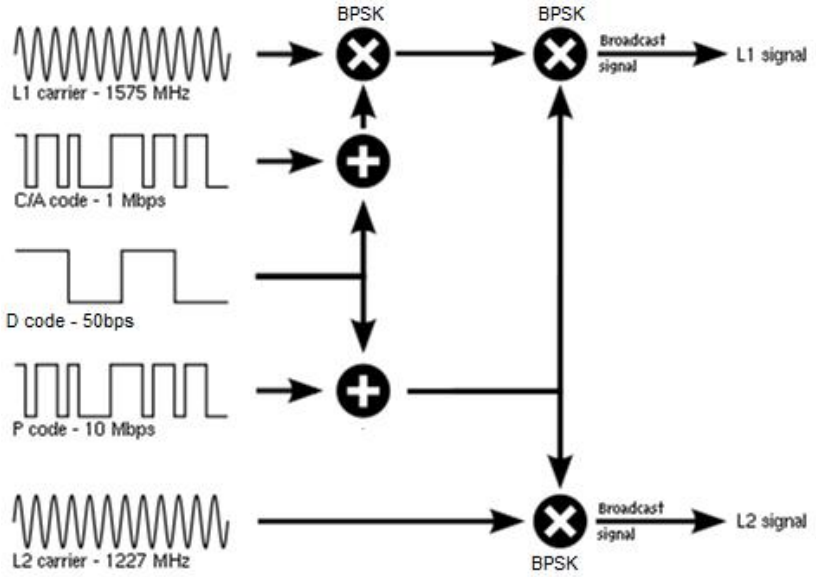


Fig.2.6: Modulazione segnali GPS

2.2.3 Rilevamento della posizione

Il rilevamento della posizione usa la tecnica della triangolazione infatti il satellite invia il suo C/A code, che è sincronizzato con il GPS Time e si ripete periodicamente ogni millisecondo. Una volta che il ricevitore riceve questo segnale esegue la correlazione con il C/A code generato localmente; anche questo dovrebbe risultare sincronizzato con il GPS Time, tuttavia, tale sincronismo non è perfetto a causa essenzialmente degli orologi utilizzati nei terminali. Si ottiene la distanza *d* dal satellite moltiplicando il ritardo *τ* ottenuto tramite la correlazione per la velocità di propagazione delle onde elettromagnetiche:

$$d = \tau \times c$$

con *c*=300.000 km/s; questa distanza è detta

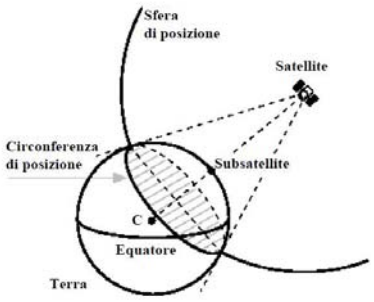


Fig.2.7: Situazione con un unico satellite

pseudorange [3]. Tramite questa distanza si traccia una sfera con il centro nella posizione occupata dal satellite nell'istante di emissione del segnale ed il raggio pari alla distanza calcolata; tale luogo interseca la superficie terrestre individuando una circonferenza, luogo dei punti nei quali può trovarsi il ricevitore (Fig.2.7). Due misure di distanza, disponibili utilizzando due satelliti, individuano due circonferenze che si intersecano in due punti di cui uno è certamente la posizione dell'osservatore; l'ambiguità fra i due punti può essere eliminata con la posizione stimata del ricevitore; considerando come ulteriore incognita anche la quota, sono necessarie tre osservazioni che individuano tre sfere, la cui intersezione individua un volume entro il quale si trova il ricevitore. Come detto, i riceventi terrestri sono equipaggiati con orologi di precisione molto inferiore rispetto a quelli installati nei satelliti, ciò può comportare una accuratezza grossolana nella misura della posizione, ma l'accuratezza può essere recuperata cercando di determinare l'errore rispetto al segnale di tempo fornito dai satelliti; l'errore dell'orologio rappresenta pertanto un'ulteriore incognita che può essere determinata con una quarta osservazione (Fig.2.8); si risolve in definitiva un sistema di quattro equazioni con le quattro incognite: latitudine, longitudine, quota ed offset dell'orologio del ricevitore.

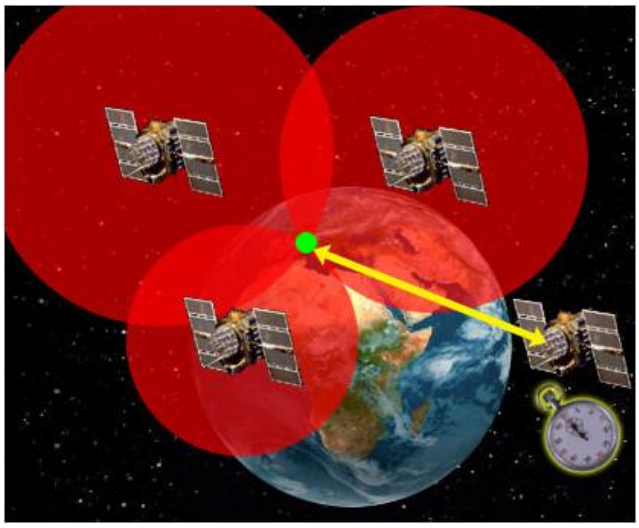


Fig.2. 8: Situazione con il quarto satellite

2.2.4 Errori del sistema

Il sistema GPS è soggetto a differenti tipi di errori, alcuni di tipo naturali altri invece, dovuti a limitazioni tecnologiche che influiscono sull'accuratezza della misura; molti di essi possono essere ridotti di ampiezza, utilizzando delle formule ottenute dopo un'attenta valutazione delle cause e dopo una serie di misure effettuate durante la fase sperimentale del sistema. I principali errori sono:

- **Errori dipendenti dai satelliti**, fra cui anche la Selective Availability e l'AntiSpoofing. Questi sono dispositivi introdotti, inizialmente, dal dipartimento della difesa degli USA per proteggere gli utilizzatori militari dal pericolo di essere ingannati da false trasmissioni e impedire ai civili e a potenze straniere ostili di usufruire al completo della capacità di localizzazione del sistema GPS [3]. Dal 1 maggio 2000 il DOD ha disattivato tali dispositivi, consentendo ai ricevitori civili un incremento della precisione. Il processo consiste nell'introdurre una piccola alterazione nel funzionamento degli orologi dei satelliti. Inoltre, le effemeridi (ovvero la posizione del satellite sul percorso) vengono trasmesse in modo leggermente diverso dal reale. Ne risulta la degradazione dell'accuratezza della posizione.

- **Errore di multipath (Fig.2.9):** derivano, principalmente, dalla combinazione dei segnali diretti con quelli riflessi dalle superfici circostanti, in particolare dalla superficie marina. Tali errori sono, quindi, dipendenti dalla natura e dalla localizzazione delle superfici riflettenti, per cui è possibile ridurli o eliminarli con un'opportuna collocazione e progettazione dell'antenna ed adottando opportune tecniche nei ricevitori. Per esempio, per le stazioni fisse si adoperano antenne di buona qualità che abbiano una scarsa sensibilità per bassi angoli di incidenza dei

segnali e che utilizzano un piano di massa o un choke ring. Il segnale ricevuto è composto da una componente diretta e una riflessa, relativa ai percorsi multipli. Un piano di massa choke ring è realizzato mediante diversi anelli conduttori concentrici che circondano il centro di fase dell'antenna, consentendo di cancellare (o attenuare) la componente riflessa.

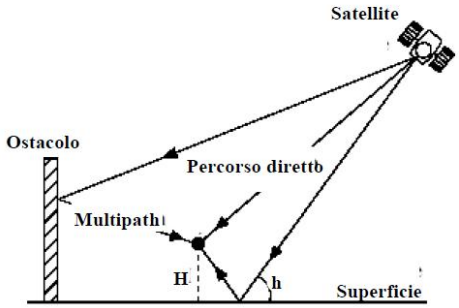


Fig.2.9: Errore di multipath

- **Errori dipendenti dal ricevitore**: Ogni ricevitore genera degli errori legati al rumore interno, alla precisione con cui lavora il correlatore, ai ritardi prodotti sia dai vari dispositivi elettronici che dal software che effettua l'elaborazione dei dati. Gli effetti di tali errori possono essere consistenti nel caso di ricevitori utilizzati su veicoli molto veloci; una progettazione accurata del ricevitore in base all'uso cui è destinato può pertanto renderli molto piccoli.

- **Errori prodotti dalla propagazione dei segnali nella ionosfera e nella troposfera:** nell'attraversamento di tali strati, le onde elettromagnetiche subiscono sia variazioni nella velocità sia delle rifrazioni che producono un allungamento dei percorsi rispetto a quelli rettilinei fra i satelliti e il ricevitore. Per la loro completa eliminazione si dovrebbe ricorrere all'utilizzo della ricezione in diversità o qualora possibile, usare il codice P.

- **Errori introdotti dal Sistema di Controllo**: la MSC nella determinazione delle orbite, nelle correzioni degli orologi può commettere degli errori.

Un aspetto importante è che, l'errore sulla posizione dipende,oltre che dagli errori appena detti, anche da un fattore scalare legato alla geometria del sistema, ossia alla distribuzione spaziale dei satelliti utilizzati nelle misure; tale fattore è denominato Fattore d'espansione dell'errore o PDOP (Position Dilution Of Precision). Nel caso della navigazione marittima o terrestre, interessa soltanto la posizione e quindi la precisione nel piano orizzontale, per cui il PDOP viene sostituito dall'HDOP (Horizontal DOP, che dipende solo dall'azimut dei satelliti). I ricevitori, utilizzando i dati contenuti nel messaggio di navigazione sono tuttavia in grado di calcolare, preventivamente, il PDOP o l'HDOP per le varie combinazioni di satelliti visibili e di scegliere, quindi, la combinazione migliore con il fattore più basso.

2.3 GPS TIME E PPS (pulse per second)

Il GPS, come già detto, può essere usato per la sincronizzazione temporale e per il time stamping. Esistono diverse scale per la misurazione del tempo, e si dividono principalmente in tre categorie fondamentali: siderale, solare e atomica. Le prime due sono legate al moto di rotazione terrestre rispetto a una direzione di riferimento nello spazio: le scale siderali rispetto alla direzione dell'equinozio; le scale solari rispetto alla direzione del Sole. La direzione di riferimento può essere quella vera o una direzione media, depurata degli effetti perturbativi. L' UT1 (Universal Time 1) è una scala solare che rappresenta l'angolo orario fra meridiano di Greenwich e direzione del Sole, depurato degli effetti del polar motion, a cui vanno aggiunte 12 ore. Le scale di tempo atomico sono invece definite da un certo numero di orologi atomici al Cesio 133 e sono le più accurate e stabili. In

particolare la scala UTC (Tempo Universale Coordinato), si basa sulla corrente di definizione del secondo fornita dal Sistema Internazionale (il secondo è la durata di 9.192.631.770 periodi della radiazione corrispondente alla transizione tra due livelli iperfini dello stato fondamentale dell'atomo di Cesio-133) che è periodicamente aggiornata e diminuita di 1s per mantenerla sincronizzata entro il secondo con la scala UT1. Sempre di tipo atomico è la scala di tempo GPST (GPS Time), che è quella a cui il sistema, e quindi il ricevitore, GPS fa riferimento. Quest'ultima differisce dall'UTC (Universal Coordinated Time) di una quantità nota. L'ora di sistema deve essere di tipo uniforme e non può quindi essere uguale all'UTC che viene "ritoccata" una o due volte all'anno aggiungendo o sottraendo 1 secondo; tale correzione comporterebbe notevoli problemi nella regolarità di funzionamento del sistema per gli scopi della navigazione. Tempo e frequenza a bordo dei satelliti sono forniti da tre o quattro orologi atomici (al Cesio ed al Rubidio) sincronizzati giornalmente dal centro di controllo terrestre a causa della lenta deriva.

La Master Station Control (MSC) di Colorado Spring, ha la funzione di assicurare un riferimento temporale comune attraverso una serie di orologi atomici sincronizzati con il GPST; il grado di precisione assicurato è tale da consentire un disallineamento al più di 1 secondo in 32 secoli.

Il ricevitore GPS permette di accedere al segnale degli orologi atomici. In effetti, il GPS non solo fornisce in tempo reale la conoscenza delle proprie coordinate geografiche, come la latitudine e la longitudine, ma anche ricevere un particolare segnale ogni secondo, il cosiddetto *one pulse per second* (1PPS), sincronizzato con il tempo UTC. Se l'antenna ricevente è mantenuta in una posizione fissa il segnale 1PPS può essere utilizzato come sistema di riferimento per il tempo, Il segnale 1PPS, fornito dai ricevitori GPS, è in assoluto il migliore standard disponibile per lunghi tempi d'integrazione.

2.4 Ricevitore GPS: Trimble Resolution T

Fig.2.10: Resolution T

Il Resolution T (Fig.2.10), è una scheda embedded GPS prodotta dalla Trimble. Essa ha un numero di canali in ricezione pari a 12 ed opera sulla frequenza L1 (1575.42 MHz). Sfrutta inoltre lo standard SPS usando il codice C/A (Coarse Acquisition) e necessita di una doppia alimentazione: 3.3V per alimentare il GPS e 5.5V per alimentare l'antenna. Il ricevitore è montato su di una motherboard in cui è presente un connettore a 8 pin (2x4) maschio, il quale ci permette l'interfacciamento, per I/O dei dati e alimentazione. Si è scelto questo ricevitore per l'alta risoluzione nelle applicazioni temporali, infatti prevede una precisione dell'ordine di 15ns (1 δ) per l'uscita PPS. L'antenna utilizzata è un antenna semisferica (Fig.2.11) del tipo choke ring è il segnale ricevuto è di circa -130 dBm alla superficie terrestre; l'antenna include dei filtri e amplificatori del segnale GPS. Essa può utilizzare uno dei due protocolli disponibili:

Fig.2.11: Antenna GPS

- Il protocollo **TSIP** (Trimble Standard Interface Protocol); protocollo a pacchetti binari che permette all'utilizzatore il massimo controllo sulla configurazione, fornendo anche l'errore di quantizzazione, che permette di ottenere una correzione sul segnale PPS. Supporta anche l'invio di comandi multipli con i rispettivi pacchetti di risposta.

- Il protocollo **NMEA** invece è uno standard industriale usato soprattutto per applicazioni marine e permette una compatibilità diretta con altri dispositivi che supportano questo standard.

Il ricevitore resolution T garantisce un'alta accuratezza, fissata la frequenza dell'oscillatore, nel generare il PPS. Quando le applicazioni richiedono maggiore precisione di quella fornita dal ricevitore GPS a basso costo, uno dei principali parametri su cui agire è quello relativo alla correzione dell'errore di quantizzazione. Poiché viene usato un oscillatore a frequenza fissa, il resolution T invia il PPS sul fronte di clock che è più vicino all'UTC/GPS PPS. La differenza, in termini di tempo, tra il fronte di salita del clock usato per generare il PPS e l'istante in cui il resolution T ha determinato l'esatta posizione in cui collocare il PPS, rappresenta la misura dell'errore di quantizzazione. L'oscillatore installato nel resolution T ha una frequenza pari a 12.504 MHz, questo equivale a un periodo di 80 ns tra due cicli di clock. Possiamo pensare quindi che il GPS per 80 ns è incapace di fornire il PPS. Il ricevitore pone il segnale PPS nel ciclo di clock vicino, ciò produce un ritardo che non è mai costante, quindi gli 80 ns di gap introducono un errore di ± 40 ns . L'uscita del resolution T deve essere posta su uno dei fronti del clock (il quale è rappresentato dalla prima forma d'onda), quindi l'ultima forma d'onda (Resolution T PPS) è quella che si ha in uscita dal GPS, mentre la forma d'onda intermedia rappresenta il segnale del tempo UTC ricevuto. Usando entrambi i fronti del clock l'errore sul PPS per il resolution T scende a ± 20 ns.

L'informazioni sull'errore di quantizzazione per ogni PPS è contenuto nel pacchetto 0x8F-AC.

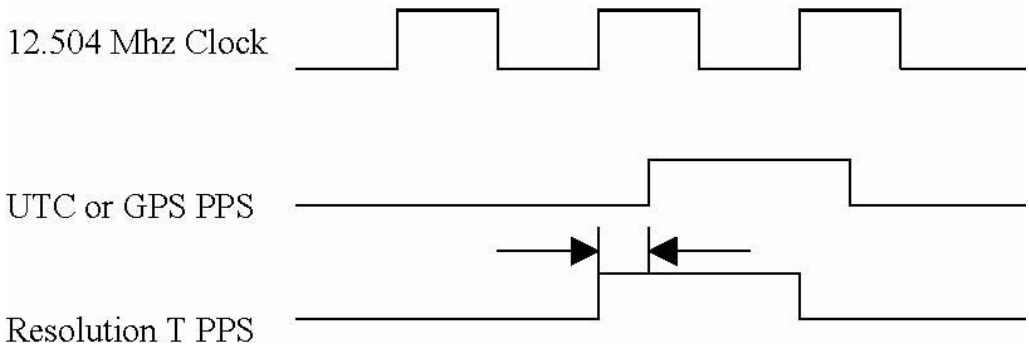


Fig.2.10: Errore di quantizzazione

2.5 Il protocollo TSIP (Trimble Standard Interface Protocol)

Il Resolution T , come già detto, può fornire i dati sulla posizione e sul tempo usando uno dei protocolli disponibili tra il NMEA ed il TSIP. Il protocollo utilizzato per questo lavoro è il TSIP, il quale è caratterizzato da un set di pacchetti per l'invio e ricezione dei dati dal ricevitore. Si è preferito questo protocollo in quanto fornisce anche l'errore di quantizzazione per ogni segnale PPS ricevuto.

Il TSIP inoltre, rispetto all'NMEA, permette la configurazione della porta seriale RS232 in modo da avere un controllo bidirezionale dei pacchetti che riceve e invia il ricevitore, inoltre i settaggi di tale porta e, in generale del GPS, vengono salvati in una memoria non volatile.

Il protocollo TSIP è basato quindi, sulla trasmissione di pacchetti di informazioni tra il dispositivo collegato e il ricevitore GPS. Ogni pacchetto include un codice che identifica il significato e il formato dei dati trasmessi nel suo interno, ed è preceduto e seguito da alcuni caratteri di controllo. Il GPS è configurato per trasmettere automaticamente in uscita i pacchetti 0x8F-AB e 0x8F-AC. Per la maggior parte degli utilizzi del GPS questi pacchetti sono sufficienti perché contengono i dati relativi al tempo, alla posizione e allo stato e funzionamento dello stesso. Di seguito sono elencati i pacchetti che riceve il ricevitore:

ID PACCHETTO	DESCRIZIONE	ID PACCHETTO DI RICHIESTA	SPEDITO
0x42	Posizione XYZ(ECEF), singola precisione	0x37	Quando la posizione fissa è elaborato
0x43	Velocità XYZ(ECEF), singola precisione	0x37	Quando la posizione fissa è elaborato
0x4A	Posizione LLA, singola precisione	0x37	Quando la posizione fissa è elaborato
0x56	Velocità ENI, singola precisione	0x37	Quando la posizione fissa è elaborato
0x58	Dati sul sistema satelliti	0x38	Quando sono ricevuti nuovi dati sul sistema
0x6D	Lista satelliti, DOPS e modalità	0x24	Quando la lista dei satelliti è aggiornata
0x83	Posizione XYZ(ECEF), doppia precisione	0x37	Quando la posizione fissa è elaborato
0x84	Posizione LLA, doppia precisione	0x37	Quando la posizione fissa è elaborato
0x8F-AB	Primo pacchetto di tempo	-	Uno per secondo
0x8F-AC	Secondo pacchetto di tempo	-	Uno per secondo

L'utente può cambiare il valore dei parametri per ottenere l'operazione desiderata, ma le modifiche attuate non vengono memorizzate automaticamente, per far questo si deve inviare il pacchetto 0x8E-26

direttamente al Resolution T, che provvederà a salvare le modifiche in memoria. Per ripristinare i parametri al valore di default, basta inviare il pacchetto 0x1E al ricevitore.

La struttura dei pacchetti TSIP è la stessa sia per le richieste che per i dati in uscita, ed è di seguito indicata:

<DLE> <id> <data string bytes> <DLE> <EXT>

Dove:

<DLE> rappresenta il byte 0x10

<EXT> rappresenta il byte 0x03

<id> rappresenta il byte che identifica il pacchetto, il quale può avere qualsiasi valore tranne <EXT> e <DLE>.

La *data string* può avere qualsiasi valore. Quando all'interno della stringa sono presenti i valori <DLE><EXT>, si può confondere tra una sequenza di valori all'interno della stringa dati e la sequenza che indica la fine del pacchetto. Per ovviare a questo inconveniente, il protocollo TSIP utilizza il byte stuffing cioè ogni byte <DLE> contenuto nella stringa dati è preceduto da un byte <DLE> extra. Quindi, si può affermare che la fine del pacchetto è rappresentata dal byte <EXT> preceduto da un numero dispari di byte <DLE>. Per cui, una stringa contenente i valori <DLE><EXT> viene trasmessa con il seguente pacchetto <DLE> <id> <DLE> <DLE> <EXT> <DLE> <EXT>.

I pacchetti che contengono tutte le informazioni necessarie da fornire per questo lavoro sono i due pacchetti 0x8F-AB e 0x8FAC (che sono sempre inviati), ed un terzo pacchetto (0x6D) che contiene il numero di satelliti in vista (che si ottiene come risposta del pacchetto 0x24 dalla tabella precedente). I dati utilizzati dal protocollo TSIP sono:

TIPI DI DATO	DESCRIZIONE
UINT8	Numero senza segno di 8 bit
SINT8	Numero con segno di 8 bit
INT16	Numero senza segno di 16 bit
SINT16	Numero con segno di 16 bit
UINT32	Numero senza segno di 32 bit
SINT32	Numero con segno di 32 bit

Di seguito vengono riportate le tabelle della struttura dei tre pacchetti con il significato dei dati:

Pacchetto 0x8F-AB lunghezza 18 byte					
Byte	**Bit**	**Item**	**Type**	**Value**	**Description**
0		Subcode	UINT8		0xAB
1-4		Time of week	UINT32		GPS seconds of week
5-6		Week number	UINT16		GPS week number
7-8		UTC Offset	SINT16		UTC Offset (seconds)
9	0	Timing Flag	Bit field	0	GPS time
				1	UTC time
	1			0	GPS PPS
				1	UTC PPS
	2			0	Time is set
				1	Time is not set
	3			0	Have UTC info
				1	No UTC info
	4			0	Time from GPS
				1	Time from user

10		Seconds	UINT8	0-59	Seconds
11		Minutes	UINT8	0-59	Minutes
12		Hours	UINT8	0-23	Hours
13		Day of Month	UINT8	1-31	Day of month
14		Month	UINT8	1 -12	Month of year
15-16		Year	UINT16		Four digits of year

Pacchetto 0x8F-AC lunghezza 67 byte				
Byte	**Item**	**Type**	**Value**	**Description**
0	Subcode	UINT8	0xAC	
1	Receiver mode	UINT8	0-7	
2	Reserved	UINT8	0-6	Reserved
3	Self-survey progress	UINT8		0-100%
4-7	Reserved	UINT32	0	Reserved
8-9	Reserved	UINT16	0	Reserved
10-11	Minor alarms	UINT16		
12	GPS decoding status	UINT8		
13	Reserved	UINT8	0	Reserved
14	Spare status 1	UINT8	0	
15	Spare status 2	UINT8	0	
16-19	Local clock bias			ns
20-23	Local clock bias rate			ppb
24-27	Reserved			Reserved
28-31	Reserved			Reserved

32-35	Temperature			Degrees C
36-43	Latitude	Double		Radians
44-51	Longitude	Double		Radians
52-59	Altitude	Double		Meters
60-63	PPS Quantization Error			Seconds
64-67	Spare			Future expansion

Pacchetto 0x6D					
Byte	**Bit**	**Item**	**Type**	**Value**	**Meaning**
0	0:2	Fix dimension	Bit field	1 -5	
0	3	fix mode	Bit field	0 -1	Auto-manual
0	4:7	Number of sv's in fix	Bit field	0-12	Count
1-4		PDOP	Single		PDOP
5-8		HDOP	Single		HDOP
9-12		VDOP	Single		VDOP
13-16		TDOP	Single		TDOP
17-n		SV PRN	SINT8	+/-(1-32)	PRN

I dati che ci interessano per il nostro lavoro sono contenuti nel pacchetto 0xF-AB da cui verranno estratti i dati per ottenere i secondi, i minuti, le ore, il giorno,il mese e l'anno corrente cioè i byte dal 10 al 14. Dal pacchetto 0x8F-AC si estrae l'errore di quantizzazione contenute nei byte 60–63. L'errore è un numero a singola precisione nel formato IEEE745, ottenuto da dati sperimentali (Fig.2.13). Per raggiungere la precisione desiderata (due cifre dopo il ns) sono sufficienti i primi tre byte.

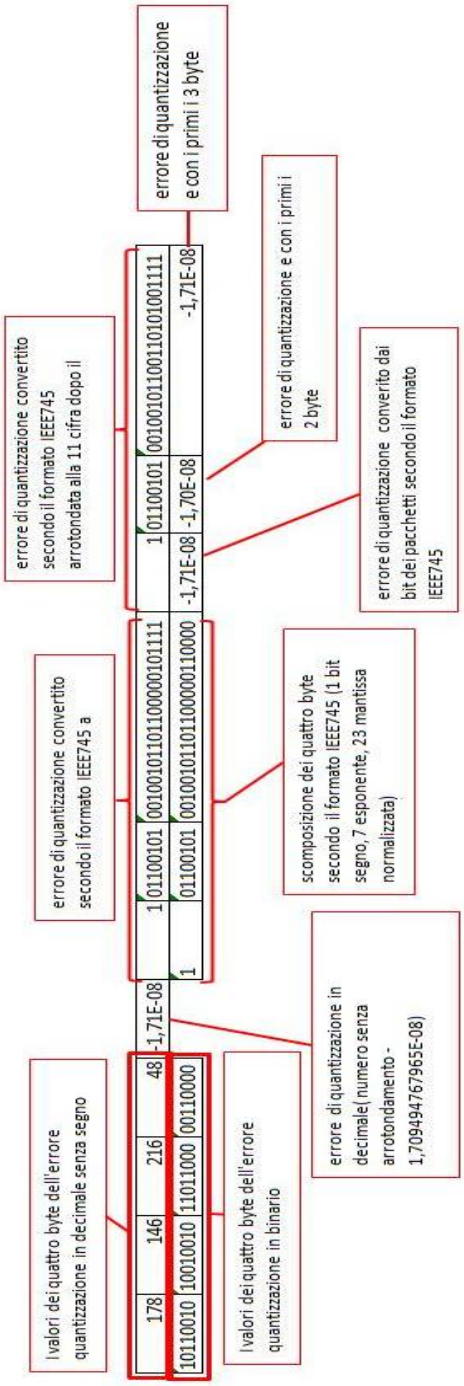


Fig.2. 11: Misura sperimentale dell'errore di quantizzazione

Nel pacchetto 0x6D, inviato dal GPS su richiesta, l'unica informazione necessaria da estrarre riguarda il numero di satelliti connessi che deve essere superiore o uguale a 4 per ottenere dei dati attendibili.

Capitolo 3

CARATTERIZZAZIONE DELLA SCHEDA EMBEDDED

3.1 Sistema embedded e Linux Embedded

I sistemi "embedded" sono dei sistemi elettronici complessi la cui peculiarità sta nel fatto che, pur essendo appositamente progettati per una determinata applicazione possono essere inseriti in sistemi eterogenei, al contrario dei sistemi general purpose che ammettono tipicamente una forte alterabilità da parte dell'utente. Questi sistemi, vengono detti embedded in virtù della loro forte interazione con l'ambiente circostante nel quale sono inseriti [6].

In generale, si può dire che un sistema embedded è la struttura di supporto al funzionamento del sistema eterogeneo e alle sue applicazioni, con un pronunciato livello di interazione con il mondo esterno. Il suo funzionamento può dipendere dall'utente, ma può derivare anche da segnali che provengono dall'esterno dovuti a particolari eventi, nello stesso modo anche il risultato dell'elaborazione non deve essere necessariamente l'output per l'utente, ma può essere anche un segnale o un insieme di segnali per il controllo di altre periferiche.

Pertanto, un sistema embedded, essendo progettato per un numero minimo di applicazioni può essere ottimizzato in modo da avere un basso consumo di potenza e poco ingombro, ma nello stesso tempo, qualora fosse necessario, poter utilizzare un elevato numero di porte I/O.

Un sistema embedded può essere pensato e sintetizzato in tre livelli (Fig.3.1):

- Livello Software Application. Questo livello è opzionale, quando il sistema progettato lo utilizza, contiene tutti i programmi utente;
- Livello Software System, anch'esso è opzionale, la sua presenza dipende sempre dal fine per cui il sistema è progettato e contiene il sistema operativo (S.O.) con tutte le applicazione per poter interrogare e dialogare con le periferiche;
- Livello Hardware, questo livello è sempre presente in ogni sistema embedded e rappresenta la parte fisica del sistema.

Fig.3.1: Livelli del sistema Embedded

Il livello hardware del sistema embedded (Fig3.2) è formato dai seguenti elementi è [6]:

- Processore. Possiede la potenza necessaria per eseguire i suoi compiti ed è l'elemento cardine del sistema;
- Memoria. La capacità di questo elemento è molto importante in un sistema poiché ha una pesante influenza sul programma da

eseguire e sul suo sviluppo. La memoria all'interno del sistema ha due compiti principali:

- Deve essere capace di memorizzare il software che deve eseguire, sia programma utente che S.O., tramite una memoria interna ROM o una esterna (EPROM, FLASH CARD). Per eseguire il S.O. ci deve essere un bootloader iniziale, le cui funzioni principali sono: inizializzazione dell'hardware sulla scheda, caricamento e avviamento del Kernel dalla memoria ROM;
- Deve memorizzare tutti i dati intermedi del programma con i dati iniziali del bootstrap.

