UNIVERSITÁ DEGLI STUDI DI PISA
LAUREA SPECIALISTICA IN INGEGNERIA ELETTRONICA

Progetto di un sistema per il filtraggio numerico di segnali audio

AUTORI

Fabio PANIA
Giuseppe PASETTI

Indice

Introduzione

I segnali audio, grazie al fatto che hanno una banda molto ridotta (20Hz-20kHz), sono molto semplici da convertire in segnali numerici. Se a questa semplicità di conversione aggiungiamo la grande flessibilità del filtraggio numerico, dovuta al fatto di poter utilizzare sistemi programmabili, ci si accorge delle grandi potenzialità che vengono offerte dall'elaborazione numerica dei segnali audio.
Sempre più spesso al giorno d'oggi le operazioni di filtraggio ed equalizzazione dei segnali audio vengono affidate a dispositivi digitali.
Il lavoro presentato in questa relazione va proprio in questa direzione; lo scopo che si è voluto ottenere è infatti quello di filtrare un qualsiasi segnale audio.
Per quanto riguarda il tipo di filtro la scelta è stata quella di utilizzare un filtraggio FIR invece di un filtraggio IIR in quanto il primo porta notevoli vantaggi in termini di semplicità implementativa.
Di seguito vengono riportate le specifiche concordate per il progetto.

Specifiche Funzionali:

- Ricevere in ingresso un segnale audio stereo;
- Poter selezionare il tipo di filtro agendo su opportuni pulsanti;
- Dare in uscita un segnale audio stereo opportunamente filtrato.

Specifiche implementative:

- Convertire un segnale audio stereo in digitale scegliendo un opportuno ADC tra quelli disponibili sul mercato;
- Effettuare un filtraggio FIR utilizzando il dispositivo FPGA Ciclone EP1C6Q240C8 di Altera;
- Avere la possibilità di variare il tipo di filtro utilizzando l'hardware messo a disposizione dalla scheda di sviluppo UP3 di SLS Corp.;
- Riconvertire i segnali digitali in uscita in segnali analogici scegliendo un opportuno DAC tra quelli disponibili sul mercato;
- Testare il funzionamento del sistema utilizzando la scheda di sviluppo UP3 di SLS Corp.

La presente relazione, che ha lo scopo di descrivere l'architettura, il funzionamento e il test del sistema progettato, si costituisce di tre capitoli il cui contenuto è di seguito esplicitato.
Il primo capitolo è dedicato alla descrizione del filtraggio numerico di segnali, in particolare al filtraggio FIR e alla valutazione dei suoi vantaggi e svantaggi.

Il secondo capitolo è diviso in due parti, una prima parte è incentrata sulla *scheda aggiuntiva* e sui suoi componenti, mentre la seconda parte è interamente dedicata alla descrizione dell'architettura implementata su FPGA.

Il terzo capitolo descrive le verifiche effettuate sul sistema in maniera da garantirne il corretto funzionamento.

Capitolo 1: Filtraggio numerico

Un filtro è un sistema progettato per alterare il contenuto spettrale di segnali in ingresso secondo un determinato profilo.

Un filtro selettivo ideale lascia passare inalterate alcune componenti frequenziali, mentre elimina completamente le altre frequenze. Il range di frequenze che lascia passare inalterate viene chiamato bandapassante, mentre il range di frequenze che vengono eliminate viene chiamata banda reiettata. Il modulo della risposta in frequenza ($|H(\omega)|$)è pari a 1 in banda passante e pari a 0 fuori. In figura 1.1 sono rappresentati i moduli della risposta in frequenza di 4 filtri: passa basso, passa alto, passa banda ed elimina banda. Partendo da questi 4 filtri è possibile, combinandoli tra di loro, ricavare qualunque altro tipo di filtro.

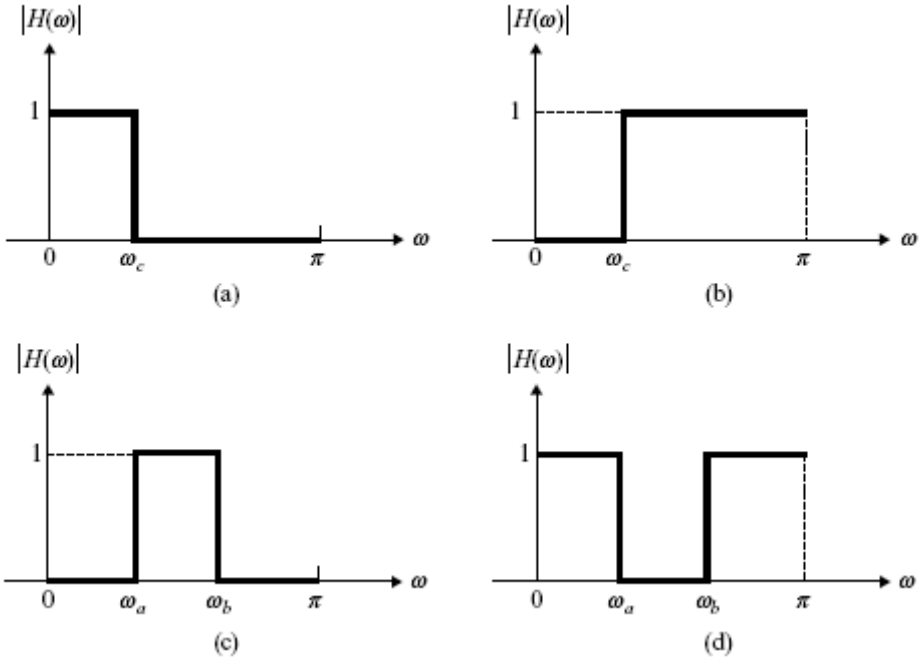


Fig. 1.1: (a) $|H(\omega)|$ passa basso; (b) $|H(\omega)|$ passa alto; (c) $|H(\omega)|$ passa banda; (d) $|H(\omega)|$ elimina banda.

Un filtro digitale è, invece, un algoritmo matematico implementato in hardware, firmware o software che opera su un segnale digitale in ingresso per generare un segnale di uscita, anch'esso digitale, le cui caratteristiche rispettano quelle definite dal tipo di filtraggio. Un filtro digitale

può essere classificato come lineare o non lineare, tempo invariante o tempo variante oppure secondo altre classificazioni. Particolare importanza ai fini del progetto rivestono i filtri lineari e tempo invarianti (LTI).
La relazione ingresso-uscita di un filtro si può esprimere nella seguente maniera:

$$Z(y[n]) = H(z)Z(x[n])$$

Dove $Z(y[n])$ è la Z-trasformata del segnale in uscita, $Z(x[n])$ è la Z-trasformata del segnale in ingresso e $H(x)$ è la Z-trasformata della risposta impulsiva del filtro.

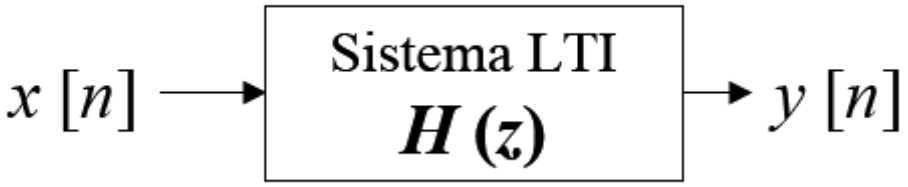


Fig. 1.1: Schema funzionale di un filtro

Ci sono due tipi fondamentali di filtri digitali LTI: filtri a risposta impulsiva infinita (IIR) e filtri a risposta impulsiva finita (FIR).
Nei filtri a risposta impulsiva infinita la risposta all'impulso (delta di Kronecker) non tende a zero, per realizzarlo è necessario reazionare il sistema e ciò può portare ad instabilità.
Nei filtri a risposta impulsiva finita la risposta all'impulso tende a zero, il sistema non necessita di essere reazionato e ciò comporta una intrinseca stabilità del sistema secondo il criterio BIBO (Bounded input bounded output).

Ora vediamo in dettaglio la seconda categoria di filtri, quelli a risposta impulsiva finita.

1.1 Filtri FIR

Come detto prima i filtri FIR sono filtri ad anello aperto, nei quali i campioni del segnale in uscita dipendono solo da un numero finito di campioni in ingresso. Questa caratteristica è un primo vantaggio rispetto ai filtri IIR in quanto necessitano di una memoria finita.

La relazione ingresso-uscita in un filtro FIR può essere descritta dalla seguente formula:

$$y[n] = \sum_{i=0}^{N} c_i x[n-i]$$

dove $x[n-i]$ sono i campioni del segnale in ingresso e c_i sono i coefficienti del filtro. Per variare la risposta in frequenza del filtro si può agire sui coefficienti c_i. Il termine N, limite superiore della sommatoria, corrisponde al numero di TAP presenti nell'architettura. Ogni TAP rappresenta un elemento di ritardo.

È possibile calcolare la risposta impulsiva e la funzione di trasferimento di un filtro FIR applicando un impulso come segnale di ingresso e valutando il segnale in uscita.

$$h[n] = \sum_{i=0}^{N} c_i \delta[n-i]$$

$$H(z) = Z(h[n]) = \sum_{i=0}^{N} c_i z^{-i}$$

Come possiamo vedere i coefficienti del filtro sono gli stessi della risposta impulsiva, e quindi in fase di progettazione è possibile limitarsi a calcolare solo questi ultimi.

L'immagine seguente è una rappresentazione grafica dell'architettura di un filtro FIR.

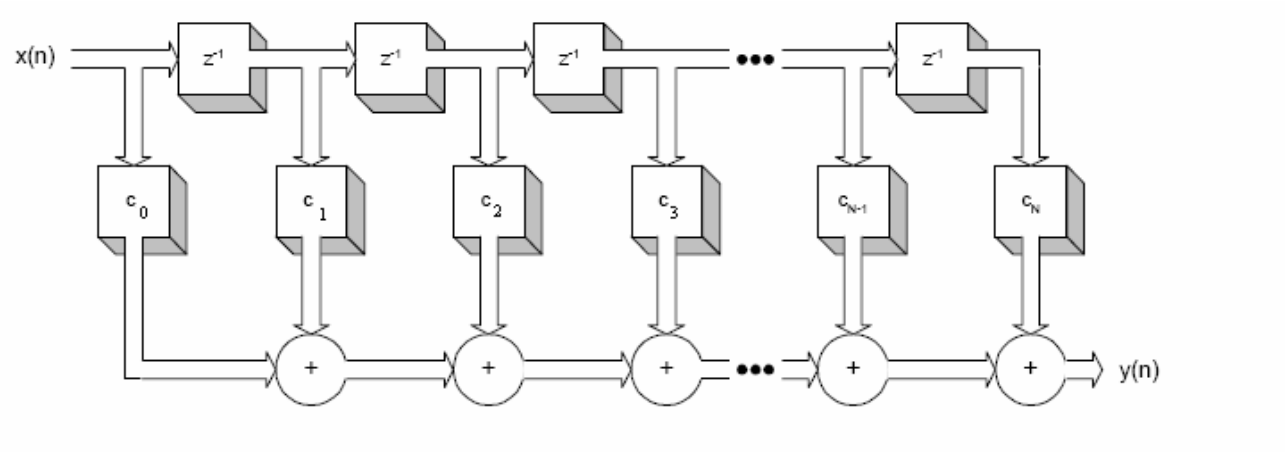


Fig. 1.2: Architettura di un filtro FIR.

La catena di termini z^{-1} è chiamata catena di ritardo discreta (tapped-delay-line) ed ad ogni z^{-1} corrisponde un ritardo di un periodo di campionamento.

1.2 Tipologia di filtri FIR

Ci sono 4 tipi di filtri FIR a fase lineare, classificabili in base alla parità del numero di TAP e in base alla simmetria dei coefficienti.

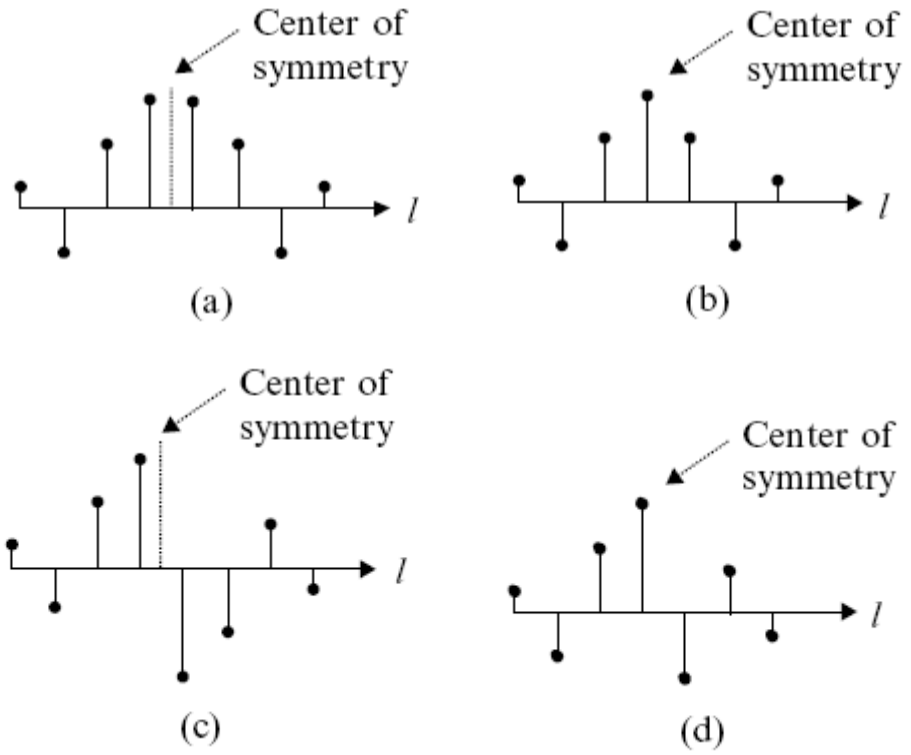


Fig. 1.4: (a) Tipo I: N pari, simmetria pari; (b) Tipo II: N dispari, simmetria pari; (c) Tipo III: N pari, simmetria dispari; (d) Tipo IV: N dispari, simmetria dispari

La risposta in frequenza per un filtro di tipo I vale sempre 0 alla frequenza di Nyquist ($f_N = f_c/2$), infatti a tale frequenza i campioni sono continuamente opposti in segno e uguali in modulo (poiché prelevo due campioni ogni periodo sfasati tra di loro di 180°), quindi la semisomma a destra del centro di simmetria è quella a sinistra sono uguali in modulo ma opposti in segno. Ecco perché questo tipo di filtro non è utilizzabile per realizzare un filtro passa alto.

La risposta in frequenza per un filtro di tipo III è sempre 0 a frequenza nulla, infatti a tale frequenza tutti i campioni del segnale sono uguali e nella sommatoria c'è sempre un termine e il suo opposto. Questo tipo di filtro non è utilizzabile per realizzare un filtro passa basso.

Con i filtri di tipo II e IV, non avendo particolari caratteristiche alle basse o alte frequenze, è possibile realizzare qualunque tipo di filtro.

Nel filtraggio FIR i coefficienti sono costanti, ma i campioni cambiano ogni periodo di campionamento. Infatti ogni campione percorre la catena di ritardo partendo dalla posizione $x(n)$ al tempo n, poi dopo un T_C occupa la posizione $x(n-1)$, poi $x(n-2)$, etc., fino a raggiungere la fine della catena di ritardo.

