ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA ED ARCHITETTURA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA,
INFORMATICA E TELECOMUNICAZIONI

Ottimizzazione di protocolli di trasmissione dati tramite interfaccia USB implementati su FPGA

Elaborato in
Elettronica dei Sistemi Digitali

Relatore
Prof.Ing. Aldo Romani

Presentata da
Giacomo Angiolini

Correlatore
Ing. Federico Thei

Sessione III
Anno Accademico 2012-2013

Alla mia famiglia
Ai coraggiosi che mi stanno accanto
A chi oggi dovrebbe essere ancora qui, D.

Indice

Introduzione

La maggior parte dei moderni dispositivi e macchinari, sia ad uso civile che industriale, utilizzano sistemi elettronici che ne supervisionano e ne controllano il funzionamento.

All' interno di questi apparati è quasi certamente impiegato un sistema di controllo digitale che svolge, anche grazie alle potenzialità oggi raggiunte, compiti che fino a non troppi anni or sono erano dominio dell' elettronica analogica, si pensi ad esempio ai DSP (Digital Signal Processor) oggi impiegati nei sistemi di telecomunicazione.

Nonostante l' elevata potenza di calcolo raggiunta dagli odierni microprocessori/microcontrollori/DSP dedicati alle applicazioni embedded, quando è necessario eseguire elaborazioni complesse, time-critical, dovendo razionalizzare e ottimizzare le risorse a disposizione, come ad esempio spazio consumo e costi, la scelta ricade inevitabilmente sui dispositivi FPGA.

I dispositivi FPGA, acronimo di Field Programmable Gate Array, sono circuiti integrati a larga scala d'integrazione (VLSI, Very Large Scale of Integration) che possono essere configurati via software dopo la produzione.
Si differenziano dai microprocessori poiché essi non eseguono un software, scritto ad esempio in linguaggio assembly oppure in linguaggio C.
Sono invece dotati di risorse hardware generiche e configurabili (denominate Configurable Logic Block oppure Logic Array Block, a seconda del produttore del dispositivo) che per mezzo di un opportuno linguaggio, detto di descrizione hardware (HDL, Hardware Description Language) vengono interconnesse in modo da costituire circuiti logici digitali.
In questo modo, è possibile far assumere a questi dispositivi funzionalità logiche qualsiasi, non previste in origine dal progettista del circuito integrato ma realizzabili grazie alle strutture programmabili in esso presenti.

Si possono quindi implementare funzionalità che vanno dalle semplici reti logiche combinatorie e sequenziali (ad esempio la cosiddetta glue logic per l' interconnessione tra diversi dispositivi integrati digitali all' interno di un sistema più complesso) fino alla realizzazione di microprocessori custom-designed e funzionalità di elaborazione digitale dei segnali analogici ad altissime prestazioni.

Molto spesso i dispositivi FPGA vengono utilizzati nella fase di progettazione di un circuito integrato custom: una volta definite le funzionalità e verificato il funzionamento , il progettista può decidere di rendere definitivo il proprio progetto ed affidarsi alle aziende specializzate (silicon foundries) per realizzare una cosiddetta ASIC (Application Specific Integrated Circuit, circuito integrato per applicazioni specifiche) a partire dal progetto realizzato mediante linguaggio di descrizione hardware.

Per quanto riguarda questo tipo di linguaggi, sono due quelli principalmente utilizzati: VHDL e Verilog.

Il linguaggio VHDL (Very High Speed Integrated Circuit Hardware Description Language, linguaggio di descrizione hardware per circuiti integrati ad altissime velocità), utilizzato nel progetto descritto in questa tesi, ha avuto origine nell' ambito del Dipartimento della Difesa USA negli anni ottanta e fu concepito con lo scopo di uniformare la descrizione del funzionamento dei dispositivi ASIC che le varie aziende fornitrici dei militari americani utilizzavano nelle loro apparecchiature.

Successivamente furono sviluppati pacchetti software in grado di interpretare il linguaggio per poter effettuare simulazioni circuitali ed eseguire la cosiddetta sintesi logica, cioè il passaggio da una descrizione formale e testuale all' implementazione fisica del circuito integrato.

Il linguaggio è poi divenuto uno standard IEEE.

Verilog ha avuto origine, nello stesso periodo, in ambito commerciale e si prefigge gli stessi scopi. Si differenzia principalmente da VHDL per la sintassi ed la tipologia di linguaggio (soprattutto per ciò che riguarda la gestione dei diversi tipi di dato/segnale) e per la minor verbosità.

Anche Verilog è divenuto standard IEEE.

In ambito industriale vengono utilizzati entrambi i linguaggi, tanto da essere considerati equivalenti.

Il flusso di progetto per la realizzazione di dispositivi utilizzando FPGA è illustrato nello schema di figura 1: una volta definiti i requisiti e pianificata una architettura di elaborazione per assolvere i compiti prefissati (Design Verification), si passa alla stesura del progetto mediante linguaggio di descrizione hardware (Design Entry).

Terminata la stesura del codice, viene eseguita la sintesi logica (Design Syntesis) mediante la quale la descrizione formale del circuito, ovverosia il codice HDL, viene trasformata in netlist, cioè vengono realizzate le interconnessioni necessarie fra le risorse dell' FPGA, creando così il circuito.

Al fine di verificare la correttezza della logica che si sta implementando è necessario eseguire una simulazione comportamentale (Behavioral Simulation), nella quale si tiene conto solamente del comportamento teorico ed ideale dei componenti utilizzati (ad esempio il tempo di propagazione dei segnali fra i gate viene considerato nullo).

Una volta verificato il funzionamento teorico dell' architettura viene eseguita l' implementazione vera e propria del circuito per l' FPGA: viene eseguito un processo denominato "Place and Route" che partendo dalla analisi della netlist determina quali fra le risorse messe a disposizione dal dispositivo debbano essere utilizzate, prediligendo quelle che permettono di realizzare l' architettura minimizzando le interconnessioni (e soprattutto le loro lunghezze).

Successivamente vengono eseguite le simulazioni circuitali che questa volta tengono conto delle non idealità dei componenti (propagation delays, register setup and hold times,…), al fine di scongiurare la presenza di alee, che comprometterebbero il funzionamento del dispositivo. Eseguito anche questo passo di verifica, viene generato un file binario (comparabile al codice eseguibile generato da un compilatore nel caso del software) che verrà poi scaricato nella memoria dell' FPGA mediante un dispositivo hardware di programmazione e che ne determinerà il comportamento, cioè le funzionalità.

A questo punto vengono eseguite le verifiche di funzionamento nel circuito elettronico fisicamente realizzato (In-Circuit Verification).

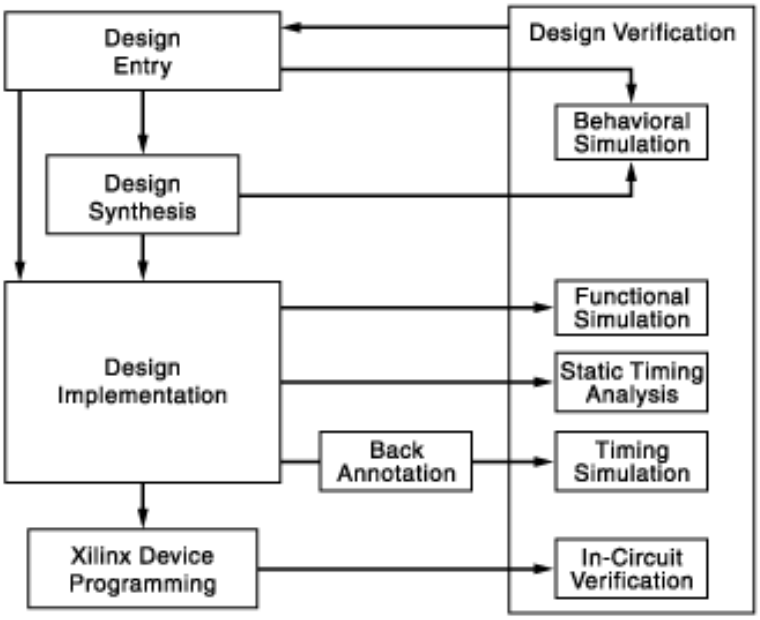


Figura 1: flusso di progetto per FPGA

Capitolo 1

Sistema analizzato

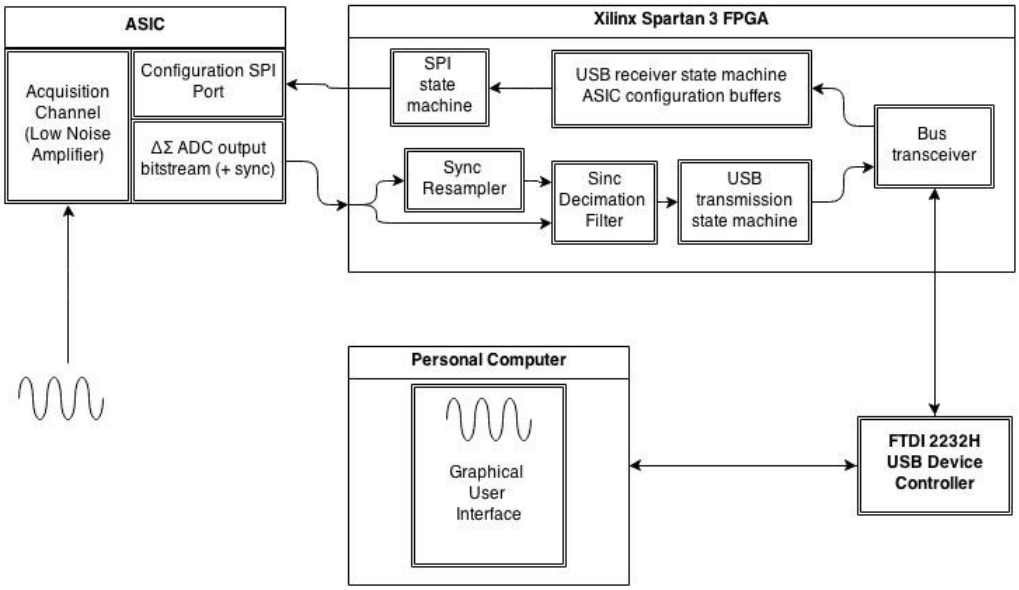


Figura 2: Schema a blocchi del sistema

Il mio lavoro di tesi ha riguardato l' analisi, dedicata all' ottimizzazione, di un dispositivo hardware il cui componente principale è costituito da un circuito integrato custom di tipo mixed signal, realizzato presso i laboratori dell' università, interfacciato con una FPGA ed un chip dedicato alla trasmissione dati mediante bus USB.

Questo dispositivo realizza, nelle dimensioni di poco maggiori rispetto ad una chiavetta usb, un sistema di acquisizione dati (nello specifico correnti nell' ordine dei picoAmpere) mediante il circuito integrato custom (da qui in poi verrà indicato genericamente come "ASIC") nel quale sono stati implementati il front-end a basso rumore per l'amplificazione del segnale in corrente, specifici blocchi di riduzione del rumore elettronico (progettati mediante tecnica Correlated Double Sampling), l' A/D converter delta-sigma e l' interfaccia di comunicazione SPI, utilizzata per la programmazione delle diverse modalità di funzionamento del circuito integrato.

Al fine di elaborare il bitstream (campioni delta-sigma), proveniente dal convertitore analogico-digitale dell' ASIC, ed inviare poi i dati al software di visualizzazione ed analisi su PC, al chip custom sono stati affiancati una FPGA Xilinx 3E ed un integrato FTDI FT2232D.

L' elaborazione eseguita dal dispositivo FPGA consiste in buona sostanza nel filtraggio dei campioni di segnale acquisiti dall' ASIC mediante un filtro decimatore di tipo $Sinc^3$ ed il successivo invio dei dati filtrati verso il PC per la visualizzazione.

La seconda funzionalità implementata nell' FPGA riguarda la ricezione dal PC delle impostazioni di configurazione per il chip e per i suoi sottosistemi costituiti da filtri, DAC ed altri circuiti, in base ai parametri (banda di acquisizione, range di correnti, stimolo in tensione da generare, ecc…) impostati nell' interfaccia grafica.

Quest' ultima invia al dispositivo un pacchetto dati di configurazione, che viene opportunamente elaborato e comunicato al chip attraverso la porta di configurazione SPI.

L' integrato FT2232D viene utilizzato in congiunzione all' FPGA per eseguire il trasferimento dati da e per il dispositivo, interfacciandosi con l' FPGA attraverso un bus parallelo ad 8 bit con opportune linee di controllo per l' accesso in lettura e scrittura, mentre si interfaccia verso il PC mediante un bus USB a specifica 2.0 Full-Speed (12 Mbyte/s massimi) .

A livello di sistema operativo, il dispositivo è riconosciuto come porta seriale virtuale, semplificando così la gestione da parte del software.

Utilizzando questo circuito integrato si può quindi evitare di dover implementare tutto il protocollo USB all' interno dell' FPGA, risparmiando preziose risorse.

Analizzando più in dettaglio la struttura del sistema nel suo complesso e l' architettura di elaborazione implementata all' interno dell' FPGA, si possono individuare due sezioni, la prima delle quali riguarda l' acquisizione dei dati: l' ASIC si interfaccia con il mondo esterno mediante tre canali, il primo dei quali, denominato "Acquisition Channel" è il canale di ingresso per il segnale da acquisire.

Di questo canale si occupa la circuiteria interna dell' ASIC, attraverso il sensore di corrente e l' ADC sigma delta, pertanto non è troppo interessante rispetto i compiti dell' FPGA.

Ciò che è importante è il bitstream in uscita dall' ADC, veicolato sulla linea OUT, che rappresenta il segnale digitalizzato: tale bitstream è ciò su cui andrà effettuata l' elaborazione da parte dell' FPGA.

Accanto al segnale che trasporta tali dati, l' ASIC fornisce anche un segnale di clock, denominato SYNC, utilizzato per la sincronizzazione del ricevitore posto sull' FPGA: in questo caso il dato valido è ottenibile campionando la linea OUT nell' istante in cui il fronte di clock scende.

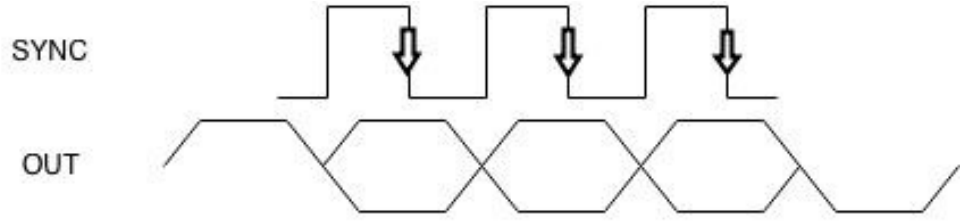


Figura 3: forme d' onda SYNC e OUT

All' interno dell' FPGA è predisposta una macchina a stati che si occupa di ricevere il dato dal chip ed effettuare il filtraggio $Sinc^3$, producendo in uscita un dato a 16 bit che verrà poi trasmesso via USB al computer, mediante un' altra macchina a stati.

La seconda sezione riguarda la configurazione del chip ASIC: tale circuito integrato presenta infatti una porta SPI in ingresso, mediante la quale è possibile scrivere un registro di configurazione a 32 bit, attraverso cui è possibile impostare vari parametri del dispositivo, come il range di acquisizione, la banda di filtraggio (oversampling ratio) ed attivare varie funzionalità, come ad esempio riferimenti di tensione e segnali di test.

L' FPGA si interfaccia con questa porta attraverso una macchina a stati, che riceve i dati riguardanti la configurazione della ASIC impostati sul PC mediante interfaccia grafica, prelevandoli dal chip FTDI (e quindi dal bus USB).

Questi dati vengono memorizzati internamente all' FPGA in registri dedicati allo scopo, dai quali poi vengono prelevati i singoli bit di configurazione, necessari a

costituire la word di configurazione a 32 bit per la ASIC, che viene infine inviata attraverso la porta SPI.