- Periferiche. Tramite queste avviene la comunicazione con il mondo esterno sia in input che in output. Le normali periferiche presenti su questi tipi di sistemi sono: porte digitali, analogiche, porte per comunicazione seriale e porta ethernet

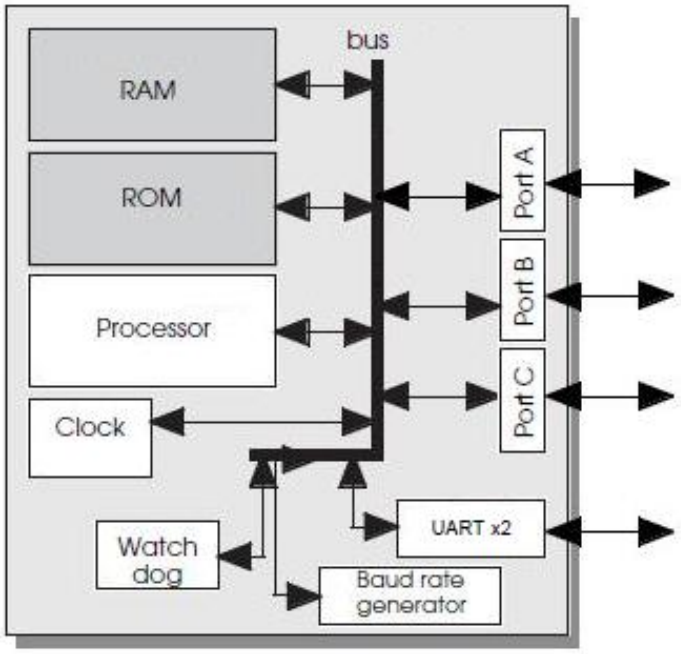


Fig.3.2: Livello Hardware

Il livello Software System, come detto, contiene il S.O. con tutte le applicazione per poter interrogare e dialogare con le periferiche. Nell'ambiente embedded si sono sviluppati vari S.O., ma il più diffuso è Linux in quanto fornisce diversi applicativi e inoltre ha caratteristiche che altri sistemi embedded non forniscono. Linux Embedded è molto vicino alle distribuzioni Linux per desktop ma è adatto per essere usato in specifici casi in quanto deve utilizzare poca memoria e un clock di frequenza inferiore rispetto a quello dei normali computer. La più grande differenza tra linux embedded e linux è la separazione tra il kernel e le applicazioni che si eseguono in user space. In Linux le applicazione si eseguono in un livello completamente diverso dal kernel, e non c'è un metodo per accedere a livello kernel, ciò per dare una maggiore protezione al sistema, cosa invece possibile nei sistemi embedded. Linux embedded non è solo il kernel (che è lo stesso delle versioni desktop) ma ha anche altre applicazioni (SSH, GCC,…). I componenti che formano il sistema Linux Embedded sono i seguenti

- Boot loader. Il suo compito principale è quello di caricare dalla memoria non volatile alla RAM il kernel ed eseguirlo;
- Kernel. È Il software che controlla l'hardware e i processi;
- Root file system. La cartella che contiene i programmi eseguiti dal kernel, questa è presente in tutte le distribuzioni di linux;
- Applicazioni. Le applicazioni create dall'utente e che il sistema deve eseguire.

Tutte questi componenti interagiscono fra di loro per eseguire i compiti necessari al funzionamento del sistema.

3.2 La FOX BOARD

La FOX Board (Fig.3.3) prodotta dall'Acmesystems è una CPU Linux Embedded di ridotte dimensioni con un microprocessore ETRAX 100LX, a 100MIPS RISC CPU costruito dalla Axis Communications.

Fig.3.3: Fox Board

Essa possiede:

- Due porte USB 1.1 Host;
- Una porta Ethernet;
- 80 pin di interconnessione di cui 48 per porte di I/O (GPIO), 3 porte seriali asincrone (una di queste può essere resa anche sincrona), I^2C Bus, 2 porte parallele;
- 3 led. Il led verde (DL3) indica che è presente l'alimentazione, il rosso (DL1) può essere usato dall'utente e il giallo(DL2) che indica l'attività di rete (gli ultimi due hanno anche l'estensione sul socket J7 (Fig.3.4), quindi utilizzabili anche come porte di output).

Le caratteristiche tecniche della FOX sono riportate nella seguente tabella:

CARATTERISTICHE HARDWARE	
Dimensioni	66 x 72 mm
Cpu	100MIPS Axis ETRAX 100LX 32 bit, RISC, 100MHz
Memory	4MB FLASH 16MB RAM
Alimentazione	5 Volt 280mA (1 watt)
Porte	1 Ethernet (10/100 Mb/s), 2 USB 1.1, 1 serial console port (J10)
Estensioni	IDE, SCSI, porte seriali e parallele, I^2C
Peso	37 gr
Temperature range	0-70 °C
CARATTERISTICHE SOFTWARE	
Kernel	Up to version 2.6.19
Server	HTTP (WEB), FTP, SSH, TELNET
SDK	Open Source
Linguaggi	C, C++, PHP, PYTHON, etc

Tutte le periferiche e controlli sono memory mapped, ossia esiste almeno un indirizzo fisico di memoria (registro) per ogni periferica nel quale ogni bit assume un particolare significato relativamente al funzionamento della periferica stessa.

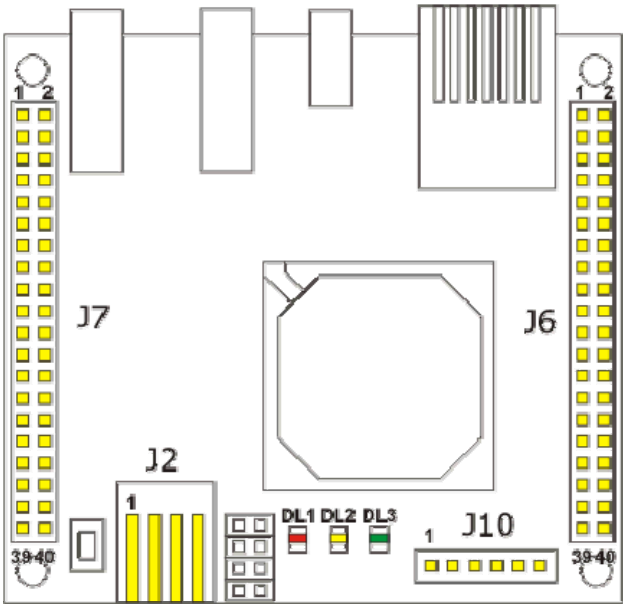


Fig.3. 4: Denominazione connettori della FOX BOARD

Il microprocessore montato sulla FOX BOARD, come già accennato, è l'AXIS ETRAX 100LX (Fig.3.5) che dispone di un supporto per Universal Serial Bus 1.1. L'ETRAX 100LX chip ha incorporato l'AXIS CRIS (Code Reduced Instruction Set) CPU che mette a disposizione il S.O..

L'ETRAX 100LX ha una 100 MIPS RISC CPU, 8 kilobyte di memoria cache per istruzioni e dati e un controllore dei DMA e delle porte I/O. Ha inoltre un basso consumo di potenza, molto importante per i sistemi embedded. L'interfaccia bus programmabile del processore supporta sia i bus dati da 16bit che quelli da 32 bit e può essere interfacciata direttamente con SDRAM, EPROM, EEPROM e FlashPROM.

Il processore è disegnato specificamente per Linux con il supporto MMU, inoltre può essere equipaggiato[2] di

- Controllo Ethernet 100Mbit/10Mbit MII (Compatible with IEEE 802.3 and Fast Ethernet standards);
- 4 porte seriali asincrone con baudrate programmabili (J6,J10);
- 2 porte seriali sincrone;
- Universal Serial Bus 1.1 Host and Device mode operation;
- 16-bit porte I/O, in cui la direzione può essere definita;
- 8 kilobyte di memoria cache;
- Linguaggio C/C++;
- Basso consumo di potenza tipicamente 350 mW.

Il microprocessore è provvisto ha un clock interno che viene impulsato con PLL da un clock esterno. Il clock esterno provvede al vettore degli interrupt interni e esterni.

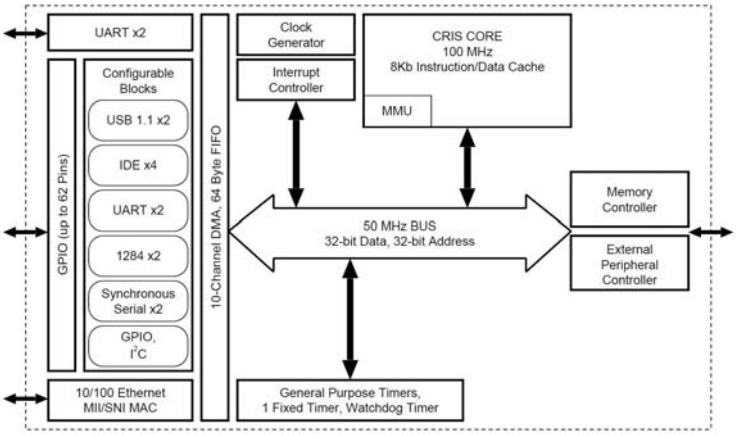


Fig.3.5: Schema a blocchi del processore ETRAX 100LX

[2] tra parentesi sono indicati i nomi dei connettori riportati in Fig.3.4 a cui si farà riferimento

L'AXIS CRIS CPU non ha bisogno di un BIOS che esegua o fornisca le funzioni di basso livello e informi il microprocessore in merito alla quantità di memoria del sistema, in quanto ha già dei bus che gli permettono di collegarsi alla memoria.

La memoria flash della FOX BOARD contiene l'intero sistema, i dati e lo spazio su cui può operare direttamente, ha due partizioni (Fig.3.6): la prima è di tipo Cramfs[3] con accesso di tipo read- only; la seconda è di tipo jffs2[4] con accesso di tipo read/write.

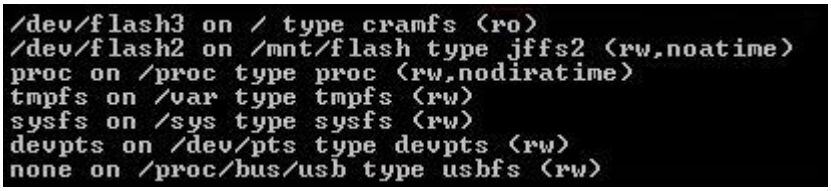


Fig.3.6: Partizioni presenti nella FOX BOARD

Si noti che il sistema non ha il boot e non è neanche presente l'immagine del kernel, poiché durante la fase di compilazione del kernel viene compilato il codice che ne esegue il suo caricamento. Il codice completo andrà ad analizzare la tabella delle partizioni per trovare la partizione bootable con la quale sarà possibile decomprimere e caricare nella RAM la zImage, e cioè l'immagine del kernel compressa. Una volta che il kernel è caricato in RAM il sistema è utilizzabile.

Le porte GPIO indicano le porte General Purpose Input/Output, ossia porte di ingresso/uscita di uso generale. La FOX BOARD possiede tre porte che vengono indicate con A, B e G, le prime due composte da 8 pin e l'ultima da 39 pin dei quali solo 29 utilizzabili sia per l'input che l'output. A queste

[3] *Compressed RAM File System* è un file system compresso disponibile per Linux di tipo read-only progettato per ottenere una gestione dello spazio il più efficiente possibile.

[4] Journaling Flash File System 2 è un Log-structured File System (scrive in blocchi adiacenti a differenza dei normali file system) utilizzato nei dispositivi provvisti di memorie flash

porte corrispondono 8 pin sul processore per le prime due e 32 per la porta G. Lo stato e la tipologia di funzionamento sono indicati in diversi registri (esaminati in seguito perché utili per la programmazione in kernel space). Queste porte sono di tipo digitale tri-state.

Nel progetto da realizzare la FOX BOARD deve eseguire calcoli in virgola mobile; la scheda emula le operazioni in virgola mobile e non le esegue a livello hardware ne consegue che i tipi in virgola mobile non possono essere usati in kernel-space.

3.3 Accesso alle porte I/O in User-space

L'accesso alle porte I/O in User-space può essere eseguito fondamentalmente in due modi: il primo utilizza il tipico metodo di astrazione dei files in UNIX, cioè l'accesso alla porta avviene tramite l'apertura di un file di dispositivo (che sono dei file tramite i quali il kernel permette la comunicazione con le unità fisiche) specifico (/dev/gpioA, /dev/gpioB, /dev/gpioC), mentre il secondo modo permette l'accesso tramite delle chiamate di sistema opportune [9].

Il primo modo permette, come detto, di accedere alla porta mediante file di dispositivo presente nella cartella /dev. Su questi file possono essere fatte tutte le operazioni ragionevoli per la manipolazione dei file (non sono state implementate perché operazioni inutili, del tipo fseek o l'apertura append).

È possibile astrarre una porta digitale tramite un file di dispositivo a caratteri; le porte sono accessibili indirizzando 8 bit per volta. Per controllare le porte è opportuno usare la chiamata di sistema ioctl che permette di leggere i singoli bit sia in input che in output (*IO_READBITS* o IO_READ_INBITS e IO_READ_OUTBITS), di definire lo stato di uno o più bit

necssario ad impostare una porta sul processore al valore logico ‘1’ (IO_SETBITS), di resettare lo stato di uno o più bit e cioè impostare una porta sul processore al valore logico ‘0’ (IO_CLRBITS) e di permettere di leggere la direzione del pin, cioè controllare se il pin è configurato come input o output (IO_READDIR o IO_SETGET_INPUT e IO_SETGET_OUTPUT).

Per utilizzare questo metodo si deve aprire il file creando un descrittore con la funzione open (*fd = open("/dev/gpioB", O_RDWR*), il primo campo indica il file di dispositivo della porta), e poi usare la funzione ioctl(*ioctl(fd,_IO(ETRAXGPIO_IOCTYPE,IO_SETBITS),1<<6*); questa istruzione non fa altro che settare a livello logico alto il terzo bit della porta A).

Questo metodo però ha uno svantaggio, quello di dover accedere ogni volta al file per poter modificare lo stato della porta ed eseguire il passaggio tra User-space e kernel-space perciò il metodo risulta leggermente lento (successivamente se ne spiegherà il motivo), infatti eseguendo un semplice loop infinito (come da codice sottostante) in cui viene ripetutamente cambiato lo stato logico un pin ed acquisendo il segnale (Fig.3.7) è possibile valutare sperimentalmente la frequenza di commutazione.

```
for (i=0;i<20;i++)
{

      ioctl(("/dev/gpioB",_IO(ETRAXGPIO_IOCTYPE,IO_SETBITS),
      1<<6);
      ioctl(("/dev/gpioB",_IO(ETRAXGPIO_IOCTYPE,IO_CLRBITS),
      1<<6);

}
```

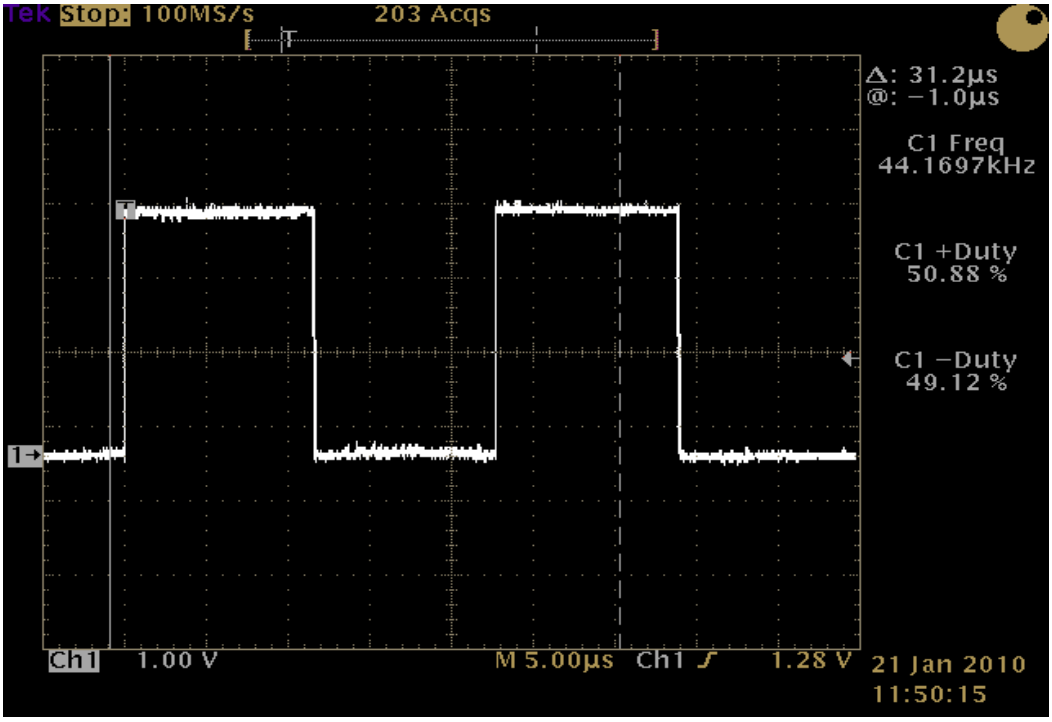


Fig.3.7: Segnale ottenuto con ioctl

Dalla figura si nota che il Duty Cycle del segnale è circa del 50%, e la frequenza è pari a 44,2 kHz.

Il secondo metodo usa le chiamate di sistema dirette (syscall è il meccanismo usato da un programma a livello utente per richiedere un servizio a livello kernel del sistema operativo) che permettano l'accesso della porta specificata. Per poter usare questo metodo con la FOX BOARD si deve utilizzare l'SDK (Software Development Kit, sono un insieme di strumenti per lo sviluppo e la documentazione di software). La libreria in cui sono definite le chiamate di sistema è linux/gpio_syscalls.h, le funzioni di questa libreria permettono l'accesso diretto al kernel senza dover ricorrere all'apertura del file associato al dispositivo che occorre nei file di dispositivo. La libreria permette di definire la direzione in input (DIRIN) e in output (DIROUT) (gpiosetdir), di definire un bit o un gruppo di bit a livello logico alto (gpiosetbits), di resettare un bit o un gruppo di bit a livello logico basso (gpioclearbits) e di effettuare il toggle del bit o del gruppo dei bit

(gpiotogglebit). Nella stessa libreria vengono definiti i nomi delle porte (PAx, PBx, PGx con x il numero del bit; i bit 8-15 e 16-23 non possono essere indirizzati come input o output singolarmente ma a gruppo PG8_15 e PG16_23).

Come fatto per il primo metodo e utilizzando la stessa porta si va a verifica la frequenza(Fig.3.8) di commutazione del pin con il seguente ciclo:

```
while(1)
{
      gpiosetbits(PORTB, PA6);
      gpiosetbits(PORTB,PA&);
}
```

Il segnale osservato con l'oscilloscopio è il seguente:

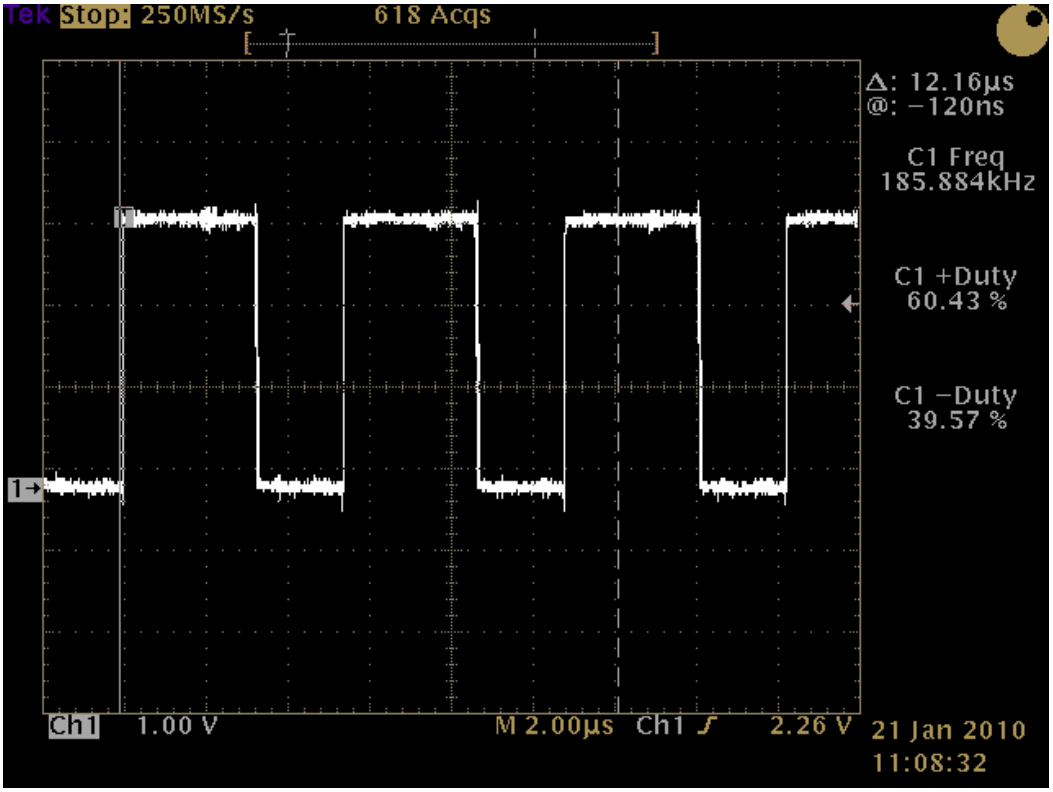


Fig.3. 8: Segnale ottenuto tramite syscall

Si nota che la frequenza pari a 185,9 kHZ, come già accennato, è molto superiore rispetto al primo metodo ma il duty cycle è diverso del 50% e non dipende dalla sequenzialità delle istruzioni. La differenza di frequenza fra i due metodi è dovuta al fatto che una chiamata di sistema, relativa a un file di dispositivo (ioctl) fa passare il processo da user-space a kernel-space e la chiamata al kernel viene fatta tramite un interrupt software il quale, normalmente, richiede molti più cicli di clock soprattutto quando è necessario salvare il precedente contesto del processo. Tutto questo degrada le performance del sistema [9].

Successivamente, ripetendo il procedimento fatto in precedenza, si è proceduto a verificare la frequenza (Fig.3.9) di commutazione del sistema, andando a variare un gruppo di bit con il seguente codice

```
while(1)
{
      gpioclearbits(PORTG, PG8_15);
      gpiosetbits(PORTG, PG8_15);
}
```

Il risultato ottenuto (Fig.3.9), è che la frequenza diminuisce leggermente rimanendo sempre superiore a quella riscontrata nel primo metodo, mentre il duty cycle si mantiene uguale:

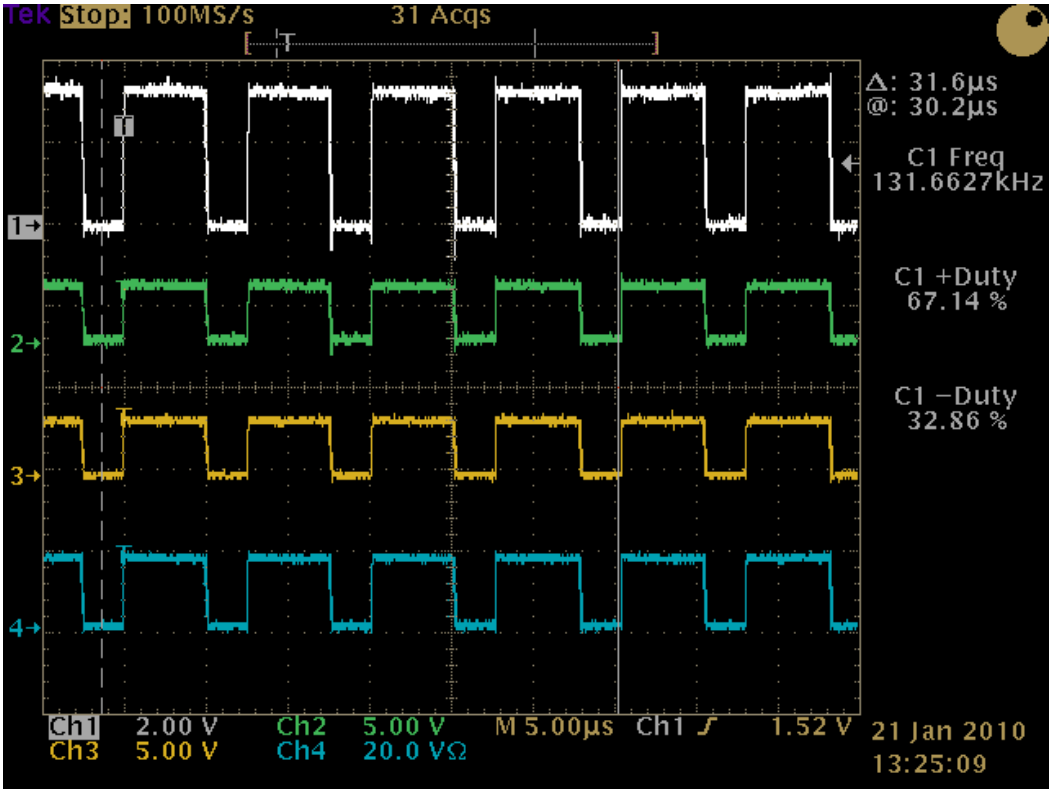


Fig.3. 9: Segnale ottenuto definendo un gruppo di bit

Essendo, quest'ultimo metodo quello più performante, tramite esso si può verificare il ritardo di commutazione fra due o più istruzioni successive e analizzare come varia la frequenza (Fig.3.10 e Fig.3.11) con il seguente codice:

```
while(1)
{
	gpioclearbits(PORTG, PG8);//segnale blu
	gpioclearbits(PORTG, PG11);//segnale grigio
	gpioclearbits(PORTG, PG12);//segnale verde
	gpioclearbits(PORTG, PG15);//segnale arancione
	gpiosetbits(PORTG, PG8);
	gpiosetbits(PORTG, PG11);
	gpiosetbits(PORTG, PG12);
	gpiosetbits(PORTG, PG15);
}
```

Si ottengono le seguenti misure:

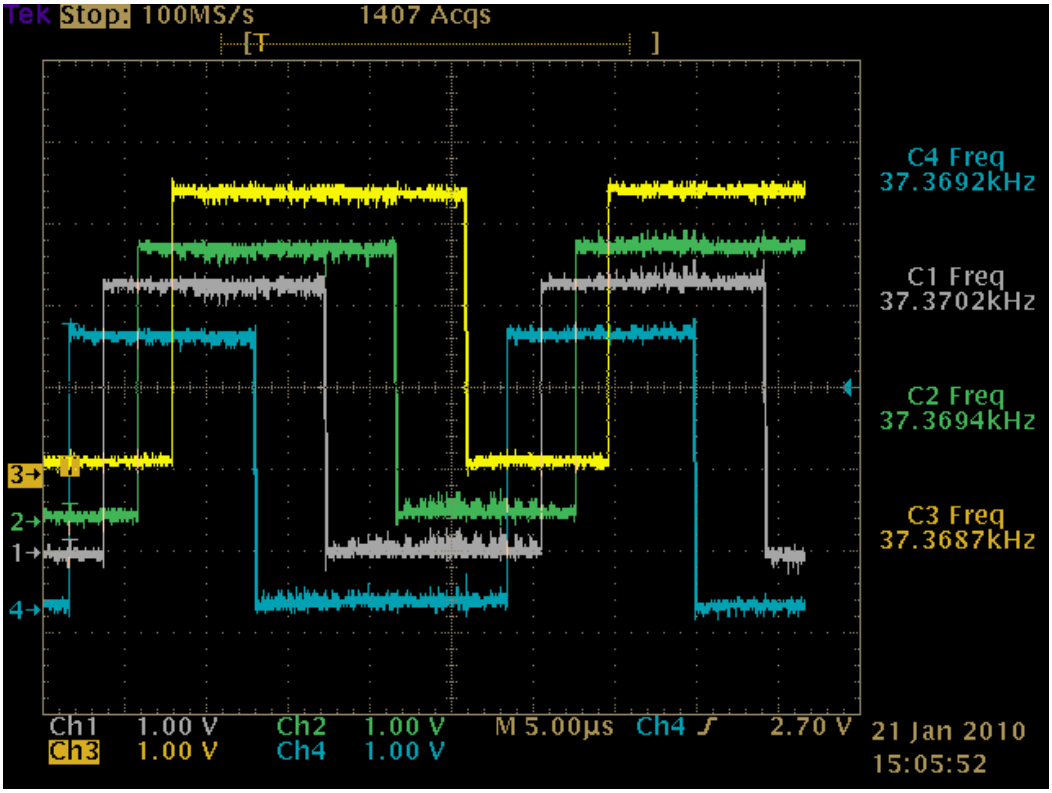


Fig.3. 10: Segnali ottenuti per verificare il ritardo di commutazione

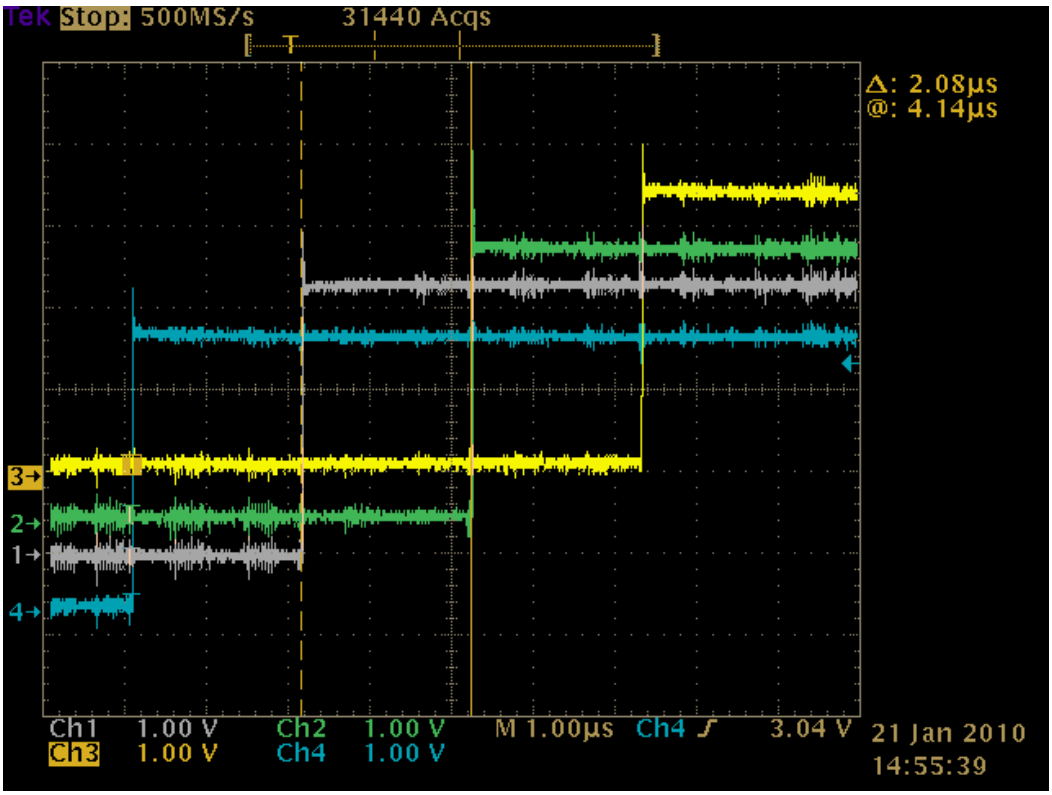


Fig.3. 11: Ingrandimento per valutare il ritardo di commutazione

Il ritardo fra due istruzioni successive è costante, ma la frequenza cala drasticamente molto probabilmente perché i bit utilizzati sono i bit che possono essere anche utilizzati in gruppo.