L’obiettivo nella progettazione di un filtro FIR è quello di determinare un set di coefficienti ($c_0, c_1, ..., c_N$) in maniera che le caratteristiche del filtro siano corrispondenti alle specifiche. Esistono varie tecniche per raggiungere questo scopo. Utilizzando il CAD MATLAB e soprattutto il tool FDATOOL è possibile ricavare con semplicità i coefficienti partendo dalla risposta in frequenza del filtro che si vuole ottenere.

Capitolo 2: Realizzazione

2.1 Scheda di sviluppo

Come già accennato nel capitolo precedente il progetto è stato realizzato sulla scheda di sviluppo Altera UP3.

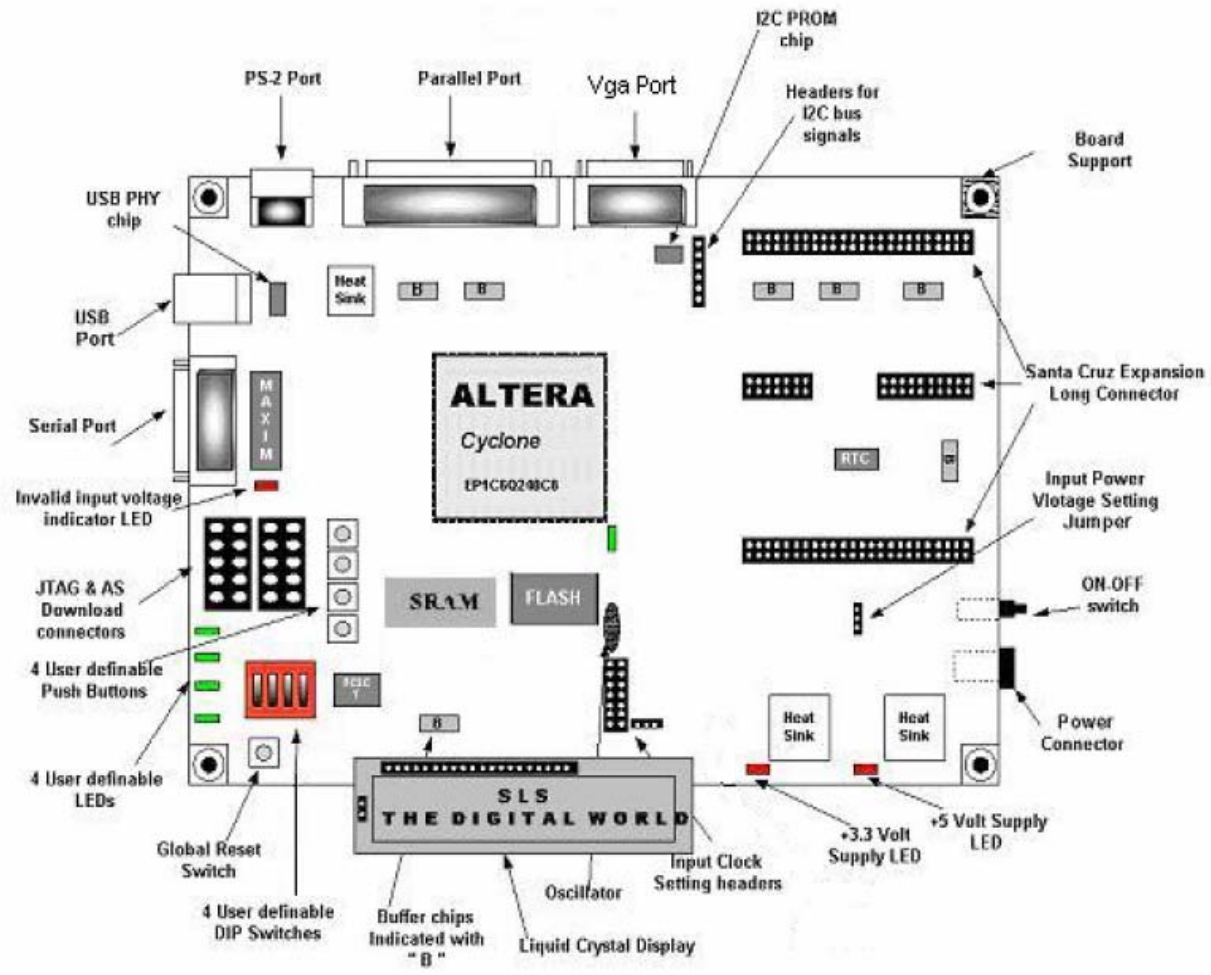
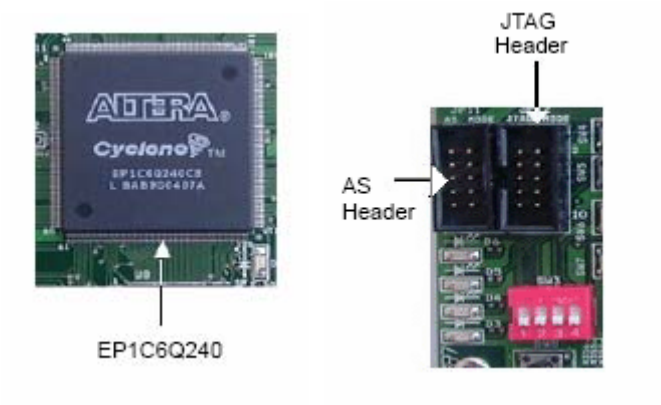


Fig. 2.1: Componenti presenti sulla scheda UP3.

Essa è equipaggiata con un FPGA □iclone EP1C6Q240C8 che può essere programmato tramite interfaccia JTAG presente sulla stessa scheda (vedi figura 2.2).



- **(b)**

Fig. 2.2: a) FPGA □iclone EP1C6Q240C8; b) Connettore JTAG.

La UP3 fornisce inoltre diverse altre possibilità di interfacciamento con il mondo esterno (tra i quali il connettore Santa Cruz), varie memorie, LEDs, pulsanti e persino un Display LCD. Non sono però presenti convertitori anologico-digitale o digitale-analogico.

Il progetto tuttavia, come già detto, ha come scopo quello di elaborare in digitale segnali audio analogici, quindi si è reso necessario realizzare una scheda che permettesse la conversione da analogico a digitale del segnale audio in ingresso, la comunicazione dell'informazione audio digitale alla scheda UP3 e la successiva ricezione e conversione del segnale filtrato in un segnale analogico da inviare a un dispositivo audio per l'ascolto.

2.2 Scheda aggiuntiva

2.2.1 Scelta dei convertitori

La scelta dei convertitori da utilizzare è stata fatta valutando diversi parametri, accuratamente scelti in modo da rispecchiare a pieno alle specifiche di progetto.
Le caratteristiche principali che i convertitori avrebbero dovuto avere sono le seguenti:

- Supportare una frequenza di campionamento 48Khz
- Alimentazione a 5 V
- Risoluzione: 16 bit
- Due canali analogici single-ended
- Package SOIC, DIP o SOL

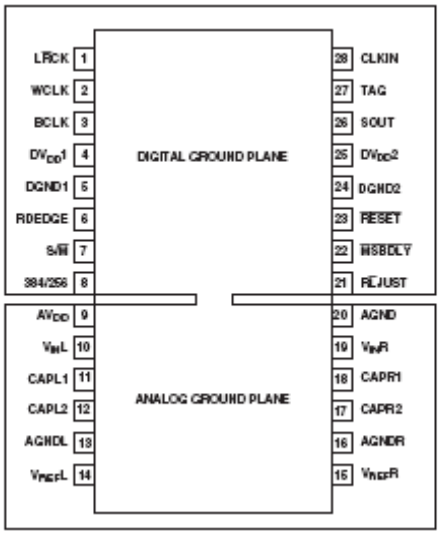
In seguito a specifiche ricerche e valutazioni è stato scelto l'ADC AD1870, mentre per quanto riguarda il DAC la scelta è ricaduta sul PCM1725.

2.2.2 AD1870

L'AD1870, prodotto dall'ANALOG DEVICE, è un ADC stereo che esegue una conversione AD a 16 bit basata sull'architettura sigma-delta utilizzato per applicazioni audio di medio-alto livello.
Le sue caratteristiche principali sono le seguenti:

- Single 5 V Power □iclon
- Single-Ended Dual-Channel Analog Inputs
- 92 dB (Typ) Dynamic Range
- 90 dB (Typ) S/(THD + N)
- 256 x f_s or 384 x f_S Input Clock
- On-Chip Voltage Reference
- Flexible Serial Output Interface
- 28-Lead SOIC Package

Di seguito viene mostrata la piedinatura dell'AD1870 e una tabella per la descrizione della funzione di ogni pin.



PIN FUNCTION DESCRIPTIONS

Pin	Input/ Output	Pin Name	Description
1	I/O	LRCK	Left/Right Clock
2	I/O	WCLK	Word Clock
3	I/O	BCLK	Bit Clock
4	I	$DV_{DD}1$	5 V Digital Supply
5	I	DGND1	Digital Ground
6	I	RDEDGE	Read Edge Polarity Select
7	I	S/M	Slave/Master Select
8	I	384/256	Clock Mode
9	I	AV_{DD}	5 V Analog Supply
10	I	$V_{IN}L$	Left Channel Input
11	O	CAPL1	Left External Filter Capacitor 1
12	O	CAPL2	Left External Filter Capacitor 2
13	I	AGNDL	Left Analog Ground
14	O	$V_{REF}L$	Left Reference Voltage Output
15	O	$V_{REF}R$	Right Reference Voltage Output
16	I	AGNDR	Right Analog Ground
17	O	CAPR2	Right External Filter Capacitor 2
18	O	CAPR1	Right External Filter Capacitor 1
19	I	$V_{IN}R$	Right Channel Input
20	I	AGND	Analog Ground
21	I	RJUST	Right/Left Justify
22	I	MSBDLY	Delay MSB One BCLK Period
23	I	RESET	Reset
24	I	DGND2	Digital Ground
25	I	$DV_{DD}2$	5 V Digital Supply
26	O	SOUT	Serial Data Output
27	O	TAG	Serial Overrange Output
28	I	CLKIN	Master Clock

- (b)

Fig. 2.3: a) Piedinatura dell'AD1870; b) Descrizione della funzione dei pin.

Il convertitore per funzionare correttamente richiede l'utilizzo di alcuni condensatori esterni da collegare come in figura 2.4:

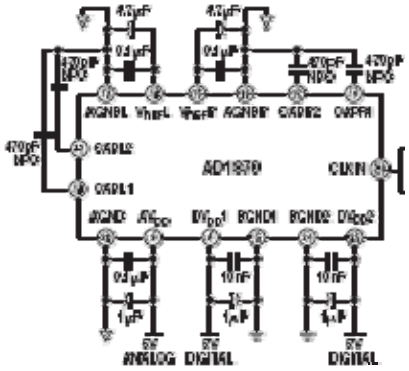


Fig. 2.4: Condensatori da collegare all'AD1870 per un corretto funzionamento

Inoltre è necessario scegliere la modalità di utilizzo dell'ADC e l'opportuna temporizzazione impostando il valore di alcuni pin. Le scelte effettuate sono le seguenti:

- RDEDGE (Read Edge Polarità Select) collegato a □iclo in modo da impostare la trasmissione dei dati sul fronte in salita di BCLK e quindi averli validi sul fronte in discesa dello stesso segnale.
- S/M (Slave/Master select) collegato all'alimentazione in modo da selezionare il modo Slave.
- 384/256 (Clock Mode) collegato a massa in modo da poter fornire un Input clock pari a 256 volte la frequenza di campionamento (f_S).
- RLJUST (Right/Left Justify) collegato a □iclo così da impostare la trasmissione dei bit in uscita giustificata a sinistra.
- MSBDLY (Delay MSB one BCLK Period) riportato in ingresso alla scheda UP3 e forzato alto in fase di programmazione dell'FPGA così da rendere semplice un'eventuale modifica di tale impostazione (qualora si richiedesse un ritardo di un periodo di BCLK nella trasmissione dei bit e quindi realizzare una trasmissione di tipo I^2S).
- TAG (Serial Overrange Output) N.C.

In base a quanto detto la temporizzazione scelta è quella rappresentata in figura 2.5.

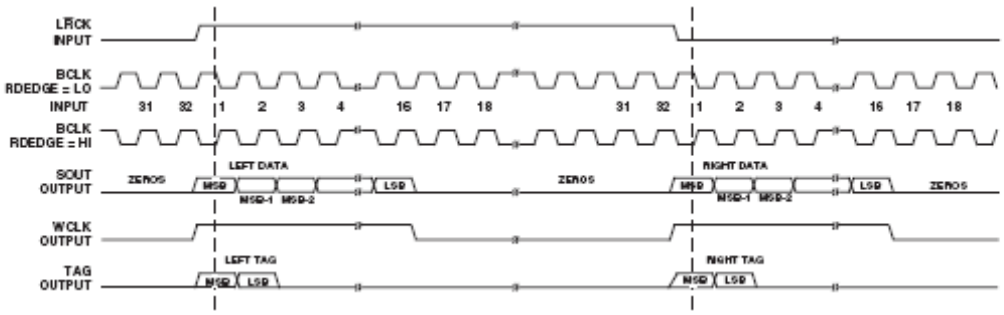


Fig. 2.5: Temporizzazione Serial Data Output: Slave Mode, Left-Justified with No MSB Delay, S/M = HI, RLJUST = LO, MSBDLY = HI.

I segnali da gestire in ingresso e in uscita dell'AD1870 sono dunque:

- $V_{IN}L$ (Left Channel Input): Ingresso analogico canale sinistro.
- $V_{IN}R$ (Right Channel Input): Ingresso analogico canale destro.
- LRCK (Left/Right Clock): Clock di frequenza pari a quella di campionamento (f_S) che deve essere fornito in ingresso al convertitore. Esso contiene l'informazione necessaria a distinguere i campioni relativi al canale destro e sinistro.