Questa macchina a stati, inoltre, utilizza la linea Slave Select per indicare se effettivamente sta trasmettendo dei dati che l' ASIC deve interpretare: ciò è dovuto alla necessità di mantenere perennemente attiva la linea di clock del bus SPI, dato che tale segnale viene utilizzato dall' ASIC come sorgente di clock.

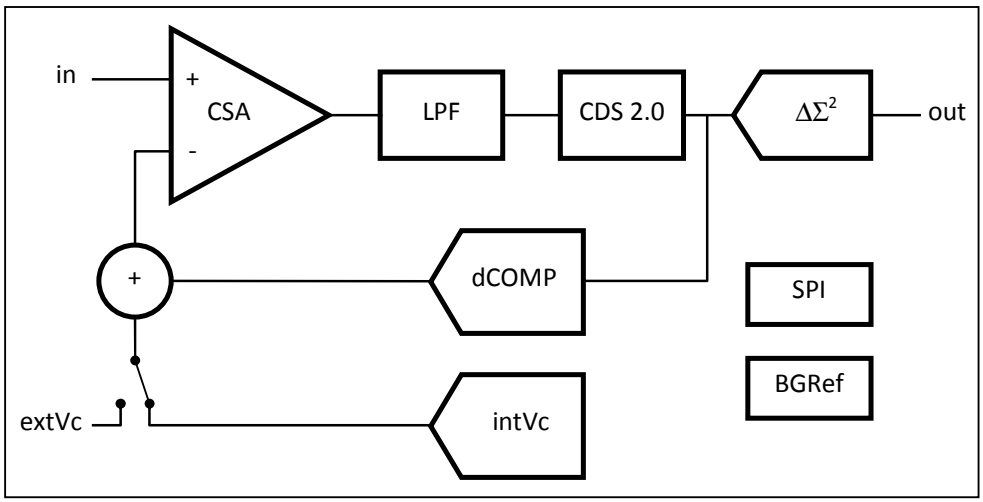


Figura 4: lo schema logico della ASIC

Nella precedente figura è schematizzata l' architettura della ASIC: al suo interno vi sono un amplificatore a basso rumore (LNA) seguito da un filtro passa basso ed un blocco di elaborazione Correlated Double Sampling per la rimozione dell' offset, il tutto retroazionato per garantirne la stabilità.

E' possibile, inoltre, inserire sul secondo ingresso dell' amplificatore una tensione (Vc) interna o esterna al chip, per poter eseguire misurazioni o test, oppure per effettuare una compensazione. L' uscita del blocco retroazionato è infine connessa ad un convertitore analogico digitale di tipo delta-sigma del secondo ordine, che garantisce un ottima linearità della conversione.

Capitolo 2

Definizione degli obiettivi

Il lavoro di ottimizzazione da me svolto ha avuto come obiettivi

- Il miglioramento della modalità di trasferimento dei dati elaborati verso il PC, al fine di ottenere il massimo throughput possibile sul bus USB
- L' adeguamento delle reti logiche preesistenti al fine di poter incrementare frequenza di clock, garantendo la funzionalità del sistema, al fine di estendere la banda di filtraggio del dispositivo fino a 100 kHz

Prima dell' implementazione delle modifiche, il sistema operava con una frequenza di clock pari a 10 MHz, che gli permetteva di campionare, acquisire ed elaborare (filtrare) segnali digitalizzati con banda sino a 10 kHz. L' incremento della frequenza di clock è stato reso necessario per concedere più cicli alla logica sequenziale (macchine a stati finiti implementate nel dispositivo) in modo che potesse elaborare il bitstream a bitrate maggiore proveniente dall' ASIC alimentata dal clock a 80 MHz, estendendo in questo modo la banda del dispositivo fino a 100 kHz. L' architettura di elaborazione originaria implementata nell' FPGA verrà descritta nei prossimi paragrafi.

La necessità di poter effettuare il filtraggio di segnali con una banda fino a 100 kHz ha imposto l' innalzamento della frequenza di lavoro del sistema, per poter incrementare il sampling rate del convertitore analogico digitale interno alla ASIC, ciò ha comportato come diretta conseguenza l' incremento della frequenza di invio dei campioni di segnale digitalizzati verso l' FPGA: difatti si è passati da un segnale di sincronismo (SYNC) pari a 1.25MHz, per un clock di sistema di 10 MHz, ad un sincronismo di 10 MHz, per un clock di 80 MHz.
In pratica, questo vuol dire che era necessario elaborare una strategia per poter

effettuare il filtraggio e l' invio dei dati ad una velocità 8 volte maggiore rispetto al sistema di partenza.

Da ciò sono derivate numerose problematiche, il cui studio e successiva soluzione sono oggetto di questa tesi.

Capitolo 3

Sistema di sviluppo

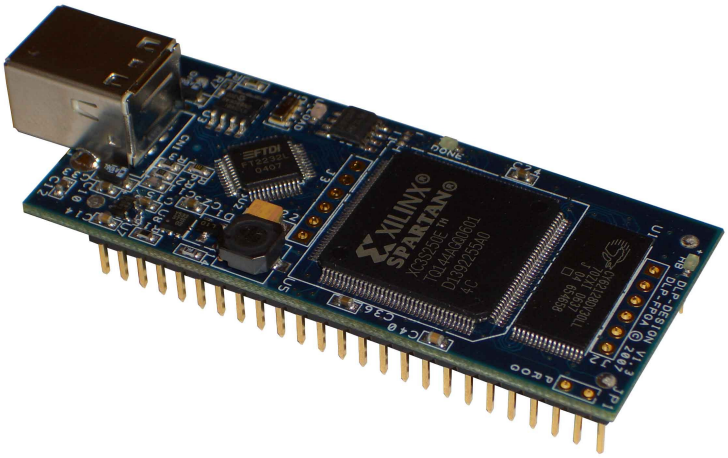
Il codice per il progetto è stato realizzato a partire da una base iniziale fornitami dal docente, sulla quale è stato svolto un approfondito lavoro di analisi e documentazione (passo necessario per comprenderne appieno il funzionamento per poi poter studiare ed applicare le modifiche necessarie), successivamente estesa secondo le specifiche richieste.

L' hardware, utilizzato a supporto dello sviluppo, era costituito da una development board DLP-FPGA, dotata della stesso hardware della scheda target (FPGA Xilinx XC3S250E e chip USB FT2232D), a meno del chip ASIC che è stato simulato internamente all' FPGA da un apposito blocco, programmabile dall' interfaccia utente su PC e grazie al quale è stato possibile verificare le funzionalità di quanto veniva man mano implementato.
L' ambiente di sviluppo di cui si è usufruito è lo Xilinx ISE Webpack 15, mediante il quale è stato sviluppato il codice VHDL utilizzato per la sintesi logica del dispositivo.
Questo pacchetto software è stato inoltre impiegato per il debugging attraverso il simulatore integrato che, mediante la scrittura di semplici test bench, ha permesso di determinare il funzionamento del codice, potendo verificare ogni singolo segnale interno al dispositivo simulato.

Di notevole aiuto allo sviluppo sono stati i documenti della ditta Xilinx, soprattutto per ciò che riguarda il corretto trattamento dei segnali ad alta velocità e l' utilizzo delle risorse interne al dispositivo FPGA, primo fra tutti il buffer FIFO, il cui impiego verrà descritto più avanti.

Il porting verso la piattaforma target è stato effettuato al termine dei test sulla scheda di sviluppo, rimappando i pin internamente connessi al blocco simulativo verso il

pin esterni dell' FPGA cui sono connesse le interfacce della ASIC, mantenendo inalterato il resto del codice.



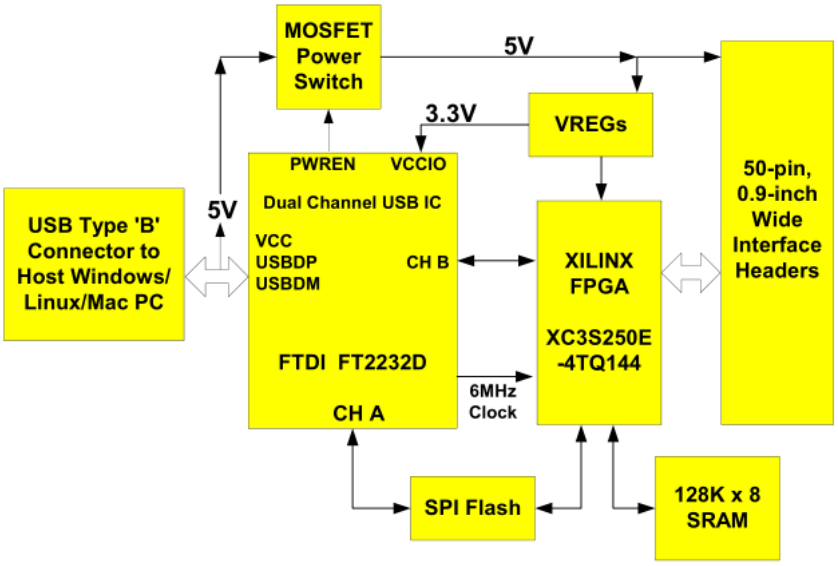


Figura 5: Immagine e schema a blocchi della scheda di sviluppo DLP-FPGA

Capitolo 4

Descrizione architettura di elaborazione iniziale

In questo capitolo verranno descritti in modo più approfondito gli elementi costituenti l' architettura di elaborazione implementata all' interno del dispositivo FPGA, nello stato iniziale prima delle modifiche apportate, con maggiore attenzione rispetto le aree in cui si è intervenuti.

Blocco di gestione del clock

Analizzando lo schema a blocchi della scheda DLP-FPGA si vede come la sorgente di clock dell' integrato Xilinx provenga dall' FTDI: a quest' ultimo è infatti connesso un risuonatore ceramico a 6MHz, che viene utilizzato da tale chip per il suo funzionamento. Il segnale di clock, utilizzato internamente ad esso, è anche presente sui pin esterni. Grazie a questa feature, non si è reso necessario (neanche sul dispositivo target) l' utilizzo di sorgenti di clock separate per l' FTDI e per l' FPGA.

Il segnale di clock a 6 MHz non viene però utilizzato direttamente all' interno dell' FPGA, ma è invece posto in ingresso ad un blocco hardware interno al circuito integrato, denominato DCM, ossia Digital Clock Manager.
Questo blocco si occupa di generare tutti i segnali di clock utilizzati nel sistema, mediante la tecnica Digital Frequency Synthesizing (la quale permette diverse combinazioni di divisione e moltiplicazione della frequenza del segnale d' ingresso, con fattori moltiplicazione/divisione configurabili).
È possibile ottenere pertanto segnali di clock anche a frequenza superiore rispetto a quella d' ingresso (a passi discreti), configurando opportunamente questo componente.
Il blocco DCM inoltre genera i segnali di reset che vengono utilizzati per la corretta

inizializzazione dei blocchi che fanno uso di registri, che, se non opportunamente azzerati, possono portare a comportamenti imprevisti del sistema implementato.

Blocco di acquisizione e filtraggio

Il convertitore analogico digitale di tipo delta sigma, implementato nella ASIC, converte un segnale analogico in un flusso dati digitalizzato, composto da una serie di uno e zeri. La densità di uno in questo flusso di dati uscenti dal convertitore è proporzionale al segnale di ingresso analogico.

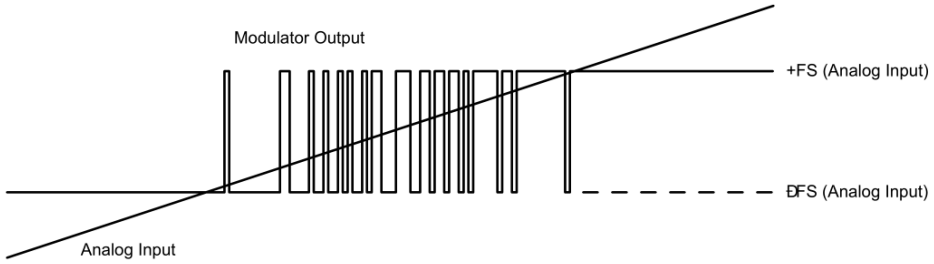


Figura 6: relazione fra segnale di ingresso (rampa) e uscita digitale del' ADC

Le tecniche di oversampling (campionamento ad una frequenza significativamente più alta di quella di Nyquist) e di noise shaping (che in pratica è l' effetto filtrante passa alto, rispetto al rumore, proprio di questo tipo di ADC) sono usate per ridurre il rumore di quantizzazione nella banda di interesse.

Lo scopo del filtraggio dei dati in arrivo dall' ASIC è quindi duplice: principalmente questa operazione è necessaria per attenuare il rumore nel segnale digitalizzato. Lo scopo secondario è quello di convertire il flusso dati seriale ad alta frequenza in un flusso dati ad alta risoluzione, a minor data rate: la cosiddetta decimazione. L' impatto dell' errore di quantizzazione sul segnale digitalizzato mediante questa tipologia di ADC è notevole, dato che il numero di bit per campione è estremamente basso. È compito del filtro decimatore eliminare il rumore indesiderato, presente nello spettro del segnale digitalizzato, al di sopra della banda di Nyquist, di modo che non si riversi nella banda di interesse (aliasing) durante il processo di decimazione.

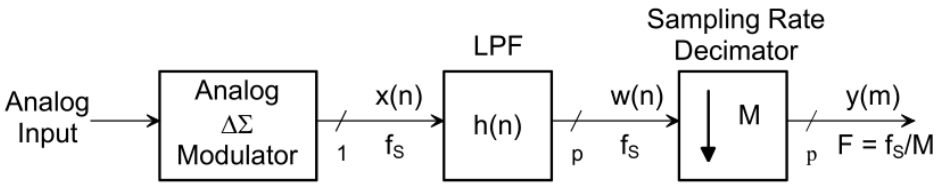


Figura 7: diagramma a blocchi della decimazione per un ADC sigma delta

Il segnale x(n) proveniente dal modulatore/convertitore delta sigma è un bitstream a frequenza fs. Questo segnale è, per prima cosa, filtrato da un filtro passa basso digitale con funzione di risposta armonica h(n), con frequenza di taglio π/M, dove con π viene indicata la frequenza normalizzata (in radianti) corrispondente alla frequenza di Nyquist (cioè metà della frequenza di campionamento fs). Il filtro h(n) rimuove tutta l' energia al di sopra della frequenza π/M dal segnale x(n), ed evita l' aliasing nel processo di decimazione, quando il segnale w(n) viene ricampionato dal Sampling Rate Decimator.

Dal punto di vista matematico, tutto ciò è riassumibile nella equazione di convoluzione:

$$y(m) = \sum_{k=-\infty}^{\infty} h(k) \cdot x(Mm - k)$$

È evidente, inoltre, come il segnale di input sia shiftato di M campioni per ciascun nuovo data computato in uscita.

Per mantenere bassi i costi del sistema, è importante il criterio con il quale questo filtro decimatore viene implementato. Tale criterio è strettamente connesso al tipo, all' ordine e all' architettura del filtro digitale utilizzato nell' implementazione. L' ordine del filtro passa basso, a sua volta, è direttamente associato ai requisiti di ripple nella banda passante e nella banda filtrata, oltre al requisito di ampiezza della banda di transizione.

Una delle migliori tipologie di architettura che riesce a soddisfare i criteri precedentemente specificati è rappresentata dai filtri Sinc: questa particolare architettura di filtraggio offre alcuni vantaggi rispetto alle tradizionali architetture per i filtri digitali. A differenza di un filtro FIR o IIR non è necessario l' uso di un blocco MAC (Multiplier-Accumulator), altrimenti indispensabile per l' operazione di

convoluzione su cui si basano tali tipologie di filtri. Non è altresì necessario disporre di una memoria in cui immagazzinare i campioni (o taps) del tipo di filtro scelto.