Siccome la FOX BOARD deve acquisire i dati da dispositivi esterni e, per far sì che essa li acquisisca in modo corretto, si deve verificare per quanto tempo i dispositivi devono mantenere in uscita il segnale da inviare. Utilizzando il pattern generation è stato creato un segnale (Fig.3.12:Traccia di colore bianco) con impulsi di durata crescente che partendo dall'impulso di durata minore pari a 1μs arrivano all'impulso di durata maggiore pari a 10 μs. Per fare questa misura la FOX BOARD esegue un loop infinito nel quale legge la porta in input e scrive senza fare alcuna operazione sulla porta di output:

```
while(1)
{
      (gpiogetbits(PORTG,PG2))?(gpiosetbits(PORTG,PG8))
                              :(gpioclearbits(PORTG, PG8));
};
```

Acquisendo il segnale in uscita (Fig.3.12 Traccia di colore verde) tramite oscilloscopio si rileva che vengono meno gli impulsi di durata inferiore, cioè quelli da 1 μs a 2 μs (alcune volte manca anche quello da 3 μs), questo è dovuto al fatto che gli impulsi sono di durata molto breve. Quindi per essere sicuri che il segnale venga letto in maniera corretta, tramite le syscall dalla FOX BOARD, il dispositivo deve assicurare lo stato del segnale per almeno 5 μs. Si rileva inoltre che, il segnale in uscita non ha la stessa durata del segnale d'ingresso, questo a causa della diversa risoluzione dell'accesso alle porte.

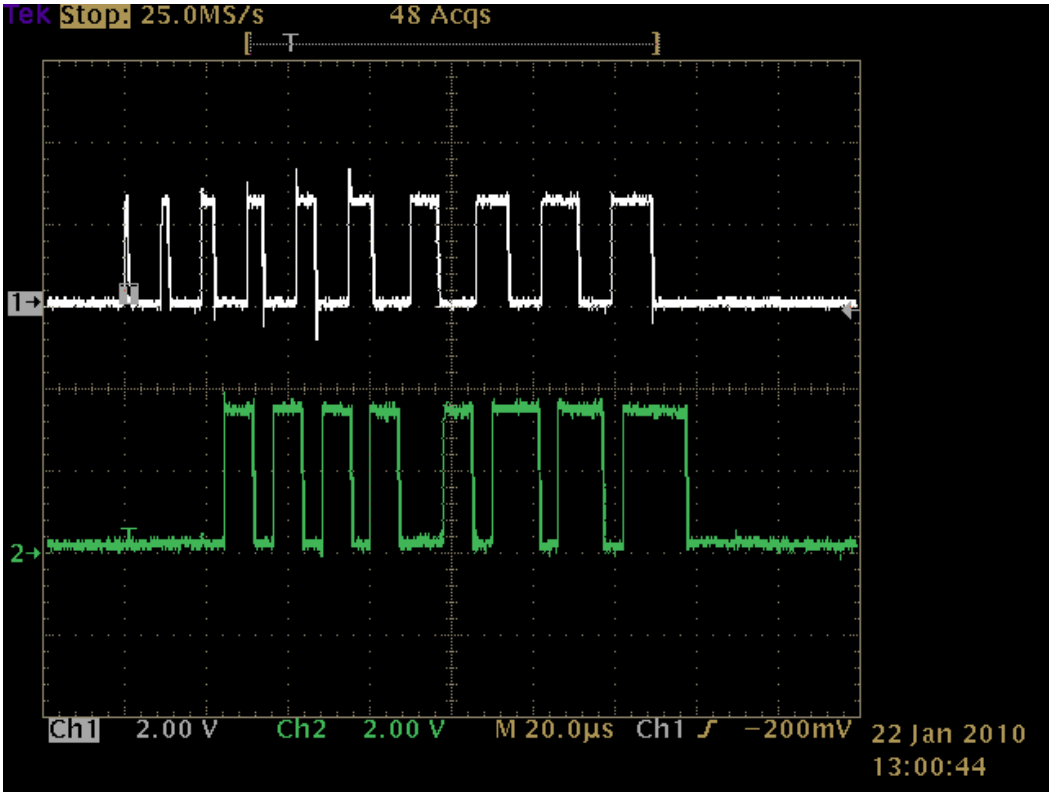


Fig.3. 12: Traccia bianca impulsi di durata crescente e la traccia verde è il risultato dell'acquisizione.

3.4 Registri delle porte I/O

I registri delle porte I/O servono per utilizzare le stesse in kernel space; ad ogni porta corrispondono dei registri diversi. Per la porta A il registro fondamentale è il R_PORT_PA_SET, un registro di sola scrittura a 32 bit di cui solo i 16 più bassi sono accessibili mentre gli altri sono riservati, i 16 bit accessibili sono così suddivisi: 8bit (quelli più bassi dei 16 bit) per il registro R_PORT_PA_DATA che è di sola scrittura e serve per fare l'output, gli altri 8 bit per R_PORT_PA_DIR, un registro di sola scrittura che setta la direzione delle porte (0 output, 1 input). Un altro registro utilizzato è l'R_PORT_PA_READ, un registro di sola lettura a 32 bit di cui solo gli 8 bit più bassi sono accessibili e servono ad acquisire i valori di input.

La struttura è simile per la porta B in quanto anche essa ha il registro R_PORT_PB_SET sempre di sola scrittura a 32 bit, tutti accessibili e così suddivisi: 8 bit (quelli più bassi) per il registro R_PORT_PB_DATA che è di sola scrittura e serve per l'output; i successivi 8 bit per R_PORT_PA_DIR che è un registro di sola scrittura che definisce la direzione delle porte; gli ultimi 16 occupati da due registri a 8 bit di sola scrittura(non verranno usati per questo progetto) e sono R_PORT_PB_CONFIG (che serve a configurare il comportamento dei pin condiviso con le porte seriali e parallele) e il registro R_PORT_PB_I2C. Infine il registro R_PORT_PB_READ che è un registro di sola lettura a 32 bit, di cui solo gli 8 bit più bassi sono accessibili e servono ad acquisire i valori di input.

La porta G ha una struttura diversa dalle precedenti: ha il registro R_PORT_G_DATA che è un registro di scrittura e lettura a 32 bit per cui viene usato sia come input che come output. Per definire la direzione dei pin della porta si usa il registro R_GEN_CONFIG, un registro a 32 bit, ma per configurarne la direzione vengono usati i bit 24(pin 0), 25(pin 8-15), 26 (pin 16-23), 27(pin 24). Gli altri bit della porta G hanno due pin sul processore uno per l'input e uno per l'output. Per vedere la struttura dei registri che si useranno si faccia riferimento all'appendice B.

3.5 Accesso alle porte I/O in Kernel-space

A livello kernel l'accesso alle porte I/O è semplice in quanto è uguale a quello che si effettua per avere accesso alla memoria o a quello che programma le periferiche memory mapped di un microcontrollore, infatti non si deve fare altro che modificare il valore del registro interessato (come descritto brevemente nel paragrafo precedente tramite operazioni bitwise).

Per effettuare la programmazione in kernel-space si devono utilizzare i moduli: sezioni di codici che si affiancano al kernel e che possono essere

compilate, caricate e scaricate indipendentemente dal resto del kernel. La loro peculiarità è la capacità di espandere le sue funzionalità a run-time (per la compilazione e il caricamento di un modulo si faccia rifermento all'appendice C).

Come per l'User-space, anche qui si sono effettuate delle misure necessarie a stabilire la funzionalità del sistema per dimensionarlo in maniera opportuna. La prima misura riguarda la frequenza di commutazione dell'uscita. Analogamente a quanto fatto in user-space, si esegue un loop infinito che va a variare lo stato di un pin (si noti e l'utilizzo degli operatori bitwise), tramite l'oscilloscopio si acquisisce il segnale(Fig.3.13) corrispondente:

```
for (;;)
{
        *R_PORT_PB_DATA=*R_PORT_PB_DATA|128;
        *R_PORT_PB_DATA=*R_PORT_PB_DATA&127;
}
```

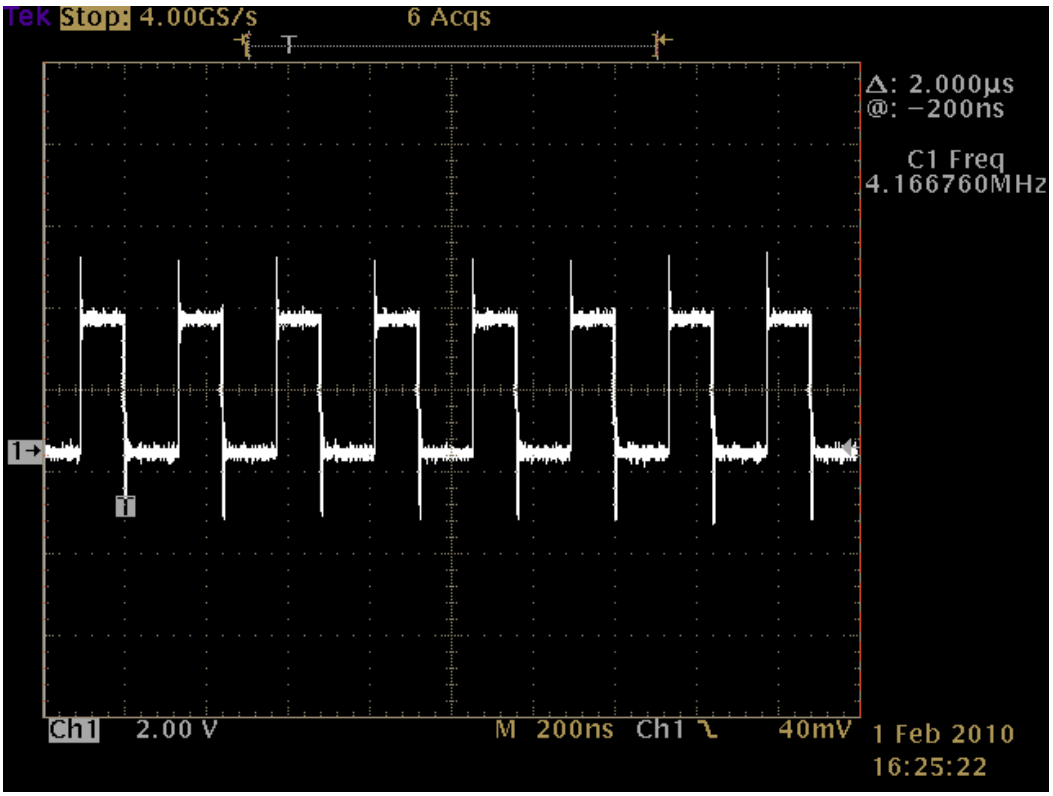


Fig.3. 13: Segnale per valutare la frequenza di commutazione

Rispetto alla stessa operazione eseguita in User-space la frequenza, pari a 4,166 MHz, è notevolmente maggiore con un guadagno di fattore pari a 21. Il fatto che la programmazione in kernel-space fa guadagnare in frequenza non deve sorprendere in quanto, in questo modo, non si deve effettuare l'interrupt per il passaggio tra lo spazio utente e kernel necessario per l'accesso alle porte in user-space.

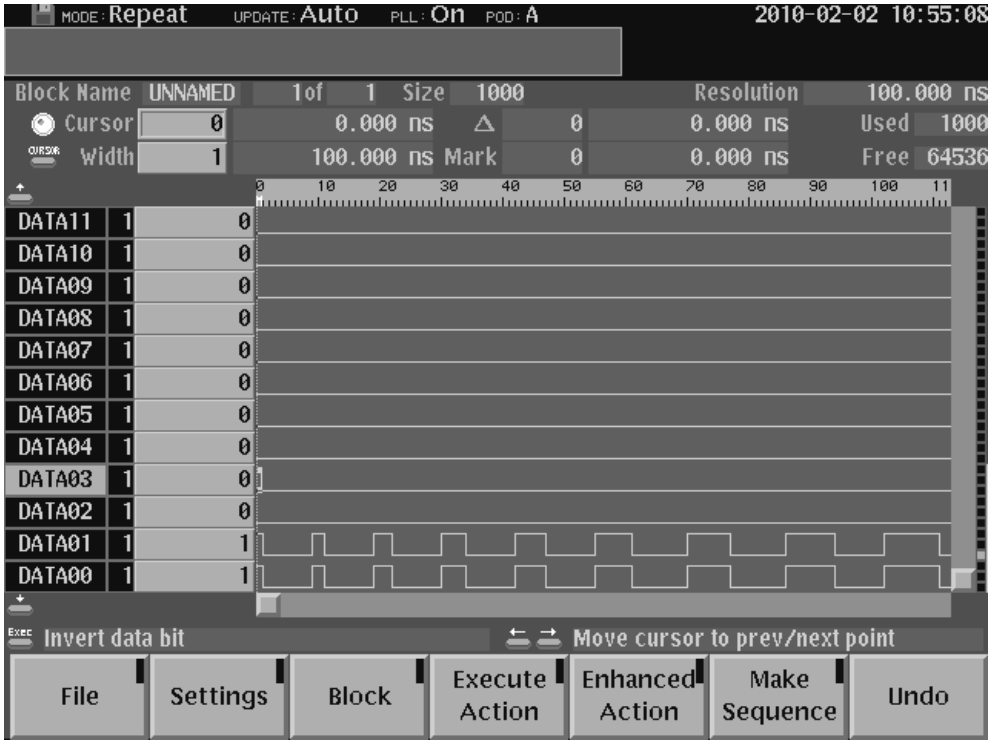


Fig.3. 14: Treno d'impulsi generato

Si è proceduto a verificare la risoluzione delle porte per calcolare il tempo minimo necessario affinché un dispositivo esterno mantenga il proprio segnale stabile (senza cambiamento di stato) senza che la FOX BOARD perda alcuna informazione. Procedendo in maniera simile a quanto fatto in user-space, si esegue un loop infinito per leggere il valore della porta in input e stamparlo senza eseguire alcuna operazione. Il segnale in input(Fig.3.15: Traccia di colore bianco), generato dal pattern generation, è un treno d'impulsi (Fig.3.14) a durata crescente, parte dall'impulso di durata minore pari a 100ns e arriva all'impulso di durata maggiore pari a 1µs.

```
while(1)
 {
   (*R_PORT_PB_DATA&16)?(*R_PORT_G_DATA|=(1<<2))
                       :(*R_PORT_G_DATA&=~(1<<2));
  }
```

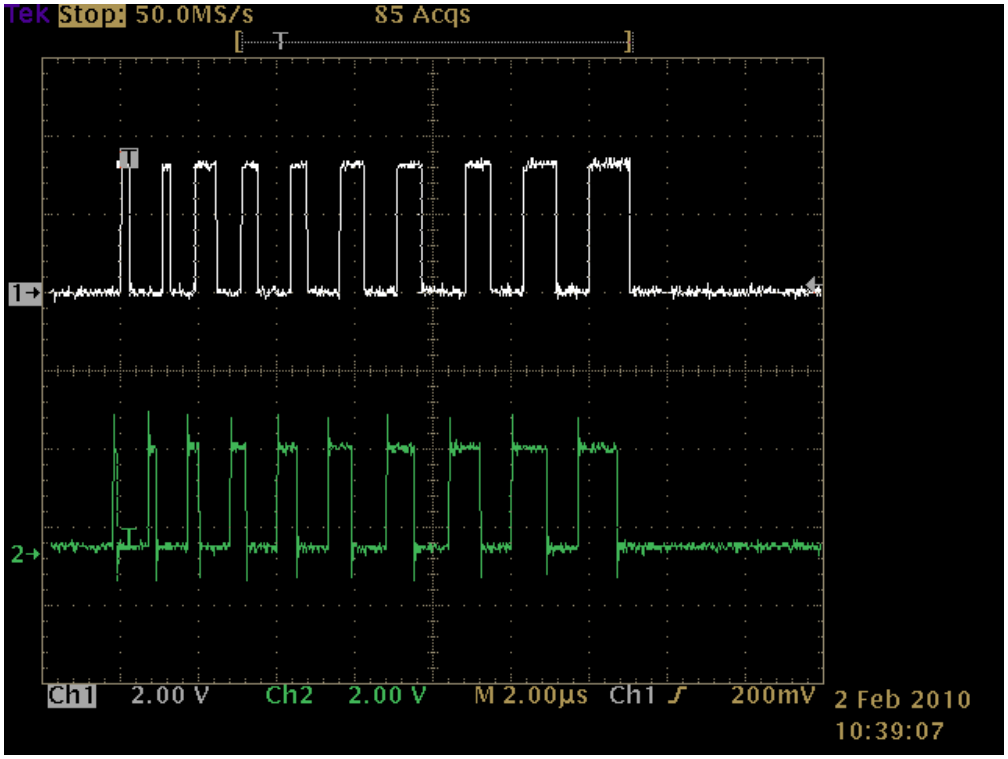


Fig.3. 15: Traccia bianca impulsi di durata crescente e la traccia verde è il risultato dell'acquisizione.

Acquisito, tramite oscilloscopio, il segnale in uscita (Fig.3.15: Traccia di colore verde) si nota che, tramite la programmazione in kernel-space si riesce ad acquisire anche l'impulso di durata inferiore di 100ns e il ritardo di elaborazione del segnale da parte della FOX BOARD è molto ridotto, come si nota dalla figura segiente (Fig.3.16).

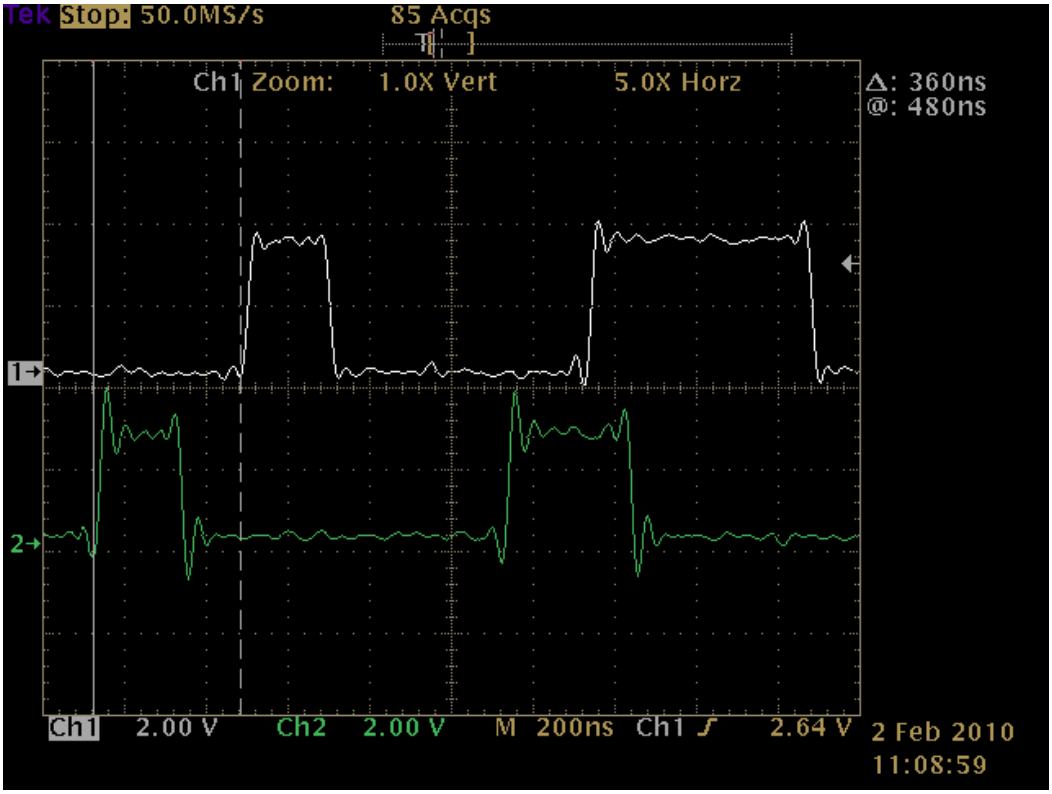


Fig.3. 16: Figura per valutare il ritardo di elaborazione

3.6 Comunicazione seriale

Le porte seriali della fox board sono quattro, una riservata come porta di debug di default che può essere usata anche come normale porta seriale attraverso una modifica del kernel (come verrà spiegato in seguito), questa porta può essere utilizzata usando il file (/dev/ttyS0) disponibile solo attraverso il connettore J10; un ulteriore porte seriale indicata con il file (/dev/ttyS1) che ha il bus in comune con la porta USB1; le ultime due porte seriali (/dev/tty2 e /dev/tty3) sono completamente disponibili sul connettore J6 e una di queste può essere configurata anche come RS485 o porta seriale sincrona. La porta di debug di default ha un baudrate pari a 115200 baud, 8 bit di dati, nessun bit di parità e un bit si stop.

La porta seriale di debug generalmente non è usata come una normale porta seriale, come avviene in questo progetto, in quanto essa ha la

funzione di debug, invia i messaggi dal kernel (i messaggi di decompressione del kernel e di avvio dell'intero sistema) e i messaggi syslogd. Per poter utilizzare la /dev/ttyS0 come una normale porta seriale si deve disabilitare la funzione di debug e l'invio dei messaggi. Per disabilitare la funzione di debug si deve modificare il kernel (Appendice A) per cui, una volta avviato make kernelconfig dalla sezione Hardware setup, disabilitare la voce Product debug-port. Invece, per disabilitare i messaggi di sistema, partendo dalla directory in cui si è installata la SDK, portarsi nella sotto directory packages/param/syslog/development-R2_01/, editare il file syslog.conf e settare a commento la prima riga (Fig.3.17) oppure, reindirizzare l'output verso il dispositivo /dev/null (Fig.3.18) (uguale risultato si può ottenere manipolando direttamente dalla FOX BOARD lo stesso file, sotto la directory /etc/, unica differenza è che ad ogni caricamento della fImage del kernel si deve rieseguire l'operazione).

```
# Log messages to the console.
#*.debug;kern.none                                    /dev/ttyS0
```

Fig.3.17:Porta di debug commentata

```
# Log messages to the console.
*.debug;kern.none                                     /dev/null
```

Fig.3.18: Porta di debug indirizzata verso /dev/null

Per eliminare i messaggi dal kernel, nella directory os/linux-2.6/arch/cris/arch-v10/boot/compressed/ cancellare il file misc.o (questa operazione deve essere fatta tramite shell con il commando rm in quanto dall'ambiente grafico non si hanno i permessi necessari alla modifica). Ricompilare, successivamente, l'immagine del kernel e trasferirla sulla FOX BOARD (appendice A), in questo modo la porta di debug diventa una normale porta seriale.

La porta seriale è gestita come un dispositivo a caratteri, data le loro peculiarità linux gestisce le caratteristiche di questi dispositivi con POSIX (è un progetto finalizzato alla standardizzazione delle API per i software sviluppati per le diverse versioni di Unix). Lo stesso POSIX fornisce un interfaccia standard per l'interrogazione e manipolazione del dispositivo, in questo caso la porta seriale. Molte delle funzioni di POSIX utilizzano il file descriptor assegnato al terminale mediante la funzione open, quindi per un corretto utilizzo si deve prima aprire la connessione con il dispositivo (il codice della seriale è riportato nell'appendice E con l'intero codice della FOX BOARD).

Una volta aperta la connessione con il dispositivo conviene salvarne le caratteristiche per poi ripristinarle a lavoro ultimato, in quanto in itinere si andranno a modificare alcune importanti caratteristiche della porta. Per salvare le caratteristiche della porta si userà una struttura termios, e successivamente un'altra struttura per impostare i nostri parametri. La struttura termios ha diversi parametri, 4 sono dei flag realizzati come maschera binaria che controllano il comportamento del terminale, di questi 4 flag, per l'impostazione della porta seriale si utilizzerà:

- Il c_cflag o flag di controllo, permette di impostare il numero di bit del dato, il numero di bit di stop, le impostazioni della parità, il funzionamento del controllo di flusso.

- Il c_lflag o flag locale, serve per controllare il funzionamento dell'interfaccia fra il driver e l'utente, come abilitare l'eco, gestire i caratteri di controllo e l'emissione dei segnali, impostare il modo canonico o non canonico (nel modo canonico i caratteri d'ingresso restano in coda finché il terminale non avrà finito di leggere una linea d'input, i dati non letti la prima volta non saranno persi ma rimarranno in coda e la gestione del driver gestirà direttamente l'input, invece nel modo non canonico

bisogna gestire tutte le eccezioni. Esistono però due specifiche principali, come vedremo VMIN e VTIME che informano il sistema di ritornare da una lettura, quando è stata acquisita una determinata quantità di dati o quando è passato un certo tempo.)

Tramite la struttura termios si possono gestire anche i caratteri speciali associati alle varie funzioni di controllo. In particolare, per il modo non canonico utilizzato nel progetto, si possono impostare:

- VMIN: Numero minimo di caratteri per una lettura;
- VTIME: Timeout, in decimi di secondo, per una lettura.

Di seguito vengono riportate le impostazioni della porta seriale da noi utilizzate (dove new è una struttura termios):

```
new.c_cflag |= CREAD; /*permette di leggere l'input del terminale,
                        altrimenti i caratteri in ingresso sono
                        scartati.*/

new.c_cflag |= B9600; //imposta il baudrate.

new.c_cflag |= CS8; //dimensione del carattere inviato.

new.c_cflag |= PARENB; //abilita il controllo di parità.

new.c_lflag &= ~(ICANON); //disabilità il modo canonico.

new.c_lflag &= ~(ECHO); /*disattiva l'eco dei caratteri in input
                          sull'output del terminale.*/

new.c_lflag &= ~(ECHOE); // disattiva la funzione del tasto erase.

new.c_lflag &= ~(ISIG); /*disabilita il riconoscimento dei
                           caratteri INTR, QUIT, e SUSP*/

new.c_cc[VMIN]=1;
```

```
new.c_cc[VTIME]=0;
```

Una volta impostata, la seguente struttura deve essere memorizzata tramite la funzione tcsetattr con il parametro TCSANOW in maniera tale che tutti i cambiamenti alle caratteristiche siano immediati. (il codice è incluso nel codice della FOX BOARD in appendice E).

3.7 Socket Netlink

Per l'acquisizione dati dall'esterno, come già visto, si hanno le migliori performance nel kernel-space, poiché in questa applicazione è necessario eseguire operazioni in virgola mobile che il kernel space non gestisce direttamente ma solo se emulate tramite software in user space. Pertanto è richiesta l'identificazione di un'interfaccia adatta per il trasferimento delle informazioni tra kernel-space e user-space. Le interfaccie più comuni sono file di dispositivo, chiamate di sistema, sistemi virtuali e il socket Netlink. I socket sono uno dei principali meccanismi di comunicazione fra processi utilizzati in ambito Unix, il socket (Fig.3.19) costituisce in sostanza un canale di comunicazione fra due processi su cui si possono leggere e scrivere dati analoghi a quello di una pipe sia kernel-space che user-space.

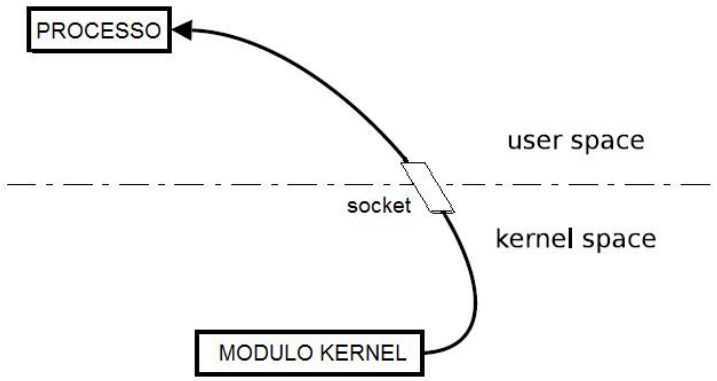


Fig.3. 19: Funzionamento del socket

I socket Netlink permettono di instaurare una comunicazione asincrona full-duplex, tra i vari processi a cui è collegato il socket; a differenza di altri tipi di socket il Netlink non permette di creare diverse comunicazione dello stesso tipo e mantenerle, in linea di principio, completamente separate.

Il socket Netlink discrimina i vari socket aperti semplicemente con un numero nel suo header, dopo l'header i dati potranno essere formattati a piacimento. I messaggi possono essere sia unicast (diretto solo a un processo) o broadcast.

Sorge pertanto la necessità di creare il socket Netlink a livello user-space e kernel-space. Mentre il kernel mette a disposizione dello user-space, tramite i suoi header, tutte le chiamate e le strutture necessarie per instaurare la comunicazione, la stessa cosa non succede in modalità kernel-space dove risulta molto più complicato instaurare la comunicazione.