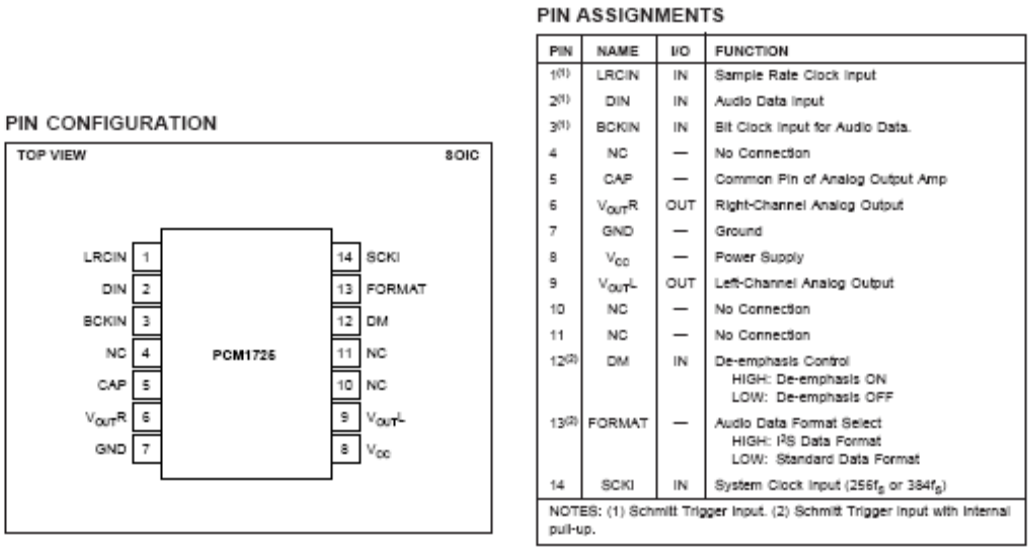
- SOUT (Serial Data Output): Bit in uscita al convertitore che codificano il segnale analogico in ingresso. Se LRCK=1 i bit sono quelli relativi al canale sinistro, se LRCK=0 i bit sono relativi al canale destro.
- WCLK (Word Clock): E' un'uscita del convertitore che segnala la trasmissione dei campioni su SOUT.
- BCLK (Bit Clock): E' un input clock di frequenza $64f_S$, pari a quella di trasmissione dei bit su SOUT.
- CLKIN (Clock in): E' il clock di sistema, che deve essere fornito in ingresso all'AD1870. Deve avere una frequenza pari a $256f_S$.
- RESET: E' il segnale di reset che verrà fornito opportunamente tramite la scheda UP3.

2.2.3 PCM1725

Il PCM1725 è un DAC stereo per applicazioni audio prodotto dalla Burr-Brown che effettua la conversione dal digitale all'analogico mediante un modulatore sigma-delta.
Le sue caratteristiche principali sono le seguenti:

- DYNAMIC RANGE: 95dB
- MULTIPLE SAMPLING FREQUENCIES: 16kHz to 96kHz
- SYSTEM CLOCK: 256fS / 384fS
- NORMAL OR I2S DATA INPUT FORMATS
- SMALL 14-PIN SOIC PACKAGE

In figura 2.6 sono riportate la piedinatura del PCM1725 e la funzione di ogni pin.



PIN ASSIGNMENTS

PIN	NAME	I/O	FUNCTION
1(1)	LRCIN	IN	Sample Rate Clock Input
2(1)	DIN	IN	Audio Data Input
3(1)	BCKIN	IN	Bit Clock Input for Audio Data.
4	NC	—	No Connection
5	CAP	—	Common Pin of Analog Output Amp
6	$V_{OUT}R$	OUT	Right-Channel Analog Output
7	GND	—	Ground
8	V_{CC}	—	Power Supply
9	$V_{OUT}L$	OUT	Left-Channel Analog Output
10	NC	—	No Connection
11	NC	—	No Connection
12(2)	DM	IN	De-emphasis Control HIGH: De-emphasis ON LOW: De-emphasis OFF
13(2)	FORMAT	—	Audio Data Format Select HIGH: I²S Data Format LOW: Standard Data Format
14	SCKI	IN	System Clock Input ($256f_S$ or $384f_S$)

NOTES: (1) Schmitt Trigger Input. (2) Schmitt Trigger Input with Internal pull-up.

a) b)

Fig. 2.6: a) Piedinatura del PCM1725; b) Descrizione della funzione dei pin.

Le connessioni del DAC sono schematizzate nella figura 2.7.

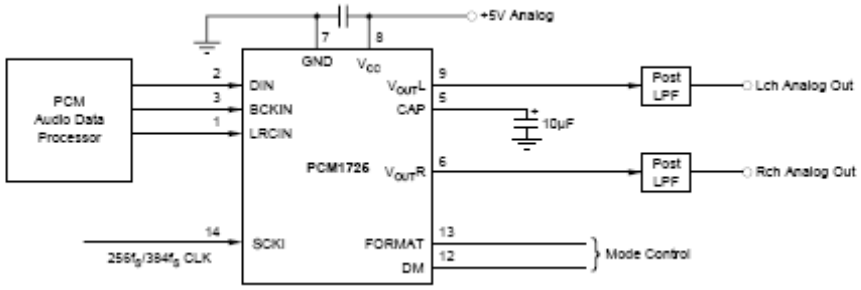


Fig. 2.7: Diagramma delle connessioni del PCM1725.

Come si vede dalle figure precedenti oltre ai pin di □iclo (GND) e di alimentazione (Vcc) è presente il pin CAP (al quale è stato collegato un condensatore, come richiesto dal datasheet del convertitore), i pin per il controllo funzionale:

- FORMAT: permette di scegliere tra 2 formati dei dati: normale (FORMAT = '0') e I2S (FORMAT = '1'). Tale segnale è stato riportato in ingresso alla scheda UP3 e forzato al livello basso in fase di programmazione dell'FPGA così da poter permettere una eventuale facile modifica del formato dei dati.
- DM: comanda la funzione di de-emphasis. Tale funzione è stata disattivata ponendo il pin a □iclo.

I pin per i segnali di interfacciamento dati audio:

- DIN (Data Input): È il terminale sul quale vengono trasmessi serialmente i campioni audio di entrambi i canali.
- LRCIN (Left Right Channel Input): contiene l'informazione necessaria a distinguere i campioni relativi al canale destro e sinistro. La sua frequenza vale f_S.
- BCKIN (Bit Clock Input): È fornito dalla scheda UP3 ed è utilizzato dal DAC per campionare i bit presenti su DIN. La sua frequenza è di 64 f_S.
- SCKI (System Clock Input): È il clock di sistema, fornito dall'FPGA tramite la UP3 a una frequenza di 256f_S.
- $V_{OUT}L$: Uscita analogica canale sinistro.
- $V_{OUT}R$: Uscita analogica canale destro.

In accordo con quanto detto precedentemente il PCM1725 riceve i dati con la temporizzazione rappresentata in figura 2.8.

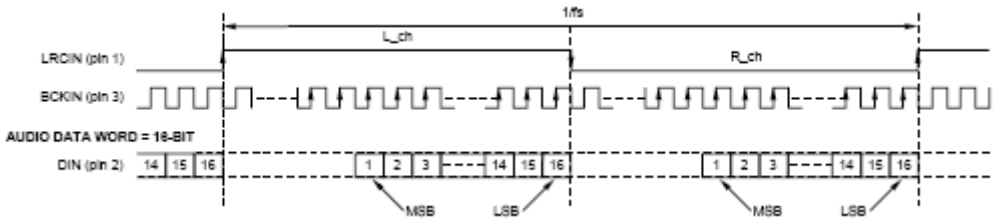


Fig. 2.8: Temporizzazione "Normal" dei Dati in ingresso al PCM1725.

2.2.4 Descrizione della topologia circuitale

In ingresso all'ADC, sia sul canale destro che su quello sinistro, è necessario un filro anti-alias. Esso è stato realizzato tramite una squadra RC e dimensionato in modo da presentare una frequenza di cut-off pari a circa 240 KHz (vedi figura 2.8), come suggerito dal datasheet dell'AD1870, così che non risulti sostanzialmente nessuna attenuazione a 20 KHz (frequenza massima dello spettro musicale).

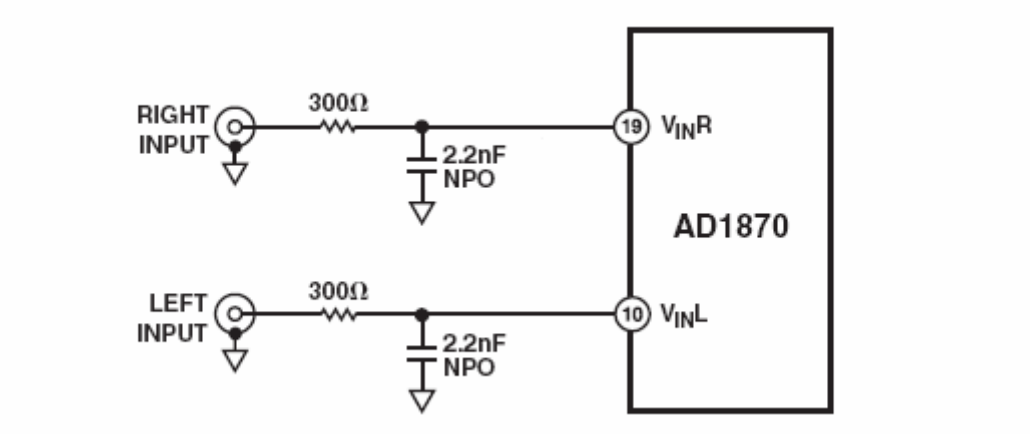


Fig. 2.9: Filtro anti-alias in ingresso all'AD1870.

Ma occorrono anche ulteriori considerazioni sui segnali applicati su $V_{IN}R$ e $V_{IN}L$. Il segnale audio in ingresso alla scheda aggiuntiva, proveniente da un qualsiasi riproduttore musicale, avrà certamente valor medio nullo, mentre l'AD1870 codifica segnali sempre positivi compresi tra un minimo e un massimo valore di tensione e il cui valore centrale è V_{REF}, che ha un valore tipico di 2.25V. Per questo motivo a monte del filtro anti-alias è necessaria una traslazione di tensione del segnale audio così da poter sfruttare al meglio la dinamica del convertitore AD. Tale traslazione è stata realizzata mediante un partitore resistivo e un condensatore, dimensionati in modo da realizzare anche un filtraggio passa alto con frequenza di taglio di circa 20 Hz, corrispondente alla frequenza minima dello spettro musicale. In figura 2.10 è rappresentato lo schema circuitale relativo al canale sinistro, quello relativo al canale destro è identico.

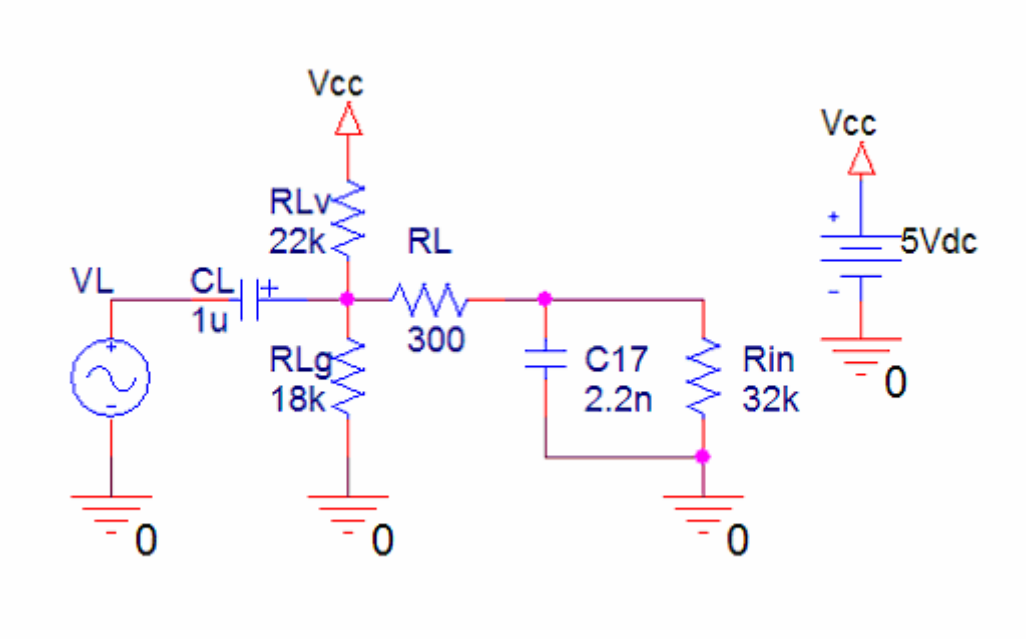
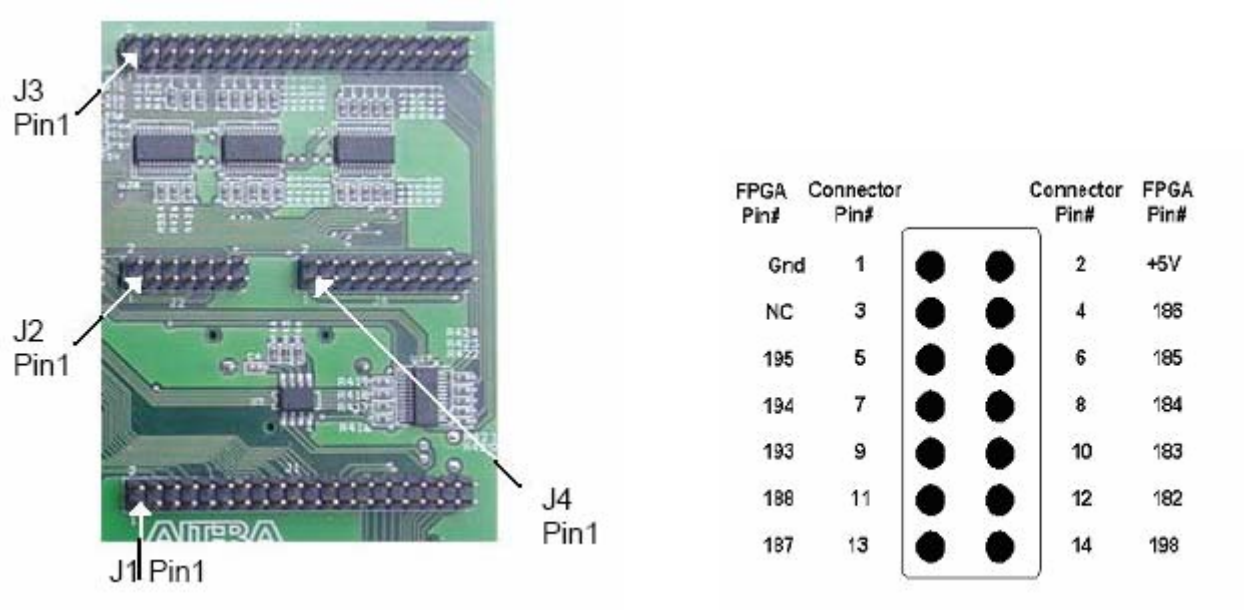


Fig. 2.10: Schema circuitale dell'ingresso sul canale sinistro dell'AD1870; Rin è la resistenza di ingresso del pin $V_{IN}L$ dell'ADC.

Infine per ottenere in uscita al sistema progettato un segnale analogico con valor medio nullo è stato realizzato, sull'uscita del PCM1725, un filtro RC passa alto dimensionato in modo da eliminare la componente continua e presentare una frequenza di taglio molto bassa. In particolare si è ipotizzato di collegare in uscita al sistema un amplificatore audio ad alta impedenza di ingresso (è sufficiente un'impedenza superiore a 3,5KΩ per avere una frequenza di taglio minore di 20 Hz).

2.2.5 Interfacciamento

Per l'interfacciamento con dispositivi di riproduzione musicale e di ascolto (quali amplificatori audio, casse o semplici cuffie) si è scelto di utilizzare connettori RCA.
L'interfacciamento con la UP3 invece è stato realizzato tramite i connettori Santa Cruz (vedi figura 2.11).



(a) (b)

Fig. 2.11: a) Connettori Santa Cruz presenti sulla scheda UP3**; b)** Connettore J2.