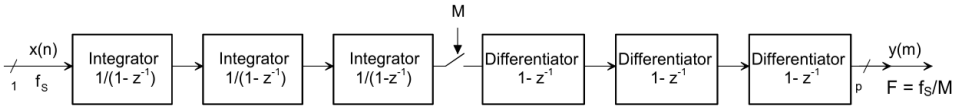


Figura 8: topologia del filtro Sinc di terzo ordine

La funzione di trasferimento desiderata è quindi implementata in modo più efficiente facendo uso di alcuni sommatori in cascata, operanti ad alta frequenza di campionamento, seguiti da altrettanti sottrattori, operanti a minore sampling rate (fs/M). Nella scelta di un filtro Sinc è opportuno tenere conto dell' ordine del modulatore delta sigma che fornisce i dati: l' ordine del filtro Sinc deve essere almeno maggiore di un fattore 1 rispetto all' ordine del modulatore, per prevenire l' entrata in banda passante dell' eccessivo aliasing del rumore al di fuori della banda passante, proveniente dal modulatore. Nel caso in esame, essendo l' ADC sigma delta del secondo ordine, è necessario implementare un filtro Sinc del terzo ordine. Il filtro inoltre ha una banda passante selezionabile in base al fattore di oversampling: le bande previste nel dispositivo (con clock di sistema a 10 MHz) sono di 10 kHz, 5 kHz, 1.25 kHz e 625 Hz.

L' FPGA si interfaccia con la ASIC attraverso il blocco di filtraggio, mediante due linee dette SYNC e OUT.

Sulla linea SYNC è presente un segnale di sincronismo, derivato dal clock di sistema e veicolato all' ASIC attraverso la porta SPI come precedentemente indicato, il cui scopo è quello di permettere la sincronizzazione del ricevitore del bitstream (in questo caso l'FPGA), di modo che il flusso di bit rappresentanti il segnale digitalizzato (flusso che è veicolato sulla linea OUT) sia correttamente interpretabile.

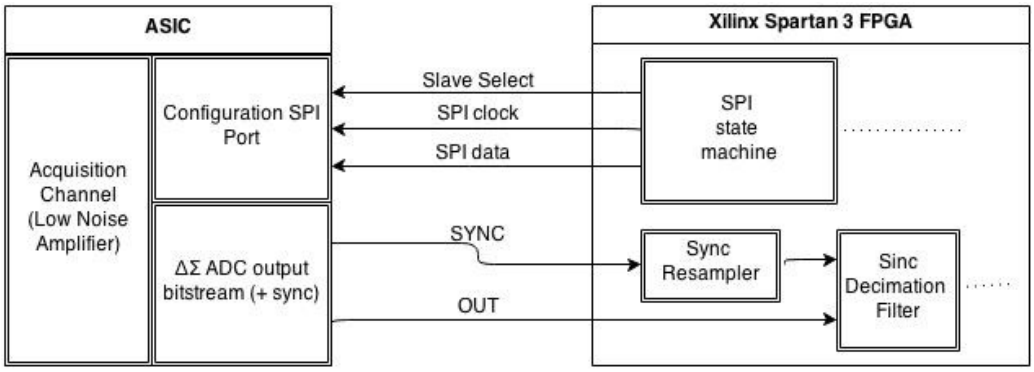


Figura 9: Schema interconnessioni FPGA-ASIC

Questo segnale di sincronismo però non viene utilizzato in maniera diretta all' interno dell' FPGA: provenendo da un sistema esterno, esso non è sincrono con i segnali interni all' FPGA, pertanto, sempre internamente all' FPGA, è stato predisposto un blocco che ne effettua il ricampionamento, utilizzando il clock di sistema come base dei tempi. Questa precauzione è necessaria in quanto, altrimenti, potrebbero generarsi problemi tali da rendere aleatorio il comportamento del sistema.

Per semplicità si farà riferimento sempre al segnale SYNC, in quanto ai fini della trattazione teorica assume lo stesso significato del segnale ricampionato. Quindi, ciò che in pratica viene trasmesso dalla ASIC è assimilabile ad un flusso seriale sincrono, anche se in realtà è semplicemente l' uscita non decimata di un ADC delta-sigma.

L' implementazione dell' architettura di filtraggio di tipo Sinc del terzo ordine può essere illustrata schematicamente nella figura 10.

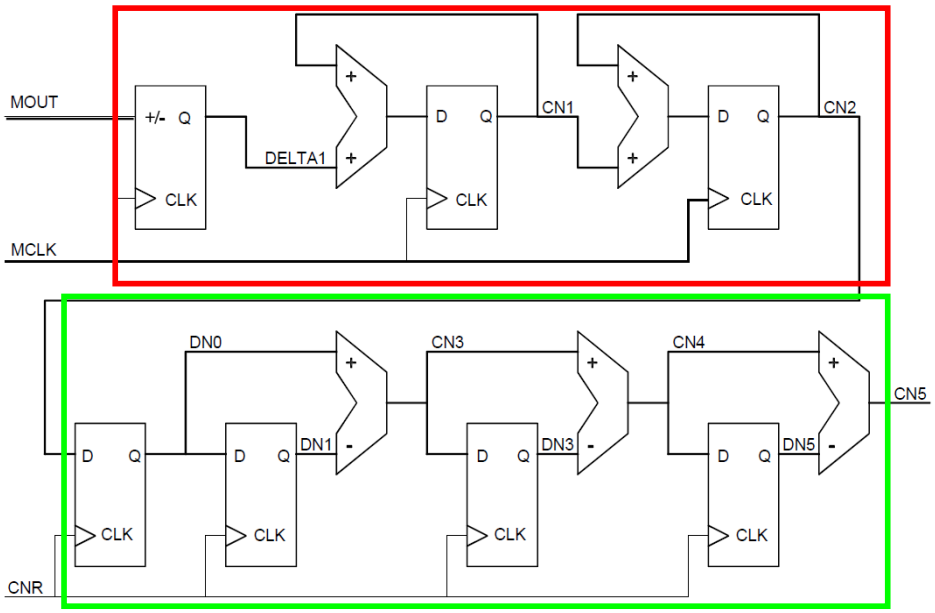


Figura 10: schematizzazione del filtraggio Sinc3

Il bitstream proveniente dall' ADC sigma delta, indicato nella figura con MOUT, entra nel primo registro, il quale campiona l' ingresso con una certa frequenza di clock. Tale segnale, indicato con MCLK è il segnale di sincronismo fornito dall' ASIC.

Il primo blocco, riquadrato di rosso, svolge la funzione di accumulatore: dopo n cicli di clock l' uscita DELTA1 rappresenta la somma dei precedenti n bit giunti dall' ADC. Tale risultato viene poi sommato (per due volte) alle precedenti somme parziali, immagazzinate nei registri posti all' uscita dei sommatori, attraverso i medesimi sommatori disposti in cascata.

Il blocco successivo, riquadrato in verde, realizza una differenziazione a più stage: ciascun blocco aritmetico prende in ingresso il valore campionato all' istante attuale dallo stage precedente e vi sottrae il valore presente all' istante di campionamento precedente, memorizzato nei registri con uscita DN*.

Questa operazione di differenziazione, però, non avviene con la stessa cadenza dell' accumulazione: essa viene eseguita dopo un certo numero n di operazioni di accumulazione e ripetuta dopo altre n, dove n dipende dal fattore di decimazione scelto, cioè dalla banda scelta per il filtro.

L' operazione di differenziazione viene quindi eseguita quando si attiva il segnale CNR = MCLK/M, dove M è il fattore di decimazione.

Ad ogni differenziazione viene ovviamente aggiornata l' uscita del blocco filtrante: dal dato presente sul bus CN5 vengono estrapolati i 16 bit significativi e questi ultimi formano il dato che verrà poi trasmesso al PC.

Blocco di simulazione chip ASIC

Il mio progetto di tesi è stato sviluppato utilizzando la scheda di sviluppo DLP-FPGA per l' FPGA Xilinx Spartan 3E. Ovviamente, essendo un prodotto commerciale general purpose, non era dotato della ASIC.
Per questo motivo, all' interno dell' FPGA è stato incluso un blocco il cui compito è quello di simulare il comportamento "alle porte" della ASIC, ovverosia è stato aggiunto al codice del progetto un blocco che presenta le stesse interfacce digitali del chip reale: da una parte può essere configurato dai parametri ricevuti da PC e dall' altra genera i segnali OUT e SYNC, che vengono poi internamente reindirizzati al blocco di filtraggio.
I segnali generati sono ovviamente semplici segnali di test, ottenuti combinando le uscite di alcuni contatori utilizzati come divisori di clock, in modo da simulare varie "tensioni di ingresso" fisse, oppure un segnale ad onda quadra.
Anche il segnale di sincronismo SYNC viene generato a partire dal clock di sistema, dividendolo per 8 (quindi 1.25 MHz con clock 10MHz oppure 10 MHz con clock 80MHz).
È possibile cambiare il tipo di forma d' onda simulata, selezionando dal software su PC uno dei bit di CHSEL (normalmente utilizzati per selezionare quale canale di acquisizione utilizzare, sul device target).

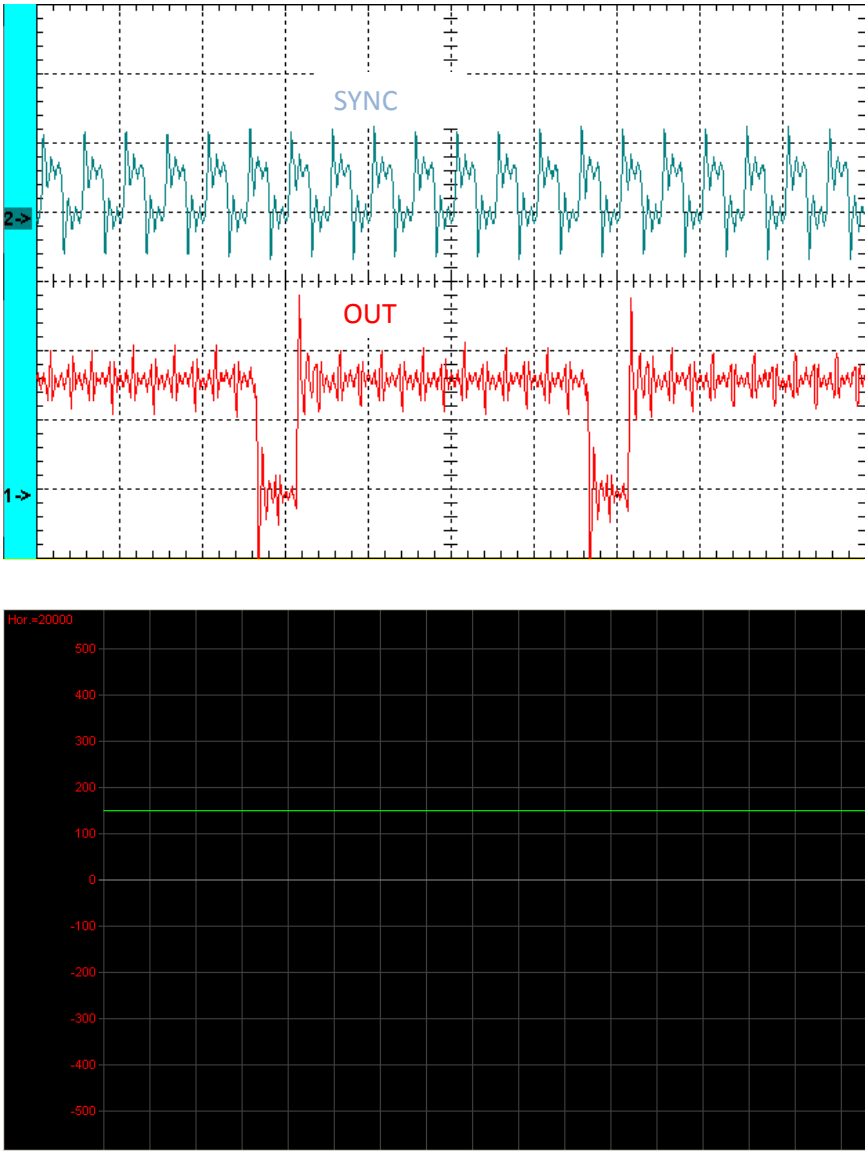


Figura 11: Oscillogrammi dei segnali SYNC e OUT e relativo segnale analogico associato, visualizzato sul software di acquisizione

Gestione USB

Il sistema si interfaccia, sia per la configurazione del chip ASIC che per la ricezione dei dati elaborati, con un personal computer dotato di interfaccia USB e sistema operativo Windows, sul quale è installato il software di acquisizione e gestione.

Il bus USB, grazie alla diffusione e facilità d' uso e soprattutto grazie alla velocità di trasferimento che garantisce, si adatta bene ad essere utilizzato oltre che da

dispositivi di memorizzazione di massa anche da strumentazione come quella in oggetto, svincolandola da interfacce proprietarie e sicuramente non universalmente diffuse.

Il protocollo USB, nella sua specifica 2.0, non è però un protocollo di semplice implementazione, come invece può esserlo ad esempio il più semplice RS-232.

Inoltre, richiedendo particolari accorgimenti riguardo le specifiche elettriche, esso non è neanche facilmente implementabile in un dispositivo programmabile, sia esso FPGA o MCU, che non lo preveda come feature. È necessario avere il cosiddetto layer fisico (PHY) costituito da un particolare transceiver ad altre prestazioni, che si occupa anche di traslare i livelli elettrici dei segnali.

Per questo motivo, ed anche per razionalizzare l' uso delle capacità messe disposizione dall' FPGA, evitando di implementare il protocollo internamente ad essa, si è ricorsi all' uso di un circuito integrato commerciale che assolve queste funzioni e che permette di essere interfacciato con l' FPGA, attraverso un bus di più semplice implementazione e di più facile gestione, senza però perdere quelle importanti proprietà del bus USB, prima fra tutte la velocità di trasferimento dati.

Il circuito integrato per l' interfacciamento USB

Questo integrato, identificato dalla sigla FT2232D, è un dispositivo prodotto dall' azienda FTDI che, grazie anche ad un particolare driver, permette di far identificare al PC un dispositivo dotato di tale chip come se fosse una periferica seriale.

Ciò permette, anche dal punto di vista del software di elaborazione su PC, alcune notevoli semplificazioni, come ad esempio la possibilità di utilizzare API standard del sistema operativo, pensate per i dispositivi seriali piuttosto che particolari librerie sviluppate appositamente per il dispositivo.

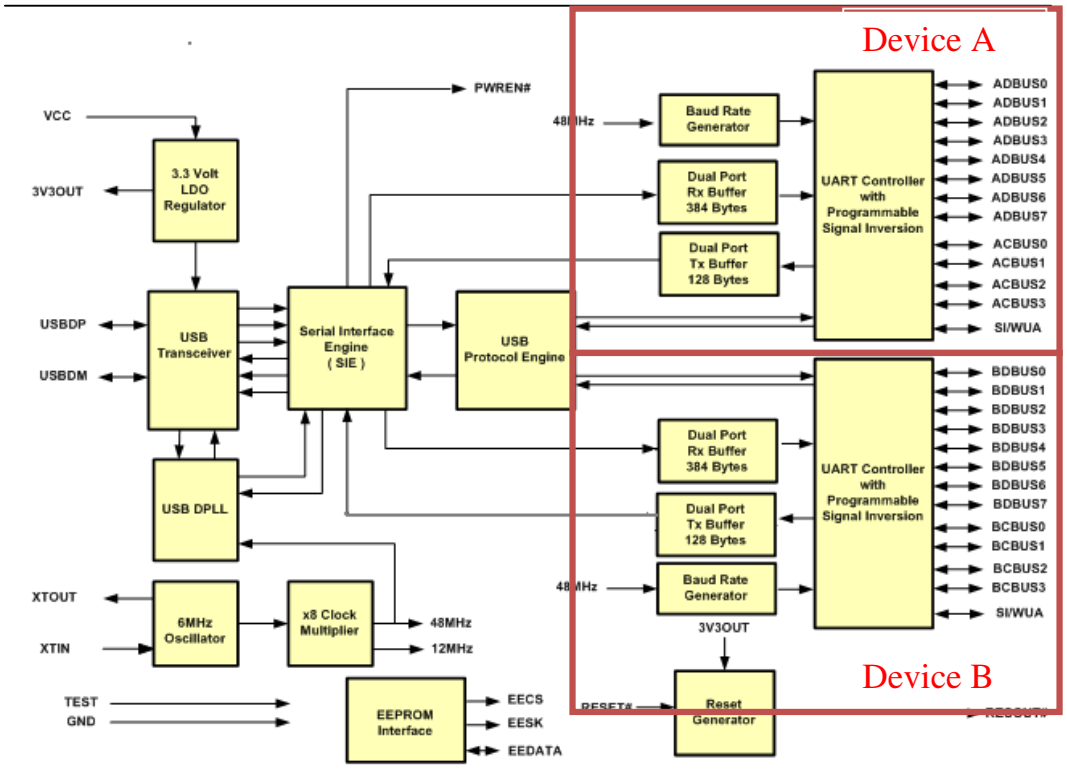


Figura 12: schema a blocchi FT2232D

L' integrato, molto versatile, è utilizzato nel sistema per due scopi diversi: come evidenziato in figura 12, al suo interno vi sono due distinti dispositivi, che svolgono le medesime funzioni e che sono distintamente utilizzabili dal PC.