Per creare il socket in user-space si deve includere la libreria linux/netlink.h (valida anche per il kernel-space), che racchiude tutti i codici numerici che possono essere usati da Netlink per instaurare correttamente la comunicazione. Per creare la comunicazione occorre definire il socket tramite l'istruzione socket (*socket(PF_NETLINK,SOCK_RAW, NETLINK_TEST)*) che restituisce un file descrittore: il primo parametro dell'istruzione definisce il tipo di socket che si vuole creare, in questo caso indica che si vuole interfacciare con il kernel; il secondo parametro la modalità d'accesso, per questo progetto si usa il SOCK_RAW per poter accedere a qualsiasi livello del messaggio inviato (esiste anche SOCK_DGRAM che permette di trasmettere pacchetti di dati o datagram di lunghezza massima prefissata e indirizzabili singolarmente) e come ultimo parametro il tipo di socket netlink che si vuole creare. Successivamente, si deve assegnare un indirizzo locale al socket, per fare ciò si utilizza la funzione bind() (bind*(sock_fd, (struct sockaddr*) &src_addr,*

sizeof(src_addr))) che richiede come primo parametro il descrittore, come secondo la struttura contenente l'indirizzo del socket (che sarà rappresentato dal PID del processo, la struttura contiene anche informazioni sul tipo di socket il netlink è identificato con AF_NETLINK) e come terzo parametro la dimensione della struttura. Si procede creando la struttura dell'indirizzo del destinatario (il PID del kernel è sempre 0). Anche se la nostra comunicazione avviene da kernel-space a user-space, il programma in user-space deve inviare prima un messaggio verso il kernel-space in modo che questo acquisisca il PID del processo. Successivamente, si crea la struttura dei dati e si va definire anche l'header del messaggio tramite la struttura nlmsghdr (che contiene informazioni sul messaggio che si invia come la lunghezza del messaggio, il PID del mittente e flag addizionali).

Creare il socket in kernel-space è più complesso perché tende ad utilizzare il socket Netlink come qualsiasi altro socket, per cui nel kernel-space sarà definita una struttura di tipo sock (definita nella libreria net/sock.h) che permette la gestione completa del socket. Si deve gestire, inoltre anche il buffer del socket tramite la struttura sk_buff (definita nella libreria linux/skbuff.h). Poi si deve creare il socket in kernel-space tramite la funzione netlink_kernel_create(*netlink_kernel_create(NETLINK_TEST,0,sendnl,THIS_MODULE)*). Questa funzione, come primo parametro deve esplicitare il tipo di socket netlink che si vuole creare e che deve corrispondere a quello inserito in user-space, il secondo parametro è riferito a un gruppo, il terzo parametro esplica la funzione richiamata in caso di richiesta d'accesso al socket e come ultimo parametro il modulo a cui si riferisce la creazione del socket.

Una volta creato il socket è possibile già interagire con esso, ma la funzione di gestione deve essere definita nello stesso modulo ed è quella indicata nella funzione di creazione del socket. Se l'user space invia un messaggio verso il kernel (cioè se il PID corrisponde a zero), il messaggio

inviato è contenuto nel buffer e viene estratto usando la funzione skb_recv_datagram (*skb_recv_datagram(sk, 0, 0, &err)*) o skb_dequeu.

La prima funzione preleva solo il primo messaggio dalla coda, lasciando inalterati gli altri, per fare ciò usa skb_dequeue(). La funzione ha come primo parametro il descrittore del socket e ultimo parametro un numero intero per discriminare tra gli errori. La seconda funzione, invece, non fa altro che estrarre il valore, decrementa la testa della lista e ritornare NULL nel caso non vi siano messaggi da leggere.

Una volta implementato il socket, viene misurato il tempo di ricezione da parte del programma dell'user-space di un messaggio inviato dal kernel-space, questo per ottenere prestazioni migliori nell'invio di un blocco di dati, per far ciò si sono inviati dei payload a lunghezza crescente da 4 byte (Fig3.20) (il valore minimo dei dati che vengono acquisiti in questa applicazione) a 30byte (Fig.3.21).

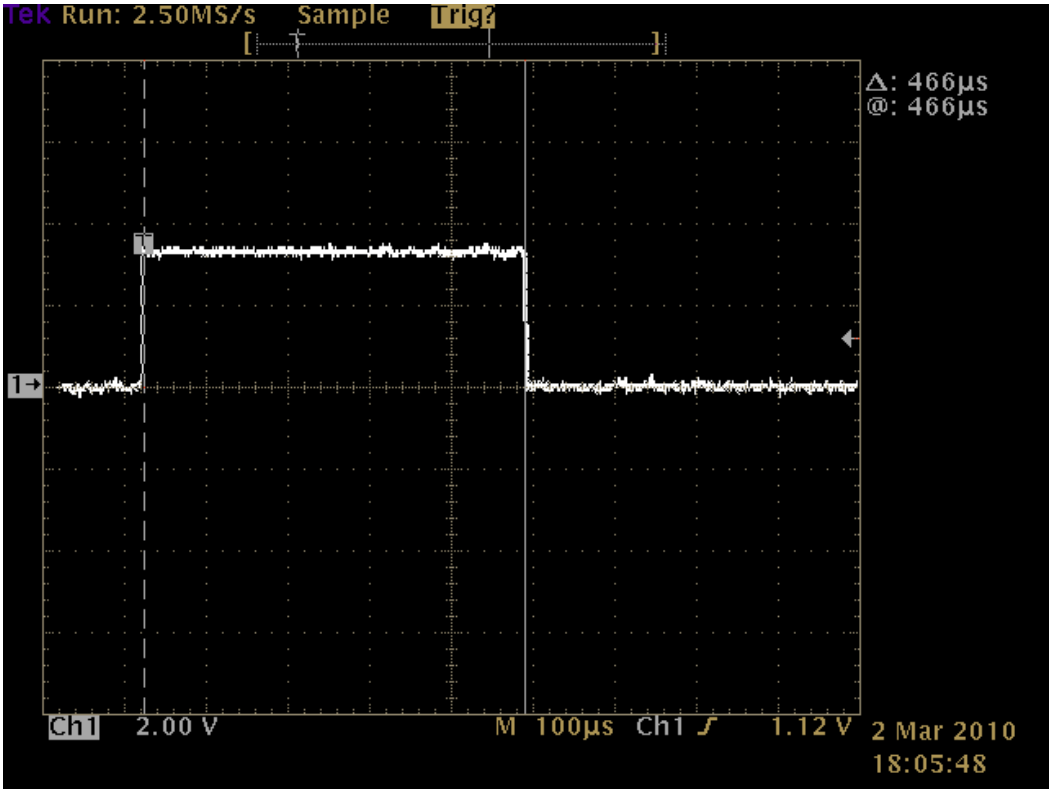


Fig.3. 20: Tempo di trasferimento di 4 byte

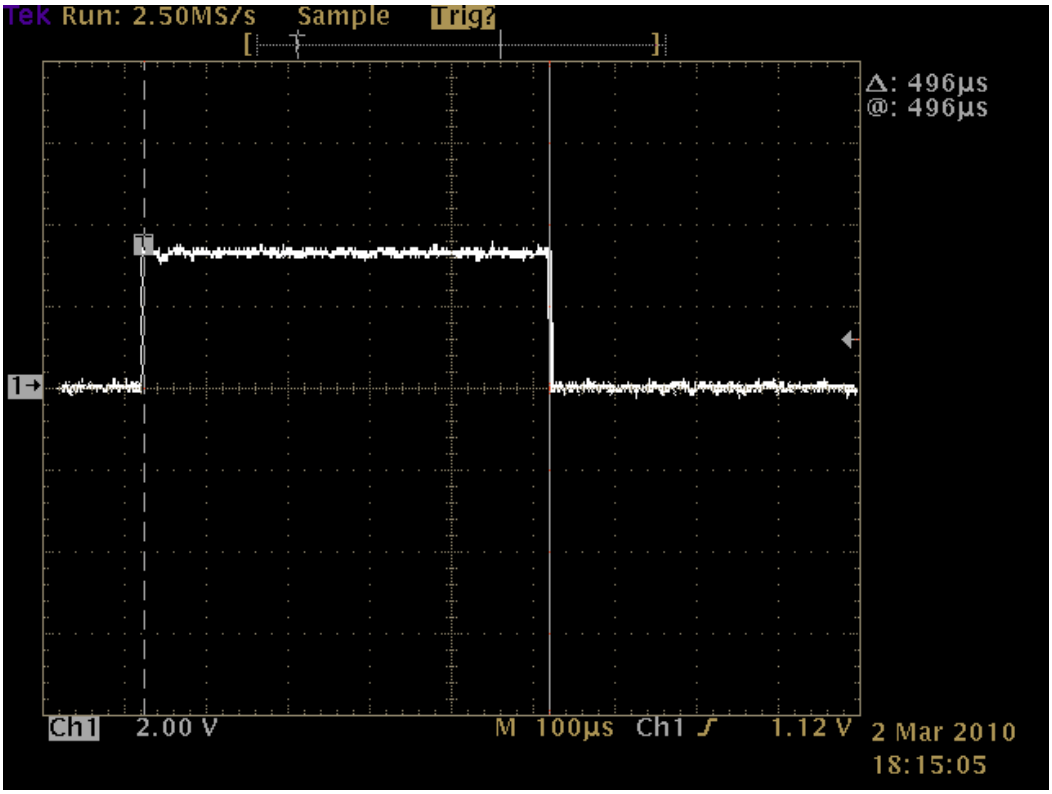


Fig.3. 21:Tempo di trasferimento di 30 byte

Si noti che, aumentando la dimensione del payload il tempo aumenta più lentamente, ciò vuol dire che la maggior parte del tempo di trasmissione si perde nell'istaurare il socket, per cui si è deciso di inviare blocchi di 30 byte (che per la trasmissione fornisce un tempo accettabile pari a 496 µs).

3.8 Interrupt

I dati che vengono dall'esterno devono essere acquisiti il più velocemente possibile dalla FOX BOARD, per far ciò si è analizzato il comportamento degli interrupt. Lo scopo per cui si sono provati ad utilizzare gli interrupt è perché a un particolare segnale in ingresso la FOX BOARD

reagisce invocando una particolare procedura di acquisizione dei dati dall'esterno.

La FOX BOARD ha diverse possibilità di eseguire gli interrupt tramite segnali provenienti dall'esterno, tra cui anche attraverso la porta A. La scelta ricade sulla porta A composta da 8 bit, tutti quanti utilizzabili come piedini di interrupt. Una volta generato l'interrupt questo viene memorizzato nell'apposito vettore (Vector Number Register Sub-Set) come per i normali IRQ di gestione delle porte I/O, gli otto bit della porta generano lo stesso vector number. Gli interrupt sono level-triggered attivi alti.

Alla porta A sono collegati anche i LED e quindi, per eseguire in maniera corretta gli interrupt, si devono disabilitare le normali funzionalità che hanno i tre led. Per far ciò si deve modificare la configurazione del kernel (Appendice A), dal sottomenu Hardware Setup (Fig.3.22), si modifica la voce Product LED Port in None e si imposta la direzione di alcuni pin della porta A, solo di quelli che devono generare gli interrupt in input, andando a modificare il valore di R_PORT_PA_DIR.

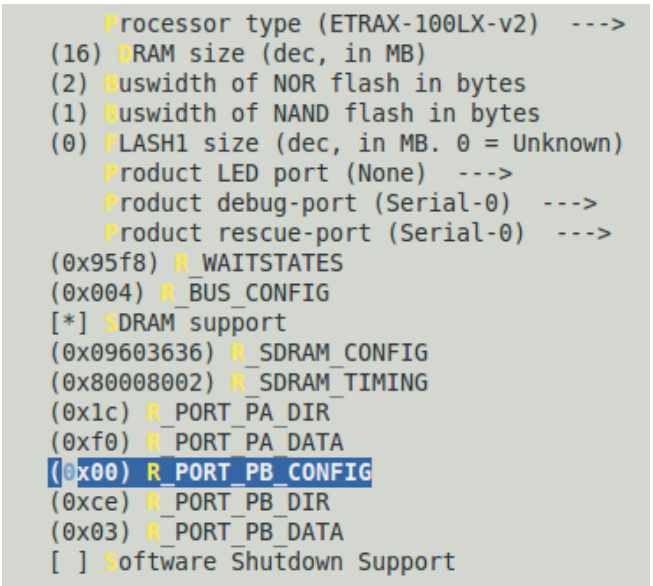


Fig.3. 22: Menù Hardware Setup

La gestione degli interrupt avviene tramite moduli kernel in quanto solo a questo livello si effettua una gestione corretta senza le limitazione degli user-space. Il vero problema nella loro gestione è quello di andare ad individuare l'interrupt una volta generato, in quanto se non si fa eseguire una particolare procedura in cui si dice cosa deve fare il sistema, quando esso vede il segnale d'interrupt lo ignora.

Il modulo di gestione dell'interrupt può essere suddiviso in tre fasi:

- la fase di caricamento deve registrare con successo la funzione di interrupt e memorizzarla nel registro degli interrupt del kernel affinché gli venga assegnato il canale richiesto.
- la fase di interrupt deve servire la funzione predisposta dell'interrupt e, al tempo, deve riportare lo stato logico del pin in cui si è effettuato l'interrupt a livello logico basso (come detto gli interrupt si attivano a livello logico alto) evitando che il sistema venga subissato da richieste di interrupt
- la fase di clean-up del modulo, consente scaricare il modulo e rilasciare tutte le risorse occupate.

Di seguito è riportato un semplice modulo di gestione degli interrupt:

```
#include <linux/interrupt.h>
#include <linux/module.h>
#include <asm/io.h>

MODULE_LICENSE("DUAL GPL/BSD");

//maschera per dire su quale pin(PORTA) si ha la richiesta di
interrupt
#define IRQ_PA_MASK 0x01

static DEFINE_SPINLOCK(gpio_lock_irq);

/*In questa funzione viene disabilitata temporaneamente la
possibilità di ricevere altri interrupt fino a cha non è soddisfatta
la funzione seguente */
static irqreturn_t irq_interrupt(int irq, void *dev_id, struct
```

```
                                    pt_regs *regs)
{
   *R_PORT_PB_DATA=*R_PORT_PB_DATA|128;
   *R_PORT_PB_DATA=*R_PORT_PB_DATA&127;
   *R_IRQ_MASK1_CLR = 0;
   /* fondamentale altrimenti genera infiniti interrupt(li vede sul
      Livello)*/
   *R_PORT_PA_DATA = 0;
   printk(KERN_ALERT "irq servito\n");
   return IRQ_HANDLED;
}

/*Funzione di inizializzazione del modulo*/
int irq_init_module(void)
{
   if (request_irq(PA_IRQ_NBR,
                   irq_interrupt,SA_INTERRUPT,"interrupt", NULL))
   {  printk(KERN_CRIT "err: PA irq for gpio\n");
      return -ERESTART;
   }
   /*setta il registro in cui si dice che il pin è abilitato
    o meno per gli interrupt*/
   *R_IRQ_MASK1_SET = IRQ_PA_MASK;
   return 0;
}

void irq_cleanup_module(void)
{
   spin_lock_irq(&gpio_lock_irq);/*disattiva tutti gli irq sul
                                   processo corrente*/
   *R_IRQ_MASK1_SET = 0;
   *R_IRQ_MASK1_CLR = 0;//elimina tutti gli interrupt
   free_irq(PA_IRQ_NBR, NULL);
   spin_unlock_irq(&gpio_lock_irq);
}

module_init(irq_init_module);
module_exit(irq_cleanup_module);
```

Le funzioni riportate nel modulo che gestiscono gli interrupt sono definite nella libreria linux/interrupt.h. La funzione per registrare l'interrupt è request_irq(come da modulo precedente), essa restituisce 0 per indicare il successo o un codice di errore, molto spesso ha come valore di ritorno –EBUSY, per indicare che un altro driver sta utilizzando la linea che si vuole utilizzare. Il primo parametro della funzione indica il numero di interrupt richiesto, il secondo parametro è un puntatore alla funzione di gestione in fase di installazione, il terzo parametro è una maschera di bit di opzioni relativi alla gestione interrupt (può assumere i seguenti valori nel caso che

l'ultimo parametro sia NULL: SA_INTERRUPT indica un gestore fast di interrupt, cioè disabilità gli attuali, SA_SHIRQ indica che l'interrupt possono essere condivisi tra i dispositivi e SA_SAMPLE_RANDOM indica che l'interrupt generato può contribuire al pool di entropia utilizzati da /dev/random e /dev/urandom per generare le chiavi di cifratura), il quarto parametro è una stringa passata a request_irq e utilizzata in /proc/interrupts per mostrare il proprietario dell' interrupt e l'ultimo parametro è un puntatore utilizzato per condividere le linee di interrupt normalmente NULL.

Ogni volta che un interrupt viene lanciato, il kernel incrementa un contatore interno, offrendo un modo per controllare se il dispositivo funziona come previsto. Per far ciò si visualizza il file /proc/interrupts(Fig.3.23).

```
[root@axis-00408c120232 /root]101# cat /proc/interrupts
           CPU0
  2:     650518          CRISv10  timer
  3:          0          CRISv10  fast timer int
  6:          0          CRISv10  ETRAX 100LX built-in ethernet controller
  8:          0          CRISv10  serial
 16:        139          CRISv10  ETRAX 100LX built-in ethernet controller
 17:        323          CRISv10  ETRAX 100LX built-in ethernet controller
 22:        152          CRISv10  serial 0 dma tr
 23:          0          CRISv10  serial 0 dma rec
 24:          0          CRISv10  ETRAX 100LX built-in USB (Tx)
 25:          0          CRISv10  ETRAX 100LX built-in USB (Rx)
 31:          2          CRISv10  ETRAX 100LX built-in USB (HC)
[root@axis-00408c120232 /root]101#
```

Fig.3. 23:File /proc/interrupts

La prima colonna della figura corrisponde al numero di IRQ, naturalmente sono elencati solo gli interrupt attivi. La seconda colonna mostra quanti interrupt sono stati eseguiti dalla CPU. Le ultime due colonne forniscono informazioni sul controller di interrupt programmabile che gestisce l'interrupt, e il nome del proprietario che ha generato l'interrupt.

Quando si vuole rilasciare la linea di interrupt si usa la funzione free_irq.

Per conoscere il tempo di risposta dell'interrupt rispetto al segnale d'interrupt esterno ed il tempo in cui il segnale deve stare a livello per essere individuato, sono state eseguite delle prove creando, attraverso un pattern generation, un treno d'impulsi a durata crescente.

Utilizzando il precedente modulo di prova delle porte GPIO (il modulo precedente) e acquisendo tramite l'oscilloscopio (FIG.3.24) sia il segnale con cui si generano le richieste che il pin 7 della porta B (che cambia stato quando l'interrupt viene servito) si ha il seguente risultato:

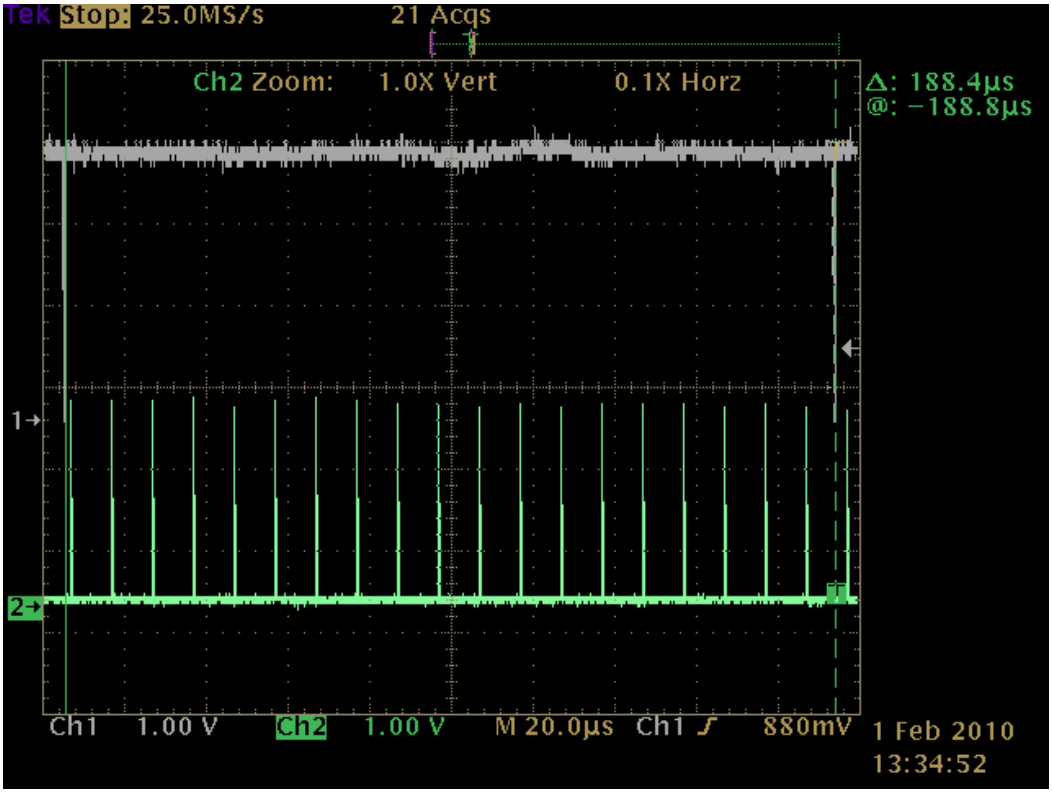


Fig.3. 24: Ritardo di risposta dell'interrupt

La traccia di colore verde è il segnale con cui si chiede l'intervento di un interrupt e quella in grigio è il segnale che indica che l'interrupt è stato servito. Si noti che, nella frequenza con cui si va a generare la richiesta d'interrupt si perdono molti eventi e quindi si deve diminuire la frequenza di richiesta. Inoltre, si deve valutare il ritardo che intercorre fra la richiesta

dell'interrupt e il tempo che l'interrupt sia servito; per valutare il ritardo si riduce la frequenza con cui si richiede l'interrupt (Fig.3.25: traccia in verde).

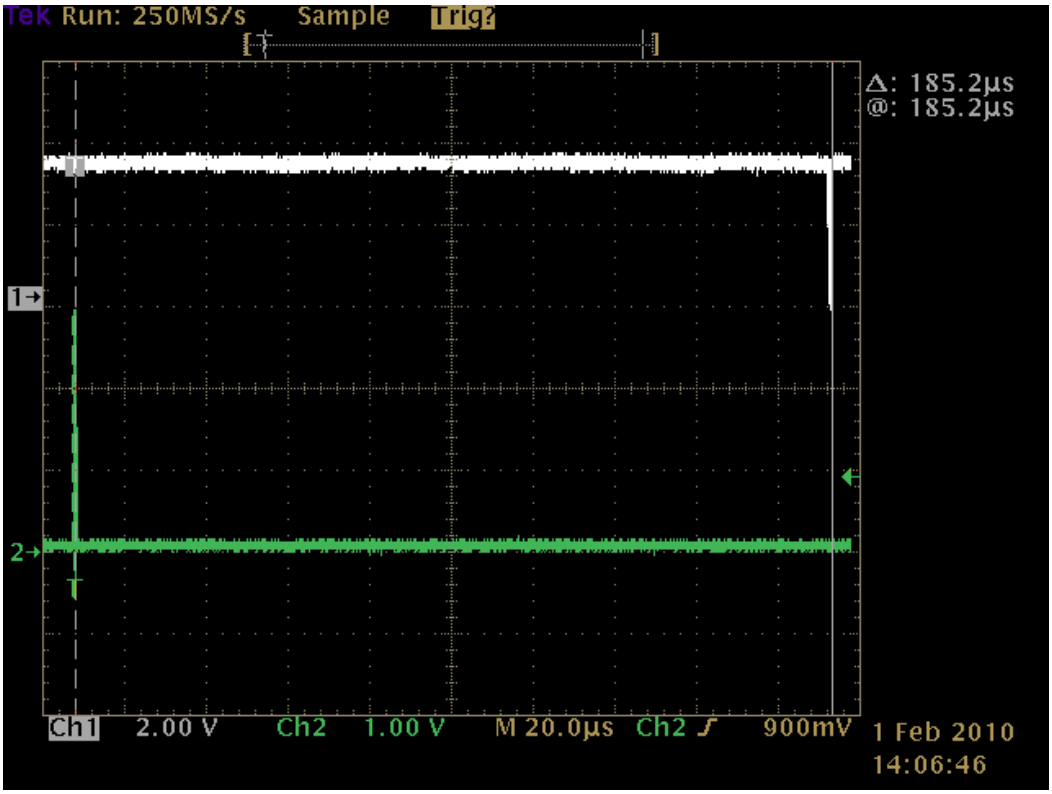


Fig.3. 25:valutazione del ritardo di risposta dell'interrupt

Si noti che il ritardo è di 186 µs, questo ritardo è eccessivo per l'esecuzione di un interrupt e in questa applicazione potrebbe creare dei problemi in quanto la FOX BOARD è bloccata mentre serve l'interrupt, quindi si è deciso di utilizzare il polling per l'interrogazione delle porte.

3.9 Operazioni in virgola mobile

La FOX BOARD è stata inserita nel progetto in quanto capace di effettuare i calcoli in virgola mobili anche se emulati. Si sono fatte alcune misure per verificare il tempo che impiega ad effettuare le operazioni, in particolare si sono valutate le addizioni in virgola mobile (Fig3.26. traccia

bianca), le divisioni (Fig3.26. traccia gialla) e le moltiplicazione (Fig3.26. traccia blu). Tutti i tempi sono calcolati per 100 operazioni.

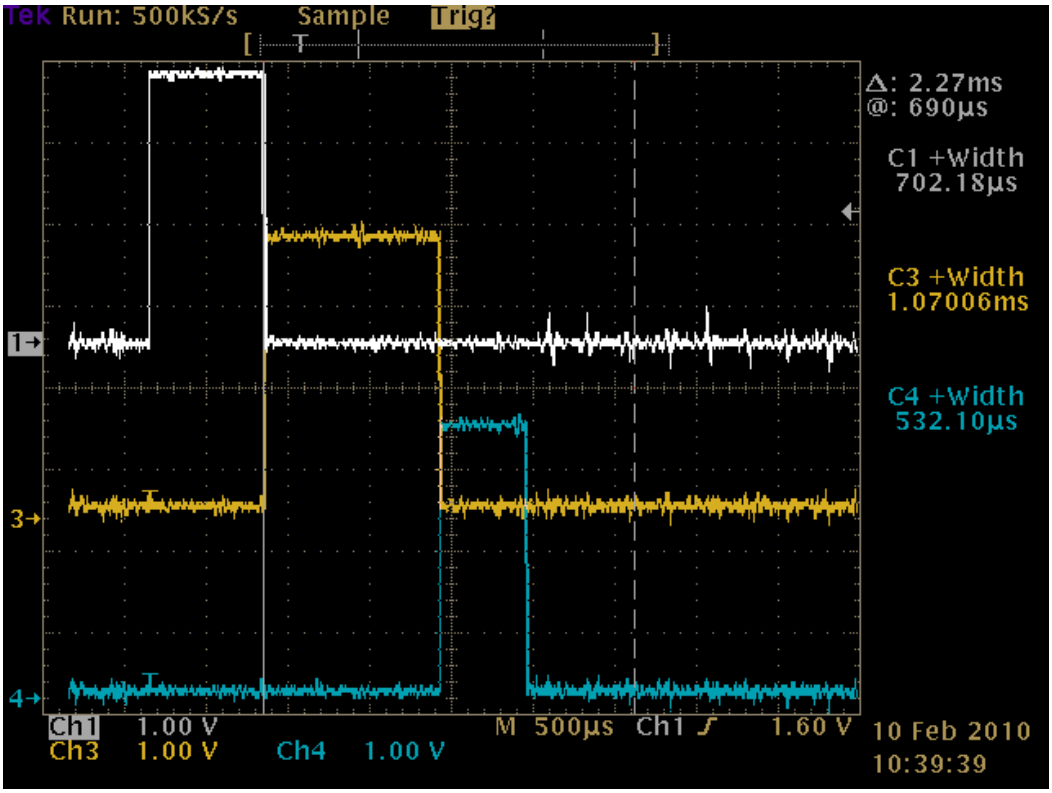


Fig.3. 26: Tempo di esecuzione di 100 operazioni aritmetiche

Si è valutato anche il tempo di scrittura in memoria (Fig.3.27: traccia blu), di estrazione degli stessi dati dalla memoria (Fig.3.27: traccia gialla) di tipo unsigned long (32 bit), il tempo di prelievo di una variabile intera e messo a disposizione delle porte di output (Fig.3.27: traccia verde). Tutti i tempi sono eseguiti per 100 operazioni.