I pin del connettore Santa Cruz J2 della scheda UP3 di figura sono stati collegati alla scheda aggiuntiva come descritto di seguito:

- *Pin 1*: collegato alla massa della scheda aggiuntiva;
- *Pin 2*: terminale con il quale la UP3 fornisce l'alimentazione alla scheda aggiuntiva;
- *Pin 3:* N.C.;
- *Pin 4:* N.C.;
- *Pin 5:* collegato al pin di RESET dell'AD1870;
- *Pin 6:* collegato al piedino FORMAT del PCM1725;
- *Pin 7:* collegato ai piedini CLKIN dell'AD1870 e a SCKI del PCM1725;
- *Pin 8:* collegato ai terminali LRCK e LRCIN rispettivamente dell'ADC e del DAC;
- *Pin 9:* collegato all'ADC tramite SOUT;
- *Pin 10:* collegato al DIN del DAC;
- *Pin 11:* collegato all'uscita WCLK dell'AD1870;
- *Pin 12:* terminale con il quale la UP3 fornisce il clock BCLK all'AD1870 e BCKIN al PCM1725;
- *Pin 13:* collegato all'ingresso MSBDLY dell'ADC;
- *Pin 14:* N.C.

2.3 Implementazione su FPGA

2.3.1 Gerarchizzazione e partizionamento

La parte □iclone□o□tivi su FPGA si è svolta con l'ausilio del CAD Quartus II di Altera. Al fine di semplificare il processo di implementazione e la fase di test è stata attuata una politica di gerarchizzazione e partizionamento dell'architettura all'interno dell'FPGA seguendo un approccio top-down. Con un primo partizionamento si è giunti ad identificare 4 macroblocchi:

- Interfaccia con ADC;
- Elaborazione e filtraggio;
- Interfaccia con DAC;
- Generazione sincronismi e controllo.

Successivamente tali macroblocchi sono stati suddivisi in vari blocchi di complessità minore, fino a giungere a blocchetti elementari quali flip-flop D e porte logiche elementari (le primitive del progetto).

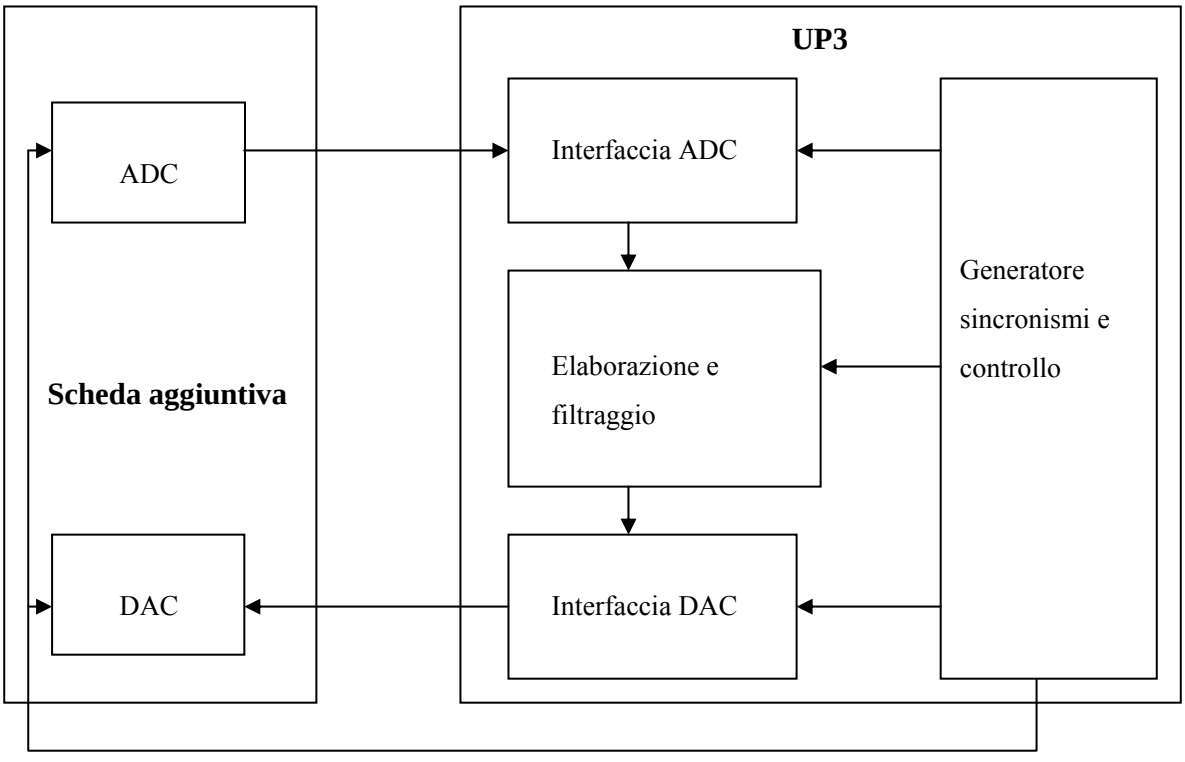


Fig. 2.12: Divisione del sistema in macroblocchi.

Nei successivi paragrafi si descriverà in dettaglio ogni singolo macroblocco.

2.3.2 Interfacciamento con ADC e DAC

Come già spiegato nel paragrafo 2.2.2, l'AD1870 campiona i segnali audio sul canale destro e su quello sinistro, codifica ogni campione con una parola lunga 16 bit e trasmette serialmente ogni bit di tale parola (partendo dal MSB) alla frequenza di BCLK sul pin SOUT. In particolare (si faccia riferimento alla figura 2.4) durante la prima metà del semiperiodo alto di LRCK vengono trasmessi i bit della parola relativa al campione del canale sinistro, mentre durante la prima metà del semiperiodo basso di LRCK vengono trasmessi i bit relativi alla parola che codifica il campione del canale destro.
Per l'elaborazione numerica dei campioni è necessario fare una conversione seriale-parallelo dei bit di ogni parola e successivamente, visto che il PCM1725 riceve i bit di ogni parola serialmente, fare una conversione parallelo-seriale.
La temporizzazione con il quale il PCM1725 riceve i dati in ingresso (rappresentata in figura 2.7) prevede che la parola relativa al campione sinistro venga ricevuta durante la seconda metà del semiperiodo alto di LRCK, mentre quella relativa al campione destro durante la seconda metà del semiperiodo basso di LRCK.

2.3.3 Sipo_manager

La necessità della conversione seriale-parallelo della trasmissione dei dati è stata soddisfatta con la creazione di un blocco Sipo_manager che contiene al suo interno due registri Sipo (Serial In Parallel Out), uno per il canale destro e uno per il canale sinistro, e un multiplexer che, a seconda del valore di LRCK, smista i campioni verso l'uno o l'altro registro.
Sipo_manager dunque dispone degli ingressi:

- LRCK: che costituirà il bit di selezione del mux interno;
- D: con il quale riceve i bit provenienti dal pin SOUT dell'ADC;
- CLK: con il quale vengono clockati i registri SIPO;
- ENABLE: che serve per l'abilitazione dei registri;
- RESET: per resettare i registri;

Mentre le uscite sono:

- QparL: con la quale il Sipo_manager trasmette la parola relativa al campione del canale sinistro al blocco elaborazione corrispondente;
- QparR: con la quale il Sipo_manager trasmette la parola relativa al campione del canale destro al blocco elaborazione corrispondente;
- ENEOUT: che serve ad abilitare il Piso_manager in uscita. Esso è alto nella seconda metà di ogni semiperiodo di LRCK, ovvero quando l'ADC non trasmette alcun dato su SOUT.

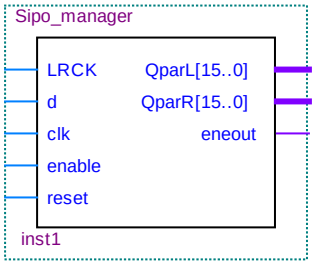


Fig. 2.13: Simbolo del blocco Sipo_manager.

2.3.4 Piso_manager

La trasformazione della trasmissione dei bit delle parole che codificano i campioni elaborati da parallela a seriale è stata effettuata con il blocco Piso_manager. Esso è realizzato con due registri PISO (Parallel In Serial Output) e un multiplexer che instrada i campioni sul registro del canale destro o sinistro a seconda del valore di LRCK.

Gli ingressi del Piso_manager sono:

- LRCK: che internamente va a costituire il bit di selezione del mux;
- DparL: con il quale il Piso_manager riceve la parola del campione del canale sinistro elaborato;
- DparR: con il quale il Piso_manager riceve la parola del campione del canale sinistro elaborato;
- CLK: con il quale vengono clockati i registri PISO;
- ENABLE_CLK: che serve per l'abilitazione dei registri;
- ENABLE: che serve a decidere quando caricare i dati sui PISO (ENABLE='0') e quando farli scorrere (ENABLE = '1').

L'uscita invece sarà solo:

- Q: con la quale vengono trasmessi serialmente i bit al DAC

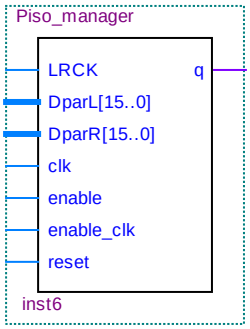


Fig. 2.14: Simbolo del blocco Piso_manager.

2.3.5 Elaborazione e filtraggio

Un fattore di merito per i filtri FIR è il numero di TAP, maggiore è questo numero e maggiore è la qualità del filtraggio che è possibile ottenere. Naturalmente questo aumento nelle prestazioni del filtro si paga con la necessità di una memoria di dimensioni maggiori, di un maggior numero di calcoli da eseguire e di una frequenza di lavoro maggiore.
Nel dispositivo in esame è stato scelto di utilizzare 16 TAP. Questa scelta è stata ritenuta un buon compromesso tra complessità e qualità del filtraggio, in quanto permette di utilizzare una memoria ridotta, non richiede l'utilizzo di un moltiplicatore ad alta velocità e si ha comunque un'effetto evidente all'udito dell'elaborazione.

Nel sistema sono presenti 2 blocchi elaborazione e filtraggio identici, uno per il trattamento del segnale relativo al canale destro ed uno per il trattamento di quello sinistro.
Ognuno di tali blocchi può essere partizionato nei seguenti sottoblocchi:

- Memoria campioni e generatore di indirizzi;
- Memoria coefficienti e selezionatore del filtro;
- Moltiplicatore e accumulatore;

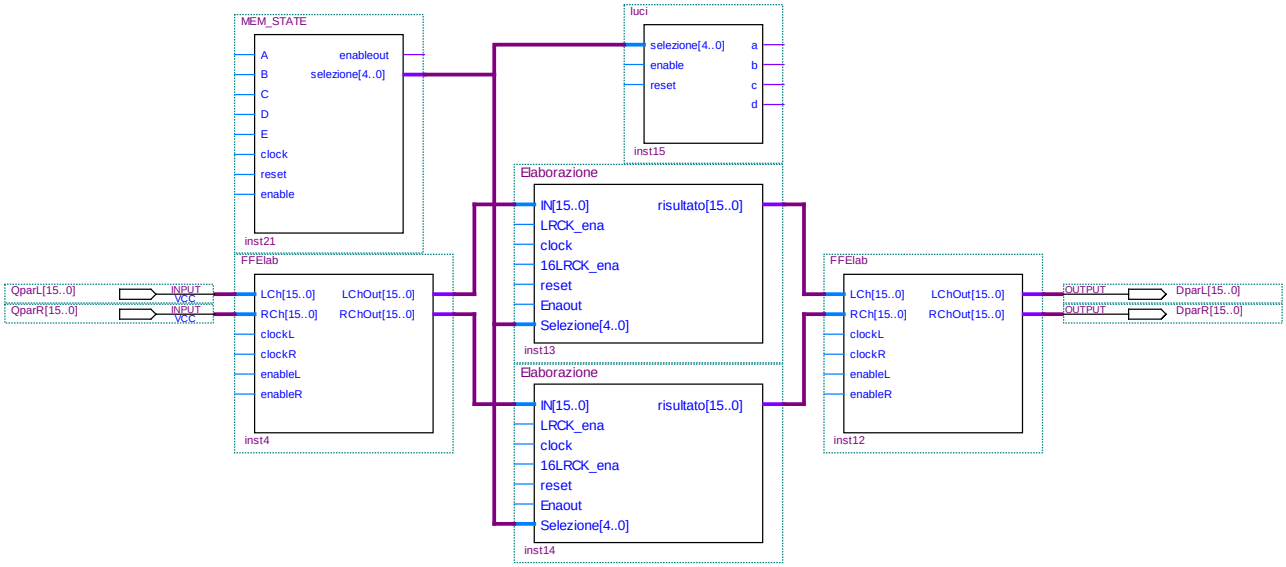


Fig. 2.15: Partizionamento macroblocco elaborazione e filtraggio.

2.3.6 Memoria campioni e generatore indirizzi

I campioni del segnale audio in ingresso al sistema vengono codificati in binario dall'ADC ma prima di essere memorizzati nella meoria_campioni vengono convertiti in complemento a 2 dal blocco Complemento_2, rappresentato in figura 2.16.

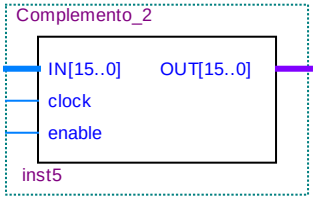


Fig. 2.16: Simbolo del blocco Complemento_2**.**

Gli ingressi di Complemento_2 sono:

- IN: con il quale il blocco riceve la parola da convertire;
- CLOCK: con il quale clockare tutti flip flop interni Complemento_2;
- ENABLE: con il quale abilitare (con una frequenza pari a LRCK) i flip flop interni al blocco.

L'uscita è:

- OUT:con la quale Complemento_2 comunica ogni parola convertita alla memoria campioni.

Nella figura 2.15 si evidenziano le connessioni tra Complemento_2 e memoria_campioni.

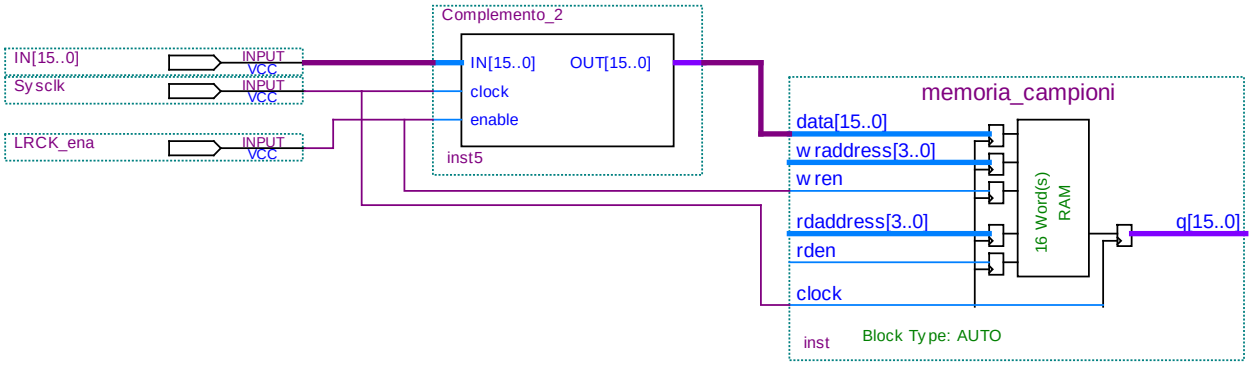


Fig. 2.17: Connessioni tra Complemento_2 e memoria_campioni**.**

La memoria campioni è stata implementata con una RAM, realizzata utilizzando una megafunction presente nel CAD QUARTUS II, in grado di instanziare ed implementare in maniera ottimizzata per l'architettura del FPGA vari blocchi complessi.