Uno di essi è esclusivamente dedicato alla programmazione della memoria Flash, contente la configurazione dell' FPGA, ed è utilizzato solo dal software di programmazione della scheda DLP-FPGA (si veda lo schema a blocchi della development board in figura 5).

L' altro dispositivo (Device B) è disponibile per essere utilizzato dall' FPGA come canale di comunicazione, sia in input che in output.

Ciascun device ha la possibilità di essere configurato per lavorare in diverse modalità: quelle più comunemente utilizzate sono l' emulazione seriale, solitamente utilizzata nel caso si connettano al chip vecchi dispositivi studiati per il bus RS-232, in modo da poterli adattare ai moderni PC, che normalmente non possiedono più questa interfaccia, e la modalità FIFO, che permette di utilizzare il device con un bus parallelo ad 8bit.

La configurazione della modalità di funzionamento è memorizzata in una memoria FLASH esterna al chip e modificabile solo mediante un apposito software su PC.

Il device A, dedicato alla programmazione della memoria Flash di configurazione per l' FPGA, è impostato per funzionare come seriale SPI (sfruttando la modalità del chip denominata MPSSE, Multi-Protocol Synchronous Serial Engine), in questo modo può essere direttamente interfacciato alla EEPROM e riconfigurare l' FPGA senza la necessità di dover utilizzare un apposito e costoso programmatore.

Il device B è invece direttamente interfacciato con i pin di I/O dell' FPGA, ed è utilizzabile dalla stessa per le comunicazioni da e per un dispositivo USB host, come appunto un personal computer.
Anche il device B è programmabile nelle diverse modalità: nel sistema corrente viene utilizzata la modalità FIFO asincrona.
Essa mette a disposizione un bus parallelo ad 8 bit bidirezionale (sul bus BDBUSx), ed alcune linee per il controllo del flusso dati (sul bus BCBUSx):

- Linea $\overline{\textbf{RXF}}$: quando è a livello logico basso, indica la presenza di un dato nel buffer FIFO del chip. Ciò significa che è stato inviato un nuovo dato dal PC, ed è opportuno effettuare uno strobe sulla linea $\overline{\textbf{RD}}$ per prelevarlo. Invece se è posta a livello alto, indica che il chip è in attesa di ricevere un nuovo byte dall' host. Tale linea viene utilizzata per sincronizzare il ricevitore.

- Linea $\overline{\textbf{RD}}$: questa linea (normalmente mantenuta inattiva a livello logico alto dall' FPGA) è il segnale che sul fronte di salita indica al chip di prelevare un nuovo dato dal FIFO interno mentre sul fronte di discesa indica all' FTDI di porre tale dato sul bus. È compito poi dell' FPGA immagazzinare per i propri scopi tale dato.

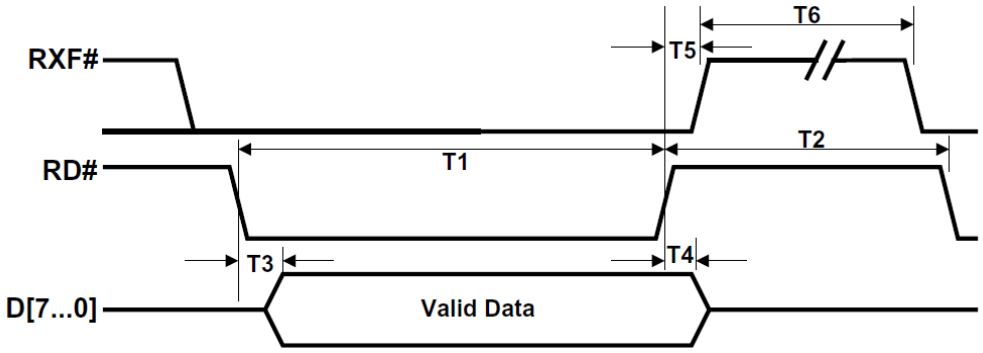


Figura 13: Ciclo di lettura

Time	Description	Min	Max	Unit
T1	RD# Active Pulse Width	50	ns	RD# Active Pulse Width
T2	RD# to RD Pre-Charge Time	50 + T6	ns	RD# to RD Pre-Charge Time
T3	RD# Active to Valid Data ****Note 19**	20	50	ns
T4	Valid Data Hold Time from RD# Inactive ****Note 19**	0	ns	Valid Data Hold Time from RD# Inactive ****Note 19**
T5	RD# Inactive to RXF#	0	25	ns
T6	RXF# inactive after RD# cycle	80	ns	RXF# inactive after RD# cycle

Figura 14: Temporizzazione ciclo di lettura

- Linea $\overline{\textbf{TXE}}$: se posta a livello logico alto, indica che il chip è occupato e non è possibile trasferirgli un nuovo dato.
  Viceversa con livello logico basso, è possibile effettuare lo strobe della linea $\overline{\textbf{WR}}$ e trasferire il dato, preventivamente posto sul bus.

- Linea $\overline{\textbf{WR}}$: è la linea che da l' abilitazione alla scrittura del dato nel buffer FIFO interno al chip FTDI.
  Una volta avvenuta la scrittura e riempito il buffer, il chip provvederà a trasferire al PC un pacchetto dati con il contenuto di tale buffer.

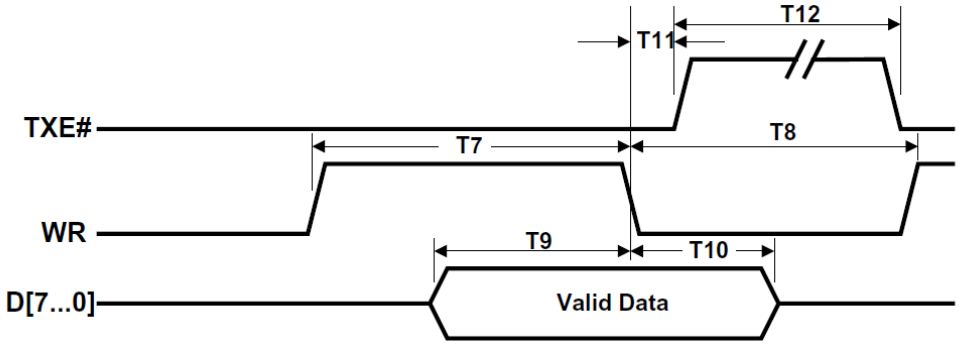


Figura 15: Ciclo di scrittura

Time	*Description*	*Min*	*Max*	*Unit*
T7	WR Active Pulse Width	50	ns	WR Active Pulse Width
T8	WR to WR Pre-Charge Time	50	ns	WR to WR Pre-Charge Time
T9	Data Setup Time before WR inactive	20	ns	Data Setup Time before WR inactive
T10	Data Hold Time from WR inactive	0	ns	Data Hold Time from WR inactive
T11	WR Inactive to TXE#	5	25	ns
T12	TXE inactive after WR cycle	80	ns	TXE inactive after WR cycle

Figura 16: Temporizzazioni ciclo di scrittura

La modalità FIFO è stata scelta per il progetto in quanto permette un più facile interfacciamento e non richiede la serializzazione dei dati.

Dato che il bus è bidirezionale, cioè è essenziale accedervi sia in lettura che in scrittura, è stato necessario implementare all' interno dell' FPGA un buffer bidirezionale ad otto bit (transceiver), la cui direzione è gestita dall' blocco di ricezione dati.

Blocco di ricezione configurazione da PC

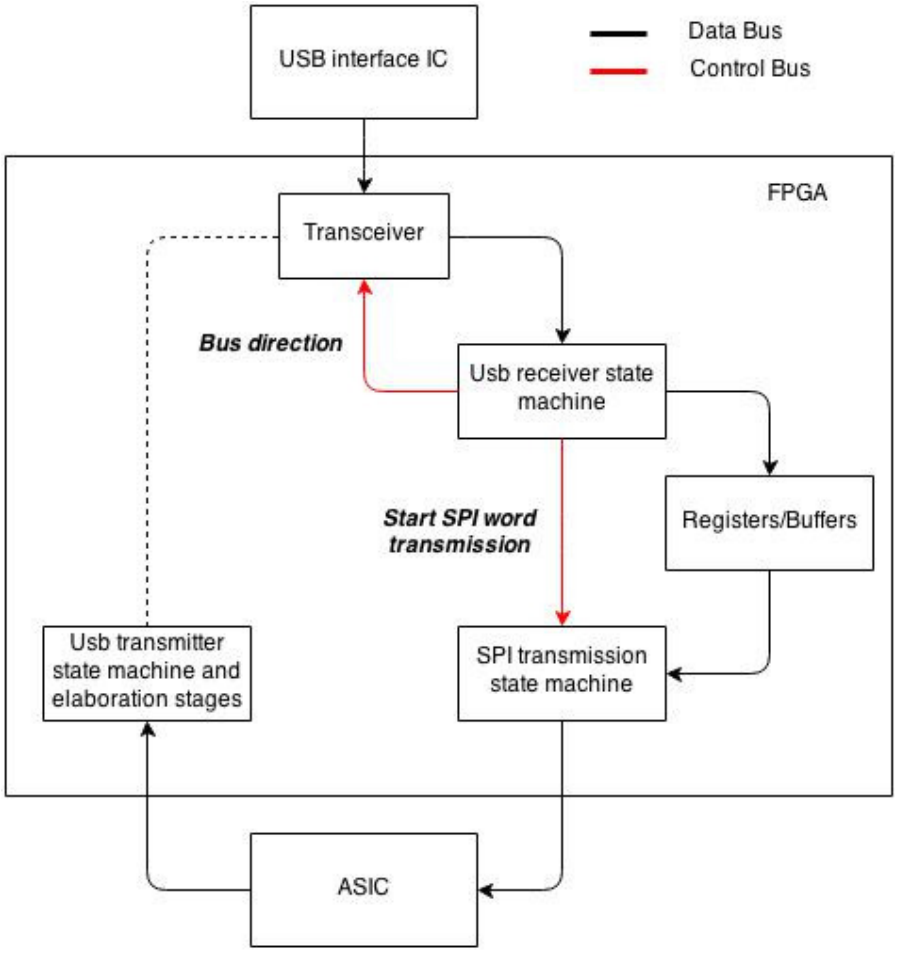


Figura 17: Schematizzazione blocco di ricezione

Come già detto in precedenza, il chip FTDI è funzionale al sistema in due ambiti:

- la ricezione dei parametri di configurazione per l' ASIC
- la trasmissione dei dati elaborati verso il PC.

Il primo di questi aspetti è gestito da una apposita sezione del codice, costituita da alcune macchine a stati finiti.

La prima di esse gestisce la comunicazione con il chip FTDI, monitorando la linea $\overline{\textbf{RXF}}$ in attesa della segnalazione di un nuovo byte ricevuto.

Qualora ve ne fossero, viene abilitato in lettura il bus dedicato all' interfacciamento con l' FTDI, commutando opportunamente il transceiver: se il byte sul bus corrisponde al byte di start (si veda la descrizione del protocollo nei paragrafi seguenti), viene resettato un contatore interno, utilizzato per l' indirizzamento di un altra apposita macchina a stati, assimilabile ad una piccola RAM a 16 byte, nella quale vengono immagazzinati i byte successivi a quello di start.
Se invece il byte è un byte di dato (e il contatore ha valore maggiore di 0), viene semplicemente immagazzinato ed il contatore incrementato, ripetendo il loop per tutti e 16 i byte.
Ovviamente se il byte di start non fosse corretto, tutta la trasmissione verrebbe scartata e la macchina a stati resterebbe in attesa di un nuovo byte di start.

Al termine della ricezione dei 16 byte di configurazione dal PC, viene triggerata una ulteriore macchina a stati che, prelevando i bit di interesse nei registri appena aggiornati, costruisce la word di configurazione a 32 bit per l' ASIC e la invia al chip mediante protocollo SPI.
Come ovvio, il bus viene rilasciato ed il transceiver è posto in modalità scrittura, cosicché il blocco di trasmissione dei dati filtrati possa trasmettere verso il PC.

La decisione di far gestire il bus-transceiver al blocco di ricezione è legata alla reattività del sistema: una variazione di impostazione sulla GUI deve essere immediatamente recepita dal sistema e comunicata alla ASIC, a discapito dell' integrità dei dati ricevuti.
Tale integrità è ovviamente non importante, come è logico pensare, nel momento in cui si sta impostando, ad esempio, una misura ma diviene importante solo al termine del setup. Difatti, una volta che il PC ha comunicato al sistema l' impostazione definitiva, il software sul computer resta in attesa dei dati senza inviare alcunché al dispositivo, di modo che la macchina a stati di ricezione rimanga inattiva e lasci libero il bus dell' FTDI per la trasmissione.

Da notare infine che il protocollo SPI è implementato in maniera particolare: la trasmissione effettiva avviene solo quando il segnale Slave Select è attivato (per un periodo equivalente a 32 bit).
Questa particolarità è dovuta al fatto che la linea di clock della porta di configurazione SPI fornisce il clock anche al resto della ASIC e deve pertanto

rimanere sempre attivo.

Da tale segnale di clock viene poi derivato il segnale di sincronismo SYNC in uscita dalla ASIC ed in ingresso all' FPGA.

Nella figura 17 è schematizzata la sezione di ricezione dati del dispositivo.

Blocco di trasmissione dati elaborati verso il PC

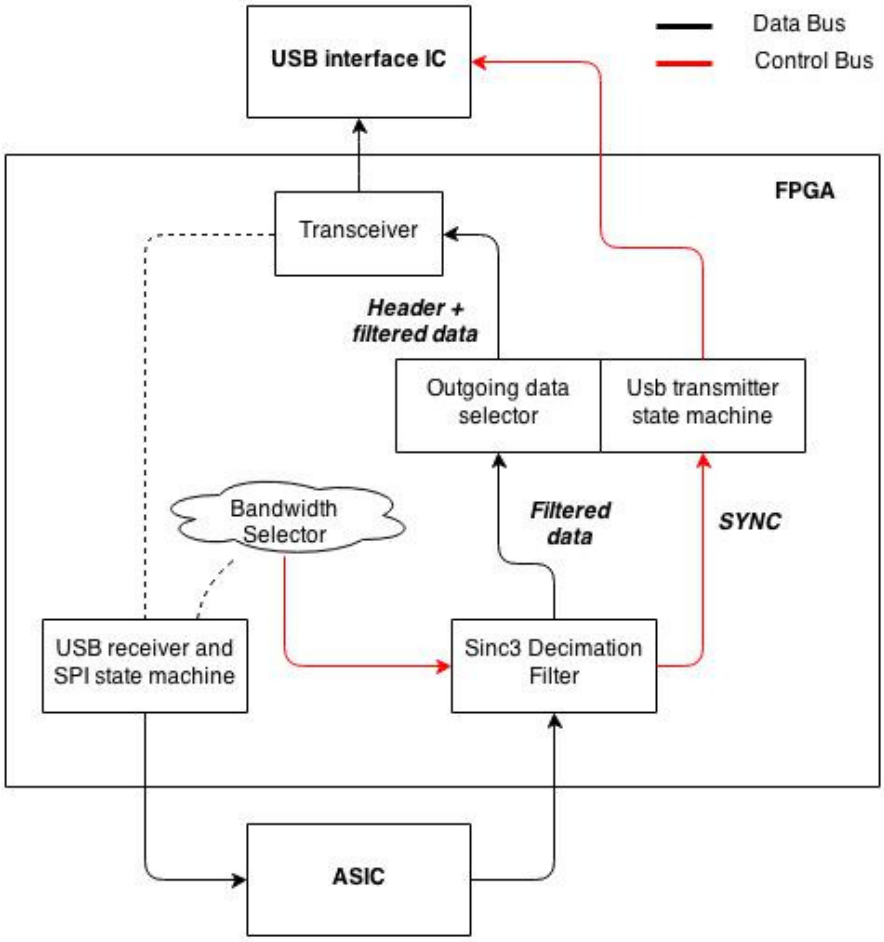


Figura 18: Schematizzazione blocco di trasmissione

La seconda sezione del codice dedicata all' I/O USB è strettamente connessa al blocco di filtraggio, visto che il suo compito è quello di inviare i dati filtrati al PC, non appena vengono prodotti da quel blocco di elaborazione.