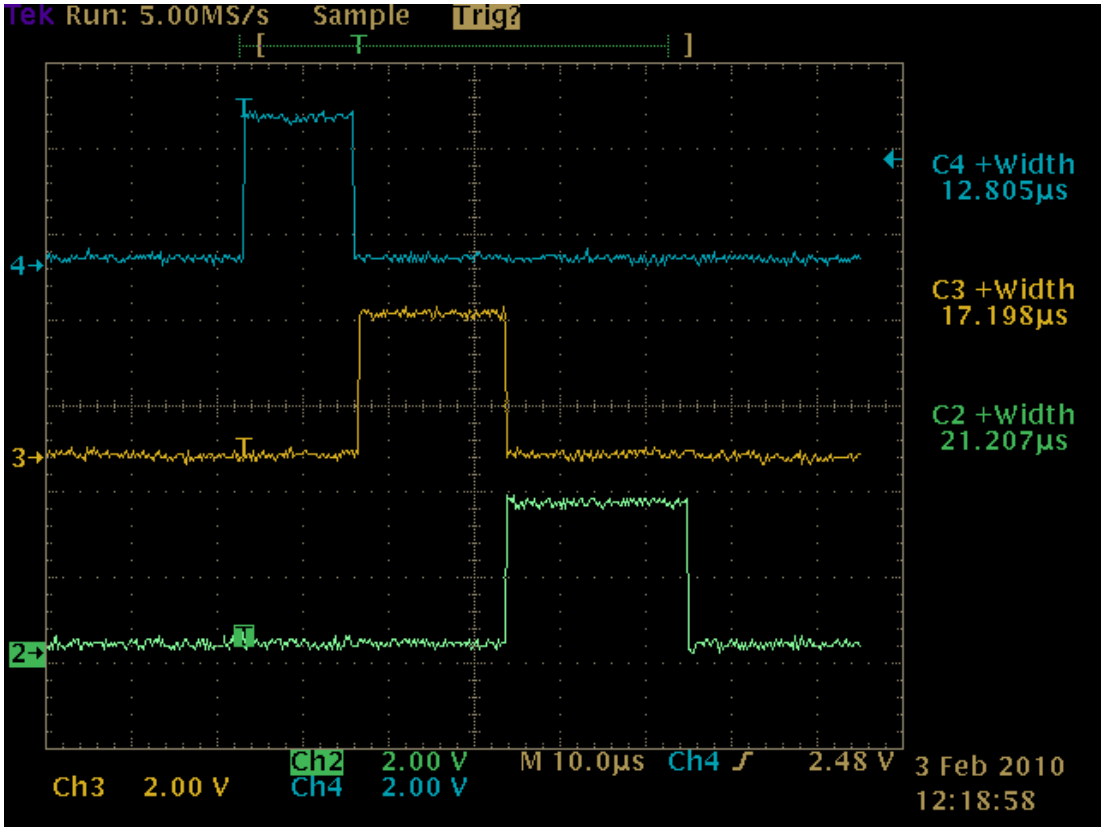


Fig.3. 27: Tempo di lettura e scrittura da memoria di 100 dati

Riassumendo abbiamo ottenuto i seguenti tempi:

OPERAZIONE	TEMPO (per singola operazione)
Addizione tra float	7,02µs
Divisione tra float	10,70µs
Moltiplicazione tra float	5,32µs
Scrittura in memoria di dati float	0,12µs
Lettura dalla memoria di dati float	0,17µs
Lettura di un intero e accesso alle porte di I/O	0,21µs

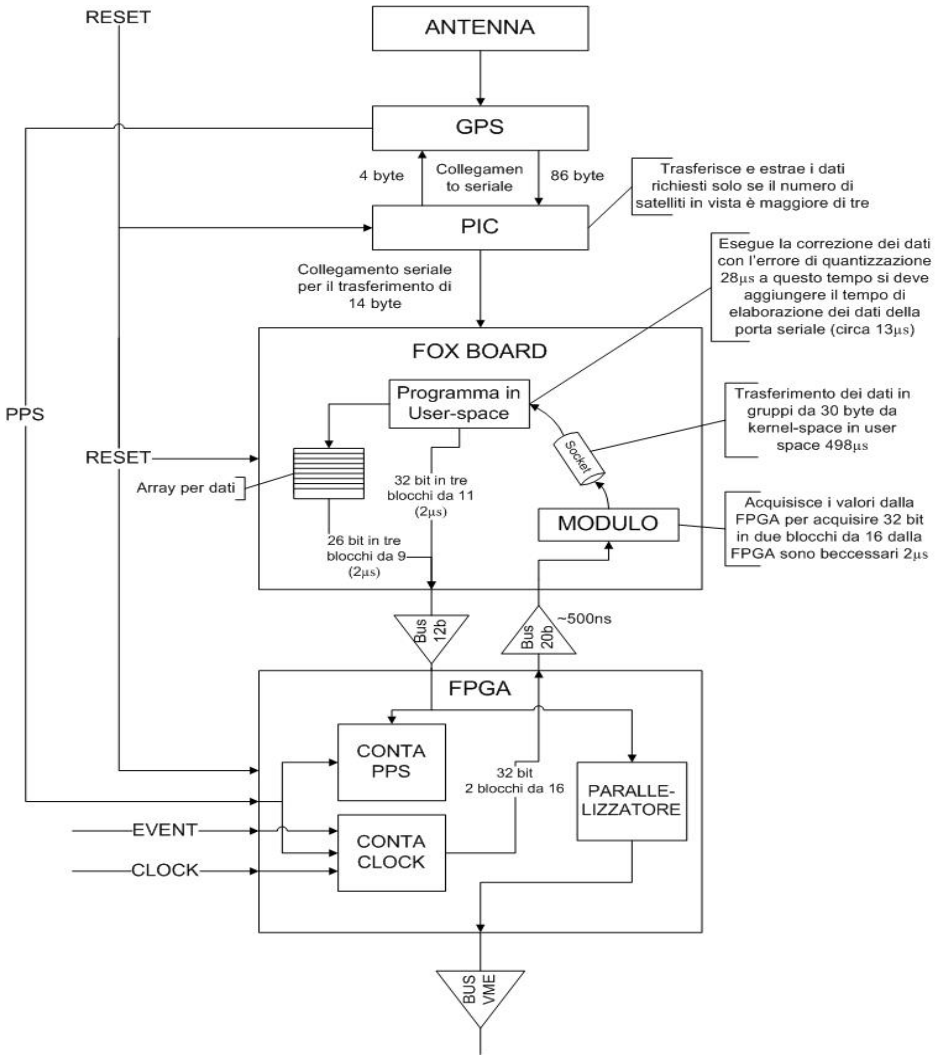
Tutti i dati ottenuti nelle misurazioni verranno successivamente utilizzati per determinare le dimensioni più adatte delle memorie interne.

Capitolo 4

SVILUPPO DEL CODICE DELLA SCHEDA VME

4.1 Teoria dell'operazione

Nella figura (Fig.4.1) sottostante è riportato lo schema a blocchi e il flusso di dati del dispositivo sviluppato nell'ambito di questo lavoro:



4. 1: Diagramma di flusso

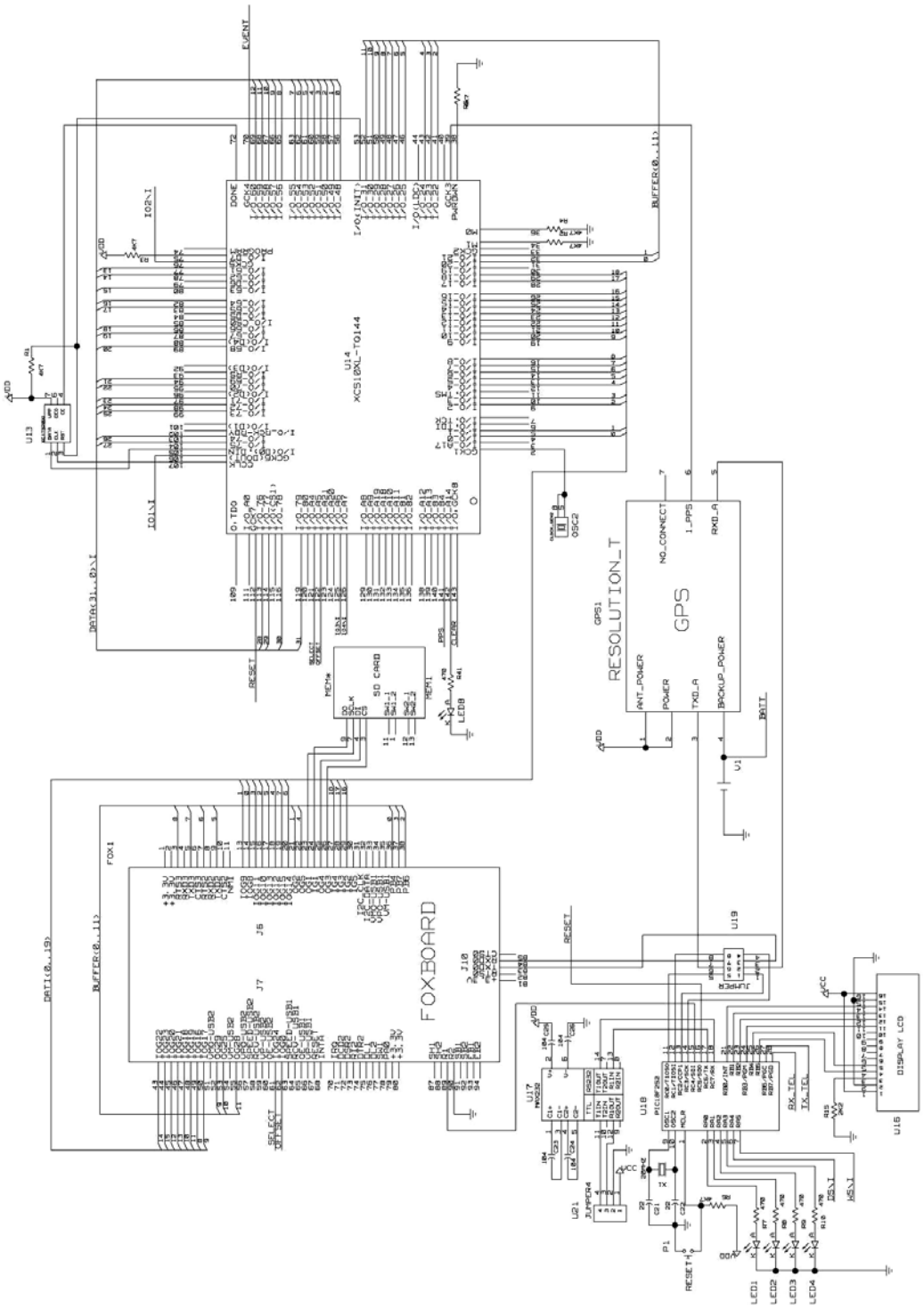
I primi due blocchi rappresentano dei dispositivi proprietari della Trimble, in particolare il ricevitore è il ResolutionT. Il terzo blocco è rappresentato da una PIC 18F252 la quale comunica con il ricevitore GPS da cui vengono acquisite le informazioni che servono, la comunicazione avviene tramite la porta seriale scambiando 86 byte dal GPS alla PIC e 4 byte nella direzione inversa, tutto è impacchettato secondo il formato del protocollo TSIP. Una volta che i dati sono stati ricevuti dalla PIC, questa esegue un controllo sul numero di satelliti visibili e accende i led, se i satelliti visibili sono maggiori di 3, estrae le informazioni temporali, le elabora e le invia tramite la porta seriale verso la FOX BOARD. Le informazioni che invia sono al massimo 14byte in 3ms; i dati in fase di sperimentazione vengono visualizzati su un dislpay per verificarne la correttezza.

Una volta che la FOX BOARD ha ricevuto tali dati, solo se la PIC vede più di 3 satelliti, i dati vengono elaborati in user-space. Il programma in user-space, acquisendo i primi valori, converte i secondi trascorsi dall'inizio dell'anno in valore binario e li invia verso il quinto blocco (di cui si discuterà successivamente). Il numero binario, formato da 32 cifre verrà trasmesso in due blocchi da 16; per fare ciò, ed essere sicuri che la comunicazione sia avvenuta correttamente, trascorrono circa 2μs. Successivamente, la FOX BOARD converte il valore binario dell'errore di quantizzazione in un numero in virgola mobile secondo lo standard IEEE745; il tale numero verrà utilizzato per correggere i dati che provengono dal quinto blocco. Una volta convertiti, questi dati, per averli a disposizione in user-space trascorrono 498μs, dopodiché vengono inseriti in una coda di 1000 elementi che viene cancellata ogni secondo. Le informazioni che arrivano dai dati immagazzinati nella coda verranno forniti al quinto blocco quando la FOX BOARD sarà disponibile ad inviarli. I dati che arrivano allo user-space sono acquisiti tramite un modulo in kernel-space e passati attraverso un socket allo user-space.

Il quinto blocco non è altro che una FPGA che esegue dei processi in parallelo e che ha in ingresso, oltre ai bus provenienti dalla FOX BOARD, il segnale PPS, un clock a 50 MHz e un ingresso di Event che rappresenta il segnale di trigger. Un primo processo è il “conta PPS” che non fa altro che incrementare il dato dei secondi trascorsi dall’inizio dell’anno ad ogni PSS che arriva. Un altro processo implementato è il “conta clock”, che può essere considerato il processo cardine; esso conta i cicli di clock dal momento in cui il segnale PPS è ricevuto. L’oscillatore che determina il clock non è detto che in un secondo faccia esattamente 50 milioni di oscillazioni per cui il conteggio deve essere normalizzato (come si vedrà nel seguito) e messo a disposizione alla FOX BOARD ogni volta che si attiva il segnale EVENT. L’ultimo processo è un semplice parallelizzatore dei dati corretti provenienti dalla FOX BOARD che, una volta disposti in parallelo vengono trasferiti verso il bus VME.

La PIC, la FOX BOARD e la FPGA hanno in ingresso anche il segnale di reset per riportare il sistema allo stato iniziale.

Nella figura (FIG.4.2) della pagina successiva è riportato lo schema elettrico dei componenti appena descritti con le varie connessioni.

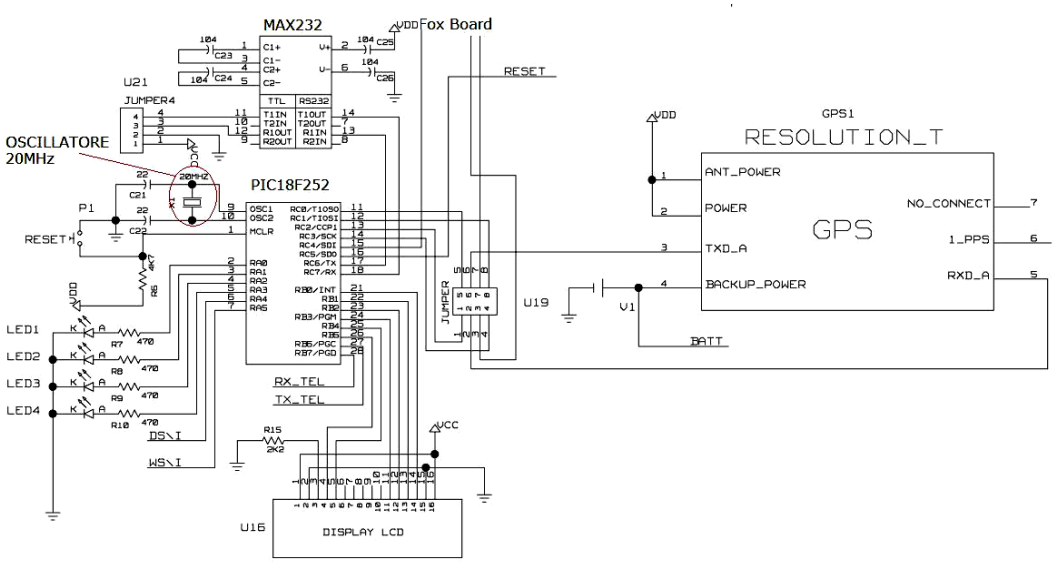


4.2: Schema circuitale

Nello schema sono presenti alcuni elementi del circuito che verranno spiegati in seguito.

4.2 Sviluppo del firmware della PIC

La PIC utilizzata per questo lavoro è il modello PIC18F252, il suo compito è l'estrazione dei dati dal protocollo usato dal ricevitore GPS Resolution T. Lo schema circuitale (Fig.4.3) della PIC sulla scheda VME è il seguente:



4.3: Schema circuitale della PIC

Nello schema circuitale oltre alla PIC e al ricevitore GPS sono presenti:

- un MAX232, che agisce quale adattatore di livello tra la PIC e l'interfaccia RS232, tramite la quale si programma la PIC attraverso il jumper4;
- un Oscillatore a 20MHz per il clock della PIC;
- 4 Led;
- un Display LCD utilizzato solo per il debbuging

Le informazioni inviate dal GPS, sono organizzate in pacchetti tramite il protocollo TSIP e, attraverso una porta seriale, trasferiti alla PIC

opportunamente programmata, che ne estrae i dati necessari al funzionamento dei successivi dispositivi.

In prima istanza si è collegato il ricevitore GPS in ingresso alla PIC e il solo display LCD in modo da poter verificare la correttezza della programmazione della PIC e che i valori estratti fossero esatti (tramite il confronto con i dati forniti dal software della Trimble). I dati, acquisiti tramite porta seriale, venivano aggiornati con cadenza di un secondo, senza ritardi.

Si esaminano ora le istruzioni che la PIC usa per l'elaborazione delle informazioni, mostrandone prima il relativo diagramma di flusso (Fig.4.4) e successivamente scendendo nel dettaglio.

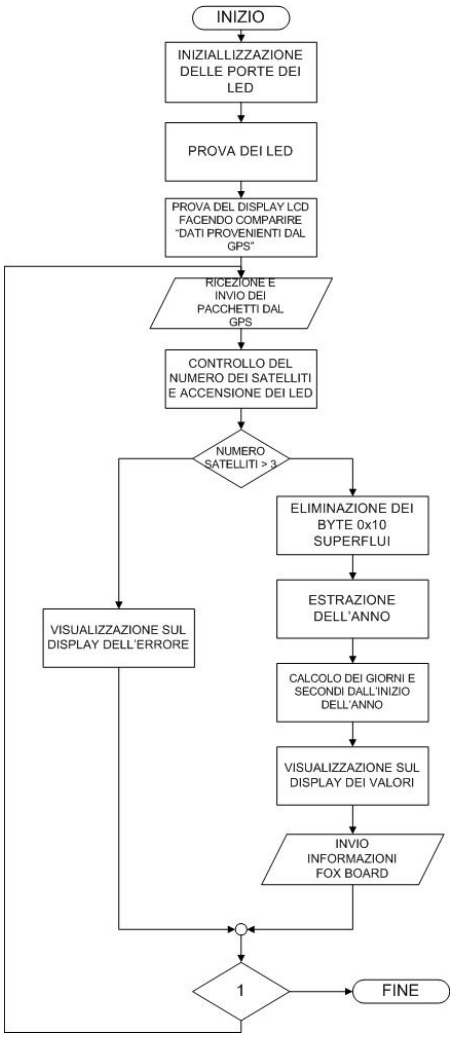


Fig.4.4: Diagramma di flusso della PIC

Procediamo con la spiegazione dettagliata del programma implementato della PIC. Dapprima vengono inizializzate le caratteristiche del display LCD e impostata come output la porta alla quale sono collegati i led e che gestisce i segnali digitali. Si definiscono poi le variabili necessarie, in particolare si definisce un vettore (variabile "day") di 12 elementi di tipo word (2 byte) in cui sono memorizzati i giorni dall'inizio dell'anno e per ogni suo mese. Si procede quindi verificando l'esatto funzionamento del display LCD attraverso la visualizzazione di una stringa; si invia il pacchetto 0x24 al GPS come richiesta di invio del pacchetto 6D, contenente il numero di satelliti, a questo punto si prelevano in sequenza i pacchetti 6D, 0x8F-AB (data e ora) e 0x8F-AC (errore di quantizzazione), provenienti dal GPS utilizzando il comando SERIN2, insieme alla parola chiave "wait", che mette in attesa la PIC per la ricezione del valore indicato che rappresenta l'ID del pacchetto GPS. Una volta individuato l'ID del pacchetto, il comando SERIN2 lo memorizza nella variabile indicata con tutte le informazioni contenute nei pacchetti (che sono stati descritti nel capitolo 2). Un grosso problema è costituito dalla lunghezza degli array in cui si devono memorizzare i pacchetti AB e AC (non ci sono problemi con il pacchetto 6D in quanto si acquisisce solo il primo byte), in quanto la loro lunghezza non è costante poiché, ogni volta che il GPS riceve all'interno della stringa dati il valore 0x10, che corrisponde anche al valore di inizio e fine del pacchetto, questo viene seguito da un valore identico extra, in modo da non confondere il valore interno alla stringa, che rappresenta un dato, con quello utilizzato dal protocollo per la separazione dei pacchetti. Tutti i byte nella stringa, successivi al valore 0x10 vengono sfasati di una posizione rispetto a quella che normalmente occupano. Per questo motivo, si deve prevedere che l'array ("pab") che contiene il pacchetto 0x8F-AB che normalmente ha una lunghezza pari a 17 byte, assumerà una lunghezza pari a 18 byte e che il pacchetto 0x8F-AC (array "pac"), che normalmente è di 66 byte, assumerà una lunghezza pari a 67 byte. Ci si deve limitare ad estrarre un byte o al massimo tre byte (questo limite è dovuto al fatto che dopo l'ultimo dato del

pacchetto ci sono i 2 byte di fine pacchetto e un byte d'inizio del pacchetto successivo) in più e comunque, nel caso in cui il valore "0x10" non sia presente nel pacchetto, c'è il rischio di acquisire i dati del pacchetto successivo. Questo vuol dire che se il pacchetto contiene un numero di byte "0x10" superiore a tre, si può verificare un errore nell'acquisizione dati.

Una volta memorizzati i pacchetti 6D,0x8F-AB e 0x8F-AC nelle rispettive variabili, si controlla che il numero di satelliti (variabile "sat") sia maggiore di tre e dopo aver shiftato la variabile, (ciò si deve fare perché i dati sui satelliti corrispondono ai 4 bit più alti del byte acquisito), si accendono i led, uno per ogni satellite visibile. Se il risultato del controllo è positivo si procede alla verifica e alla eliminazione dell'eventuale presenza dei byte "0x10" superflui tramite un ciclo per ogni pacchetto.

Successivamente, si estrae l'anno che nell'array caricato è diviso in due byte, per cui anno (variabile "anno") sarà una variabile di tipo word. Le informazioni sui secondi, minuti, ore, giorno del mese e mese sono di tipo byte senza segno e quindi accessibili direttamente dall'array dove sono state memorizzate. Conoscendo il mese, è possibile calcolare i giorni trascorsi tramite il vettore creato ad inizio d'anno e memorizzare il dato in una variabile di tipo word (variabile "daystart"). Si controlla, di seguito, se l'anno è bisestile e, in caso affermativo, si aggiunge un giorno al conteggio appena fatto. Una volta calcolati i giorni trascorsi dall'inizio dell'anno si può trasformare il dato in secondi. Sapendo che il numero di secondi contenuti in un anno è pari a 31536000, che corrisponde ad un numero binario di 25 bit, occorre una variabile di tipo long (32 bit).

Il secondo pacchetto (array "pac") contiene l'errore di quantizzazione che verrà prelevato ed inviato direttamente. Le posizioni sono 59, 60, 61 e 62 (come si può rilevare dalle tabelle del protocollo presenti nel capitolo 2) e, come già detto, di questo pacchetto verranno inviati solo i primi 3 byte. Al

termine di questo processo, viene visualizzato il tutto sul display LCD, per una ulteriore verifica dei dati elaborati.

Tramite un'altra porta seriale si trasferiscono le informazioni verso la FOX BOARD: si inviano prima i secondi trascorsi dall'inizio dell'anno (variabile "sec") con a seguito un carattere speciale che indica alla FOX BOARD che l'invio della informazione sui secondi è terminata. Il carattere speciale è fondamentale quando l'informazione, che può essere lunga da una cifra a un massimo 10 cifre, è composta da cifre uguali a zero e che è inutile inviare. Di seguito si invia l'errore di quantizzazione.

Se le informazioni non risultano attendibili, cioè il numero di satelliti è minore di 3, sul display verrà visualizzata la stringa: "Numero di satelliti insufficiente".

Questo procedimento viene ripetuto continuamente: Si deve considerare che il collegamento a un numero di satelliti che permettono di considerare attendibili le informazioni avviene dopo qualche minuto dall'accensione del GPS e se le condizioni di ricezione sono buone.

Una volta programmata la PIC attraverso la porta seriale e il MAX 232, il ricevitore GPS viene collegato alla PIC chiudendo il jumper della scheda come in Fig.4.3 per permettere in questo modo alla PIC di ricevere o trasmette i dati dalla porta verso il ricevitore GPS. Si verificano, inoltre, i dati che vengono visualizzati sul display:

- UTC;
- Data;
- Giorni trascorsi dall'inizio dell'anno;
- Numero di satelliti

tramite l'ausilio del software messo a disposizione della Trimble. Il codice della PIC è riportato nell'Appendice D e in Fig.4.5 la parte della Scheda VME in cui è montata la PIC.

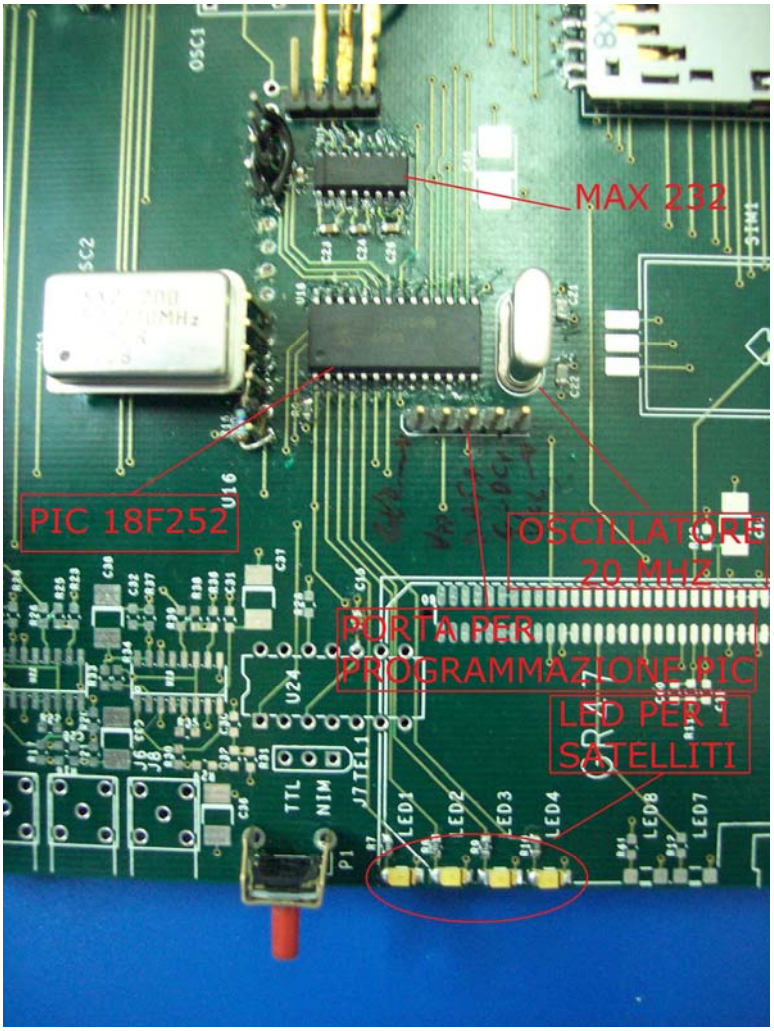


Fig.4.5: Circuito della PIC sulla scheda VME

4.3 Sviluppo del programma della FOX BOARD

La FOX BOARD è una parte fondamentale della scheda e allo stesso tempo l'anello debole in quanto l'oggetto più lento, per questo è lei che deve "pilotare" gli altri dispositivi.

La FOX BOARD è connessa sia alla PIC, tramite la porta seriale, che alla FPGA, tramite le normali porte GPIO. Di seguito (Fig.4.6) è riportato lo schema circuitale e i collegamenti con gli altri dispositivi.

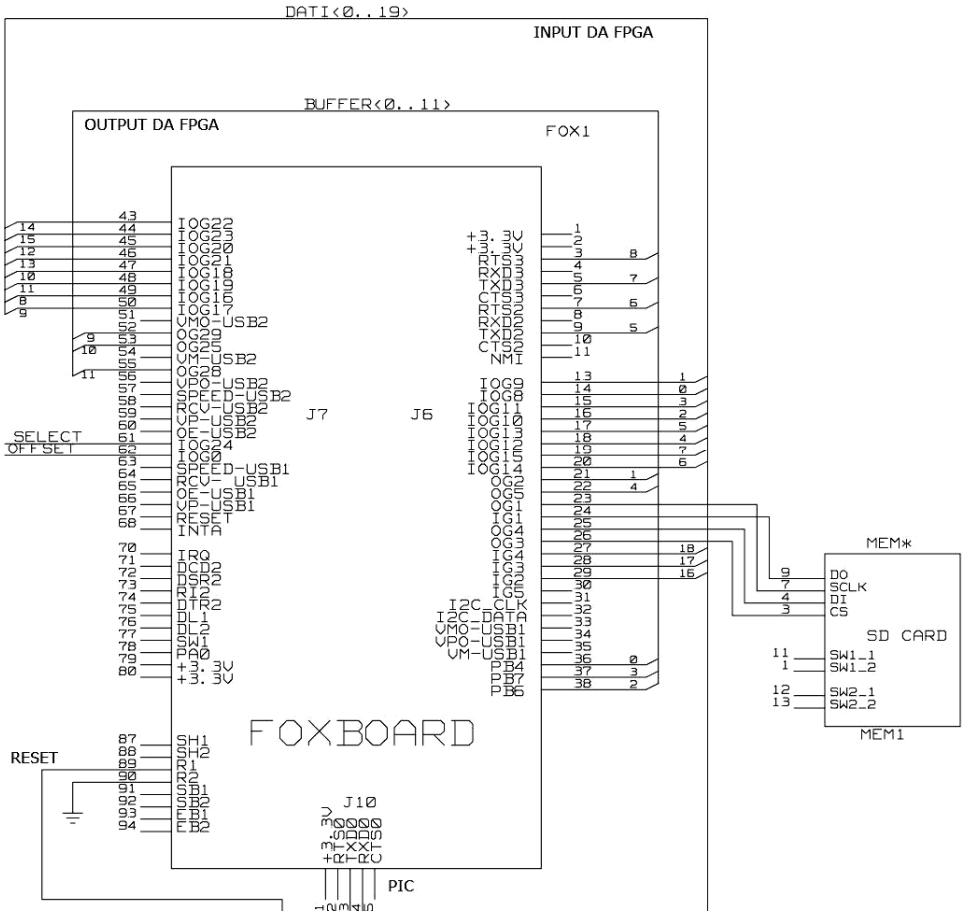


Fig.4.6: Schema circuitale della FOX BOARD

Come si può notare dalla figura la FOX BOARD è equipaggiata anche di un lettore di SD-CARD per il caricamento dei programmi. Per far riconoscere questo dispositivo si deve fare una modifica nel menuconfig (Appendice A) nella sezione Driver settings e abilitare la voce Enable MMC Support.