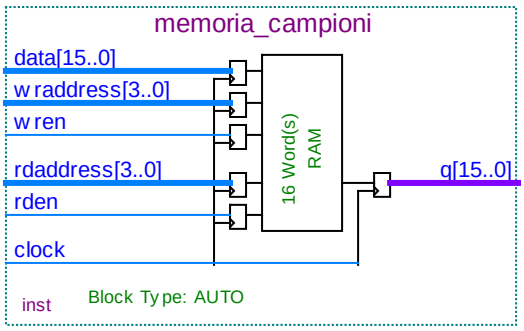


Fig. 2.18 : Simbolo della memoria RAM

La parte di generazione indirizzi deve generare sia gli indirizzi in scrittura che in lettura per le due RAM e gli indirizzi in lettura per le ROM in cui sono contenuti i coefficienti dei filtri.
Gli indirizzi in scrittura sono generati da un contatore modulo 16, in modo da implementare una memoria circolare. Questo contatore conta ogni ciclo di LRCK, in maniera che ogni nuovo campione venga scritto nell'elemento di memoria successivo.

Gli indirizzi in lettura della RAM devono avere una corrispondenza precisa con quelli della ROM in cui sono contenuti i coefficienti così da moltiplicare i giusti campioni con i giusti coefficienti. In figura 2.15 è possibile vedere la relazione tra i primi ed i secondi.
Per generare gli indirizzi in lettura è necessario oltre ad un contatore modulo 16, che, diversamente da quello utilizzato in lettura conta ogni 16LRCK, in maniera che ad ogni LRCK sia tornato allo stato iniziale, sia l'utilizzo di un sommatore.

Il sommatore serve ad aggiungere un offset agli indirizzi nella memoria campioni, in maniera da permettere la lettura contemporanea dalle due memorie e la successiva moltiplicazione dell'ultimo campione ricevuto e del primo coefficiente, del campione precedente e del secondo coefficiente, etc.

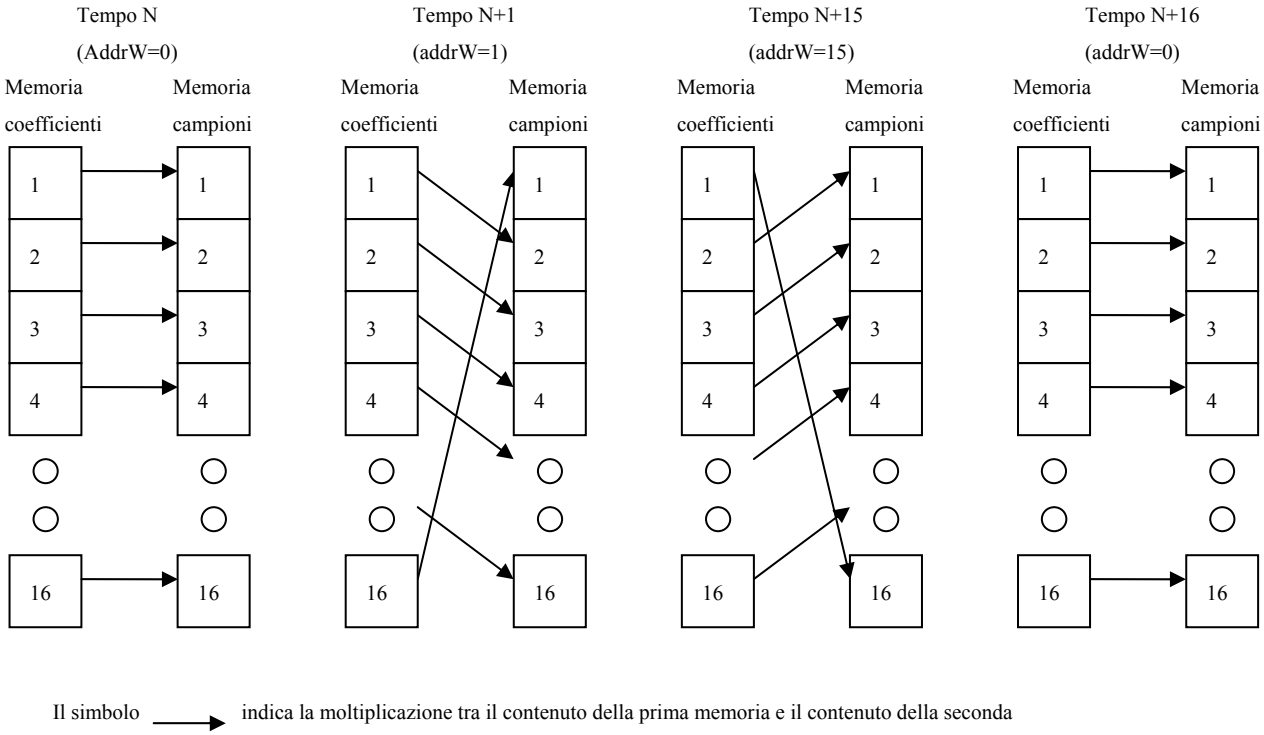


Fig. 2.19: Corrispondenza tra indirizzi campioni e indirizzi coefficienti.

L'indirizzo in lettura alla ROM viene direttamente generato del contatore modulo 16 che conta ogni fronte positivo di16LRCK.

Per generare gli indirizzi in lettura alla RAM è necessario sommare un offset all'indirizzo della ROM, e questo offset è proprio l'indirizzo in scrittura alla RAM. L'indirizzo in scrittura indica proprio l'ultimo campione ricevuto e quindi il campione che deve essere moltiplicato con l'indirizzo 0 della memoria coefficienti.

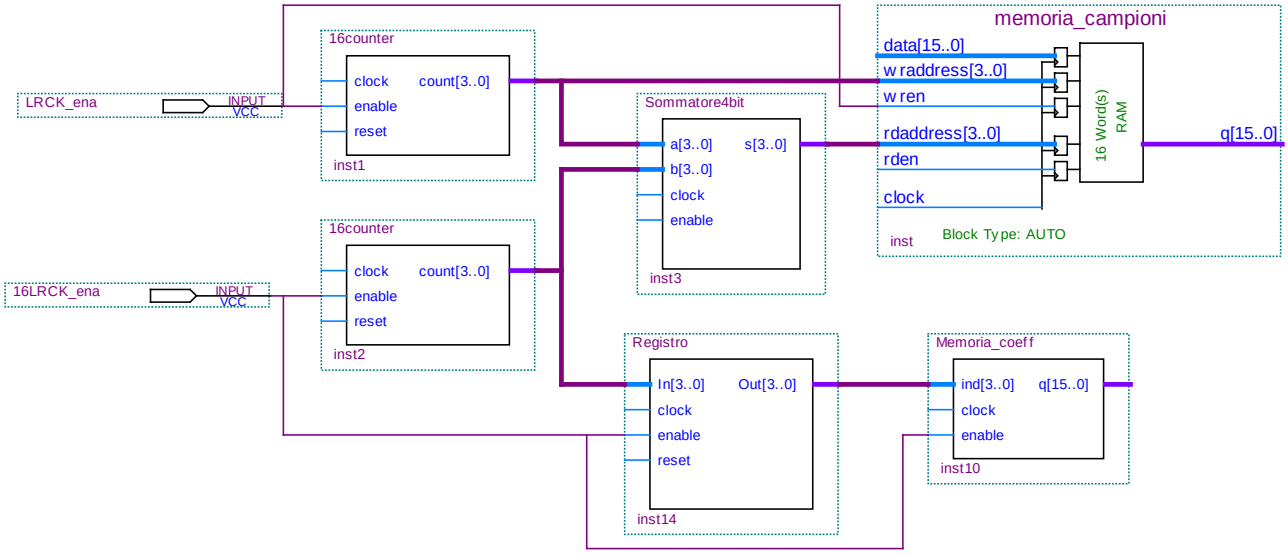


Fig. 2.20: Generazione indirizzi in scrittura e in lettura.

2.3.7 Memoria coefficienti e scelta filtro

Questo blocco può essere partizionato in 2 sottoblocchi distinti: la memoria coefficienti e la circuteria di scelta del filtro.

Esistono 5 memorie coefficienti separate, una per ogni filtro, tutte indirizzate con l'indirizzo generato del blocco generatore indirizzi. Come i campioni memorizzati nella memoria campioni, anche i coefficienti sono codificati in complemento a due. Siccome i coefficienti richiesti da un filtro FIR sono numeri decimali minori di 1, si è reso necessario scegliere una rappresentazione che permettesse di avere sia una complessità ridotta, sia una risoluzione ed un range adatti allo scopo.

Considerando il fatto che il range non è molto elevato, la scelta è ricaduta su una rappresentazione in virgola fissa, memorizzando i coefficienti moltiplicati per 2^{15} in maniera da sfruttare tutti e sedici i bit della ROM, ed evitare l'overflow, proprio perché i coefficienti di qualsiasi filtro FIR sono minori di 1.

Nella memoria coefficienti 1 sono stati memorizzati i coefficienti relativi ad un filtro passa alto con una frequenza di taglio di 10kHz. È stata scelta una frequenza di taglio così elevata in maniera da avere un effetto filtrante molto marcato. Il filtro è a fase lineare e con coefficienti antisimmetrici, questa è l'unica scelta possibile con un numero pari di coefficienti.

Nella memoria coefficienti 2 sono stati memorizzati i coefficienti di un filtro passa basso con frequenza di taglio 100Hz. Il filtro è a fase lineare con coefficienti simmetrici. Anche per questo filtro questa è l'unica scelta possibile.

Nella memoria coefficienti 3 sono stati memorizzati i coefficienti relativi ad un filtro taglia banda con una frequenza centrale di 4,5 kHz ed una banda reiettata di 7 kHz. Questo filtro non può essere realizzato con fase lineare in quanto il numero di coefficienti pari comporta che o ad alta frequenza, o a basa frequenza il modulo della risposta in frequenza tenda a 0.

Nella memoria coefficienti 4 è stato inserito un filtro enfatizza medi (passa banda), con frequenza centrale di 5 kHz ed una banda di 6 kHz. Questo filtro è a fase lineare e con coefficienti simmetrici.

Nella memoria coefficienti 5 è stato memorizzato un coefficiente pari a 2^{15} all'indirizzo 0, mentre a tutti gli altri indirizzi è stato memorizzato il coefficiente 0. Utilizzando questa memoria coefficienti otteniamo in uscita un segnale identico a quello in ingresso, ad eccezione di un ritardo.

La scelta della memoria coefficienti da utilizzare viene realizzata successivamente da un multiplexer pilotato da MEM_STATE.

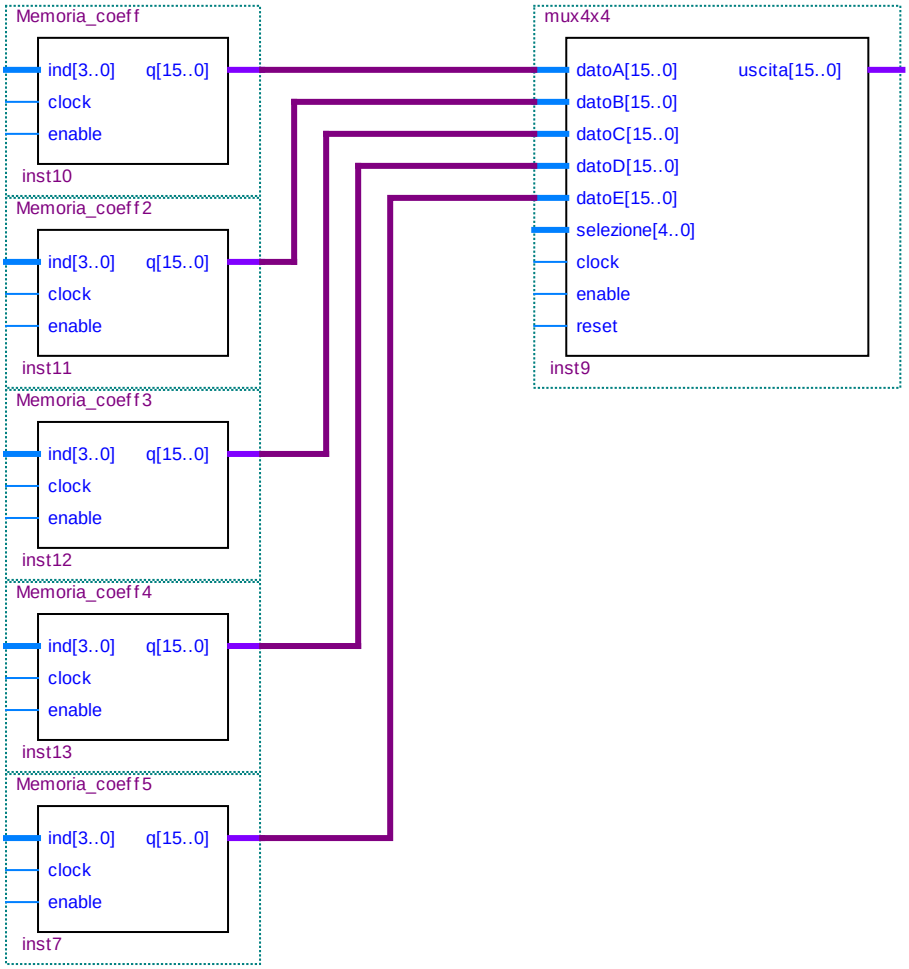


Fig. 2.21: Memorie coefficienti e multiplexer.

La parte relativa alla selezione del filtro si può ancora suddividere in due parti separate

- MEM_STATE: riconosce il pulsante premuto e segnala il filtro da utilizzare;
- LUCI: in base al filtro selezionato accende uno dei LED presenti sulla scheda UP3.

- **MEM_STATE**

Questo blocco serve per selezionare quale memoria coefficienti utilizzare, e quindi quale tipo di filtraggio realizzare.
In ingresso accetta i seguenti segnali:

- A: Serve per attivare il primo banco di coefficienti;
- B: Serve per attivare il secondo banco di coefficienti;
- C: Serve per attivare il terzo banco di coefficienti;
- D: Serve per attivare il quarto banco di coefficienti;
- E: Serve per attivare il quinto banco di coefficienti;
- CLOCK: è il clock di tutti i flip-flop;
- RESET: resetta la macchina a stati interna e porta a 0 tutti i flip-flop;
- ENABLE: serve ad abilitare il sistema a essere sensibile ai cambi di filtro e ad abilitare l'uscita.

In uscita il blocco genera i seguenti segnali:

- SELEZIONE: Indica al blocco elaborazione ed al blocco luci quale filtro è stato selezionato;
- ENABLEOUT: Serve ad abilitare o meno l'uscita.

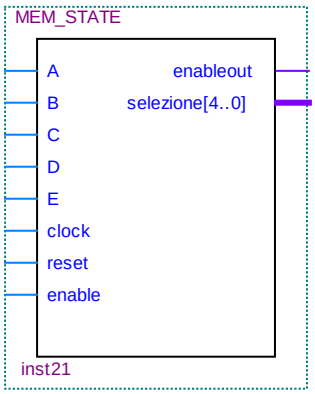


Fig. 2.22: Simbolo del blocco MEM_STATE.