Come precedentemente accennato, il blocco di filtraggio lavora con una base dei tempi prefissata, dipendente dalla frequenza del segnale di sincronismo proveniente dall' ASIC. Questo segnale alimenta un contatore binario che funge da divisore di frequenza: ricordando ciò che si è detto nel paragrafo in cui viene descritto il blocco filtrante, il blocco sottrattore viene attivato dopo che un certo numero di bit vengono accumulati nei registri del blocco sommatore.
Pertanto è stata predisposta una rete combinatoria che in base all' oversampling ratio (parametro inviato da PC), cioè in base alla banda del filtro desiderata, attiva il segnale di sottrazione: dividendo la frequenza del segnale di sincronismo per un numero M, potenza del 2, si ottiene il segnale necessario. Conseguentemente, aggiornandosi l' uscita del blocco filtrante, viene attivato anche il segnale di abilitazione all' invio al PC del dato filtrato.

Il contatore binario controlla anche un' altra macchina combinatoria, assimilabile nel comportamento ad una piccola ROM a 5 bytes che, indirizzata dai 5 bit di tale contatore, discrimina quale dato porre sul bus di trasmissione: quando il contatore ha valore compreso fra 0 e 3 vengono posti sul bus in successione i byte dell' header, mentre quando il valore del contatore è pari a 4 oppure 5, vengono posti sul bus rispettivamente il byte alto ed il byte basso della word a 16 bit del dato filtrato (si faccia riferimento al prossimo paragrafo per una descrizione completa del pacchetto dati). Per valori maggiori del contatore, sul bus viene poso il valore 0. Negli stati con dati validi, cioè per valori del contatore compresi fra 0 e 5, si attiva un secondo segnale di abilitazione alla trasmissione.
È evidente come questa architettura preveda delle restrizioni importanti alle tempistiche per poter trasmettere un dato al computer, mettendo a disposizione un tempo massimo pari al periodo che trascorre fra due successive attivazioni del primo segnale di abilitazione all' invio dei dati.

La macchina a stati che gestisce l' invio dei dati al PC è piuttosto semplice: ricordando che il sistema (allo stato originale) funzionava a 10 MHz, è stata operata la scelta di implementare la comunicazione in scrittura con l' FTDI facendo uso di wait states statici.
La macchina a stati attende che vi sia un fronte di salita del segnale di sincronismo SYNC, condizione che si verifica a metà del periodo in cui è presente sul bus il byte

da trasmettere, dopodiché attiva il segnale $\overline{\mathbf{WR}}$ diretto verso l' FTDI nel caso in cui sia effettivamente necessario trasmettere un dato, in base ai segnali di abilitazione provenienti dal blocco filtrante e dalla "ROM": in questo modo la macchina a stati da l' abilitazione alla scrittura solo quando è necessario, in modo da non trasmettere più volte la stessa informazione.

Il segnale $\overline{\mathbf{WR}}$ resta in questo stato per due cicli di clock (questi sono i wait states statici), sufficienti (a 10 MHz) per soddisfare le tempistiche imposte dalle specifiche dell' FTDI, senza controllare lo stato della linea $\overline{\mathbf{TXE}}$.

Come si vedrà questo ha comportato notevoli problemi nel momento in cui si è tentato di innalzare la frequenza di clock del sistema.

Capitolo 5

Protocolli di comunicazione

Il dispositivo utilizza 3 tipi di pacchetto dati nel suo ciclo di funzionamento:

- il pacchetto dati (4 byte di header e 2 di dati) per trasferire il segnale digitale filtrato dal dispositivo al PC
- il pacchetto di 16 byte proveniente dal PC e generato dal software di interfaccia, contenente i parametri di configurazione per la ASIC e per l' FPGA (banda del filtro ecc…)
- la word da 32 bit per la configurazione della ASIC generata dalla macchina a stati di trasmissione SPI.

Ciascuno di essi ha un proprio formato ben preciso ed è schematizzabile come segue.

Pacchetto dati (6 bytes)

Questo pacchetto dati viene scambiato fra il PC ed il dispositivo e trasporta i dati filtrati, che vengono trasferiti al computer per essere visualizzati, oppure per ulteriori elaborazioni. Esso è costituito da un preambolo di quattro byte, detto header, utilizzato dal software su PC per sincronizzarsi con il flusso dati. Di seguito a questi 4 byte ne vengono trasferiti altri due, che contengono il dato vero e proprio. Essendo tale dato a 16 bit, il byte più significativo è posto prima (cioè viene trasmesso prima) del byte meno significativo.

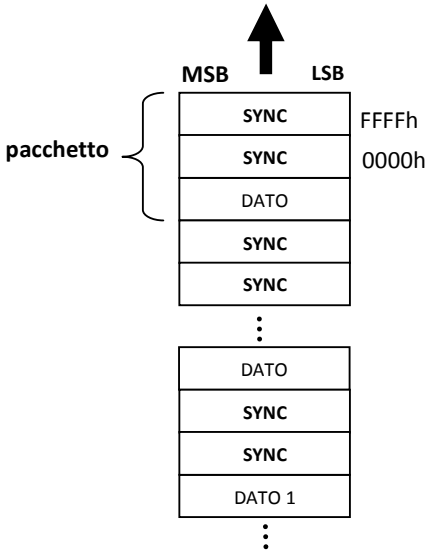


Figura 19: una rappresentazione del pacchetto dati elaborati

Pacchetto configurazione da PC (16 byte)

Tale pacchetto è stato studiato per essere genericamente applicabile ad un certo numero di dispositivi di acquisizione dati, simili a quello in esame, pertanto presenta anche campi non significativi per questa applicazione. In generale, esso trasporta le informazioni riguardanti la configurazione del dispositivo, impostate sulla GUI, come ad esempio:

- banda di filtraggio
- fondo scala
- abilitazione/disabilitazione accessori del chip (filtri, tensioni di riferimento, ecc…)
- pilotaggio DAC interno alla ASIC (valore per la tensione di riferimento, generazione onda triangolare di test…)

Il pacchetto è così conformato:

	MSb							LSb	
START	80h								1
DACint0	0	DAC6int	DAC5int	DAC4int	DAC3int	DAC2int	DAC1int	DAC0int	2
DACint1	0	0	0	0	0	DAC9int	DAC8int	DAC7int	3
DACext0	0	DAC6ext	DAC5ext	DAC4ext	DAC3ext	DAC2ext	DAC1ext	DAC0ext	4
DACext1	0	0	0	0	0	DAC9ext	DAC8ext	DAC7ext	5
CHSEL	0	CHSEL-6	CHSEL-5	CHSEL-4	CHSEL-3	CHSEL-2	CHSEL-1	CHSEL-0	6
CONFIG0	0	CFG0-6	CFG0-5	CFG0-4	CFG0-3	CFG0-2	CFG0-1	CFG0-0	7
CONFIG1	0	CFG1-6	CFG1-5	CFG1-4	CFG1-3	CFG1-2	CFG1-1	CFG1-0	8
CONFIG2	0	CFG2-6	CFG2-5	CFG2-4	CFG2-3	CFG2-2	CFG2-1	CFG2-0	9
CONFIG3	0	CFG3-6	CFG3-5	CFG3-4	CFG3-3	CFG3-2	CFG3-1	CFG3-0	10
CONFIG4	0	CFG4-6	CFG4-5	CFG4-4	CFG4-3	CFG4-2	CFG4-1	CFG4-0	11
CONFIG5	0	CFG5-6	CFG5-5	CFG5-4	CFG5-3	CFG5-2	CFG5-1	CFG5-0	12
CONFIG6	0	CFG6-6	CFG6-5	CFG6-4	CFG6-3	CFG6-2	CFG6-1	CFG6-0	13
CONFIG7	0	CFG7-6	CFG7-5	CFG7-4	CFG7-3	CFG7-2	CFG7-1	CFG7-0	14
CONFIG8	0	CFG8-6	CFG8-5	CFG8-4	CFG8-3	CFG8-2	CFG8-1	CFG8-0	15
CONFIG9	0	CFG9-6	CFG9-5	CFG9-4	CFG9-3	CFG9-2	CFG9-1	CFG9-0	16

Figura 20: il pacchetto di configurazione da PC

Il primo byte ricevuto è il byte START, che rappresenta l' header per tutto il pacchetto e ne identifica la validità (come precedentemente visto, il blocco di ricezione della configurazione effettua un check su tale header).

I valori più rilevanti per quanto concerne l' elaborazione sull' FPGA sono:

- i bit che identificano la banda di filtraggio richiesta
- il bit che indicano la richiesta di abilitazione della onda triangolare di test
- i bit che complessivamente formano la word rappresentate un valore di tensione generabile dal DAC interno alla ASIC, utilizzato per generare una tensione di riferimento oppure una forma d' onda, se tale valore viene fatto variare nel tempo
- Il bit per la dis/abilitazione della modalità fast 10/80 MHz
- i restanti bit (quelli necessari e rilevanti) vengono trasferiti alla ASIC senza elaborazioni.

Word di configurazione per la ASIC (32 bit)

La via per configurare la ASIC è costituita, come precedentemente indicato, dalla porta SPI connessa all' FPGA. Quando viene ricevuto il pacchetto da PC, descritto precedentemente, una macchina a stati si occupa di elaborarlo e preparare questa word di 32 bit da inviare mediante la porta SPI.

La word che viene costruita ha la seguente forma:

CFG1-0	Sync Sel - Select the sync signal: 0=sync core 1 / 1=sync core 2
CFG1-1	SDO Sel - Select the SDO signal: 0=SDOe core 1 / 1=SDO core 2
CFG1-2	Range (0=±200pA / 1=±20nA)
CFG1-3	Fast (0=slow mode (10kHz) / 1=fast mode (100kHz))
CFG1-4	Subtractor: 1=Activate subtractor for time-varying Vc
CFG1-5	TP1: 1=Connect test-pad TP1
CFG1-6	TP2: 1=Connect test-pad TP2
CFG2-0	TP3: 1=Connect test-pad TP3
CFG2-1	TP4: 1=Connect test-pad TP4
CFG2-2	VcmForce1: 1=Force positive LNA input to VcmA (core 1)
CFG2-3	VcmForce2: 1=Force positive LNA input to VcmA (core 2)
CFG2-4	EXT_CAP: 1=Connect CAP pin to positive LNA input
CFG2-5	VcInt: 1=Activate DAC output
CFG2-6	En_dComp: 1=Activate digital-offset compensation
CFG3-0	OSR64
CFG3-1	OSR128
CFG3-2	OSR512

CFG3-3	OSR1024
CFG3-4	VcInt Channel Sel: 1=apply DACint to all channels (non richiesta su board CarbonioTest)
CFG3-5	EN-triangular: 1=enable the generation of the triangular wave
CFG3-6	RESET - 1=reset; 0=off

Figura 21: word di configurazione per la ASIC, a meno dei bit per il DAC

Come si può vedere, essa è costituita da un sottoinsieme dei bit del pacchetto ricevuto da PC e precedentemente descritto.

Capitolo 6

Descrizione delle diverse soluzioni

Al fine di giungere al soddisfacimento degli obiettivi prefissati, sono state intraprese diverse vie di progetto, l' ultima delle quali si è poi rivelata vincente. Comunque, prima di giungere a tale soluzione definitiva, sono stati necessari alcuni step intermedi di sperimentazione: dai risultati ottenuti si è potuto dedurre la soluzione ottima.

Prima soluzione adottata

Il primo tentativo per raggiungere l' obiettivo fissato è consistito nel semplice innalzamento della frequenza di lavoro del sistema, da 10 MHz ad 80 MHz.
Per poter incrementare la frequenza è stato necessario utilizzare un blocco DCM doppio, in quanto non è possibile ottenere la frequenza di 80 MHz manipolando i parametri di configurazione di un singolo blocco DCM, avendo in ingresso un segnale a 6 Mhz. Ponendone in cascata due, invece, è possibile avere un controllo più fine sulla frequenza, ed ottenere quella desiderata, a meno di un piccolo aumento di jitter sul segnale.

Questo cambiamento di clock ha comportato un aumento della velocità della ASIC con conseguente aumento del flusso di dati in ingresso all' FPGA.
Di conseguenza, vi è stato anche un aumento della velocità di filtraggio che ha prodotto un incremento di flusso dati in uscita dal filtro decimatore (il blocco filtrante non è stato modificato in quanto il suo funzionamento non è dipendente dalla frequenza di clock).
Dualmente, vi è stato un restringimento del tempo a disposizione per le reti a valle del filtro per poter trasmettere il dato filtrato al PC, previo invio dell' header, così come previsto dal protocollo.
Con questa nuova situazione, l' architettura di elaborazione, ottimamente funzionante nel caso a 10 MHz, è risultata non più adeguata in quanto,

alla frequenza di 80 MHz, il filtro genera in uscita un nuovo dato ogni 6.4 µs e ciò si scontra con le tempistiche richieste sia dal chip FTDI che con quelle della macchina a stati che lo pilotava. Quest' ultima, infatti, non permetteva di effettuare degli invii in veloce successione a causa del suo metodo di funzionamento, che faceva uso di wait states statici: un certo numero di cicli di ritardo inseriti in una macchina a stati funzionante a 10 MHz generano un effetto diverso rispetto a quanto accadrebbe se tale macchina fosse fatta funzionare ad 80 MHz, ed in questo caso l' effetto è quello di non riuscire a trasferire correttamente i dati verso il chip di interfaccia USB, a causa del mancato rispetto dei timing specificati dal costruttore di tale integrato.

Nelle figure seguenti vengono mostrati i segnali di interesse nei due casi, a 10 ed 80 MHz. Si tenga conto che la durata del segnale *ftdi_txe* non rispecchia la durata reale di tale segnale, in quanto è il frutto di una semplice simulazione del chip FTDI implementata nel testbench. In realtà la durata del segnale può variare di volta in volta in maniera dipendente da ciò che accade sul bus USB. Tale simulazione è però stata utile per verificare il funzionamento "logico" di quanto implementato: la verifica sperimentale successiva ha poi confermato o smentito gli esiti delle varie simulazioni.

Il segnale *tx_usb_en* identifica, sul suo fronte di salita, l' istante in cui un nuovo dato è stato elaborato dal blocco di filtraggio ed è pronto per essere inviato al PC.
Come è evidente dal confronto di tale segnale con il segnale di clock *clk10m_out* (a 10 MHz) ed il segnale *qout(2)* (che è il segnale SYNC), il sistema ha disposizione tutto il tempo necessario per effettuare la trasmissione dei sei byte (bus *usb_dout*), soddisfacendo le tempistiche dell' FTDI.
Infatti il segnale *tx_usb_en* non ha un nuovo fronte di salita (cioè non c'è un nuovo dato da trasmettere) prima che il ciclo di trasmissione termini: ai sei impulsi di scrittura per il dispositivo FTDI corrispondono altrettanti impulsi della linea TXE, interpretabili in questo caso come conferma di corretta trasmissione (acknownledge).
Nella simulazione ad 80 MHz (si noti il segnale clk10m_out rispetto a clk6m_in, clock a 6MHz fornito all' FPGA dall' FTDI) è evidente che a fronte di 6 impulsi di write (*ftdi_wr*), vi sono solo 3 impulsi di "acknownledge" (*ftdi_txe*). Ciò ovviamente significa che solo 3 byte su 6 vengono effettivamente trasmessi al PC. Considerando che per ogni dato a 16bit (2 byte) devono essere trasmessi anche quattro byte di

intestazione, per un totale di 6 byte, è evidente come sia impossibile effettuare la trasmissione diretta dei dati filtrati. Pertanto l' approccio "diretto" è stato immediatamente scartato.