Il codice sviluppato per la FOX BOARD è abbastanza complesso in quanto si basa su due programmi distinti che lavorano su differenti livelli di Linux: il programma principale, che lavora a livello di user-space, deve

stabilire la connessione tramite porta seriale e socket con l'altro programma che risiede in kernel-space, oltre a dover eseguire tutte le operazioni e le conversioni. Il programma in kernel-space, invece, deve "solo" instaurare il socket per la comunicazione e acquisire le informazioni della FPGA e inviarle, dopo averle trattate, verso il programma in User-space. Il motivo per cui abbiamo sviluppato un modulo in kernel-space, come già detto, è dovuto al fatto che il tempo di interrogazione alle porte di I/O e molto inferiore rispetto all'User-space.

Si esamina prima il programma in User-space, del quale si riporta il diagramma di flusso per chiarire ulteriormente le sue funzioni e poi il modulo in kernel-space con relativo diagramma di flusso: entrambi i programmi sono scritti in Linguaggio C.

Nella figura (Fig.4.7) seguente è riportato il diagramma di flusso del programma in user-space.

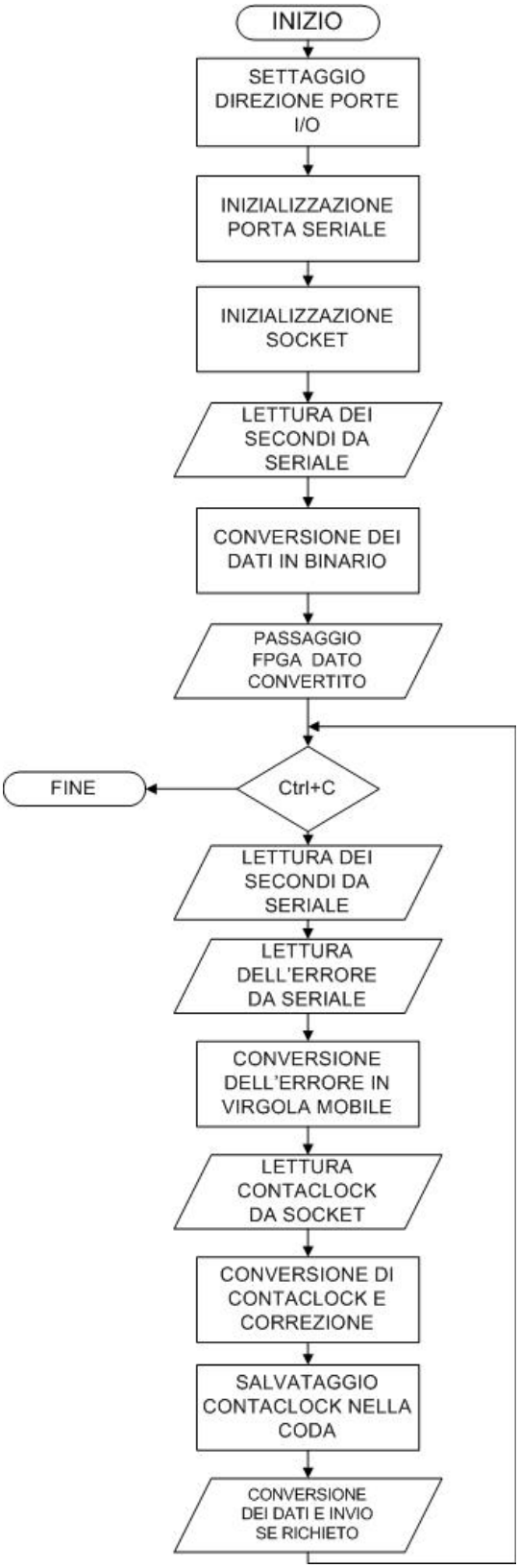


Fig.4.7: Diagramma di flusso del programma in user-space

Il programma, dopo la fase iniziale di dichiarazione delle variabili e strutture, definisce la direzione della porta PB4 in output (per default è di input), fatto ciò, richiama due funzioni scritte all'inizio del programma. La prima funzione instaura la connessione della porta seriale e richiede quale parametro il file di dispositivo della porta usata (/dev/ttyS0), mentre la seconda funzione instaura il socket netlink.

Nella funzione di inizializzazione della porta seriale, si definisce una struttura termios ("newser") che permetterà di modificare le caratteristiche della porta (velocità, controllo di parità, …) e di adattarle al progetto. Successivamente, utilizzando il file di dispositivo passato alla funzione, con la funzione open si stabilisce la connessione della porta seriale. La funzione open restituisce un descrittore ("fdser") che, se è maggiore di zero, è indice di connessione instaurata, altrimenti esce dal programma. Dopodiché, viene controllato l'esistenza del terminale d'input predefinito (questo avviene tramite la chiamata ad un'altra funzione) e, dopo aver instaurato la connessione con la porta seriale, si definisce il comportamento della porta nel caso d'arrivo di segnali d'allarme e si definiscono le caratteristiche della porta seriale con la funzione tcsetattr.

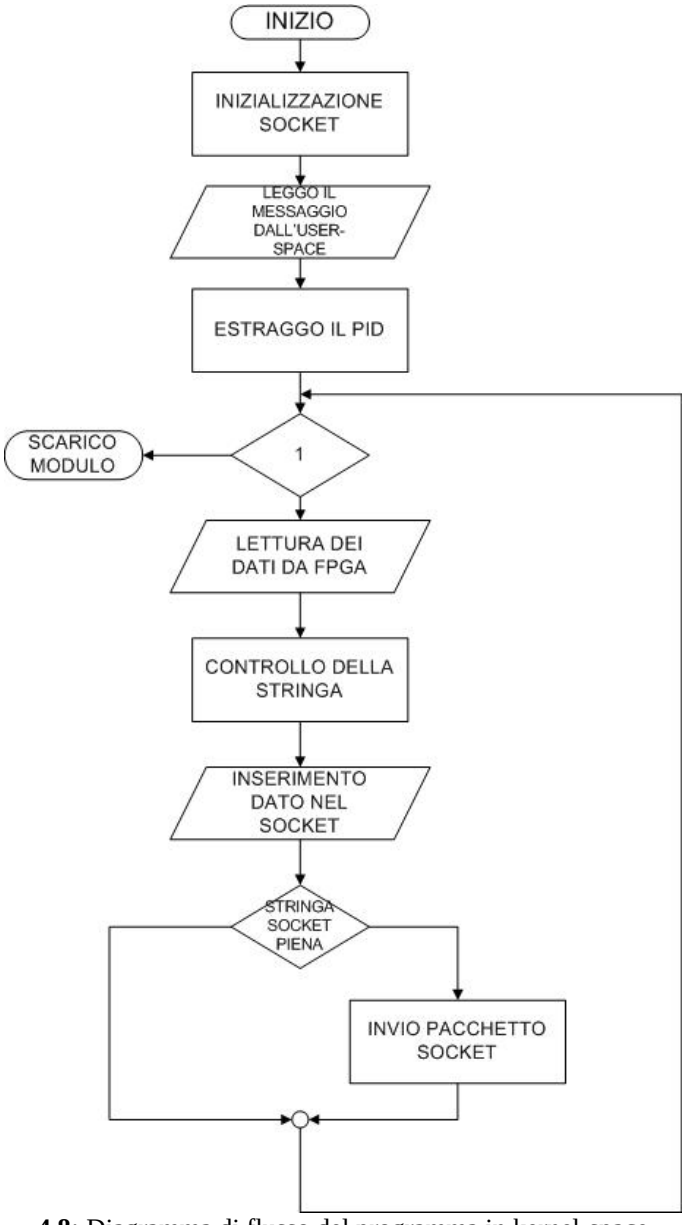
La funzione d'inizializzazione del socket NetLink, instaura il socket tramite la funzione socket che ne restituisce il descrittore ("sockfd"), definisce i campi della struttura ("srcaddr") contenenti l'indirizzo e il tipo di socket del mittente e, tramite la funzione bind(), assegna un indirizzo locale al socket. Successivamente, si definiscono i campi della struttura ("destaddr") che contiene l'indirizzo e il tipo di socket del destinatario (nel nostro caso l'indirizzo del destinatario è sempre zero in quanto è il kernel), e i campi dell'header dei messaggi che s'inviano. Si invia un messaggio al kernel affinché questo salvi il PID del processo per utilizzarlo poi, come destinatario dei messaggi inviati nel socket,

Instaurate le due connessioni, nel programma principale si va ad acquisire, tramite la porta seriale, l'informazione dei secondi trascorsi dall'inizio dell'anno. Una volta acquisiti i dati, tramite un ciclo che termina dopo 10 letture o quando incontra la lettera "P" e salvati i valori all'interno di un array ("sec"), questi sono convertiti, prima in un numero intero ("secondi") e successivamente in un numero binario da fornire alla porta GPIO in modo da essere acquisito dalla FPGA. Il bus che connette la FOX BOARD alla FPGA è di solo 12 bit, ciò crea un problema relativamente al trasferimento di un intero di 32 bit. Considerato che i secondi calcolati dall'inizio dell'anno occupano al massimo 25 bit, si è deciso di trasmettere l'informazione in due blocchi da 10 bit e un blocco da 5 bit. Prima d'iniziare la conversione, si invia un avviso alla FPGA settando un bit a livello alto per indicargli il transito di un valore, questo bit rimane alto per tutto il tempo del passaggio; in seguito si effettua la conversione settando le porte d'uscita della FOX BOARD. Alla fine della conversione di ogni blocco (questa avviene tramite operatori bitwise e alla fine di ogni blocco il dato viene shiftato di 10 posizioni per poter utilizzare un ciclo), viene posto a livello alto un ulteriore pin per far eseguire il salvataggio del dato alla FPGA (questo procedimento si ripete per i tre blocchi).

Dopo aver inviato i dati alla FPGA, si esegue un ciclo, corrispondente a quello principale, che termina premendo Ctrl+C. All'interno del ciclo leggo dalla porta seriale, prima i secondi trascorsi dall'inizio dell'anno e poi i byte dell'errore di quantizzazione che vengono salvati in un array ("errore"). Una volta acquisito questo dato si esegue la conversione del numero in virgola mobile secondo il formato IEEE745. Si riceve, dal socket NetLink, l'informazione riguardante il numero di clock al momento dell'evento di trigger dall'ultimo PPS (ogni messaggio che si riceve dal socket riporta 7 informazioni su altrettanti eventi) e se ne effettua la conversione trasformandolo in una misura di tempo che viene corretta con l'errore di quantizzazione. Il dato così corretto, viene salvato in una coda contenente i

dati da trasferire alla FPGA, che a loro volta, prima di essere inviati vengono convertiti in binario (lo stesso procedimento è applicato ai secondi trascorsi dall'inizio dell'anno all'avvio del programma). Successivamente, si rimane in attesa del socket per un secondo tramite un ciclo, quello principale, che viene ripetuto finché non termina il programma.

Analizziamo ora il programma in kernel-space, di cui nel seguito viene riportato il relativo diagramma di flusso(Fig.4.8)



4.8: Diagramma di flusso del programma in kernel-space

Come ogni modulo, è composto da tre parti. La prima parte (“nlinit”) è costituita dalla funzione che esegue la creazione del kernel tramite la funzione netlink_kernel_create e come parametro richiede il puntatore alla funzione di gestione del socket. Nel nostro caso questa funzione è la nlinput, (seconda parte), la terza e ultima parte è quella fondamentale con la funzione nlclean che elimina il socket.

Le funzioni di creazione ed eliminazione sono semplici in quanto costituite da funzioni del Linguaggio C e Linux, invece la funzione di gestione, interamente sviluppata, è la parte in cui ci si è concentrati di più in quanto è questa la funzione che effettua la gestione del socket e la comunicazione della FPGA.

La funzione nlinput, dopo aver dichiarato le strutture necessarie alla gestione del socket, sapendo che il programma in user-space ha inviato un messaggio nel socket, preleva dal buffer del socket il datagramma relativo al messaggio dello User-space al fine di identificare il PID del processo user-space con il quale comunicare, quindi viene salvato il PID. Successivamente, il modulo entra in regime e si pone a livello logico alto un pin della FOX BOARD per avvertire la FPGA di inviare i dati immagazzinati. Il dato che la FPGA trasferisce è un numero intero senza segno di 32 bit, ma il bus a disposizione per la comunicazione tra FPGA e FOXBOARD e solo di 20 bit, per cui l’FPGA deve inviare due blocchi da 16 elementi. L’invio dei due blocchi è gestito da un impulso che viaggia su un altro pin e che comunica alla FPGA quando è il momento di sostituire il primo blocco in uscita con il secondo. Il primo blocco viene salvato in una variabile intera senza segno e poi convertito in decimale e salvato in una ulteriore variabile (“dato”, variabile che poi conterrà il valore del conta clock), di seguito si estrae il secondo blocco il quale viene convertito e sommato al precedente valore della variabile dato e successivamente è posto a livello logico basso il pin per indicare alla FPGA che non si riceve più il dato. Quindi si converte in stringa la variabile (“dato”) e la si inserisce nel buffer: per ogni variabile “dato” si

devono inviare 4 caratteri, in quanto questa variabile è di 32 bit. Se i dati immagazzinati nella stringa sono 7 o è arrivato il nuovo segnale PPS e allora si definisce l'header del messaggio da inviare e si trasferisce il messaggio nel socket. Questo procedimento è ripetuto all'infinito.

Nella figura seguente (Fig.4.9) è mostrata la parte della scheda VME in cui è montata la FOX BOARD, l'intero programma è riportato in Appendice E.

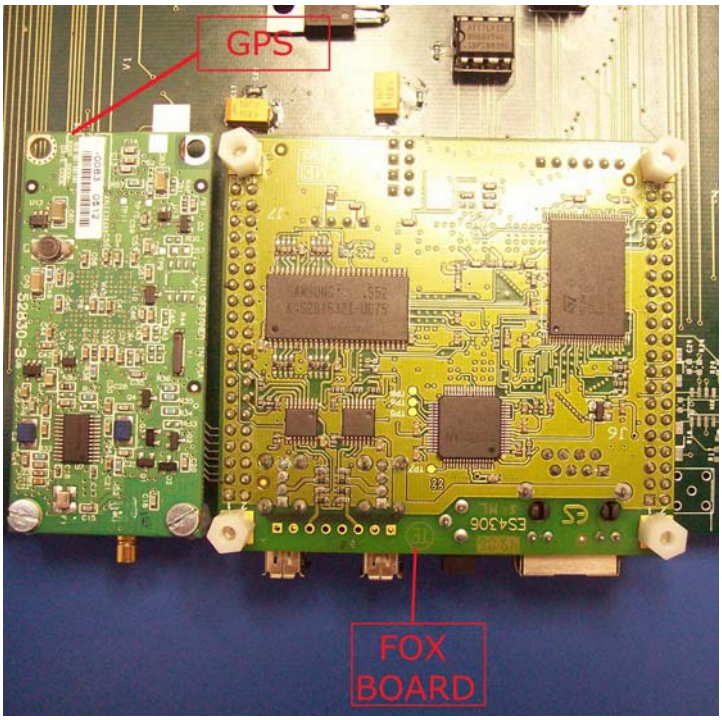


FIg.4.9: Circuito della FOX BOARD sulla scheda VME

4.4 Sviluppo del firmware della FPGA

Le FPGA (Field Programmable Gate Array), sono delle matrice di elementi più o meno complessi che possono essere programmati per realizzare la funzione logica desiderata. Le stesse interconnessioni tra gli elementi possono essere programmate per connettere tra loro i diversi circuiti. Le FPGA costituiscono un'importante evoluzione nel mondo dei dispositivi programmabili poiché forniscono un'elevata potenza di calcolo e di

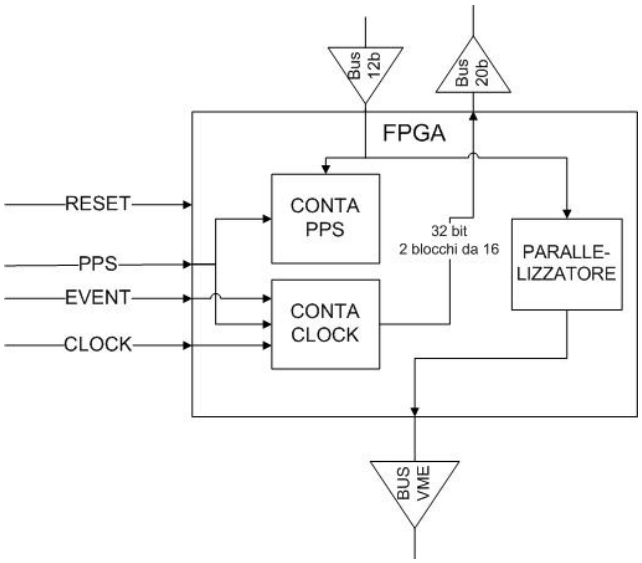
connessione, inoltre i collegamenti locali del dispositivo sono condivisi da pochi elementi logici determinando limitati ritardi di propagazione dagli stati

In commercio esistono diverse FPGA, per questo progetto si è utilizzata una SPARTAN-XCS10XL TQ 144 della XILINX in quanto ha un adeguato numero di connessioni I/O, e la possibilità di programmazione sul campo. Le sue caratteristiche tecniche sono:

Celle logiche	Massimo numero di Gate	Range Tipico di Gate	Matrice CLB	CLB totali	Numero FLIP FLOP	I/O	Bit di RAM totali
466	10000	3000-10000	14x14	196	616	112	6272

Il codice viene trasferito nell'FPGA saldata sulla scheda da una EEPROM montata su zoccolo e opportunamente programmata.

La rete implementata nella FPGA deve assolvere il compito di doppio contatore e, come evidenziato precedentemente, di parallelizzatore. Di seguito è riportato uno schema a blocchi (Fig4.10), che visualizza i segnali di ingresso, di uscita e le varie interconnessioni tra i due contatori implementati nella FPGA:



4.10: Schema a blocchi della FPGA

I segnali provenienti dall'esterno della FPGA, oltre ai bus di comunicazione con la FOX BOARD, hanno la seguente funzione:

- RESET: azzera tutti i valori dei segnali interni e delle uscite, per riportare allo stato iniziale il processo in caso di un arresto accidentale;
- PPS: è il segnale proveniente dal GPS;
- EVENT: segnale di trigger degli eventi;
- CLOCK: segnale dell'oscillatore al quarzo a 50MHz.

Nella figura successiva (Fig.4.11) è rappresentato lo schema circuitale delle connessioni della FPGA con la FOX BOARD e con gli altri elementi della scheda VME:

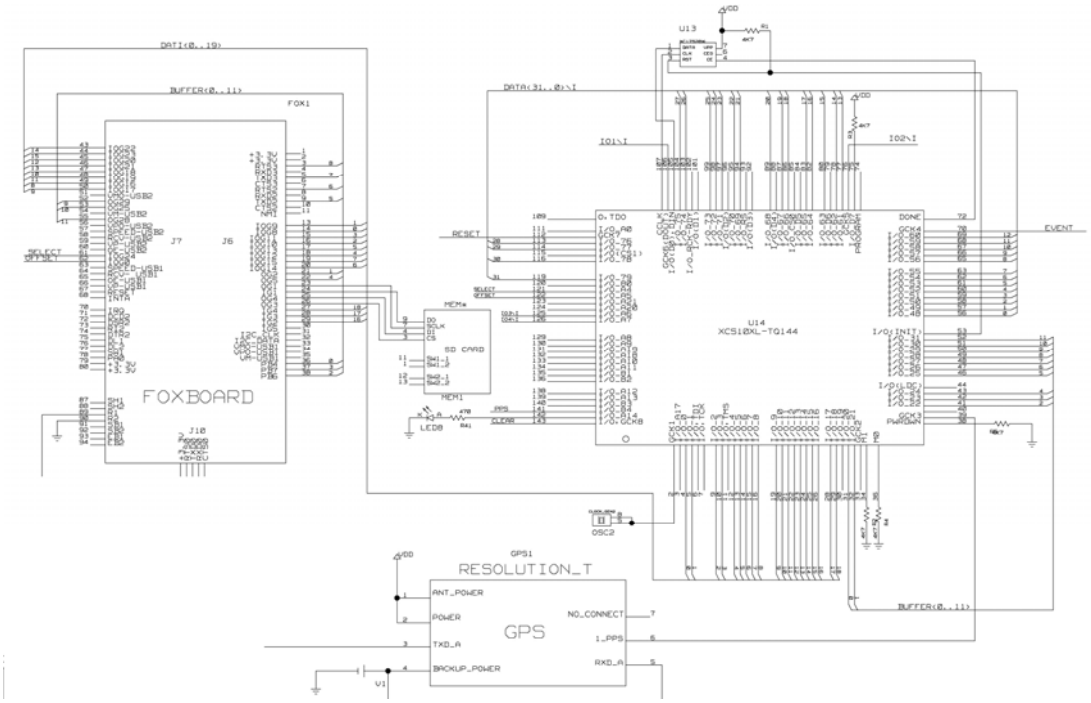


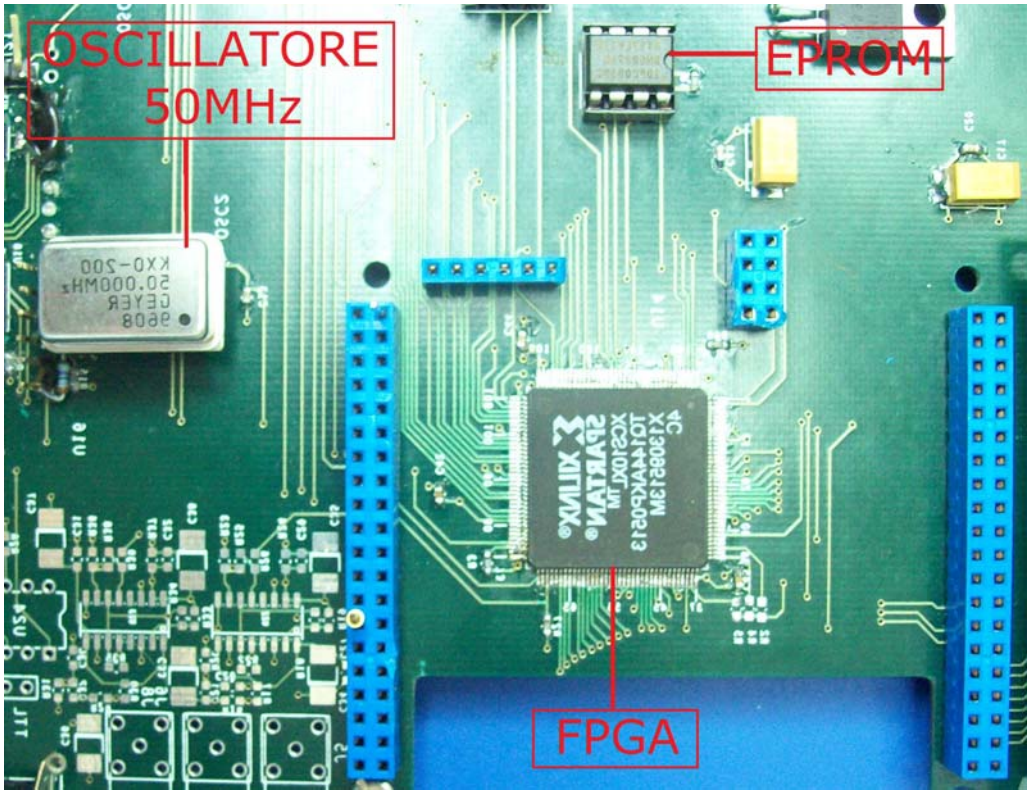
Fig.4.11: Schema circuitale della FPGA e dei suoi collegamenti

Esaminiamo ora il funzionamento logico dei blocchi interni alla FPGA. Il “CONTA CLOCK” esegue il conteggio dei cicli di clock dall’arrivo dell’ultimo PPS fino alla ricezione dell’impulso del segnale EVENT; utilizzando il quarzo a 50MHz si riesce a misurare anche le frazioni del secondo. Per far ciò si utilizzano due segnali PPS ritardati rispetto a quello reale in modo da prevenire l’arrivo del PPS, salvare il conteggio precedente con la ricezione del primo dei due segnali e con il secondo resettare il contatore. Si è riusciti a normalizzare i cicli di clock calcolando l’errore che si commetteva rispetto al precedente PPS, per cui, per entrare in regime il blocco conta clock deve aspettare almeno 3 PPS. Una volta acquisito il dato dei cicli di clock all’arrivo dell’evento, per essere trasmesso viene spezzato in due blocchi poiché il dato è di 32 bit e il bus di comunicazione è di solo 20 bit e inviato alla FOX BOARD

Il CONTA PPS è un semplice contatore a cui è assegnato un valore iniziale pari al valore dei secondi dall’inizio dell’anno, dato passato dalla FOXBOARD e che, ad ogni fronte di salita del segnale PPS viene incrementato di un passo.

Il blocco “PARALLELIZZATORE” è un processo implementato per richiedere i dati alla FOX BOARD che li trasmette a blocchi a causa di un insufficiente bus di comunicazione, unirli in unico blocco per poi trasmetterlo sul bus VME.

Nella figura seguente (Fig.4.12) è mostrata la parte della scheda VME in cui è montata la FPGA, l’intero programma è riportato in Appendice E; la sua funzionalità è stata verificata attraverso una simulazione.



4.11: Circuito della FPGA sulla scheda VME

Conclusioni e sviluppi futuri

Si è riusciti ad effettuare il time stamping degli eventi e la sincronizzazione degli apparati, tramite l'utilizzo di diversi dispositivi programmati in maniera opportuna per il raggiungimento dell'obiettivo.

Nello sviluppo del modulo elettronico i punti fondamentali trattati sono i seguenti:

- sviluppo del firmware della PIC;
- sviluppo del firmware della FPGA;
- sviluppo del programma della FOX BOARD;
- test dei programmi.

Grazie all'utilizzo del ricevitore GPS si è ottenuto un riferimento temporale di alta precisione, infatti il ricevitore per comunicare utilizza il protocollo TSIP, il quale permette di avere la massima precisione ottenibile. Le informazioni temporali per effettuare il time stamping sono state estratte dai pacchetti del protocollo tramite una PIC per la quale è stato sviluppato un firmware adatto. Il problema di avere una precisione dell'ordine del nanosecondo è stato risolto utilizzando una FPGA, sincronizzata con il GPS tramite il PPS, con un clock esterno ottenuto con un oscillatore al quarzo da 50MHz. La FPGA riceve i secondi trascorsi dall'inizio dell'anno e viene incrementata ad ogni PPS, affiancata da un contatore che riparte ad ogni PPS e che permette di effettuare il time-stamping relativamente alle frazioni del secondo. Tramite un particolare algoritmo si è riusciti anche a normalizzare i cicli di clock dell'oscillatore e la FPGA implementa inoltre la parallelizzazione dei dati.

Di seguito, si è sviluppato il programma del sistema embedded, FOX BOARD, la quale combina i dati provenienti dalla PIC e dalla FPGA per correggere il dato e successivamente fornirlo in corrispondenza di una richiesta.

Gli sviluppi futuri riguardano l'integrazione del modulo con il bus VME, prevedere una FPGA con una memoria maggiore in modo che la FOX BOARD una volta corretto il dato lo invii direttamente alla FPGA la quale avrà il compito di gestire la comunicazione con il bus VME inoltre, si dovrebbe prevedere almeno un ingresso di clock della FPGA sia per il bus di input che di output per avere in questo modo dei processi sensibili alla commutazione ed eventualmente valutare la possibilità di adoperare un'altra tipologia di sistema embedded che non simuli le operazioni in virgola mobile ma che le effettui a livello hardware diminuendo così i ritardi di elaborazione.

APPENDICE A

Compilazione e caricamento del Kernel

Per effettuare la compilazione del kernel, modificato per questa applicazione, si deve installare sulla propria macchina la SDK in ambiente Linux o utilizzando una macchina virtuale negli ambienti windows inoltre, tramite la SDK si possono anche compilare direttamente i programmi senza l'utilizzo del webcompiler.

Una volta installata la SDK, dalla cartella principale occorre spostarsi nella sottodirectory devboard-R2_01 (con il comando cd) e lanciare ./configure; quando il sistema ha completato la configurazione dei parametri si può lanciare il primo comando **make** con **menuconfig** (Fig A.1) (*make menuconfig*). Dopo che la SDK ha letto tutte le configurazioni si aprirà la seguente schermata:

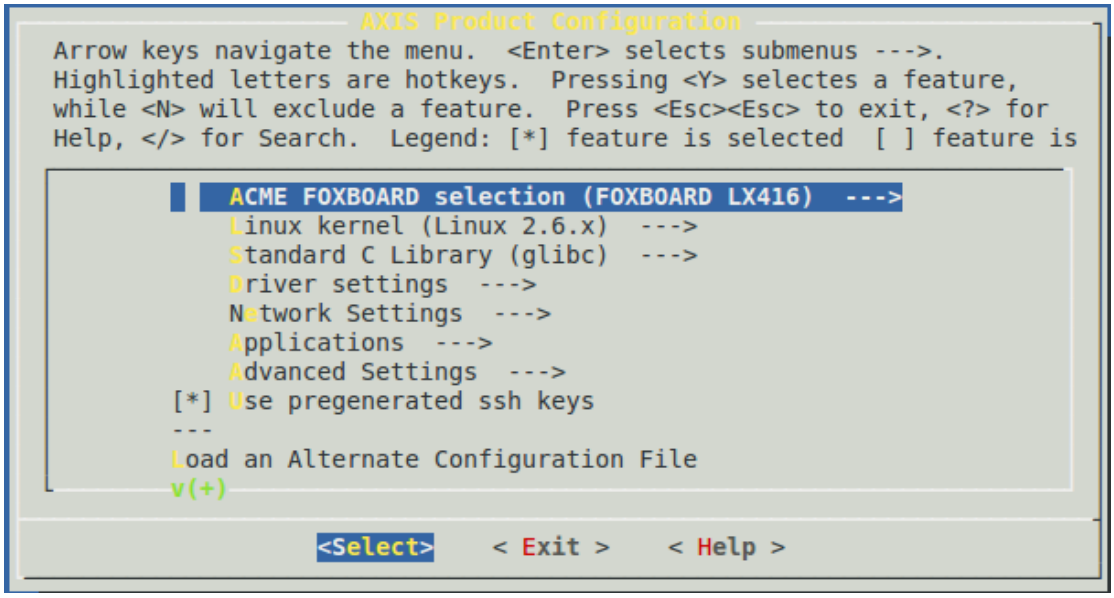


Fig. A. 1: Schermata menuconfig

Nella schermata si può scegliere la versione della FOX BOARD (nel nostro caso LX 416), la versione del kernel tra le 2.4 e la 2.6, la libreria che si utilizza in C e l'attivazione di altre periferiche supportate dalla FOX BOARD che di default sono disattivate.