I segnali A, B, C, D sono collegati ai Push Button Switches (vedi figura 2.20 a) e quindi hanno la forma di impulsi. Essi vengono dapprima sincronizzati utilizzando una struttura come quella in figura 2.23, poi viene inviato in ingresso ad una macchina a stati che rileva la pressione del tasto e resetta il primo flip-flop con il segnale Ack.

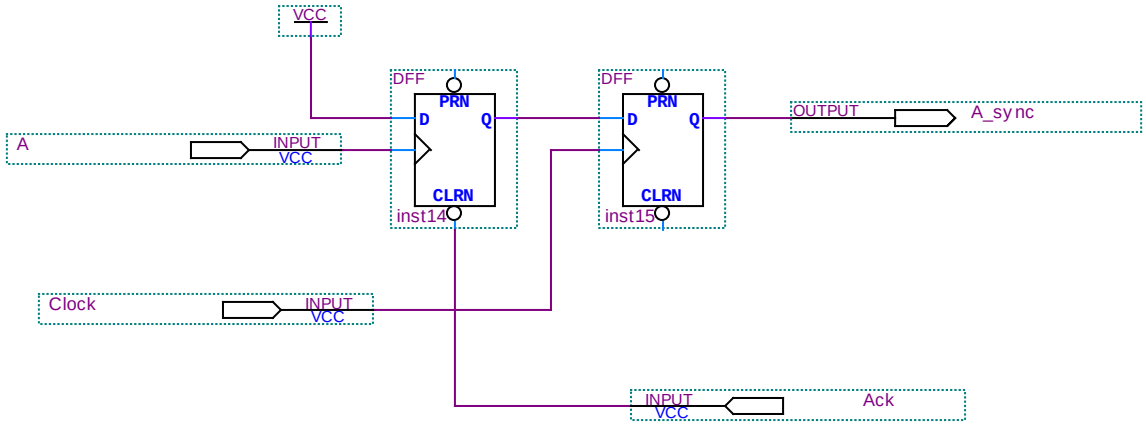
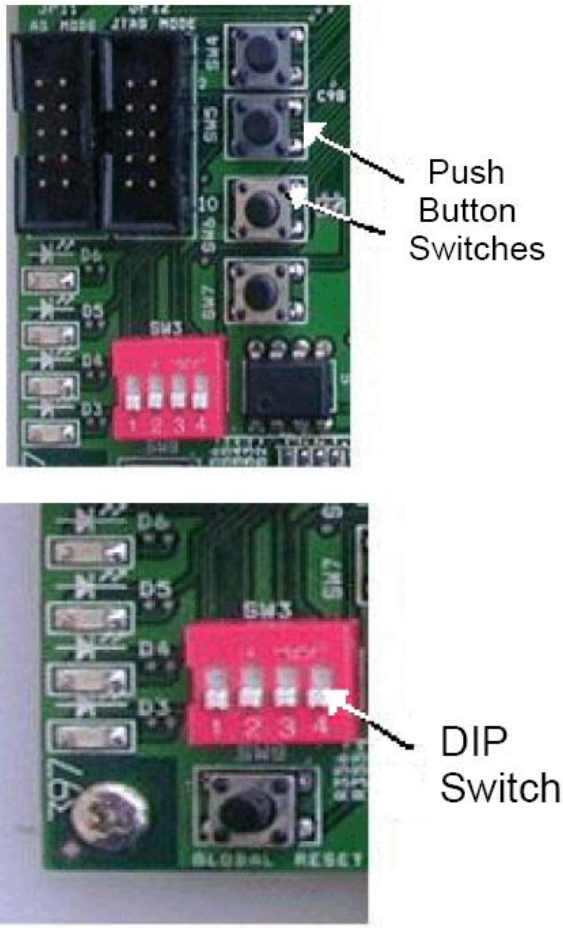


Fig. 2.23: Protezione sui push switch.

La macchina a stati ogni volta che rileva la pressione di un tasto, cambia stato, invia il segnale di Ack a tutti i flip-flop di protezione, e cambia l'uscita SELEZIONE in modo da variare i coefficienti del filtro.
Il segnale E è collegato a un piedino DIP 4 (vedi figura 2.24 b) per cui è necessaria una circuiteria per la sincronizzazione sensibile al livello (rappresentata in figura 2.25).



(a) (b)

Fig. 2.24: (a) Fotografia dei push switches; (b) Fotografia dei DIP switches

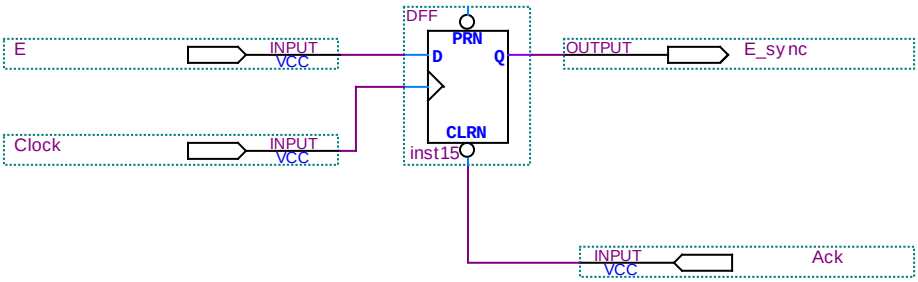


Fig. 2.25: Protezione sui DIP.

Pulsante	Selezione[ ]	Filtro
A	1	Passa alto
B	2	Passa basso
C	4	Taglia Banda
D	8	Enfatizza Medi
E (on)	16	No Filtro

Tabella 2.3: Corrispondenza pulsante-filtri

- **LUCI**

Il blocco LUCI ha il compito di accendere i 4 LED presenti sulla scheda UP3 in base al filtro selezionato.

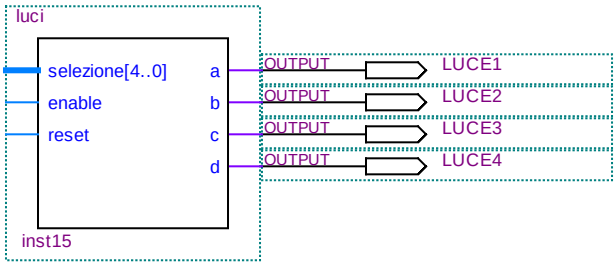


Fig. 2.26: Simbolo del blocco LUCI.

Nella tabella seguente viene esplicitata la corrispondenza tra il filtro selezionato e i LED attivi, indicando anche il valore della variabile SELEZIONE.

Selezione[ ]	Filtro	LED attivi
1	Passa alto	Solo LED 1
2	Passa basso	Solo LED 2
4	Taglia Banda	Solo LED 3
8	Enfatizza Medi	Solo LED 4
16	No Filtro	Tutti attivi

Tabella 2.4: Corrispondenza filtro-LED.

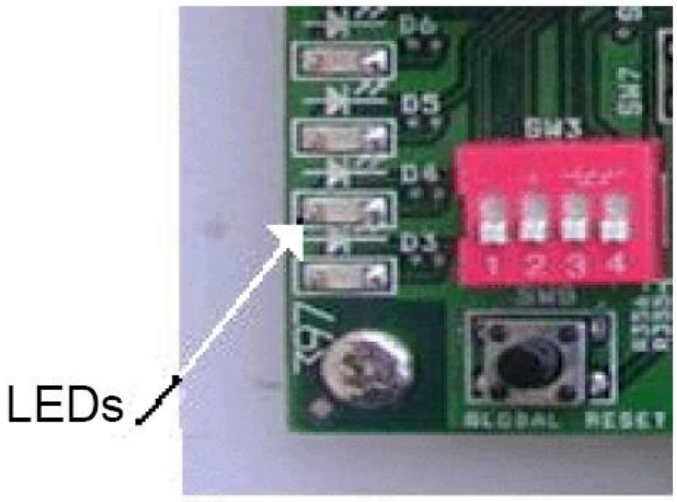


Fig. 2.27: Fotografia dei LED presenti sulla scheda UP3

2.3.8 Moltiplicatore e accumulatore

Il filtro FIR necessita di un N moltiplicatori e di un sommatore ad N termini, dove N è il numero di TAP più uno.

Siccome 32 (16 per ogni canale audio) moltiplicatori a 16 bit richiedono un numero enorme di sommatori, questo approccio rischia di saturare le risorse del FPGA. Per ovviare a questo problema la scelta è ricaduta sull'utilizzo di un solo moltiplicatore per ogni canale, che effettua serialmente le varie moltiplicazionie e pone i risultati parziali in un accumulatore.

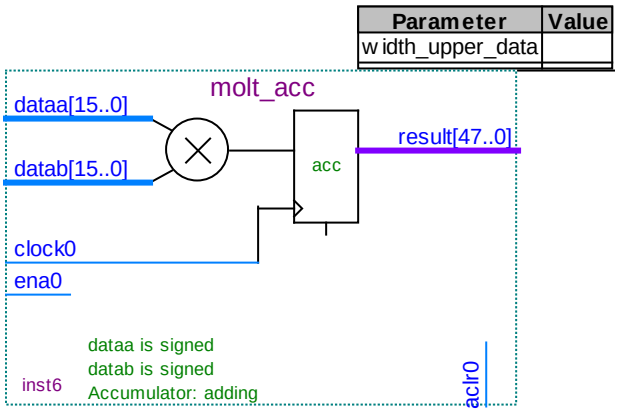


Fig. 2.28: Simbolo del moltiplicatore e accumulatore

Il segnale di azzeramento dell'accumulatore viene generato da una macchina a stati (Reset_acc) che, in base all'indirizzo in lettura alla ROM, genera sia un impulso di cancellazione dell'accumulatore, sia un impulso di data ready per la lettura della somma corretta. Il segnale di

azzeramento viene inviato anche ogni volta che viene attivato il segnale di reset (vedi figura 2.29).

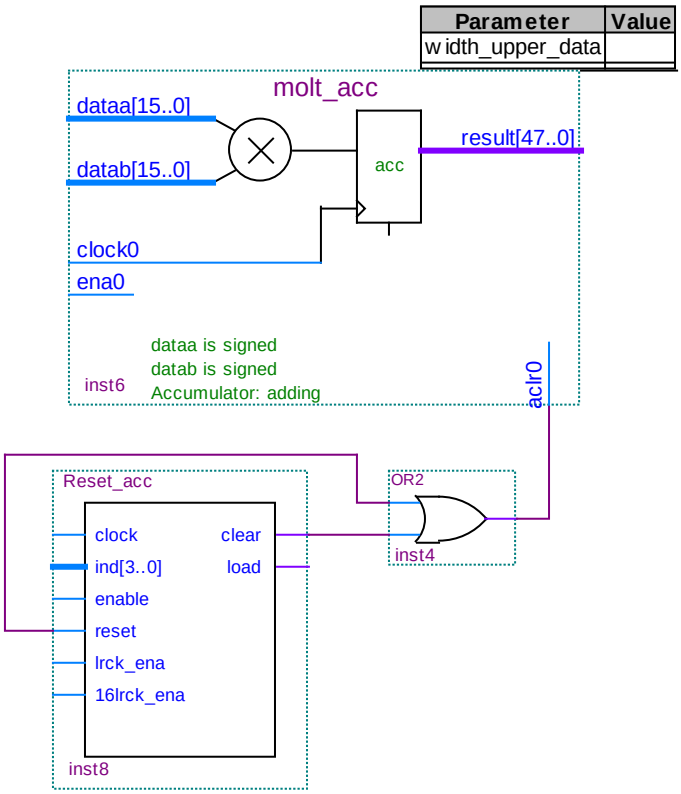


Fig. 2.29: Connessioni tra mol_acc e Reset_acc.

Siccome i coefficienti nella ROM sono moltiplicati per 2^{15}, è necessario dividere il segnale in uscita per lo stesso valore, realizzato in maniera semplice scartando i 15 bit meno significativi dal segnale in uscita dall'accumulatore.
Poi dei rimanenti 33 bit i 17 più significativi vengono scartati in maniera da ottenere un segnale di uscita su 16 bit, compatibile con l'ingresso del DAC.

2.3.9 Generazione sincronismi

Il sistema è clockato da USB_CLOCK, che è un clock a 48 MHz generato autonomamente dalla scheda UP3. Questo clock viene diviso per 4 da un prescaler (blocco divisore4) in maniera da ottenere un system clock di 12 MHz, clock di funzionamento del sistema.

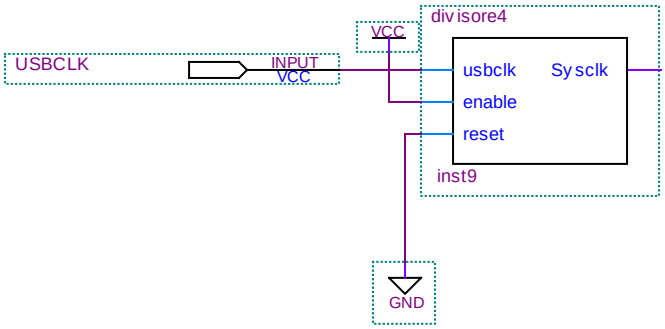


Fig. 2.30: Blocco divisore4.

Siccome la frequenza di system clock è 256 volte la frequenza di campionamento (f_S, pari a LRCK), ne segue che quest'ultima è pari a 46,875 kHz. Il clock di bit (BCK) è 64 volte la frequenza di campionamento e quindi pari a 3 MHz.

Il progetto sincrono statico prevede di utilizzare un clock unico per tutti i flip-flop. Dato che il sistema necessita di lavorare a velocità differenti, e soprattutto di generare dei segnali di clock per pilotare l'ADC e il DAC, è stato necessario realizzare un blocco (clocks) che genera i sincronismi che abilitano i vari blocchi.

In particolare, per il corretto funzionamento del sistema bisogna generare due clock: BCK (bit clock) e LRCK (Left/right clock) e 5 sincronismi interni.

Come pin in ingresso il blocco richiede:

- SYSCK: system clock, serve per clockare tutti i flip-flop interni;
- ENABLE: abilita il generatore di sincronismi;
- RESET: serve per resettare il generatore di sincronismi.

In uscita il blocco è in grado di generare:

- LRCK: che viene inviato in ingresso all'ADC e al DAC per il loro corretto funzionamento;
- BCK: che viene inviato in ingresso all'ADC e al DAC per il loro corretto funzionamento;
- Bck_ena: abilitazione interna; è attivo per un ciclo di system clock ogni fronte in salita di BCK;
- LrckR_ena: abilitazione interna; è attivo per un ciclo di system clock ogni fronte in salita di LRCK;

- LrckL_ena: abilitazione interna; è attivo per un ciclo di system clock ogni fronte in discesa di LRCK;
- 16lrck_ena: abilitazione interna; è attivo per un ciclo di system clock ogni $\frac{16}{T_c}$.