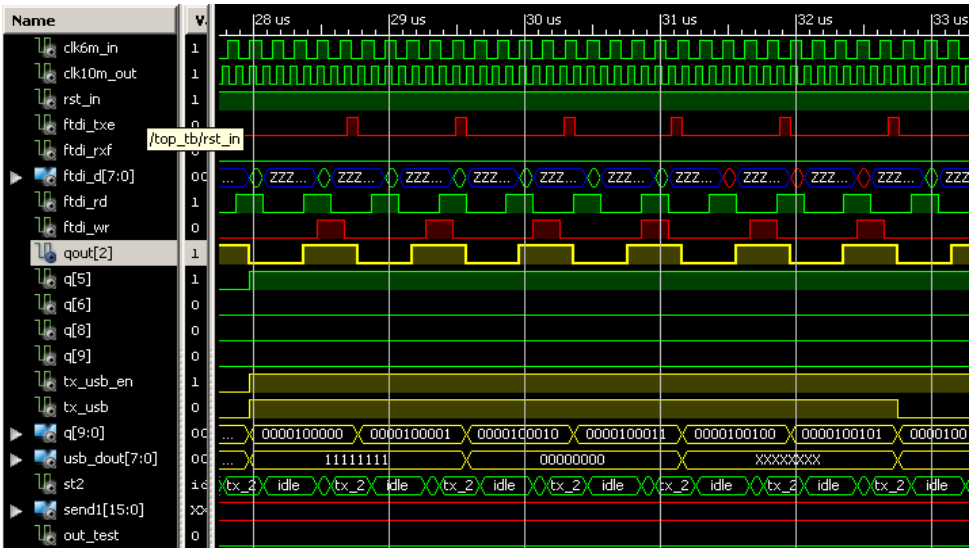


Figura 22: simulazione ciclo di trasmissione a 10 MHz

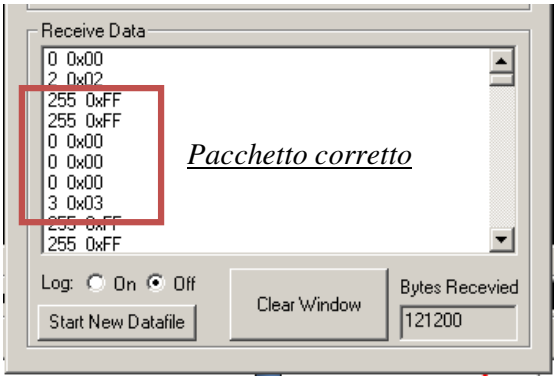


Figura 23: corretta ricezione dati a 10 MHz

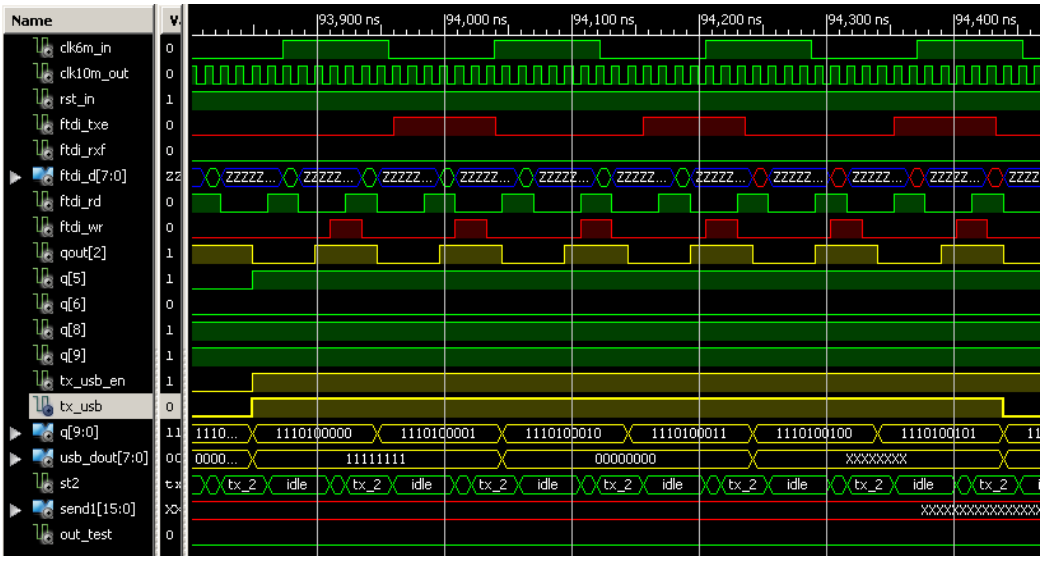


Figura 24: simulazione ciclo di trasmissione a 80 MHz

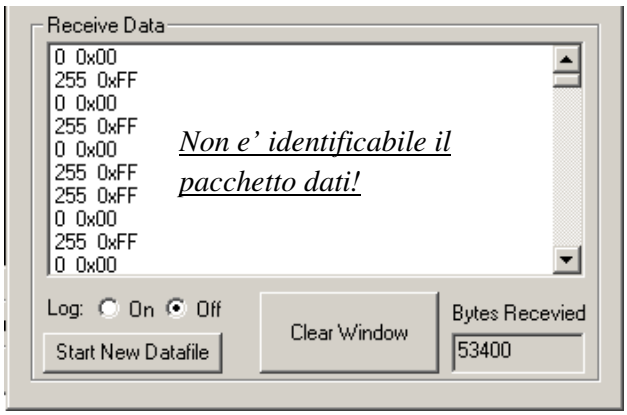


Figura 25: non corretta ricezione dati a 80 MHz

Seconda soluzione adottata

Alla luce di quanto evidenziatosi durante lo sviluppo della prima soluzione, si è reso necessario un approccio più radicale al problema.

Basandosi sulle tempistiche specificate nel datasheet del chip FTDI, si è optato per una riscrittura della macchina a stati finiti che si occupa dell' interfacciamento con tale chip e della generazione del pacchetto da inviare al computer: invece di calcolarsi il numero di cicli di attesa ed inserire dei wait states fissi tra un abilitazione del segnale di write e la successiva, per garantire la conformità alle tempistiche del chip FTDI, si è fatto in modo di monitorare la linea $\overline{\mathbf{TXE}}$, che rappresenta lo stato del bus di trasmissione verso il chip (1: chip occupato, non è possibile trasmettere il dato, 0: chip non occupato, linea libera e possibilità di trasmettere un dato), ed attivare la linea di write non appena il chip manifestasse lo stato "non occupato".

Successivamente la macchina a stati avrebbe atteso l' attivazione della linea $\overline{\mathbf{TXE}}$ e la sua successiva disattivazione (strobe), dove nel tempo trascorso fra l' attivazione e la disattivazione, il chip avrebbe trasferito il dato al PC, ripetendo poi di nuovo il ciclo con un nuovo byte da trasferire.

Sulla carta, ed alla prova del simulatore, le tempistiche erano perfettamente rispettate.

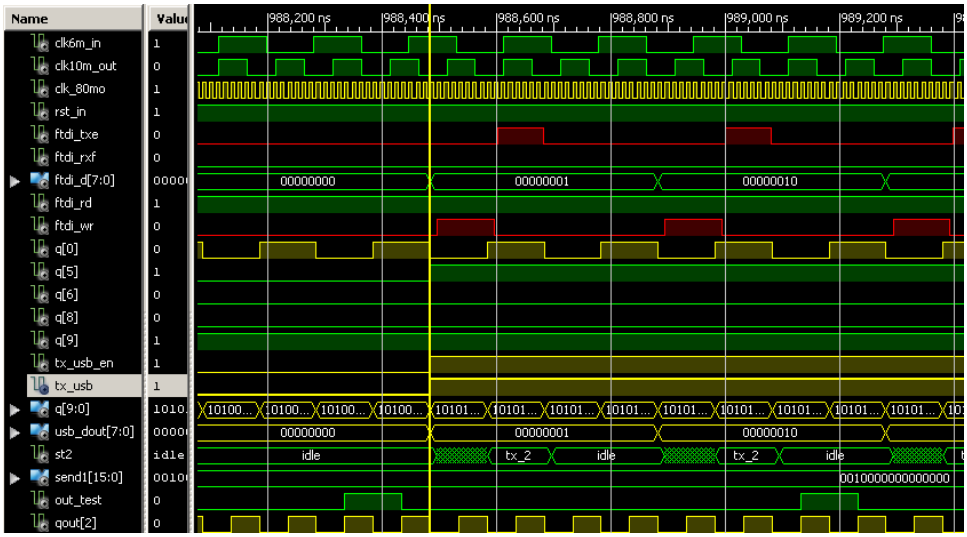


Figura 26: dettaglio del ciclo di trasmissione ad 80 MHz, versione 2

Nella precedente figura è evidenziato come, con un clock ad 80 MHz si sia riuscito ad ottenere (in apparenza) una corretta temporizzazione: nelle fasi di test sono stati sostituiti i byte "veri", cioè quelli di header e di dato, con una sequenza incrementale da 1 a 6 (bus *usb_dout*), per identificarli sulla applicazione di debug e come si vede, i cicli di scrittura verso l' FTDI (strobe di *ftdi_wr* e acknownledgement di *ftdi_txe*) vengono correttamente effettuati.

Alla prova dei fatti, però, era evidente come vi fossero delle pesanti perdite di dati, ed il flusso ricevuto dal PC era pressoché identico a quello mostrato nella figura 25, cioè profondamente corrotto.

Un primo tentativo di ottimizzazione di questa soluzione è stato quello di sfruttare l' intero periodo nel quale un byte veniva posto sul bus dalla rete logica connessa alla base dei tempi (la ROM da 5 bytes): invece di iniziare il ciclo di scrittura a metà del tempo di byte, si è implementato un meccanismo per eseguirlo un ciclo di clock dopo che il dato venisse prodotto dal filtro (come si vede in figura 26 il segnale *ftdi_wr* si attiva poco dopo il fronte di salita del segnale *tx_usb_en*) e, contemporaneamente, è stato allungato il periodo di presenza sul bus dei singoli byte, in modo da poter aver a disposizione il tempo necessario per soddisfare le tempistiche imposte, necessarie al corretto trasferimento di dati verso l' FTDI.

Nonostante queste ulteriori ottimizzazioni, verificate come potenzialmente valide al simulatore, le perdite persistevano.

Si è proceduto quindi all' analisi mediante l' oscilloscopio dei vari segnali di interesse: sono stati esposti all' esterno dell' FPGA sui pin del connettore header della development board, i segnali interni *ftdi_txe* e *ftdi_wr*, congiuntamente al segnale *tx_usb_en* (in figura 27 sono mostrati i primi due).

A questo punto era evidente che nel periodo del segnale *tx_usb_en*, cioè il periodo disponibile per effettuare la trasmissione di 6 byte, venivano mediamente eseguite 3 sole scritture verso l' FTDI.

Inoltre, era ben evidente il fatto che la linea $\overline{\textbf{TXE}}$ non sempre rimaneva alta per il medesimo tempo, cioè il chip non impiegava sempre lo stesso tempo per trasferire i dati al PC: dopo un certo intervallo di tempo, difatti, la linea rimaneva alta per periodo superiore al millisecondo, non permettendo la trasmissione di un notevole numero di pacchetti.

Questo problema era dovuto al fatto che il chip FTDI al suo interno è dotato di un

buffer FIFO con una certa profondità e per la cui scrittura è necessario mediamente più di 1 us. Una volta riempito questo buffer, il chip impiega numerosi cicli temporali per trasferirlo in blocco verso il PC: in questo modo sia per il tempo non nullo necessario per trasferire l' intero pacchetto al computer, sia per le latenze introdotte dal driver lato PC (che per sua natura esegue il prelevamento dei dati dalle periferiche connesse al computer mediante polling), il ritardo introdotto era notevole.

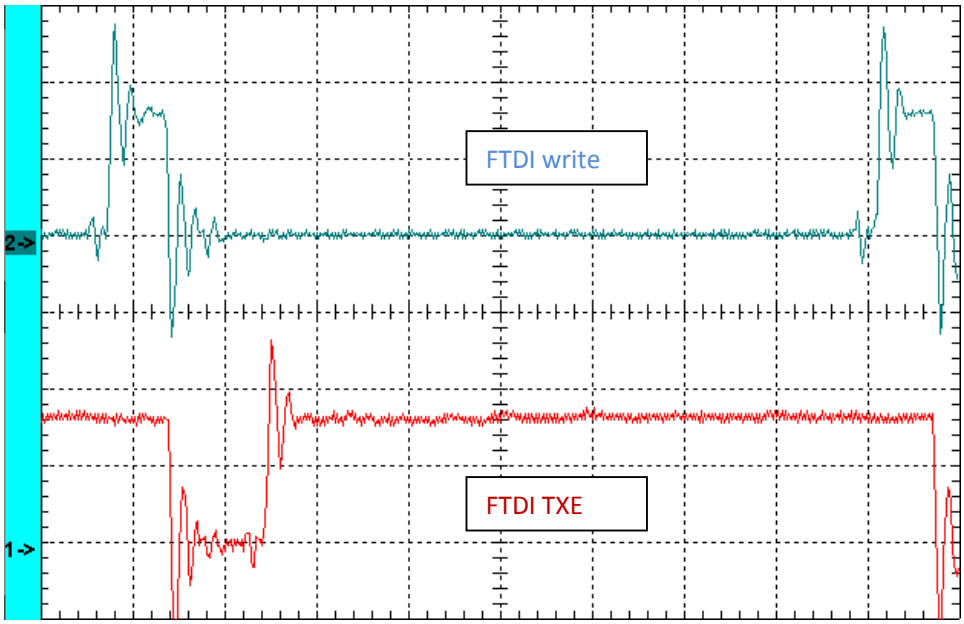


Figura 27: oscillogramma dei segnali ftdi_wr e ftdi_txe durente la scrittura di un byte, il periodo del segnale TXE è pari a circa 1us

Durante lo sviluppo di questa seconda soluzione è stato comunque effettuato un passo in avanti, in quanto la macchina a stati di gestione della trasmissione dei dati è stata resa indipendente dalla frequenza di lavoro del sistema, avendo fatto a meno dei wait states "statici" che venivano introdotti nella vecchia versione e che sarebbe stato necessario ricalcolare e modificare per ogni variazione della frequenza di lavoro.

Il miglioramento introdotto da questa architettura è stato riscontrabile dal suo perfetto funzionamento in tutto il range di frequenze di clock da 10 MHz a 45 MHz, con la sola modifica del moltiplicatore di clock nel blocco DCM e nessun altro

aggiustamento al codice, pur non funzionando poi alla frequenza target di 80 MHz per i problemi sopra esposti.

Parallelamente ai vari tentativi di ottimizzazione, dato che la macchina a stati per il solo interfacciamento con l' FTDI era stata giudicata valida, è stato effettuato un test per verificare quale fosse l' effettiva velocità massima di trasferimento dati ottenibile con il dispositivo FTDI.