Il secondo comando **make** è **com kernelconfig** (Fig.(*make kernelconfig*), dopo averlo lanciato si aprirà la seguente schermata:

Fig. A. 2: Schermata kernelconfig

Da qui si può gestire il funzionamento di tutte le periferiche (input, dispositivo a carattere, file system, porte seriali, attivazione del lettore SDCARD) e la configurazione di base delle porte (se sono di I/O o se sono utilizzati per altri scopi): sottomenu Hardware setup.

Per default la FOX BOARD possiede solo alcuni comandi della shell di linux, per aggiungerli si lancia **make** con **busybox** (Fig. A.3) (*make busybox*) ottenendo la seguente schermata:

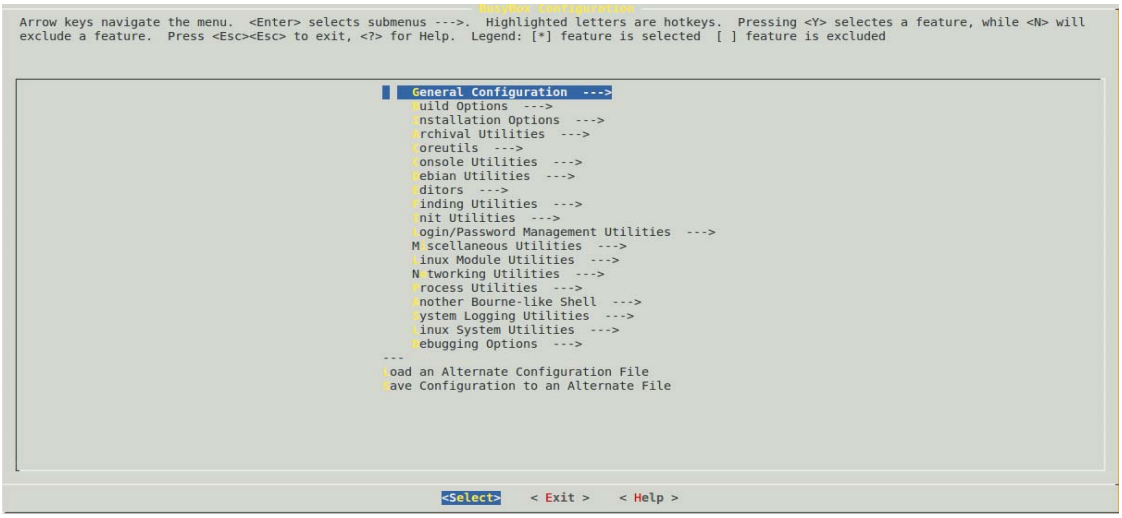


Fig. A. 3: Schermata busybox

Una volta finita la configurazione del kernel si lancia nuovamente ./configure e successivamente il comando make per la ricompilazione del kernel e generazione della fImage.

Una volta creata la fImage occorre trasferire FOX BOARD, il metodo usato in questo progetto usa la porta ethernet e il web server della stessa FOX BOARD. Aprendo un broswer e scrivendo nella barra degli indirizzi l'indirizzo IP della FOX BOARD (192.168.0.90) si apre l'home page della fox board (Fig. A.4).

Fig. A. 4: Home Page FOX BOARD

In fondo alla pagina ci sono due pulsanti che permettono di selezionare due tipi di upgrade (caricamento kernel)

- Update FOX firmware - system area che abilita l'update (caricamento del nuovo kernel) solo sulla partizione di sola lettura della scheda.

- Update FOX firmware - entire system che va a riscrivere tutta la FLASH memory.

Prima di selezionare una delle due opzioni, si deve specificare la posizione della fImage e cliccare su uno dei due bottoni, il browser chiederà conferma dell'operazione ed apparirà poi la seguente schermata (Fig. A.5):

```
Preparing system for upgrade ...
Starting run level 4 (stop most daemons) ...
Waiting for run level 4 to start ...
Run level 4 started.
Waiting for run level 4 to finish ...
... waiting ...
Run level 4 finished.
Stopping some remaining processes.
   sending TERM signal ...
Most processes stopped.
The file system will be upgraded after reboot.
Unmounting file system /var ...
   umount /var failed with exit status: 1
Unmounting file system /mnt/flash ...
Receiving new firmware ...
No HWID in bootblock, continuing anyway.
Erasing old file system ...
   1%    3%    5%    7%    9%   11%   13%   15%   16%   18%   20%   22%
  24%   26%   28%   30%   32%   33%   35%   37%   39%   41%
```

Fig. A. 5: Schermata del caricamento della fImage

Una volta terminata la cancellazione del sistema e la riscrittura, la FOX BOARD verrà riavviata automaticamente e sarà possibile utilizzarla con le proprie configurazioni.

APPENDICE B

Registri delle porte della FOX BOARD

Di seguito sono riportati i registri e i campi della **porta A**:

Registro **R_PORT_PA_DATA,** di sola scrittura, e suoi campi:

Bit(s)	Nome	Descrizione	Range
7 - 0	data_out	Data byte to general port PA.	0 - 255

Registro **R_PORT_PA_DIR** e suoi campi, il registro è di sola scrittura e serve per definire la direzione dei pin:

Bit(s)	Nome	Descrizione	Range
7	dir7	Questo bit seta la direzione (**input** o **output**) del bit 7 della porta PA.	0=input 1=output
6	dir6	Questo bit seta la direzione (**input** o **output**) del bit 6 della porta PA.	0=input 1=output
5	dir5	Questo bit seta la direzione (**input** o **output**) del bit 5 della porta PA.	0=input 1=output
4	dir4	Questo bit seta la direzione (**input** o **output**) del bit 4 della porta PA.	0=input 1=output
3	dir3	Questo bit seta la direzione (**input** o **output**) del bit 3 della porta PA.	0=input

			1=output
2	dir2	Questo bit seta la direzione (**input** o **output**) del bit 2 della porta PA.	0=input 1=output
1	dir1	Questo bit seta la direzione (**input** o **output**) del bit 1 della porta PA.	0=input 1=output
0	dir0	Questo bit seta la direzione (**input** o **output**) del bit 0 della porta PA.	0=input 1=output

Registro **R_PORT_PA_READ,** di sola scrittura, e suoi campi:

Bit(s)	**Nome**	**Descrizione**	**State/Range**
31 - 8	Reserved	-	
7 - 0	data_in	Dati della porta PA.	0 - 255

Di seguito sono riportati i registri e i campi della **porta B**:

Registro **R_PORT_PB_DATA,** di sola scrittura, e suoi campi:

Bit(s)	**Nome**	**Descizione**	**Range**
7 - 0	data_out	Byte d'uscita dei pin che sono configurati come I/O della porta PB.	0-255

Registro **R_PORT_PB_DIR** e suoi campi, il registro è di sola scrittura e serve per definire la direzione dei pin:

Bit(s)	Nome	Descrizione	Range
7	dir7	Questo bit seta la direzione (**input** o **output**) del bit 7 della porta PB. Il campo dir7 è ignorato se il campo scsi1 di R_PORT_PB_SET è pari a enph, o se il campo syncser3 di R_PORT_PB_SET èpari a ss3extra o se il campo cs7 in R_PORT_PB_SET è pari a cs.	0=input 1=output
6	dir6	Questo bit seta la direzione (**input** o **output**) del bit 6 della porta PB. Il campo dir6 è ignorato se il campo cs6 in R_PORT_PB_SET è pari a cs.	0=input 1=output
5	dir5	Questo bit seta la direzione (**input** o **output**) del bit 5 della porta PB. Il campo dir5 è ignorato se il campo cs5 in R_PORT_PB_SET è pari a cs.	0=input 1=output
4	dir4	Questo bit seta la direzione (**input** o **output**) del bit 5 della porta PB. Il campo dir4 è ignorato se il campo scsi0 di R_PORT_PB_SET è pari a enph, o se il campo syncser1 di R_PORT_PB_SET è pari a ss1extra o se il campo cs4 in R_PORT_PB_SET è pari a cs.	0=input 1=output
3	dir3	Questo bit seta la direzione (**input** o **output**) del bit 3 della porta PB. Il campo dir3 è ignorato se il campo cs3 in R_PORT_PB_SET è pari a cs.	0=input 1=output
2	dir2	Questo bit seta la direzione (**input** o **output**) del bit 2 della porta PB. Il campo dir2 è ignorato se il campo cs2 in R_PORT_PB_SET è pari a cs.	0=input 1=output
1	dir1	Questo bit seta la direzione (**input** o **output**) del bit 1 della porta PB.	0=input 1=output

		Il campo dir1 è ignorato se il campo i2c_en in R_PORT_PB_SET è pari a on.	
0	dir0	Questo bit seta la direzione (**input** o **output**) del bit 0 della porta PB. Il campo dir0 è ignorato se il campo i2c_en in R_PORT_PB_SET è pari a on.	0=input 1=output

Registro **R_PORT_PB_READ,** di sola scrittura, e suoi campi:

Bit(s)	**Nome**	**Descrizione**	**State/Range**
31 - 8	Reserved	-	
7 - 0	data_in	Dati della porta PB.	0 - 255

Di seguito sono riportati i registri e i campi della **porta G**:

Registro **R_PORT_G_DATA** e i suoi campi per leggere e scrivere la porta G, il registro è di scrittura e lettura:

Bit(s)	**Nome**	**Descrizione**	**Range**
31 - 0	data	Attraverso questo registro si può leggere e scrivere i pin G0-31.	

Alcuni campi del registro **R_GEN_CONFIG** permettono di definire la direzione dei pin della porta G:

Bit(s)	**Nome**	**direzione**	**State**
27	g24dir	Questo bit seta la direzione (input o output) del pin g24 (se	0 = in

		configurato)	1 = out
26	g16_23dir	Questo bit seta la direzione (input o output) del pin g16-23 (se configurato)	0 = in 1 = out
25	g8_15dir	Questo bit seta la direzione (input o output) del pin g16-23 (se configurato)	0 = in 1 = out
24	g0dir	Questo bit seta la direzione (input o output) del pin g0 (se configurato)	0 = in 1 = out

Per maggiori informazioni sui registri e sul multiplexing delle interfacce si veda il riferimento [8].

APPENDICE C

Compilazione, caricamento e scaricamento di un modulo kernel

I moduli sono sezioni di codice che permettono l'espansione delle funzionalità del kernel durante la sua esecuzione tramite altre parti di codice che devono essere caricati e\o scaricati: questa versatilità, rappresenta una delle migliori caratteristiche di Linux. Un modulo del kernel può implementare un device driver, un file system o un protocollo di networking. Il supporto di Linux per la gestione dei moduli si divide in tre componenti:

1. il manager dei moduli, provvede al caricamento in memoria dei moduli, al controllo della loro attività e alla loro interazione con il resto del kernel;

2. la registrazione dei driver, permette ai moduli di informare il kernel che un nuovo driver è disponibile aggiornando una tabella dinamica;

3. la risoluzione dei conflitti, un meccanismo che permette a driver diversi di riservare risorse hardware e di proteggere queste risorse dall'uso accidentale da parte di un altro driver.

Per creare un modulo per la FOX BOARD partendo dalla cartella in cui è stata installata la SDK, occorre spostarsi nella sotto-directory drivers (cd os/linux-2.6/drivers), e creare la cartella che dovrà contenere il modulo. Una volta creata ed aperta la cartella si può dar vita al modulo (per esempio nano infoPA.c, il codice è quello riportato in figura (Fig.C.1)è un semplice modulo che stampa su un file di sistema i registri delle port)

```c
/*nella prima libreria sono definite le macro per risalire alla versione del kernel correntee per utilizzare il modulo,
  mentre nella seconda è definita per esempio la funzione printk l'analoga della funzione printf in C*/
#include <linux/module.h>
#include <asm/io.h>

/*se non si dichiara una licenza GPL ( licenza open source di GNU ) durante la compilazione del modulo si avrà un warning
riferito alla licenza non GPL.*/
MODULE_LICENSE("DUAL GPL/BSD");

/*Un modulo del kernel deve avere al minimo due funzioni: init_module, chiamata al momento del caricamento, e cleanup_module,
chiamata al momento della rimozione. La prima funzione si occupa dell'inizializzazione del modulo: ricerca dell'hardware e
registrazione del nuovo device driver all'interno delle tabelle del kernel; la seconda ha invece il compito di rilasciare
le risorse usate dal modulo e di cancellare il device driver dalle tabelle del kernel*/

void infoPA_cleanup_module(void)
{ /*Non devo fare nulla*/}

int infoPA_init_module(void)
{  /*printk ha varie priorità in base hai messaggi che si vuole dare al kernel: 1. KERN_EMERG   "<0>" sistema instabile
     2. KERN_ALERT   "<1>" stato di allerta 3. KERN_CRIT    "<2>" stato critico 4. KERN_ERR     "<3>" stato di errore
     5. KERN_WARNING "<4>" stato di pericolo 6. KERN_NOTICE  "<5>" condizione normale 7. KERN_INFO    "<6>" informazione
     8. KERN_DEBUG   "<7>" messaggio di debug           */
   printk(KERN_ALERT "\nIndirizzi della porta PA:\n - R_PORT_PA_SET  = %8X\n - R_PORT_PA_DATA = %8X\n - R_PORT_PA_DIR  = %8X\n",
          (int)R_PORT_PA_SET, (int)R_PORT_PA_DATA, (int)R_PORT_PA_DIR);
   printk(KERN_ALERT "\nValori della porta PA:\n - R_PORT_PA_SET  = %8X\n - R_PORT_PA_DATA = %8X\n - R_PORT_PA_DIR  = %8X\n",
          *R_PORT_PA_SET, *R_PORT_PA_DATA, *R_PORT_PB_DIR);
   *R_IRQ_MASK1_SET = 0x0F;
   printk(KERN_ALERT "R_IRQ_MASK1_SET__pa0__BITNR: %X\n\n", R_IRQ_MASK1_SET__pa0__BITNR);
   return 0;
}

/*INIT & EXIT*/
module_init(infoPA_init_module);
module_exit(infoPA_cleanup_module);
```

Fig.C. 1: Esempio di modulo

Successivamente, all'interno della stessa cartella in cui si è memorizzato il file .c, si deve creare il file Makefile (nano Makefile) contenente uno dei seguenti comandi:

- obj-y += nome modulo.o (obj-y += infoPA.o)
- obj-m += nome modulo.o (obj-m += infoPA.o),

la differenza tra i due comandi è che con il primo comando il modulo sarà compilato e inglobato all'interno del kernel come modulo del kernel stesso, invece con il secondo comando il modulo viene solo compilato. Un modulo compilato è privo di errori se, nella cartella che lo contiene, dopo la fase di compilazione compare un file con estensione .ko (kernel object).

Si passa poi a modificare il file Makefile spostandosi nella sottodirectory Drivers (cd .. se la posizione corrente è quella dove il file .c è stato creato) si apre il Makefile (nano Makefile) e alla fine del file si aggiunge la riga obj-y += cartella/ (cartella corrisponde alla cartella in cui si è creato il modulo, nel nostro caso obj-y += modulo/).

Per compilare il modulo in modo non monolitico è sufficiente lanciare make nella directory del kernel e ricompilare l'intero kernel, oppure lanciare il comando make module in os/linux-2.6/,(in entrambi i casi ricordare prima di eseguire il comando . init_env in devboard-R2_01).

Una volta compilato il modulo, nella directory dove si è creato il file .c, il Makefile ha salvato un file con estensione .ko (infoPA.ko) che deve essere trasferito sulla FOX BOARD (si può anche usare scp).

Dopo il trasferimento, per caricare il modulo si utilizza il comando insmod (Fig.C.2) che lo scarica tramite rmmod (Fig.C.2); per vedere le informazioni stampate tramite printk, si deve leggere il file messages (questo perchè nel comando si è usato la macro KERN_ALERT; lo stesso effetto si avrebbe con KERN_INFO) presente in /var/log/ (Fig.C.2):

```
[root@axis-00408c120232 /root]101# insmod /mnt/flash/infoPA.ko
Using /mnt/flash/infoPA.ko
[root@axis-00408c120232 /root]101# tail /var/log/messages
Jan  1 00:13:34 axis-00408c120232 kernel:  - R_PORT_PA_SET  = B0000030
Jan  1 00:13:34 axis-00408c120232 kernel:  - R_PORT_PA_DATA = B0000030
Jan  1 00:13:34 axis-00408c120232 kernel:  - R_PORT_PA_DIR  = B0000031
Jan  1 00:13:34 axis-00408c120232 kernel:
Jan  1 00:13:34 axis-00408c120232 kernel: Valori della porta PA:
Jan  1 00:13:34 axis-00408c120232 kernel:  - R_PORT_PA_SET  =       F3
Jan  1 00:13:34 axis-00408c120232 kernel:  - R_PORT_PA_DATA =       F3
Jan  1 00:13:34 axis-00408c120232 kernel:  - R_PORT_PA_DIR  =        0
Jan  1 00:13:34 axis-00408c120232 kernel: R_IRQ_MASK1_SET__pa0__BITNR: 0
Jan  1 00:13:34 axis-00408c120232 kernel:
[root@axis-00408c120232 /root]101# rmmod infoPA
```

Fig.C. 2: /var/log/messages e comandi insmod e rmmod

Se utilizzando il comando insmod non si mette l'estensione .ko, il sistema carica il modulo presente nel kernel indipendentemente dalla pathname in cui viene lanciato il comando.

APPENDICE D

Codice della PIC

```
INCLUDE "modedefs.bas" 'funzioni porta seriale
TRISA=0  'setto i pin della porta A come uscite
adCON1 = 7 'setta la porta A come porta digitale
'definisce la frequenza in MHz dell'oscillatore
DEFINE      OSC   20
'posizione del bootloader in maniera da non sovrascrivere
DEFINE  LOADER_USED 1
' parametri del display LCD
DEFINE      LCD_DREG PORTB
DEFINE      LCD_DBIT 0
DEFINE      LCD_EREG PORTB
DEFINE      LCD_EBIT 4
DEFINE      LCD_RSREG PORTB
DEFINE      LCD_RSBIT 5
DEFINE      LCD_LINES 2
DEFINE      LCD_BITS 4
DEFINE  LCD_COMMANDUS 2000
DEFINE  LCD_DATAUS 50
'imposta il bit di parità della comunicazione
DEFINE SER2_ODD 1
'imposta il numero di bit dei dati in questo caso 8
DEFINE SER2_BITS 9
'assegna i pin della PIC a cui sono collegati i led a una
variabile
L1 VAR PORTA.0
'controlla il funzionamento del led(1 on, 0 off)
L1=1:PAUSE 100:L1=0
L2 VAR PORTA.1
L2=1:PAUSE 100:L2=0
L3 VAR PORTA.2
L3=1:PAUSE 100:L3=0
L4 VAR PORTA.3
L4=1:PAUSE 100:L4=0
PAUSE 100
anno VAR WORD
'giorni dall'inizio dell'anno
daystart VAR WORD
'vettore  in  cui  memorizzo  i  giorni  passati  dall'inizio
‘dell'anno
day VAR WORD[12]
```

```
day[0]=0:day[1]=31:day[2]=59:day[3]=90:day[4]=120:day[5]=151
day[6]=181:day[7]=212:day[8]=243:day[9]=273:day[10]=304
day[11]=334
'count per i cicli
i VAR BYTE
x VAR BYTE
'variabile dove memorizzo il pacchetto 8F-AB
pab VAR BYTE[18]
'variabile dove memorizzo il pacchetto 8F-AC
pac VAR BYTE[67]
sat VAR BYTE 'numero satelliti
'variabile in cui memorizzo i secondi dall'inizio dell'anno
sec VAR LONG
main:
 'prova del funzionamento del display
 'cancella tutto il contenuto del display
 LCDOUT $FE, 1 : LCDOUT "DATI PROVENIENTI"
'scrive all'inizio della seconda riga
 LCDOUT $FE, $C0, "DAL GPS"
 start:
  'richiesta del pacchetto 6D tramite il pacchetto 24 per il
  'num. di satelliti
  SEROUT2 PORTC.2,8276,[$10,$24,$10,$03]
  'ricevo il pacchetto 6D e estraggo il numero di satelliti
  SERIN2 PORTC.0,8276,[WAIT($10), wait ($6D),sat]
  'ricevo il pacchetto AB e per estrarre la data e ora
  SERIN2 PORTC.0,8276,[WAIT($8F),WAIT($AB),STR pab\18]
  'ricevo il pacchetto AC e per estrarre l'errore di
  'quantizzazione
  SERIN2 PORTC.0,8276,[WAIT($8F),WAIT($AC),STR pac\67]
  'siccome sat è una variabile di 8 bit, memorizza il byte 6D
  'ma l'informazione del numero dei satelliti è solo i 4 bit '
  'più alti del byte,quindi lo shifto per avere il valore
  'corretto
  sat=sat>>4
  'controllo il numero di satelliti per accendere i led
  SELECT CASE  sat
      CASE 0
        L1=0
        L2=0
        L3=0
        L4=0
      CASE 1
        L1=1
        L2=0
        L3=0
        L4=0
      CASE 2
        L1=1
        L2=1
        L3=0
        L4=0
      CASE 3
        L1=1
        L2=1
        L3=1
```

```
      L4=0
    CASE IS > 3
      L1=1
      L2=1
      L3=1
      L4=1
END SELECT
'elaborazione dati: controllo i dati ricevuti sono
'affidabili solo se vedo più di 3 satelliti
IF sat>3 THEN
    daystart=0
    'cancello i possibili doppi 10 presenti nel pacchetto AB
    FOR i=0 TO 17
        IF (pab[i]==$10)&&(pab[i+1]==$10)THEN
            FOR x=i TO 16
                pab[x]=pab[x+1]
            NEXT
        ENDIF
    NEXT
    'cancello i possibili doppi 10 presenti nel pacchetto AC
    FOR i=0 TO 66
        IF (pac[i]==$10)&&(pac[i+1]==$10)THEN
            FOR x=i TO 65
                pac[x]=pac[x+1]
            NEXT
        ENDIF
    NEXT
    'salvo il valore dell'anno
    anno.Byte1 = pab[14]
    anno.Byte0 = pab[15]
    'giorni dall'inio dell'anno
    daystart=day[pab[13]-1]+pab[12]
    'controllo se l'anno bisestile e se il mese è successivo
    ' a febbraio
    IF(((anno//4==0)&&(anno//100!=0))||(anno//400==0)&&
        pab[13]>2)THEN
            daystart=daystart+1     'aggiungo un giorno
    ENDIF
    'calcolo i secondi dall'inizio dell'anno
    sec=3600*(24*daystart+pab[11])+pab[10]*60+pab[9]
    'stampo sul display le informazioni del gps di data e
    'ora, num staelliti
    LCDOUT $FE, 1,DEC pab[12],"/",DEC pab[13],"/",DEC anno
    LCDOUT $FE,$C0,DEC pab[11],":",DEC pab[10]
    LCDOUT ":",DEC pab[9]," "
    LCDOUT DEC daystart," ", DEC sat
    'passo le informazioni tramite seriale alla FOX BOARD
    'invia l'informazione sui secondi
    SEROUT2 PORTC.1,8276,[DEC sec]
    'uso la lettera P per indicare alla FOX BOARD che è
    'finito il dato sui secondi
    SEROUT2 PORTC.1,8276,["P"]
    'invio l'errore di quantizzazione alla FOX BOARD
    SEROUT2 PORTC.1,8276,[pac[59]]
    SEROUT2 PORTC.1,8276,[pac[60]]
    SEROUT2 PORTC.1,8276,[pac[61]]
```

```
  ELSE
     'stampo l'informazione che i satelliti sono insufficienti
      LCDOUT $FE, 1,"Satelliti: ", DEC sat
      LCDOUT $FE,$C0,"insufficienti"
  ENDIF
 GOTO start
END
```

APPENDICE E

Programma della FOX BOARD

Prima viene riportato il codice del modulo:

```
/*****************************************************************
  il modulo ottiene un messaggio dallo user-space, ottenengo il PID
  del processo, poi kernel acquisisce i dati dalle GPIO e invia
  questi dati tramite il socket
*****************************************************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/types.h>
#include <linux/socket.h>
#include <linux/netlink.h>
#include <net/sock.h>

// costanti e macro per il modulo NETLINK
#define NETLINK_TEST        30
#define MAX_PAYLOAD_CHAR    28
#define MAX_PAYLOAD_LENGTH  sizeof(struct nl_payload_t)

MODULE_LICENSE("GPL");

/* La struttura è definita arbitrariamente e deve contenere il pid
   del processo mittente e i dati da stampare  Il campo `multicast'
   indica se il messaggio è di tipo multicast*/
typedef struct nl_payload_t
{
   int sender_pid;
   int multicast;
   char str[MAX_PAYLOAD_CHAR];
} nl_payload_t;

static struct sock *nl_sk = NULL;

/*nl_input è la funzione che deve essere passata al momento della
creazione   dello stesso socket, la funzione riceve dal processo in
user-space il pid e poi acquisisce e invia i dati al kernel in un
ciclo interno */
void nlinput(struct sock *sk, int len)
{
   // puntatore al buffer  in cui si memorizzano i dati
   struct sk_buff *buf = NULL;
   // struttura dell'header
   struct nlmsghdr *nlh;
   //è la variabile usata per l'acquisizione dalla porte
   unsigned int val,val1;
   //varabile per salvare il PID del processo utente
   int Pidprocesso;
```

```
/*dato è il valore ricavato in questo momento invece dato1 è il
  valore ricavato nell'istante precedente in questo modo sono
  sicuro confrontandoli di non acquisire più volte lo stesso dato
  e inoltre riesco a dire anche quando è arrivato il PPS*/
unsigned int dato,dato1;
/*conta dati è un contatore che mi indica quanti dati ho inserito
  nella stringa da inviare*/
int countdati;
int err,i;
//payload
struct nl_payload_t *payload;
/*aspetto di ricevere un messaggio dall'user-space da cui poter
  prelevare il PID in maniera tale discriminare il destinatario
  del messaggio*/
// Prelevo il datagramma dalla coda del buffer.
buf = skb_recv_datagram(sk, 0, 0, &err);
//prelevo il la struttura del messaggio compreso l'header NETLINK
nlh = (struct nlmsghdr *)buf->data;
// prelevo il payload, ossia i dati effettivi
payload = (struct nl_payload_t *)NLMSG_DATA(nlh);
// A questo punto conosco il mittente.
/* Questo è controllo di sicurezza nel caso che il PID inviato è
   lo stesso del kernel ma questo non può quasi mai succedere  */
if(payload->sender_pid == 0)
     return;
//salvo il PID del processo user space
Pidprocesso=payload->sender_pid;
dato1 =0;
/*inizio la lettura dei dati che in inizia la FPGA, i dati
  inviati dalla FPGA hanno lunghezza 32 bit e il bus di
  comunicazione è di solo 20 bit quindi si inviano due blocchi da
  16 bit e si avverte la FPGA di cambiare il blocco che invia
  mandando un impulso sul pin IG4, invece tramite il piedino IG3
  si dice alla FPGA che la FOX BOARD può ricevere dati*/
countdati=0;
while(1)
{
   /*setto a uno il piedino IG3 per dire alla FPGA che posso rice
     vere dati*/
   *R_PORT_G_DATA=*R_PORT_G_DATA|(1<<3);
   dato=0;
   //ciclo per l'acquisizione dei dati sulla porta G
   for (i=0;i<2; i++)
   {
         val = *R_PORT_G_DATA;
         /*questo controllo serve per far inviare l'impulso solo
           una volta*/
         if (i==0)
         {
           /*in questo modo salvo le due parti meno significative
             dei due dati per capire se sono eventi diversi o
             sempre lo stesso*/
           val2=val1;
           val1=val;
           *R_PORT_G_DATA=*R_PORT_G_DATA|(1<<4);
           *R_PORT_G_DATA=*R_PORT_G_DATA&~(1<<4);
```

```
            }
            /*in questo modo prendo solo i primi 16 bit di val,
              e nel secondo ciclo questi verranno anche shiftati
              alla posizioni più significative*/
            dato=dato + ((val&4095)<<(i*16));
        }
        /*pongo a zero il pin IG3 per avvertire la fpga che non sto
          leggendo le porte GPIO*/
        *R_PORT_G_DATA=*R_PORT_G_DATA&~(1<<3);
        /*in questo modo sarò sicuro che si tratta di un altro dato
          passato e non del dato precedente*/
        if (val2!=val1)
        {
            for (i=0;i<4;i++)
               payload->str[i+countdati]=(char)((dato>>(31-8*i)));
            countdati++;
            dato1=dato;
        }
        /*questo caso riguardo quando arriva il PPS è il dato è zero e
          quindi dato1 è maggiore in questo caso invio le informazioni
          oppure quando risulta che la stringa ha 7 dati*/
        if ((dato<dato1)||(countdati==7)
        {
            //invio il messaggio al user-space
            NETLINK_CB(buf).pid = 0;
            // PID a cui è destinato il messaggio
            NETLINK_CB(buf).dst_pid = Pidprocesso
            NETLINK_CB(buf).dst_group = 0;
            //invio il messaggio
            netlink_unicast(nl_sk, buf, payload->sender_pid,
                            MSG_DONTWAIT);
            /* procedo ad inseire il nuovo dato nel caso sono
               all'interno  del nuovo PPS*/
            if (countdati<7)
            {
              /* sono sicuro che sono all'inizio della stringa quindi
                 scrivo nelle prime posizioni*/
                 for (i=0;i<4;i++) payload->str[i]=(char)((dato>>(31-
                                                          8*i)));
                 countdati=1;
                 dato1=0;
            }
        }
    }
}


/*nl_init_module non fa altro che creare il socket NETLINK e
  associare la funzione da richiamare*/
int nlinit(void)
{
    nl_sk = netlink_kernel_create
          (NETLINK_TEST,0,nlinput,THIS_MODULE);
    if(nl_sk)
      printk(KERN_INFO "Socket creato: %u\n", (unsigned)nl_sk);
    else
```

```
     printk(KERN_ALERT "Socket non creato: %u\n", (unsigned)nl_sk);

   return 0;
}


/*nl_cleanup_module deve assolutamente eliminare il socket
NETLINK.*/
void nlclean(void)
{
   sock_release(nl_sk->sk_socket);
}

module_init(nlinit);
module_exit(nlclean);
```