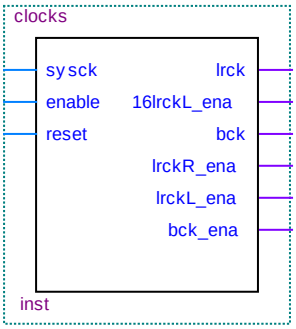


Fig. 2.31: Simbolo del blocco clocks

Per descrivere internamente la funzione svolta dal blocco abbiamo utilizzato il linguaggio testuale AHDL.

Sono stati utilizzati come primitive 3 contatori, uno modulo 4, uno modulo 16 ed uno modulo 256.

Ogni contatore serve per dividere sysck e generare un impulso di sincronismo quando viene rilevata una determinata configurazione dell'uscita.

Nel contatore modulo 4 viene rilevata la configurazione '2' e questa genera bck_ena, mentre il bit più significativo serve a generare BCK.

Nel contatore modulo 16 viene rilevata la configurazione '0' e questa genera 16lrck_ena.

Nel contatore modulo 256 viene rilevata la configurazione 0 e questa genera LrckL_ena, la configurazione 128 genera LrckR_ena, mentre il bit più significativo genera LRCK.

Nell'immagine 2.32 è rappresentata la temporizzazione dei segnali di ingresso e di uscita del blocco.

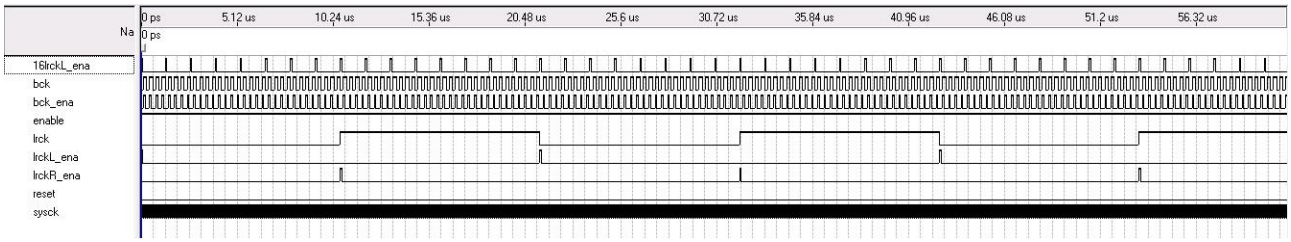


Fig. 2.32: Temporizzazione dei sincronismi.

2.3.10 Generazione segnali di controllo

- **Enable**

Il segnale di enable viene generato all'esterno attraverso il DIP 1 e, dopo aver attraversato un flip-flop di protezione, viene smistato all'interno del sistema.
L'enable viene inanzitutto inviato al blocco MEM_STATE ed al blocco Clocks. Per abilitare correttamente il Sipo_manager si è reso necessario effettuare l'AND tra il segnale di enable globale ed il sincronismo Bck_ena, come imposto dal progetto sincrono statico. Per l'abilitazione del Piso_manager invece è stato effettuato l'AND tra il segnale ENAOUT generato dal Sipo_manager, il segnale di enable globale (enable_g) ed il segnale di dis_out generato dal generatore di reset.

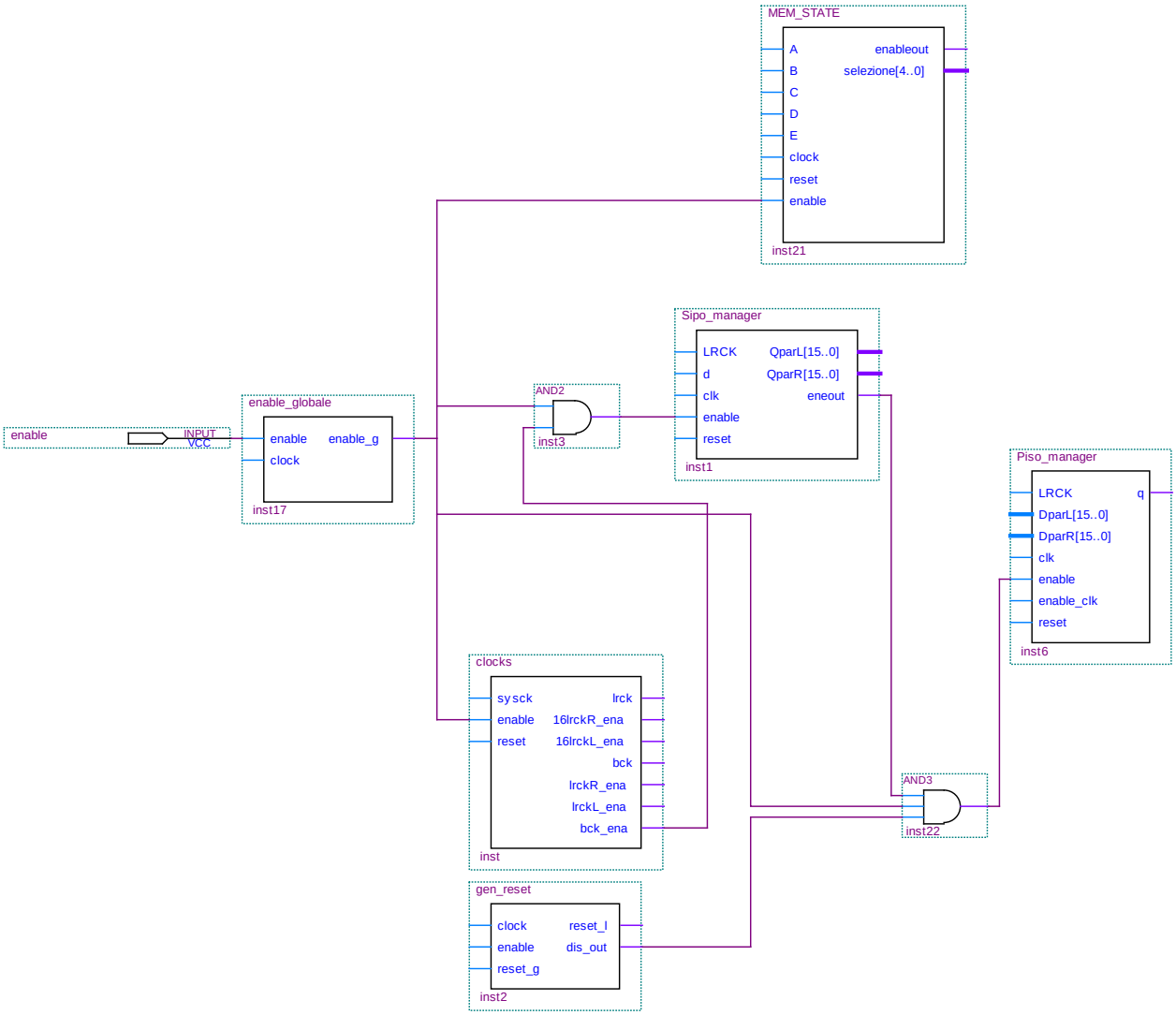


Fig. 2.33: Albero dell'enable.

- **Reset**

Il segnale di reset viene prelevato dall'integrato TL7705BCP della Texas Instruments. Questo integrato è un controllore della tensione di alimentazione, se questa scende sotto 3 V viene mandato un segnale di reset all'intera scheda UP3. Il segnale di reset viene inviato anche se viene premuto il pulsante global_reset presente sulla scheda UP3.

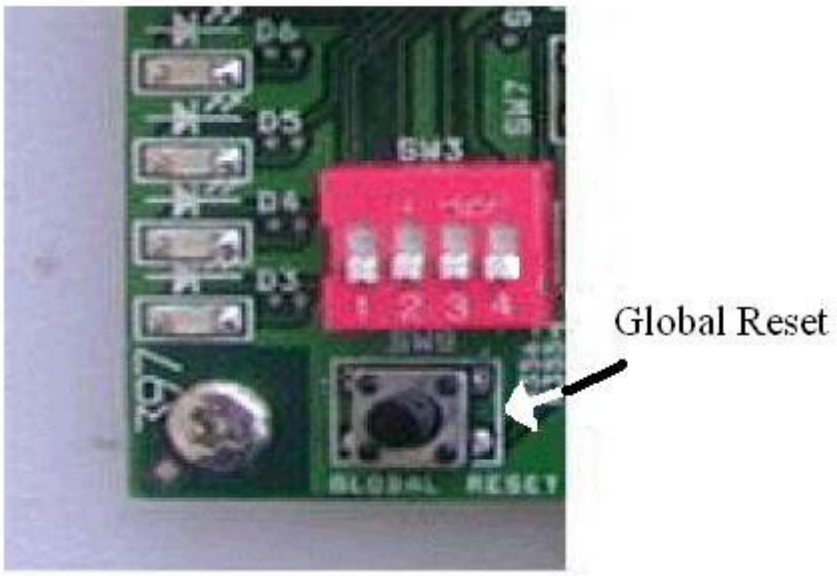


Fig. 2.34: Pulsante di global reset.

All'interno del FPGA il segnale di reset viene prima inviato ad un flip-flop di protezione, poi viene inviato ad una macchina a stati che a sua volta manda un segnale di reset a tutto il sistema. Questa macchina a stati serve anche per generare un impulso di reset all'avvio del sistema (subito dopo la programmazione dell'FPGA), e a disabilitare l'uscita in caso di reset.
La figura 2.35 rappresenta l'albero di distribuzione del reset.

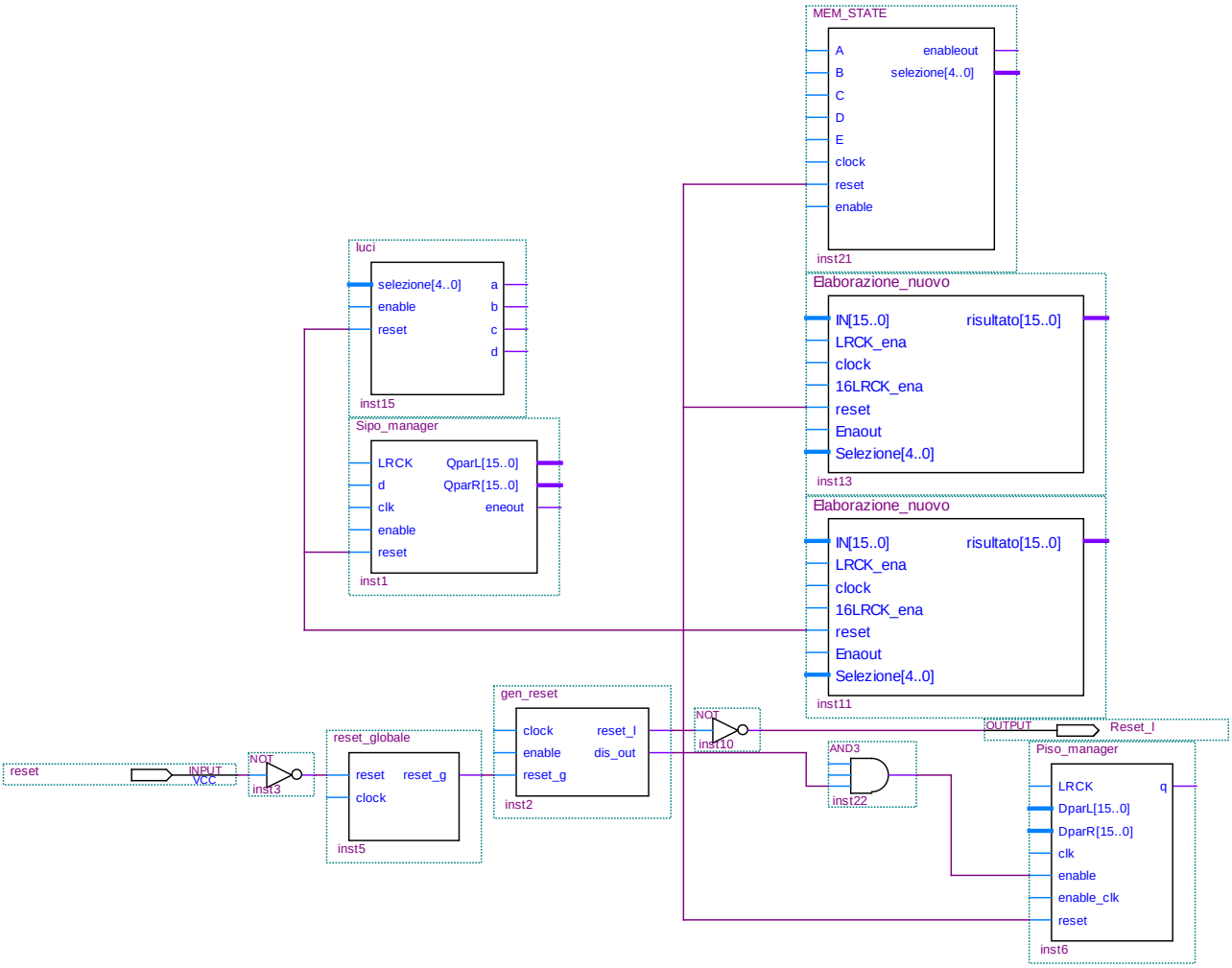


Fig. 2.35: Albero di distribuzione del reset.

Capitolo 3: Verifica e simulazione

Il partizionamento del sistema in vari sottoblocchi ha permesso una notevole semplificazione di tutta la fase di test e collaudo, permettendo la verifica dapprima di ogni singolo blocco indipendentemente dagli altri e poi, salendo di livello di partizionamento, del corretto assemblaggio e collegamento dei blocchi tra di loro.
La verifica del progetto può essere suddivisa in due fasi:

- Verifiche funzionali;
- Veriche sperimentali.

La prima ha richiesto l'esecuzione di simulazioni (dapprima funzionali e poi timing) e una valutazione di esse al fine di verificarne la consistenza.
La seconda è invece consistita in test sperimentali sul sistema e nel successivo confronto dei risultati con quelli ottenuti dalle simulazioni eseguite nella fase precedente.

3.1 Verifiche funzionali

- *Sipo_manager*

Per prima cosa si è simulato il funzionamento del blocco Sipo_manager. Sono stati generati dei bit in ingresso ed è stata verificata la correttezza della conversione seriale-parallelo e dell'instradamento verso il canale destro e sinistro.
In figura 3.1 è riportata la waveform risultante dalla simulazione.

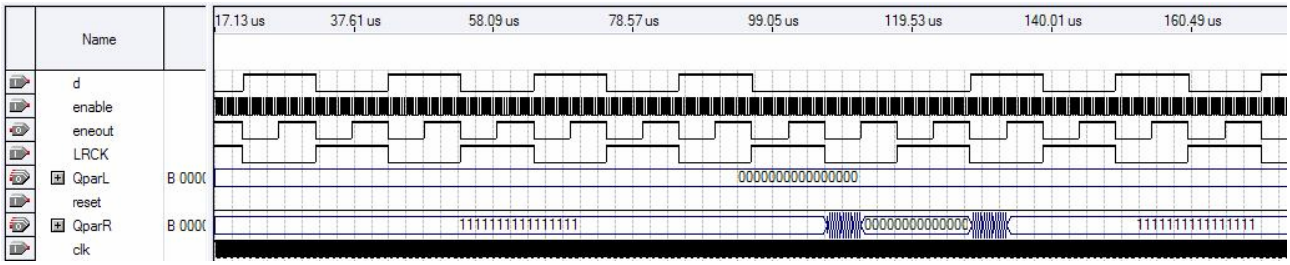


Fig. 3.1: Simulazione funzionale del Sipo_manager.