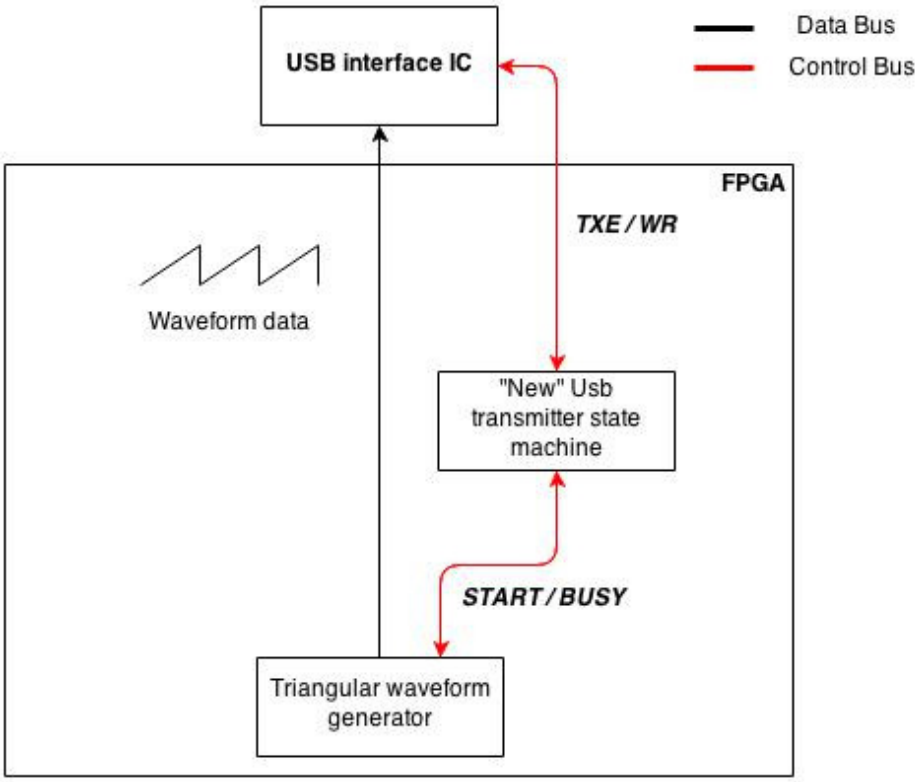


Figura 28: architettura sistema di test per determinazione della massima velocità di trasferimento dati

Per eseguire questo test è stato sviluppato un codice VHDL apposito, attorno alla nuova macchina a stati, modificata con l' aggiunta di un segnale di busy e di un segnale di start, utilizzabili dagli altri componenti del sistema all' interno dell' FPGA per la sincronizzazione: il segnale busy indica se sta avvenendo una trasmissione dati e ci si deve porre in attesa; il segnale start, qualora la macchina a stati non fosse occupata, avvia la trasmissione di un nuovo byte.

Non è stato possibile utilizzare direttamente la linea $\overline{\textbf{TXE}}$ per questo scopo, in quanto la trasmissione dati inizia nel momento in cui la linea write va a zero: la linea

$\overline{\text{TXE}}$ si attiva dopo un tempo non trascurabile a frequenze alte, e ciò potrebbe portare a comportamenti erronei del sistema. (Si veda il diagramma delle temporizzazioni dell' FTDI in scrittura, in particolare il periodo di tempo indicato come T11).

Accanto alla macchina a stati che si occupa dell' interfacciamento con l' FTDI ne è stata implementata un' altra (interfacciata ovviamente con la prima) per l' invio di un dato di test (una forma d' onda triangolare a dente di sega) in modo sincrono con l' FTDI, cioè un nuovo dato da trasmettere veniva creato solo all' avvenuta trasmissione del dato precedente (controllando la linea di busy e attivando la linea di start di conseguenza, ovviamente ponendo sul bus i dati da inviare prima di dare il segnale di start).

È stato quindi verificato che il massimo bit rate ottenibile dal dispositivo, con l' uso del driver di emulazione seriale, è pari a 7.5 Mbyte al secondo.

Tale bit rate è stato posto come obiettivo finale, ovverosia l' architettura definitiva per essere ritenuta soddisfacente, quando impostata sulla banda di filtraggio a 100 kHz, avrebbe dovuto comunicare i dati filtrati, ovviamente senza perdite di dati, a tale bit rate.

Visto il parziale insuccesso della soluzione appena illustrata si è deciso di modificare ancora più profondamente l' architettura di elaborazione: come si è visto il blocco filtrante è una macchina a stati "sincrona" (rispetto al sincronismo SYNC), mentre la sezione che gestisce l' USB è essenzialmente "asincrona", perciò si è deciso di inserire un buffer fra le due sezioni in modo da far interagire correttamente le due parti, rendendo più indipendente la sezione asincrona da quella sincrona.

Terza soluzione adottata

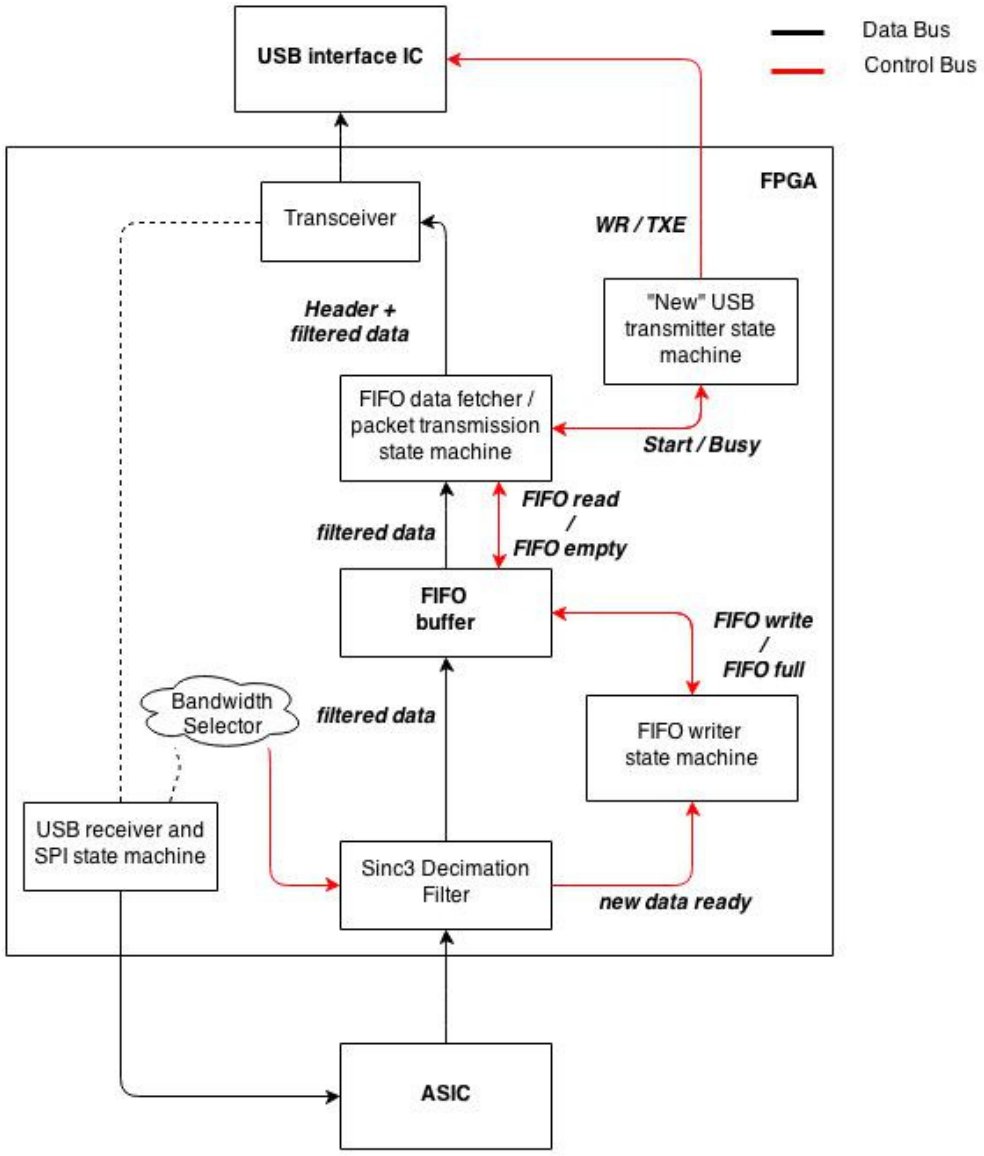


Figura 29: architettura definitiva della sezione di trasmissione dati filtrati

Dopo aver individuato le maggiori problematiche nei due precedenti tentativi di soluzione si è deciso di inserire un buffer fra il blocco di filtraggio e la macchina a stati dedicata alla trasmissione USB, in modo da rendere quest' ultima più indipendente dalle tempistiche relative all' acquisizione ed al filtraggio dei dati e contemporaneamente ovviare al problema legato ai tempi morti introdotti dall' FTDI, prefiggendosi lo scopo di ottenere il massimo throughput.

La forma di bufferizzazione scelta per lo scopo è quella di tipo FIFO (First-In First-Out). Questo tipo di buffer è sostanzialmente una memoria a due porte, condivise fra due utilizzatori: da una porta un utilizzatore può scrivere uno o più dati, finché la memoria non si riempie, mentre dall' altra porta un secondo utilizzatore può prelevare dati finché ce ne sono. Le due operazioni di lettura e scrittura possono essere non sincronizzate, come in questo e nella maggior parte dei casi.
Normalmente tale tipo di bufferizzazione viene utilizzato in tutti quei casi in cui vi è una diversa velocità del flusso dei dati fra parti del sistema.
Un buffer di questo tipo è stato posto fra il filtro decimatore e la macchina a stati di trasmissione: l' integrazione del buffer ha richiesto una profonda modifica dell' architettura di elaborazione: è stato necessario implementare due ulteriori macchine a stati finiti per poterlo gestire, una dedicata al prelevamento del dato dal blocco di filtraggio e al suo corretto immagazzinamento, l' altra dedicata al prelevamento del dato dal FIFO e alla successiva trasmissione, assieme ai byte di header.

La libreria di componenti Xilinx fornisce un blocco FIFO personalizzabile per quanto riguarda la dimensione del dato da memorizzare ed il numero totale di dati memorizzabili (profondità del buffer) oltre che per quanto riguarda i segnali di controllo utili per la sincronizzazione, sia dello scrittore del buffer (nel nostro caso il blocco filtrante) sia del lettore (il blocco di trasmissione USB): a questo proposito vi sono segnali di buffer pieno, buffer vuoto, buffer overflow (errore in scrittura), buffer underflow (errore in lettura).

Tale customizzazione, e conseguente generazione del blocco da istanziare nel codice VHDL, viene eseguita attraverso un apposito wizard a corredo dell' ambiente di sviluppo.
In questo progetto si è optato per un buffer di 8192 kword, ciascuna di 16 bit.

In riferimento a quanto detto in precedenza, la prima delle due macchine a stati finiti, indicata nello schema a blocchi come "FIFO writer state machine" si occupa di attendere un nuovo dato dal blocco filtrante, evento segnalato come in precedenza dalla transizione di un segnale apposito (in pratica è il segnale *tx_usb_en* rinominato), e trasferire poi tale dato al buffer: ciò avviene con successo, effettuando uno strobe sulla linea di write, collegata al buffer, per la durata di un ciclo di clock,

qualora il buffer FIFO non fosse già pieno di dati in attesa di invio (situazione segnalata dallo stesso buffer mediante un filo dedicato allo scopo). Il successo dell' operazione di scrittura viene segnalato mediante un segnale di acknownledge dal FIFO.

La seconda macchina a stati, indicata nel diagramma con il blocco denominato "FIFO data fetcher/packet data transmission state machine", racchiude in se più funzionalità: si interfaccia con la macchina a stati di trasmissione USB, implementata e descritta in precedenza, e con il bus dati diretto verso l' FTDI.
Per prima cosa essa pone in sequenza su tale bus i byte di header, facendo partire la trasmissione dati mediante strobe della linea start, connessa alla macchina a stati di trasmissione, ed attendendo la fine di tale trasmissione, monitorando la linea busy. Appena viene segnalato il termine della trasmissione, e ripetuto il ciclo per tutti e 4 i byte di header, viene eseguito il prelevamento di un dato (fetch) precedentemente immagazzinato nel FIFO dall' altra macchina a stati già descritta: il prelevamento avviene se il buffer non è vuoto (situazione anche in questo caso indicata dallo stesso buffer mediante un segnale dedicato allo scopo) eseguendo uno strobe sulla linea read, collegata al buffer stesso, per la durata di un ciclo di clock. Il successo dell' operazione di lettura viene segnalato mediante un segnale di acknownledge dal FIFO.
A quel punto, viene connessa la parte alta del bus di uscita del buffer FIFO (bit da 15 a 8) con il bus diretto verso l' FTDI ed eseguito un ciclo di trasmissione, al cui termine viene di nuovo commutato il bus, connettendo questa volta i bit da 7 a 0 del bus di uscita del buffer FIFO, ed eseguito l' ultimo ciclo di trasmissione.
Questo intero ciclo, compresa la trasmissione dei byte di header, viene ripetuto finché il buffer non si svuota (condizione che non si verifica mai in quando l' ASIC invia sempre nuovi dati). Nel caso in cui si verificasse tale condizione la macchina attenderebbe che un nuovo dato fosse scritto nel buffer, ricominciando il ciclo appena descritto.

Il sistema precedentemente descritto permette quindi di rendere più indipendente la sezione di trasmissione dati rispetto le tempistiche rigidamente imposte dalla base dei tempi con la quale è sincronizzata la sezione di filtraggio.

Riguardo la macchina a stati di ricezione, essa non è stata modificata, se non per l' aggiunta di un segnale, utilizzato per la sincronizzazione con la macchina a stati di trasmissione, di modo che quando viene attivata la prima, la seconda non tenti di eseguire scritture sul bus. La macchina a stati di ricezione è inoltre pilotata sempre dal clock a 10 MHz, dato che non vi è nessuna necessità per la quale debba funzionare alla velocità maggiore.

È stato da subito evidente che questa soluzione ha portato notevoli migliorie alla situazione: utilizzando l' applicazione di test che permette di visualizzare i singoli byte ricevuti, era evidente come il protocollo veniva rispettato, ed i dati trasmessi arrivavano a destinazione senza corruzione. Era inoltre possibile, dato che i pacchetti potevano essere correttamente interpretati dal software su PC, visualizzare una forma d' onda su tale software di acquisizione: ciò ha evidenziato che nonostante tutto, ancora sussistevano alcuni problemi.

Nel codice sviluppato sulla scheda DLP-FPGA, era stato implementato, sin dal prima soluzione tentata, un blocco che generava un' onda a dente di sega posto a valle del blocco filtrante, con le stesse temporizzazioni del dato filtrato (cioè il dato da trasmettere veniva aggiornato con la stessa cadenza, sia che fosse il dato uscente dal filtro, sia che fosse quello dell' onda triangolare di test).
Tale blocco, che una volta azionato dall' interfaccia su PC escludeva il bus proveniente dal filtro e si sostituiva ad esso, serviva appunto per verificare il corretto trasferimento dei dati in quanto era difficile correlare "ad occhio" i valori dei byte con le forme d' onda (seppur di test) filtrate.
Invece, l' output di un contatore free-running a 16 bit, cioè la rampa, era più facilmente verificabile in quanto, anche con un singolo dato mancato, la variazione sulla rampa visualizzata era sempre evidente sotto forma di glitch.

La prima implementazione dell' architettura sopra descritta presentava, purtroppo, numerosi ed evidenti glitch, come mostrato nella figura 30.