Ecco il codice del programma in user-space:

```
/*-----------------programma in user-space----------------------*/
/*Name    : foxboard.c                                         */
/*il programma mi deve leggere legere  le informazioni dalla    */
/*porta seriale e convertirle nel formato richiesto, successi-  */
/*legere  l'informazioni del socket NetLink e convertile, ese-  */
/*guire una divisione e somma, salvarlo in una coda e rinconver-*/
/*tire il numero quando richiesto dalla FPGA                    */
/*--------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <stdarg.h>
#include <signal.h>
#include <math.h>
#include <linux/gpio_syscalls.h>
#include <linux/netlink.h>
#include <time.h>

 //costante per la conversione in tempo dei passi
#define Freq 50000000
/* numero di protocollo che andrà scritto nell'intestazione dei
   pacchetti*/
#define NETLINK_TEST        30
/* dimensione della stringa in byte del payload per la struttura
   nl_payload_t */
#define MAX_PAYLOAD_CHAR    28
// macro per calcolare la dimensione dei dati utili incapsulati
```

```
#define MAX_PAYLOAD_LENGTH  sizeof(struct nl_payload_t)

// definizione dalla struttura  del nostro payload
typedef struct nl_payload_t
{
   int sender_pid;              // Process ID del Mittente
   //se è 0 il messaggio è unicast altrimenti multicast
   int multicast;
   char str[MAX_PAYLOAD_CHAR];  // Stringa dati
} nl_payload_t;

//strutture in cui salvo le vecchie caratteristiche della seriale
struct termios oldstdin;
struct termios oldser;
// descrittore seriale
int fdser;

//descrittore socket
int sockfd;
//struttura per l'indirizzo del destinatario e del mittente
struct sockaddr_nl srcaddr, destaddr;
// header di spedizione e ricezione
struct nlmsghdr *nlh, *nlhrecv;
struct msghdr msg;
struct iovec iov;
//struttura del payload
struct nl_payload_t p;

/* apertura della connessione tramite seriale tty_dev rappresenta
   l'indirizzo della porta passato da l main*/
void openser(char* tty_dev)
{
  //nuova struttura in cui salvo le caratteristiche
  struct termios newser;
  // apertura della seriale
  fdser=open(tty_dev,O_RDWR| O_NONBLOCK);
  //controllo se l'apertura è effettivamente avvenuta
  if (fdser<0)
  {
      fprintf (stderr,"errore apertura seriale %s\n",
             strerror(errno));
      exit(EXIT_FAILURE);
  }
  //controllo sul terminale d'input predefinito
  if (stdin_init()<0) {
      printf("stdin non aperta %s\n", strerror(errno));
      exit(EXIT_FAILURE);
  }
  //vengono ignorati i segnali d'interruzione del processo
  if (signal(SIGINT, closeser) == SIG_IGN) signal (SIGINT, SIG_IGN);
  if (signal(SIGHUP, closeser) == SIG_IGN) signal (SIGHUP, SIG_IGN);
  if (signal(SIGTERM,closeser)== SIG_IGN) signal (SIGTERM, SIG_IGN);
  //setto le caratteristiche della porta seriale
  tcgetattr(fdser,&oldser);
  tcgetattr(fdser,&newser);
  newser.c_cflag |= CREAD;
```

```
  newser.c_cflag |= B9600;
  newser.c_cflag |= CS8;
  newser.c_cflag |= PARENB;
  newser.c_lflag &= ~(ICANON);
  newser.c_lflag &= ~(ECHO);
  newser.c_lflag &= ~(ECHOE);
  newser.c_lflag &= ~(ISIG);
  newser.c_cc[VMIN]=1;
  newser.c_cc[VTIME]=0;
  tcsetattr(fdser, TCSANOW, &newser);
  return 0;
}

// funzione per stampare tramite porta seriale
void printftty(char *format, ...)
{
  va_list argptr; //puntatore alla lista
  char buffer[200];
  va_start(argptr,format); //setta il puntatore all'inizio di format
  //trasferisce su buffer il contenuto di argptr
  vsprintf(buffer,format,argptr);
  va_end(argptr);
  // scrive sulla porta seriale
  write(fdser,buffer,strlen(buffer));
}

//chiudo la connessione della seriale
void closeser (int signum)
{
      tcsetattr(STDIN_FILENO,TCSANOW,&oldstdin);
      // ripristino i valori
      if (fdser>0) tcsetattr (fdser,TCSANOW,&oldser);
      close(fdser);
      printf("Exit\n");
      exit(0);
}

//modifico il terminale d'input
int stdin_init(void)
{
  struct termios tattr;
  //controlla che esista il file descriptor del terminale input
  if (!isatty (STDIN_FILENO))
  {
       fprintf (stderr,"stdin is not a terminal\n");
       return -1;
  }
  /*salvo la vecchia struttura in modo da essere
    ripristinata alla fine*/
  tcgetattr (STDIN_FILENO, &oldstdin);
  tcgetattr (STDIN_FILENO, &tattr);
  tattr.c_lflag &= ~(ICANON | ECHO);
  tattr.c_cc[VMIN] = 0;
  tattr.c_cc[VTIME] = 0;
  tcsetattr (STDIN_FILENO, TCSAFLUSH, &tattr);//setto i nuovi valori
  return 0;
```

```
}

//instaurazione del socket NetLink
/*  una volta costruita la struttura msg (che comprende iov, nhl
    e payload), la struttura verrà usata sempre sia per ricezione
    che per trasmissione, in quanto si vanno a scrivere i campi
    della struttura e non facendo un memcpy*/

void instsocket()
{
   /*apertura del socket di tipo Netlink per la comunicazione con il
     kernel (PF_NETLINK)*/
   sockfd = socket(PF_NETLINK,SOCK_RAW,NETLINK_TEST);
   // setto le caratteristiche del mittente(questo processo)
   srcaddr.nl_family = AF_NETLINK;
   /* ricavo l'identificativo del processo per discriminare tra i
      diversi processi che lo possono utilizzare*/
   srcaddr.nl_pid    = getpid();
   srcaddr.nl_groups = 0;
   /*assegno un indirizzo locale al socket, se l'operazione non
     riesce esce dal programma*/
   if(bind(sockfd,(struct sockaddr*)&srcaddr,sizeof(srcaddr)))
     exit(EXIT_FAILURE);
   /*riservo lo stesso spazio dell'indirizzo del mittente
     in memoria all'indirizzo del destinatario*/
   memset(&destaddr, 0, sizeof(srcaddr));
   //setto la struttura dell'indirizzo del destinatario
   destaddr.nl_family= AF_NETLINK;
   //con pid=0 indico che il messaggio è destinato al kernel
   destaddr.nl_pid   = 0;
   destaddr.nl_groups= 0;
   // setto l'header del messaggio in memoria
   nlh = (struct nlmsghdr*)malloc(NLMSG_SPACE(MAX_PAYLOAD_LENGTH));
   nlh->nlmsg_len = NLMSG_SPACE(MAX_PAYLOAD_LENGTH);
   nlh->nlmsg_pid = getpid();
   nlh->nlmsg_flags = 0;
   //svuoto il buffer della memoria
   memset(&p, 0, sizeof(p));
   /*invio il primo messaggio verso il kernel per poter memorizzare
     il pid*/
   strncpy(p.str, "Kernel", MAX_PAYLOAD_CHAR-1);
   p.sender_pid = getpid();//fisso il pid del sender
   /*trasferisco il payload nel pacchetto netlink*/
   memcpy(NLMSG_DATA(nlh), &p, MAX_PAYLOAD_LENGTH);
   /*configuro la variabile msg che è fondamentale per l'invio nel
     socket*/
   iov.iov_base = (void *)nlh;
   iov.iov_len = nlh->nlmsg_len;
   msg.msg_name = (void *)&destaddr;
   msg.msg_namelen = sizeof(destaddr);
   msg.msg_iov = &iov;
   msg.msg_iovlen = 1;
   //Forza la scrittura di tutti i dati bufferizzati dallo stream
   fflush(stdout);
   //invio il messaggio al socket
   sendmsg(sockfd, &msg, 0);
```

```
}

// int ==> 4 byte

int main()
{
  char rxChar;
  //variabile dei byte dell'errore di quantizzazioni
  char errore[3];
  //variabile dei byte dei secondi dall'inizio dell'anno
  char sec[10];
  int secondi;
  double mant; //mantissa
  double err;  //errore di quantizzazione
  int esp=0;
  int i=0,j;
  // variabile per contare il tempo di un secondo
  clock_t inizio;
  //dato passato dal socket
  unsigned int contaclock;
  float dati[1000]; //coda in cui si memorizza i dati
  float val; //valore da mettere nella coda
  //indici di gestione coda
  int inizio,fine;
  // PB4 di deflault è d'input in questo modo lo fisso di output
  gpiosetdir(PORTA, DIROUT, PA0);
  //apertura della porta seriale
  openser("/dev/ttyS0");
  //instaurazione del socket
  instsocket();
  /*lettura dei secondi dall'inizio dell'anno per il passaggio alla
    FPGA*/
  rxChar='a';
  while((i<10) && (sec[i-1]!='P'))
  {    //lettura da porta seriale
       if (read(fdser,&rxChar,1)>0)
       {
           sec[i]=rxChar;
           i++;
       }
  }
  secondi=0;
  i-=2;
  for (j=0;j<=i;j++)
  { //conversione in valore decimale
    secondi+=(sec[i-j]-48)*(int)pow(10,j);
  }
  /*conversione del valore in numero binario per il passaggio alla
    FPGA, il per il passaggio dei valori è di 12 bit è il valore da
    passare è di 32 bit, siccome dei 32 bit massimo 25 bit vengono
    utilizzati s'inviano 2 blocchi da 10 bit e un blocco da 5.
    Subito metto un bit(OG28) a livello logico alto  per indicare
    alla FPGA che invio il dato, una volta finita l'elaborazione del
    primo blocco metto a livello logico alto un altro bit(OG25/OG29)
    (se leggo il secondo/terzo blocco), che sarà rimesso subito
    basso, per dire alla FPGA  di salvare i valori e poi ripeto
```

```
  questo per tutti tre i blocchi. L'elaborazione consiste nel
  convertire il numero in binario e di passarlo tramite le GPIO
  una volta finito il passaggio, metto a livello basso il bit che
  ho alzato all'inizio*/
i=0;
/*setto a livello logico alto il bit per dire alla fpga che invio
   il dato*/
gpiosetbits(PORTG, PG28);
/*conversione del valore in binario e passaggio alle gpio. Per
  convertirlo in binario uso i bitwise tramite 10 if con
  altrettanti bit del numero per vedere il valore binario
  corrispondente a quel particolare bit, dopo i dieci if shifto il
  numero di 10 posizione per poter in questo modo utilizzare il
  ciclo*/
while(i<3)
{
  (secondi&1)?gpiosetbits(PORTG, PG28):gpioclearbits(PORTG, PG28);
  (secondi&(1<<1))?gpiosetbits(PORTB, PB4)
                  :gpioclearbits(PORTB, PB4);
  (secondi&(1<<2))?gpiosetbits(PORTG, PG2)
                  :gpioclearbits(PORTG, PG2);
  (secondi&(1<<3))?gpiosetbits(PORTB, PB7)
                  :gpioclearbits(PORTB, PB7);
  (secondi&(1<<4))?gpiosetbits(PORTB, PB8)
                  :gpioclearbits(PORTB, PB8);
  (secondi&(1<<5))?gpiosetbits(PORTG, PG5)
                  :gpioclearbits(PORTG, PG5);
  (secondi&(1<<6))?gpiosetbits(PORTG, PG6)
                  :gpioclearbits(PORTG, PG6);
  (secondi&(1<<7))?gpiosetbits(PORTG, PG7)
                  :gpioclearbits(PORTG, PG7);
  (secondi&(1<<8))?gpiosetbits(PORTG, PG30)
                  :gpioclearbits(PORTG, PG30);
  (secondi&(1<<9))?gpiosetbits(PORTG, PG31)
                  :gpioclearbits(PORTG, PG31);
  i=i+1;
  if (i==1)
  {
     gpiosetbits(PORTG, PG25);
     gpioclearbits(PORTG, PG25);
  }
  else if(i==2)
  {
     gpiosetbits(PORTG, PG29);
     gpioclearbits(PORTG, PG29);
  }

  secondi=secondi>>10;

  /*lo riporto a zero qui per aspettare qualche microsecondo per
    essere sicuri della lettura da parte della FPGA*/

}
gpiosetbits(PORTG, PG28); //riporto a zero il valore
printf("press ctrl-c to exit\n"); // per uscire dal ciclo
while(1)
```

```
{
    i=0;
  rxChar='a';
  while((i<10) && (sec[i-1]!='P'))
  {
     if (read(fdser,&rxChar,1)>0)
     {
         sec[i]=rxChar;
         i++;
     }
  }
  // acquisisco l'errore di quantizzazione
  i=0;
  while (i<3)
    {
          if (read(fdser,&rxChar,1)>0)
          {
            //leggo i byte dell'errore di quantizzazione
            errore[i]=rxChar;
            i++;
          }
    }
 /*in questo punto sono sicuro che il GPS ha inviato altre
   informazioni alla PIC ciò vuol dire che è arrivato un altro PPS
   e quindi posso cancellare gli elementi della coda presente, per
   cancellare azzero gli indici.  ATTENZIONE: partendo da zero
   spreco una posizione ma risparmio dei controlli, che si
   dovrebbero fare usando un numero maggiore di 1000 che indica
   coda vuota*/
    inizio=0;
  fine=0;
  /*l'errore di quantizzazione è scritto secondo lo standard
    IEEE745, 1 bit di segno, 8 di esponente e 24 di mantissa (noi
    ne usiamo solo 16 in quanto ci permettono di raggiungere la
    precisione desiderata). La mantissa è scritta in forma
    normalizzata cioè 1,011.... e per questo mant è inizializzata
    a 1, si è deciso d'inviare l'errore di quantizzazione come
    caratteri in modo da raggruppare gli 8 bit in un unico
    valore*/
    mant=1;
    esp=0;
    //calcolo dell'esponente analizzando il primo byte
    for (i=1;i<8;i++)
          esp+=((errore[0]&(1<<(i-1)))?1:0)*pow(2,i);
  /*aggiungo all'esponente il suo LSB presente nel bit meno
    Significativo*/
    esp+=(errore[1]&(1<<7))?1:0;
    //calcolo il della mantissa
    for (i=1;i<8;i++)
          mant+=((errore[1]&(1<<(7-i)))?1:0)*pow(2,-i);
    for (i=8;i<16;i++)
          mant+=((errore[2]&(1<<(15-i)))?1:0)*pow(2,-i);
  //calcolo l'errore di quantizzazione
    err=((errore[0]<0)?-1:1)*pow(2,esp-127)*mant;
    /*eseguo un ciclo in maniera che rimango in ascolto del socket
      per un secondo*/
```

```
inizio = clock();
while (clock-inizio<100)
{
/*adesso ricevo i messaggi dal netlink esso invia 28 byte cioè
  7 dati alla volta al quale aggiungo l'errore di
  quantizzazione e lo salvo*/
//ricezione del messaggio
recvmsg(sockfd, &msg, 0);
// copio lil payload dalla memoria alla struttura locale p
memcpy(&p, NLMSG_DATA(nlh), MAX_PAYLOAD_LENGTH);
for(i=0;i<7;i++)
{
    /*estraggo le informazioni dalla mia stringa, sono sicuro
      che il primo bit è il MSB del nostro dato, per ricavare
      il numero decimale sapendo che i bit sono ordinati
      shifto a blocchi di 8 in quanto C, vede il caratteri
      anche come numeri da 0 a 255*/
     contaclock=(p.str[0+4*i]<<24)+(p.str[1+4*i]<<16);
     conta clock+=(p.str[2+4*i]<<8)+(p.str[3+4*i]);
    //calcolo del dato necessario per memorizzarlo in memoria
    val=(float)(contaclock/freq) + (float)err;
    dati[fine]=val;
    fine++;
    //se la coda è vuota inizializzo gli indici
    if (inizio==0) inizio=0;
    //gestione indici
    //la coda è piena riparto dall'inizio
    if ((fine==1001))  fine=1;
    if ((fine==inizio)|| gpiogetbits(PORTG,PG3))==1 )
    {
        if (fine==inizio) fine=1000;
        while(inizio<fine)
        {
          //invio i valori alla FPGA
          i=0;
          /*invio il dato diviso in quattro blocchi e
            convertendolo in binario come fatto in
            precedenza*/
          while(i<4)
          {
           (dati[inizio]&1)?gpiosetbits(PORTG, PG28)
                           :gpioclearbits(PORTG, PG28);
           (dati[inizio]&(1<<1))?gpiosetbits(PORTB, PB4)
                           :gpioclearbits(PORTB, PB4);
           (dati[inizio]&(1<<2))?gpiosetbits(PORTG, PG2)
                           :gpioclearbits(PORTG, PG2);
            (dati[inizio]&(1<<3))?gpiosetbits(PORTB, PB7)
                           :gpioclearbits(PORTB, PB7);
            (dati[inizio]&1<<4))?gpiosetbits(PORTB, PB8)
                           :gpioclearbits(PORTB, PB8);
            (dati[inizio]&1<<5))?gpiosetbits(PORTG, PG5)
                           :gpioclearbits(PORTG, PG5);
            (dati[inizio]&1<<6))?gpiosetbits(PORTG, PG6)
                           :gpioclearbits(PORTG, PG6);
            (dati[inizio]&1<<7))?gpiosetbits(PORTG, PG7)
                           :gpioclearbits(PORTG, PG7);
```

```
                    (dati[inizio]&1<<8))?gpiosetbits(PORTG, PG30)
                                        :gpioclearbits(PORTG, PG30);
                    (dati[inizio]&1<<9))?gpiosetbits(PORTG, PG31)
                                        :gpioclearbits(PORTG, PG31);
                    gpiosetbits(PORTG, PG25);
                    dati[inizio]&= dati[inizio]&>>10;
                    i=i+1;
                    gpioclearbits(PORTG, PG25);
                  }
                  //leggo il successivo dato nella coda
                  inizio++;
                }
                if (fine==inizio)
                {
                   inizio=0;
                   fine=0;
                }
              }
        }
     }
  return EXIT_SUCCESS;
}
```

APPENDICE F

Codice della FPGA

Segue il codice riguardante il conta clock

```
----------------------------------------------------------------
--questo programma calcola i passi di clock trascorsi da quando
--abbiamo ricevuto l'ultimo pps, all'arrivo dell'evento, salva
--i passi di clock attuali e li corregge con l'errore commesso
--nel secondo precedente in questo modo si riesce a normalizza-
--re, in quanto in un secondo non siamo sicuri che il contatore
--esegua esattamente 50e6 passi, una volta quest'operazione il
--dato viene spezzato in due parti (il bus dati verso il mondo
--esterno  di 20 bit). I dati segmentanti verrano immagazzinati
--in array di dimensione opportunamente calcolate( le dimensioni
--sono calcolate in base al tempo la fox  impiegata a fare le
--operazioni necessarie alla correzione del dato), l'array verra'
--svuotato una volta che la fox invia il segnale che e' pronta a
--ricevere i dati
--Per entrare in regime mi servano 3 PPS:
-- 1 PPS: mi ricavo il valore dei conteggi effettuati e se l'errore
--        commesso devo essere aggiunto o sottratto e l'errore di
--        passi(errore) che ho compiuto, il problema che
--        quest'errore deve essere proporzionato al numero di cicli
--        di clock presenti rispetto 50e6, per far questo dovrei
--        fare una divisione, questo non possibile, per risolvere il
--        problema aspetto il 2 PPS
-- 2 PPS: sapendo il numero di passi che ho sbagliato in precedenza
--        e supponendo che questi non variano tra due istanti
--        successivi, vado a trovarmi la parte piu' piccola del
--        dell'intervallo (50e6) su cui commetto l'errore di +/-1
--        questa parte  chiamata segmento
-- 3 PSS: ogni volta che il mio conteggio  multiplo di segmento
--        vado ad effettuare la correzione del dato
----------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_misc.all;
use ieee.std_logic_unsigned.all;


entity conclo is
    port (
        clk   : in std_logic;
        PPS   : in std_logic;
```

```
        istance : in std_logic; --evento esterno per salvare i passi
        reset : in std_logic;
        prot : in  std_logic_vector(4 downto 0);
        dato : out std_logic_vector(15 downto 0)
    );

end conclo;

architecture contatore of conclo is

--uscite di sei flip-flop per ritardare il PPS e avere in questo
-- modo anche l'isstante precedente per salvare i dati necessari per
-- il calcolo dell'errore
signal Q1, Q2, Q3, Q4, Q5, Q6, PPS0, PPS1 : std_logic;
--l'evento esterno non ha una durata fissa, con un filtro
-- antirimbalzo lo rendo un impulso di durata fissa
signal istance0, Y1,Y2,Y3 : std_logic;
signal d1,d2,d3,dato1 : std_logic_vector(25 DOWNTO 0);
signal errore,ERROR1,ERROR2,error0 : std_logic_vector(25 DOWNTO 0);
signal errinc,errinc0,errinc1 : std_logic_vector(25 DOWNTO 0);
signal step : std_logic_vector(25 DOWNTO 0);
signal count,segm,segmento : std_logic_vector(25 DOWNTO 0);
signal piuno,menouno,piuno1,menouno1 :std_logic_vector(25 DOWNTO 0);

--costante per fare incremento e decremento
constant  uno : std_logic_vector(25 DOWNTO 0) :=
                                        "00000000000000000000000001";
--costante per calcolare l'errore
constant  m50 : std_logic_vector(25 DOWNTO 0) :=
                                        "10111110101111000010000000";

begin

-- ritarda la ricezione del pps e lo rende pi stabile
process(clk)
  begin
  if (clk'event and clk = '1') then
    Q1 <= PPS;
    Q2 <= Q1;
    Q3 <= Q2;
    Q4 <= Q3;
    Q5 <= Q4;
    Q6 <= Q5;
  end if;
end process;

--segnale PSS ricavato all'istante precedente utilizzato per
-- memorizzare i valori necessari alla correzione
PPS1 <= Q1 and Q2 and (not Q3);
--segnale di pps usato per il reset di tutti i processi che indica
-- che arrivato il nuovo segnale PPS
PPS0 <= Q3 and Q4 and (not Q5);

--stabilizza il segnale dell'evento
process(clk)
  begin
```

```
  if (clk'event and clk = '1') then
    Y1 <= istance;
    Y2 <= Y1;
    Y3 <= Y2;
  end if;
end process;
-- segnale evento stabile
istance0<= Y1 and Y2 and (not Y3);

--contatore dei cicli di clock fra due PPS
process (clk, pps0,reset)
  begin
  if (PPS0 = '1' or reset = '1') then
    d1 <= (others => '0');
    d2 <= (others => '0');
  elsif (clk'event and clk = '1') then
    d1 <= d3 + uno;
    d2 <= d1;
   end if;
end process;

--mi esegue la corezzione ogni segmento, il segmento non altro che
-- il valore di cui supero l'errore preedente
process (clk, pps0,reset)
  begin
  if (PPS0 = '1' or reset = '1')then
    step <= (others => '0');
  elsif (clk'event and clk = '1') then
     step <= step + uno;
     d3 <= d2;
    if (step = segmento) then
     step<=(others => '0');
     d3 <= d2 + piuno - menouno;
     --d1 <= d3;
    end if;
  end if;
end process;

--contatore all'indietro per vedere le parti che mancono
process (clk, pps0, reset)
   begin
   if (PPS0 = '1' or reset = '1') then
      error0 <= m50;
      error1 <= (others => '0');
   elsif (clk'event and clk = '1') then
     if (error0 /= "0") then
       error0 <= error0-uno;
       error1 <= error0;
       piuno1 <= uno;
       menouno1 <= (others => '0');
     else --l'errore puo' essere maggiore
       errinc0 <= errinc0 + uno;
       errinc1 <= errinc0;
       piuno1 <= (others => '0');
       menouno1 <= uno;
     end if ;
```

```
   end if;
end process;

--salvataggio dell'errore prima che arriva il PPS
process (PPS1)
   begin
   if (pps1'event and pps1 = '1') then
     errore <= error1 + errinc1;
     piuno<=piuno1;
     menouno<=menouno1;
   end if;
end process;

--conto i numeri di errori comessi sul dato attuale
process(clk, PPS0, reset)
  begin
  if (PPS0 = '1' or reset = '1') then
      count <= (others => '0');
      segm <=(others => '0');
  elsif (clk'event and clk = '1') then
    count <= count + uno;
    --controllo l'errore che sto commettendo sull'attuale conteggio
    --rispetto al precedente
    if(count > errore) then
       count<=(others => '0');
       segm<= segm + uno;
    end if;
  end if;
end process;

--si salva la lunghezza del segmento perpoi utilizzarla per
--correggere il dato
process (PPS1)
   begin
   if (PPS1'event and pps1 = '1') then
            segmento <= segm; --salvo segmento
   end if;
end process;

--salvataggio del dato dei cicli di clock al momento dell'evento
process(istance0)
   begin
   if (istance0'event and istance0 = '1') then
            dato1 <= d3;
   end if;
end process;

--invio i dati alla FOX BOARD
process(prot)
   begin
   if(prot(0) = '1' and (prot(0) = '0')) then
            dato<=dato1(15 downto 0);
   end if;

   if((prot(1) = '1') and (prot(0) = '1')) then
            dato<="000000"&dato1(25 downto 16);
```

```
   end if;
end process;

end contatore;
```

Segue il codice riguardante il conta PPS:

```
----------------------------------------------------------
--il seguente programma  un conteggio di PPS,con offeset
--inziale pari ai secondi trascorsi dall'inizio dell'anno
----------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_misc.all;
use ieee.std_logic_unsigned.all;


entity conpps is
    port (
        PPS   : in std_logic;
        reset : in std_logic;
        prot : in  std_logic_vector(3 downto 0);
        dato : in std_logic_vector(9 downto 0)
    );

end conpps;

architecture contatore of conpps is

signal dato1,dato2 : std_logic_vector(25 DOWNTO 0);
--costante per fare incremento e decremento
constant  uno : std_logic_vector(25 DOWNTO 0) :=
                                        "00000000000000000000000001";

begin

--contatore dei pps
process (pps,reset)
  begin
  if (reset='1') then
    dato1 <= (others => '0');
    dato2 <= (others => '0');
  elsif (pps'event and pps = '1') then
    dato2 <= dato2 + uno;
  end if;
end process;


--ricevo ida ti dalla fox board
process(prot,dato)
   begin
   if((prot(0) = '1')and (prot(1) = '0') and prot(2)='0') then
      dato1(9 downto 0)<=dato(9 downto 0);
   elsif((prot(1) = '1') and (prot(0) = '1') and prot(2)='0') then
```

```
      dato1(19 downto 10)<=dato(9 downto 0);
   elsif((prot(1) = '0') and (prot(0) = '1') and prot(2)='1') then
      dato1(25 downto 20)<=dato(5 downto 0);
   end if;
end process;

end contatore;
```

BIBLIOGRAFIA

[1]. Il progetto EEE, http://www.centrofermi.it/eee/.

[2]. Jean Marie Zogg - “GPS Basic Intrduction to the system Application overiew”

[3]. Artech House Telecommunications Library – “Understanding GPS Principles and Applications” -Elliott D. Kaplan, Christopher Hegarty

[4]. O'REILLY – “Linux Device Driver” - Jonathan Corbet, Alessandro Rubini and Greg Kroah-Hartman -

[5]. O'REILLY - “Understanding The Kernel Linux” - Daniel P. Bovet and Marco Cesati

[6]. Newnes – “Embedded Systems Design” -Health Steave

[7]. Acmesystem http://www.acmesystems.it/

[8]. AXIS ETRAX 100LX Designer’s Reference - http://www.axis.com/products/dev_etrax_100lx/

[9]. FoxDoc.pdf - di Calzoni Pietro scaricabile da scaricabile dal sito http://www.lugman.org

[10]. FOX Board BOOT- di Calzoni Pietro scaricabile da scaricabile dal sito www.lugman.org

[11]. NetLink.df - di Calzoni Pietro scaricabile da scaricabile dal sito http://www.lugman.org

[12]. Micro Engineering Labs - “PicBasic Pro Compiler”

RINGRAZIAMENTI

Ringrazio il Prof. Marco Panareo, per avermi permesso di svolgere la tesi presso l' Istituto Nazionale di Fisica Nucleare, per la costante disponibilità, per il compito di relatore svolto con grande impegno.

Ringrazio il gruppo di lavoro del Laboratorio di Elettronica dell' INFN, in particolare il Dott. Alessandro Corvaglia, che mi ha " sopportato" durante tutto il periodo dello svolgimento della tesi consigliandomi e aiutandomi a superare le difficoltà incontrate per realizzare questo lavoro; il Dott. Pietro Creti per avermi introdotto nel difficile mondo della programmazione VHDL; il Dott. Carlo Pinto per avermi aiutato nel montaggio della scheda e a Roberto Assiro per essersi reso disponibile ad aiutarmi in qualsiasi momento, e Fulvio Ricciardi per avermi aiutato nel capire il mondo del software libero.

Un particolare ringraziamento a mia madre Maria Lucia, mio padre Pietro, mia sorella Annachiara, nonna Mina, nonno Biagio, zio Enzo, zia Anna e Nonna Lucia che mi hanno sostenuto, incoraggiato e aiutato nei momenti di difficoltà affrontando sacrifici senza i quali non avrei raggiunto questo traguardo e per i quali sarò sempre grato. Agli amici Mariella e Angelo Neve sempre presenti al momento del bisogno con i loro saggi consigli.

Un ultimo grazie a tutti gli amici (Alberto, Luvi, Giò, Matteo, Beppe, Giuseppe, Paolo, Fra, Francesco, Sara, Alberto, Salvatore, Andrea, Valeria, Cri, Giancarlo, Cosimo) che mi sono stati vicini in questi anni e con cui ho passato esperienze bellissime .