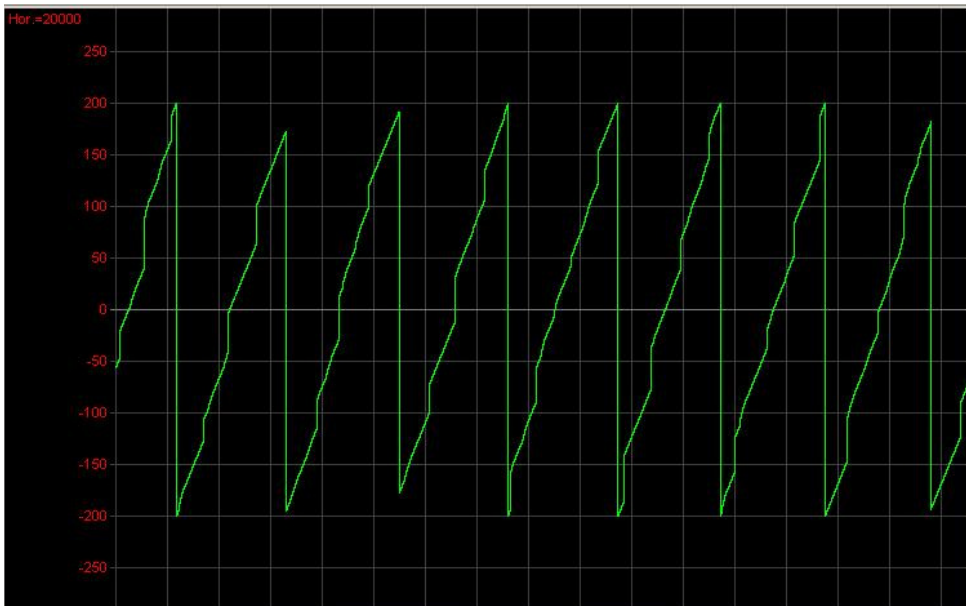


Figura 30: segnale di test con evidente perdita di dati.

Analizzando all' oscilloscopio i segnali di buffer overflow (attivo alto), proveniente dal FIFO, ed il segnale di busy (anch' esso attivo alto), proveniente dalla macchina a stati di trasmissione USB, era evidente come il buffer veniva riempito molto più velocemente di quanto riusciva ad essere svuotato: infatti si notava che proprio in corrispondenza dell' attivazione del segnale di buffer overflow, la macchina a stati per la trasmissione usb era in attesa della fine di un trasferimento dati verso il PC ad opera dell' FTDI. (figura 31)

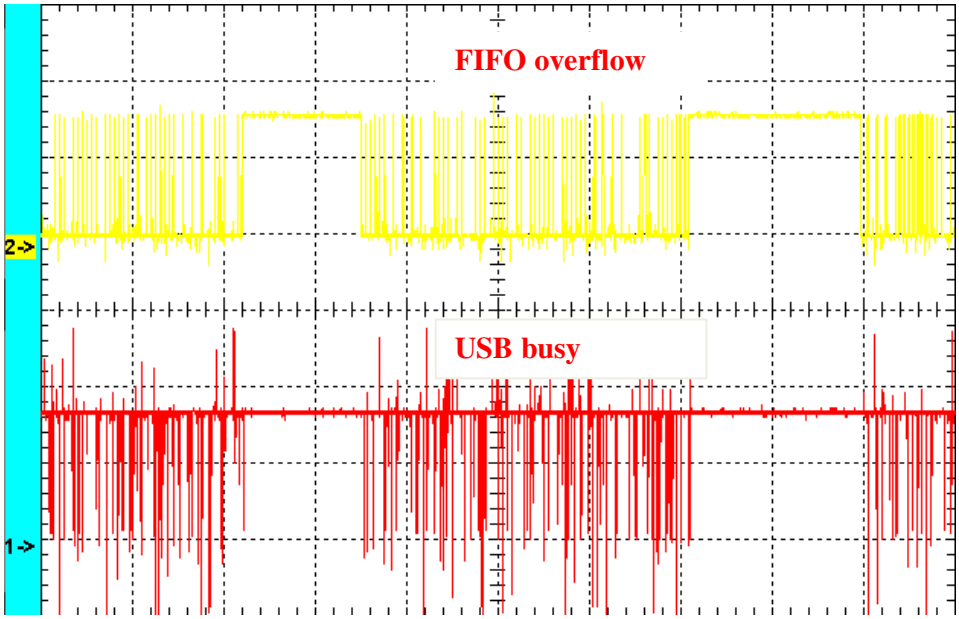


Figura 31: oscillogrammi dei segnali fifo_overflow e usb_busy nel caso con perdite di dati

Dal punto di vista della tempistica, un nuovo dato da trasmettere veniva generato ogni ~6.4 µs (periodo del segnale di dato pronto da immagazzinare nel FIFO), mentre per trasmettere un dato verso il PC (header + dato) erano necessari mediamente più di 1 µs a byte, come precedentemente affermato, ed era evidente, sempre da misure all' oscilloscopio, che l' FTDI rimaneva occupato per periodi molto lunghi (effetto combinato della latenza del driver su PC e del trasferimento "in blocco" dei dati memorizzati nella memoria interna dell' FTDI): ciò ovviamente comportava l' accumulo di dati non trasmessi nel FIFO.

La tesi sopra esposta è stata poi avvalorata lasciando funzionare per un po' il sistema: all' "accensione" la forma d' onda presentava la pesante deformazione illustrata nella figura precedente, ma man mano che il tempo passava, (minuti o secondi, senza precisa periodicità e quindi con totale aleatorietà) la situazione “migliorava”, come mostrato in figura 32.

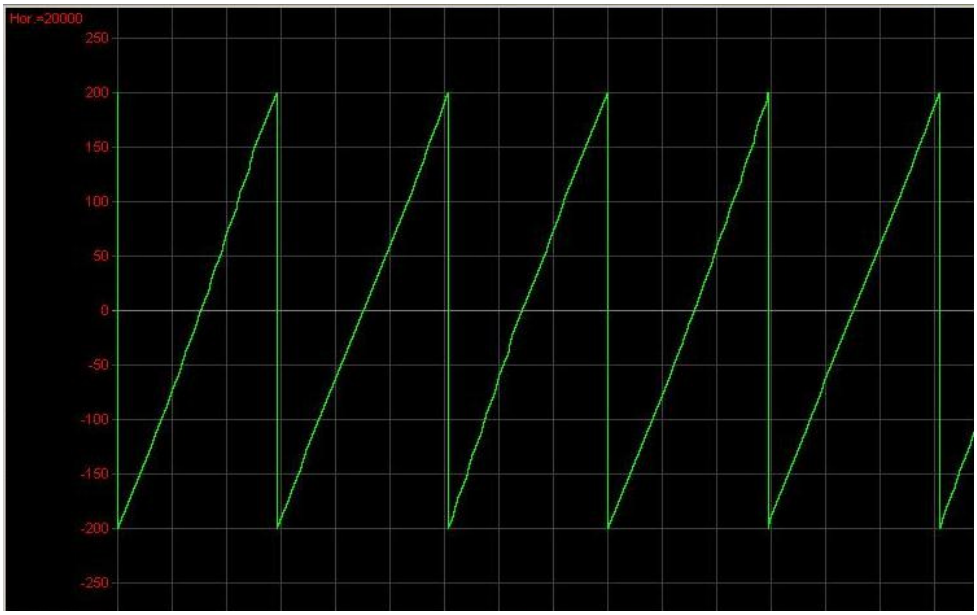


Figura 32: segnale di test con perdita di dati meno evidente, ma comunque presente.

In questo caso, si evidenziavano sull' oscilloscopio minori tempi morti dell' FTDI, e quindi minori saturazioni del FIFO, ma era comunque evidente che alcuni dati venivano persi, cioè non venivano memorizzati nel buffer perché nell' istante in cui era necessario memorizzarli, il buffer FIFO era già saturo. Inoltre capitava poi che, da questa situazione "migliore", si ritornasse a quella peggiore, con cadenza imprevedibile e ciò era dovuto esclusivamente alle latenze del driver sul PC.

Fortunatamente però questo problema è stato velocemente risolto: il pacchetto dati era effettivamente strutturato in modo poco bilanciato perché per ogni dato da trasmettere era necessario inviare anche i 4 byte di header.
Ciò costituiva un notevole overhead, ed un eccessiva ridondanza.
Perciò, grazie anche alla flessibilità del software di acquisizione su PC, si è deciso di cambiare il formato del pacchetto dati, trasmesso dal device al computer, mantenendo i 4 byte di sincronizzazione ma allargando la sezione dati, passando da uno a 64 valori consecutivi trasmessi, per un totale di 132 byte, abbattendo notevolmente l' overhead e permettendo così la trasmissione fluida e senza glitch dei

dati.

È stato inoltre precauzionalmente allargato il buffer FIFO, fino alla dimensione precedentemente indicata (circa 8 kword), in modo da avere sufficiente spazio anche nelle eventualità in cui la trasmissione dovesse essere interrotta per un periodo prolungato, sempre nell' ordine dei millisecondi.

Al fine di verificare il funzionamento corretto, e prolungato, del dispositivo, è stato eseguito un ulteriore test, consistente nell' acquisizione dei dati mediante il software su computer, per un certo periodo di tempo. Il programma ha permesso di salvare i dati in un file binario, interpretato in Matlab da un apposito script. Riordinando poi in vettori separati i valori rappresentanti ciascuna rampa, e confrontandoli fra loro (mediante un altro script), è stato possibile verificare la consistenza dei dati ricevuti (figura 33).

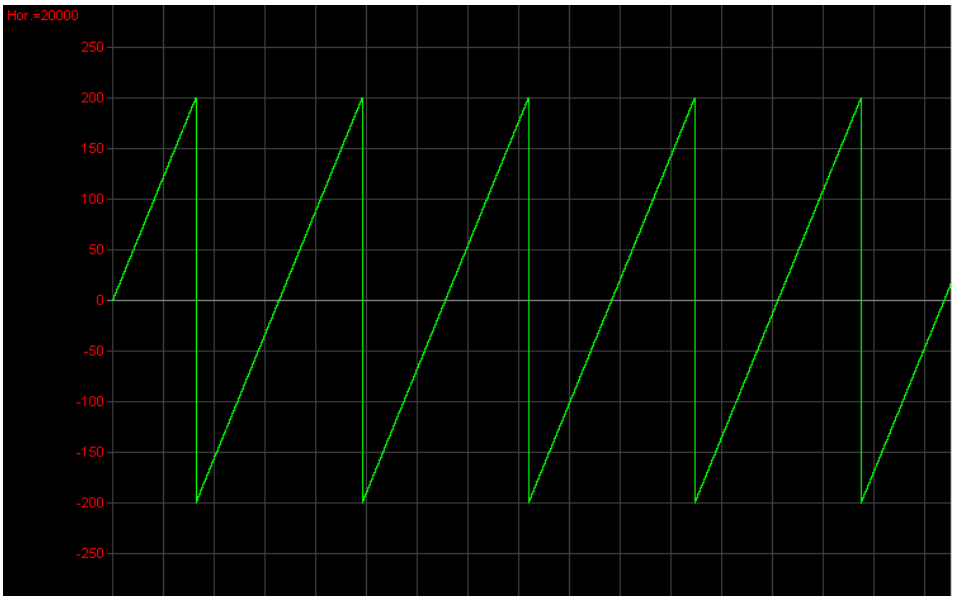


Figura 33: segnale di test correttamente ricevuto.

Porting sulla scheda target

Terminato la fase di sviluppo e debugging della maggior parte del codice sulla development board, si è proceduto ad effettuare il porting sulla scheda target, dotata della vera ASIC e non più del blocco simulativo.

Per fare ciò è stato necessario rimappare i pin di I/O, precedentemente assegnati, in quanto l' FPGA usata nella scheda target aveva un diverso package rispetto a quella della development board, ed inoltre è stato modificato il DCM in modo che generasse correttamente i segnali di clock a partire da una sorgente a 12 MHz e non più a 6.

Effettuati i vari test di funzionamento, è stata poi implementata una funzione particolare che permette di commutare la frequenza di clock delle macchine a stati descritte in precedenza, attraverso un multiplexer apposito, comandato da uno dei parametri inviati dal software su PC, di modo che il sistema possa funzionare sia a 10 Mhz che a 80 Mhz, permettendo così di avere due diverse bande di filtraggio con lo stesso parametro di oversampling e due diverse frequenze di campionamento a livello della ASIC.
Ciò significa che se il sistema funziona a 10 MHz, le bande di filtraggio selezionabili sono quelle elencate in precedenza, nel paragrafo in cui viene descritto il blocco filtrante, mentre se il clock è di 80 Mhz le bande selezionabili sono 100kHz, 50 kHz, 12.5 kHz e 6.25 kHz.

Porting su nuova scheda di sviluppo

Paralleamente a quanto descritto in precedenza, è stata svolta un attività legata al porting del codice VHDL su una nuova piattaforma hardware, dotata di FPGA Spartan 3 con maggiore capacità e soprattutto dotata di chip di interfacciamento USB FT2232H, che a differenza del chip impiegato nell' hardware descritto in precedenza, permette trasferimenti di dati a maggior bitrate. È stato necessario modificare l' architettura di elaborazione sia per quanto riguarda l' interfacciamento (ovviamente i nuovi chip, FPGA e FTDI, hanno una diversa piedinatura) sia per quanto riguarda la gestione del clock: in questa scheda, essendo presente una memoria DDR2, che però non viene utilizzata per il progetto, è necessario avere una sorgente di clock più

stabile rispetto al risuonatore ceramico e a più alta frequenza: è perciò presente un oscillatore al quarzo con frequenza di 66.666 MHz (la frequenza di funzionamento della memoria DDR è di 133 MHz, la frequenza dell' oscillatore è quindi facilmente moltiplicabile per ottenere questa frequenza particolare). Per ciò che riguarda il progetto, è stato quindi necessario modificare il blocco DCM in modo che elaborasse il segnale di clock a 66.666 MHz per poter infine fornire i clock utilizzati nell' architettura di elaborazione.

Purtroppo, per alcune problematiche legate al diverso interfacciamento del dispositivo FTDI con l' FPGA (i ruoli del device A e del device B sono scambiati) ed alla diversa signature presente nella memoria Flash collegata al chip per la USB, il software su computer non riusciva a collegarsi al dispositivo.

Per tanto i test effettuati si sono limitati alla verifica della correttezza dei dati trasferiti utilizzando il programma che permette di visualizzare i dati "grezzi" provenienti dal bus USB.

Capitolo 7

Conclusioni

Grazie alle funzionalità implementate e alle modifiche apportate al codice VHDL preesistente, è stato possibile ampliare le capacità del dispositivo, permettendogli di poter acquisire ed elaborare segnali con banda fino a 100 kHz.
Inoltre si è predisposta l' architettura di elaborazione per ulteriori e future estensioni delle funzionalità.

Nei vari capitoli si è cercato di fornire una panoramica generale, ma non generica, del sistema completo, focalizzandosi sulle tematiche coinvolte nelle fasi di progettazione. Si è poi cercato di illustrare il percorso che ha portato alle scelte di progetto effettivamente implementate, dando una descrizione astratta (ma aderente a quanto effettivamente realizzato) delle soluzioni architetturali vagliate e poi applicate.

Dal punto di vista didattico, potendo lavorare su un sistema reale, è stato possibile confrontarsi con problematiche di progetto sia teoriche che pratiche e tangibili, acquisendo capacità di debugging e conoscenze più approfondite riguardo i dispositivi FPGA: il percorso compiuto attraverso le diverse soluzioni al problema progettuale hanno permesso di acquisire familiarità sia con gli strumenti messi a disposizione dalla azienda produttrice della FPGA (ambiente di simulazione in primis) sia con questioni progettuali rilevanti, come la gestione di segnali di clock relativamente veloci e la presenza simultanea, nella stessa architettura di elaborazione, di parti strettamente sincrone (rispetto ad una certa base dei tempi) e di parti asincrone, con le relative problematiche evidenziate in precedenza.

Bibliografia

[1] “Spartan-3 Generation FPGA User Guide: Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families UG331 (v1.5)”, Xilinx Inc., January 2009

[2] “Combining the ADS1202 with an FPGA Digital Filter for Current Measurement in Motor Control Applications”, M. Oljaca, T. Hendrick, Texas Instruments Incorporated, June 2003

[3] “DLP-FPGA USB - FPGA Module Datasheet”, DLP Design, Version 1.4, November 2010

[4] “FT2232D Dual Usb To Serial Uart/Fifo IC Datasheet”, Future Technology Devices International Ltd, Version 2.05, 2010

[5] “Protocolli di trasmissione USB FTDI – PC”, F. Thei, Versione 3, Novembre 2013