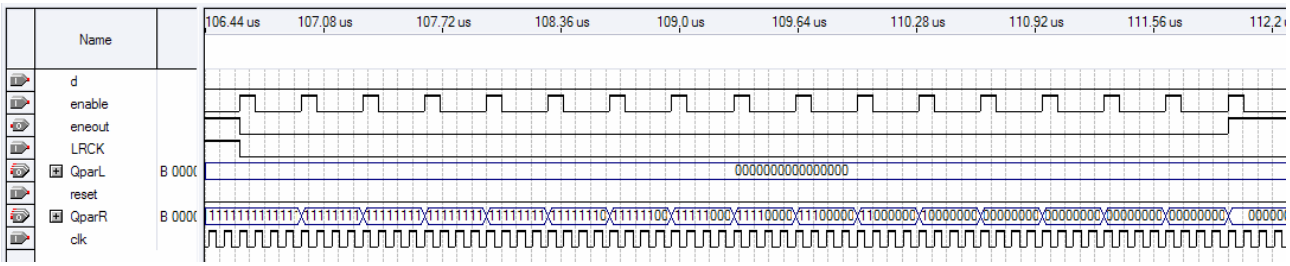


Fig. 3.2: Particolare della simulazione del Sipo_manager: caricamento del registro Sipo.

- *Piso_manager*

Successivamente è stato posto sotto esame il blocco Piso_manager. Anche in questo caso la simulazione è stata svolta inviando i segnali in ingresso e verificando l'instradamento seriale dei bit in uscita con la giusta temporizzazione.

In Figura 3.3 si mostra la relativa waveform generata dal CAD QUARTUS II.

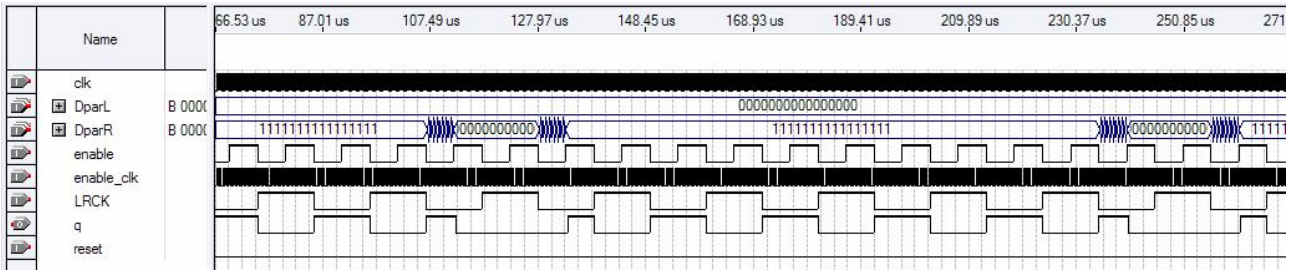


Fig. 3.3: Simulazione funzionale del Piso_manager.

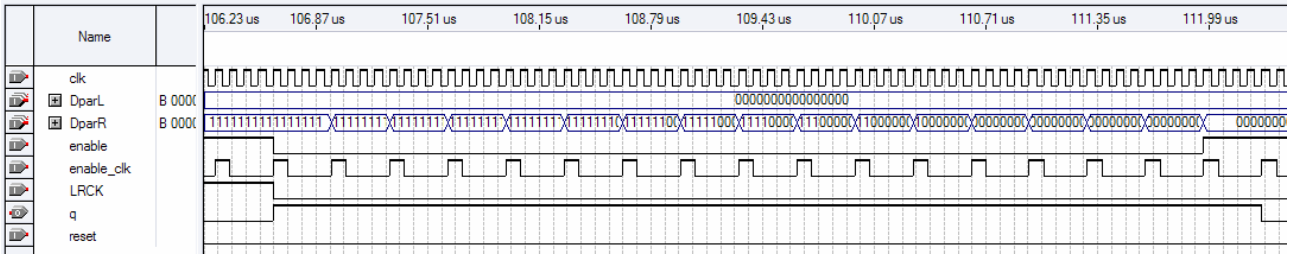


Fig. 3.4: Particolare della simulazione del Piso_manager: scaricamento del registro Piso.

- *Generatore sincronismi*

Il generatore di sincronismi è stato analizzato a fondo al fine di verificare che i segnali generati a partire da SYSCK in ingresso (generato a sua volta dal prescaler interno) avessero i giusti rapporti di fase e di frequenza tra di loro. Inoltre questo blocco deve generare anche due clocks da mandare al DAC e all'ADC, anch'essi in fase con i segnali di sincronismo interni.

In figura 3.5 è rappresentato l'esito della simulazione.

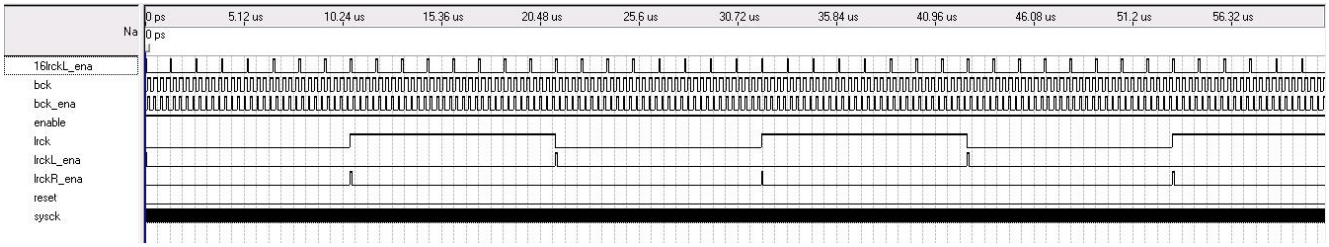


Fig. 3.5: Simulazione funzionale di Clocks.

- *Verifica interfacciamento ADC-DAC*

Dopo aver verificato singolarmente ogni blocco, è stata effettuata una verifica a più alto livello, collegando insieme i blocchi Sipo_manager, Piso_manager e il generatore di sincronismi. È stata aggiunta una barriera di flip-flop tra il Sipo_manager ed il Piso_manager, utile alla corretta sincronizzazione dei segnali.

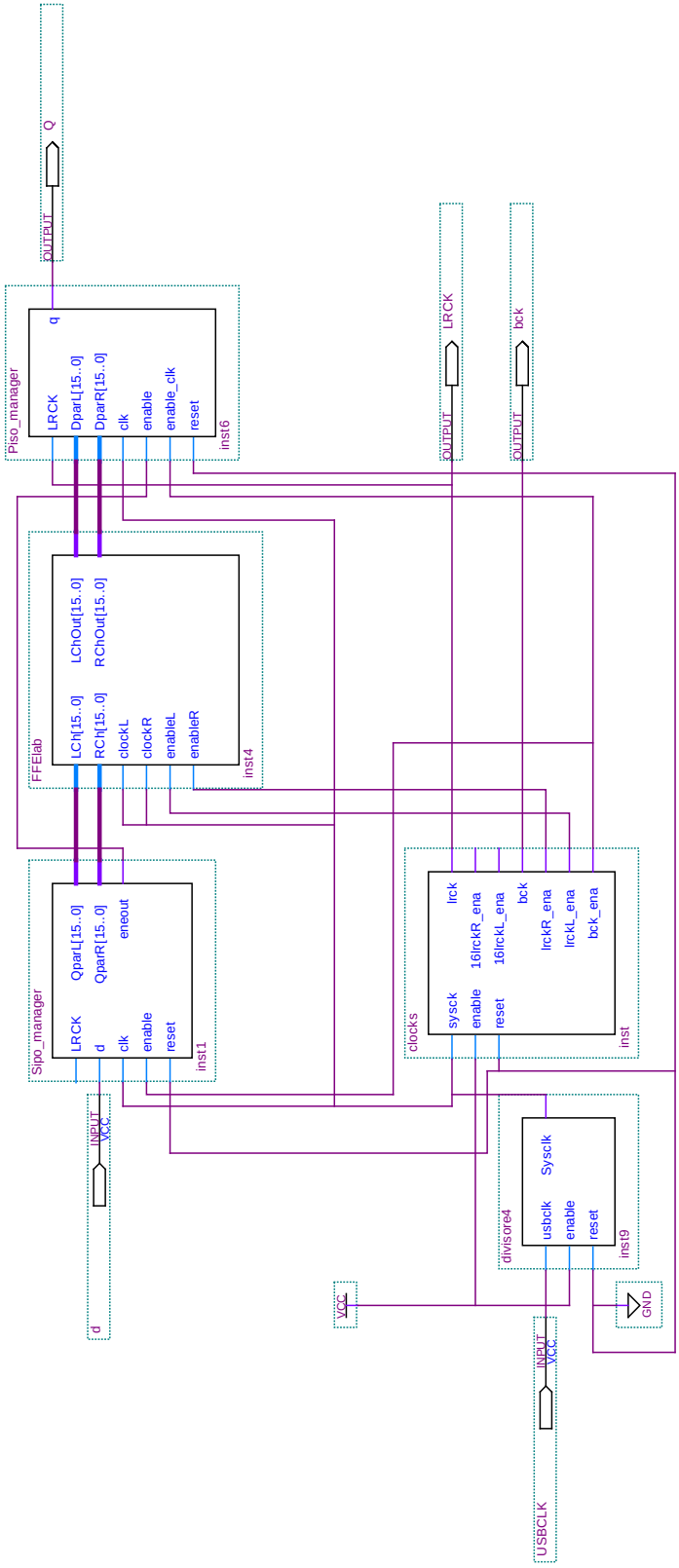


Fig. 3.6: Architettura implementata per la verifica dell'interfacciamento con ADC e DAC.

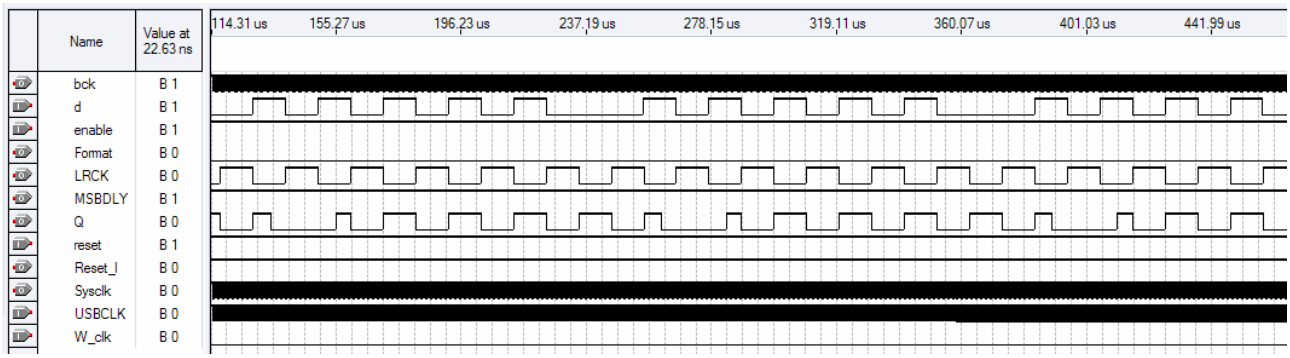


Fig. 3.7: Temporizzazione dell'interfacciamento ADC-DAC, relativo all'architettura di figura 3.6.

- *Generazione indirizzi*

Per verificare la correttezza nella generazione degli indirizzi, è stato realizzato un vettore contatore e posto in ingresso al blocco elaborazione destro e sinistro. In ogni cella della memoria coefficienti 5 è stato memorizzato il valore dell'indirizzo della cella stessa. In questa maniera è stato possibile controllare in maniera semplice che il moltiplicatore-accumulatore realizzasse le giuste operazioni.

In figura 3.8 si evidenzia la temporizzazione della scrittura della memoria campioni cosi come descritta sopra.

In figura 3.9 si evidenziano invece la temporizzazione in lettura della memoria campioni e coefficienti, i segnali di controllo ed i risultati delle operazioni compiute dal moltiplicatore-accumulatore.

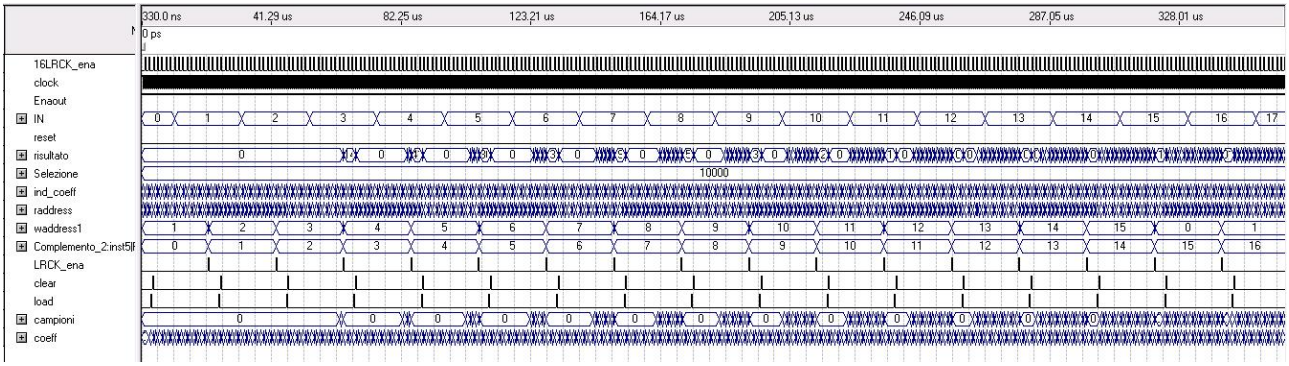


Fig. 3.8: a) Temporizzazione in scrittura sul canale sinistro.

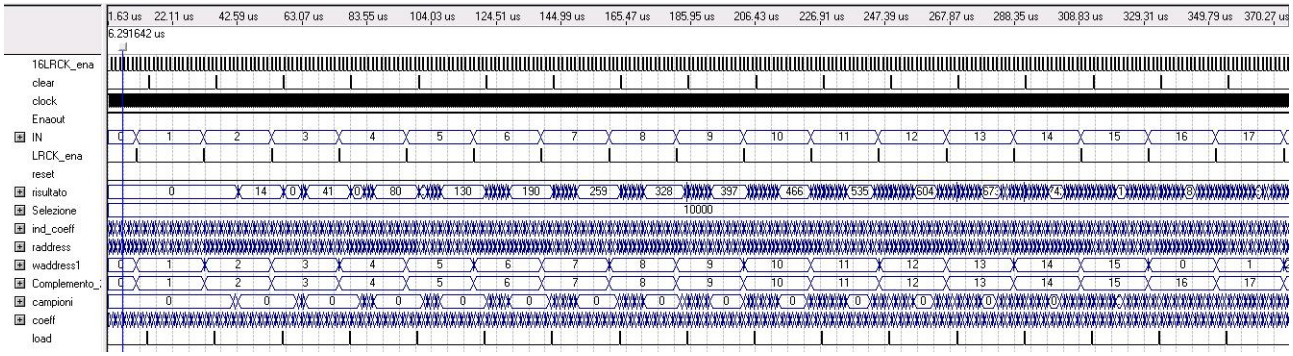


Fig. 3.8: b) Temporizzazione in scrittura sul canale destro.

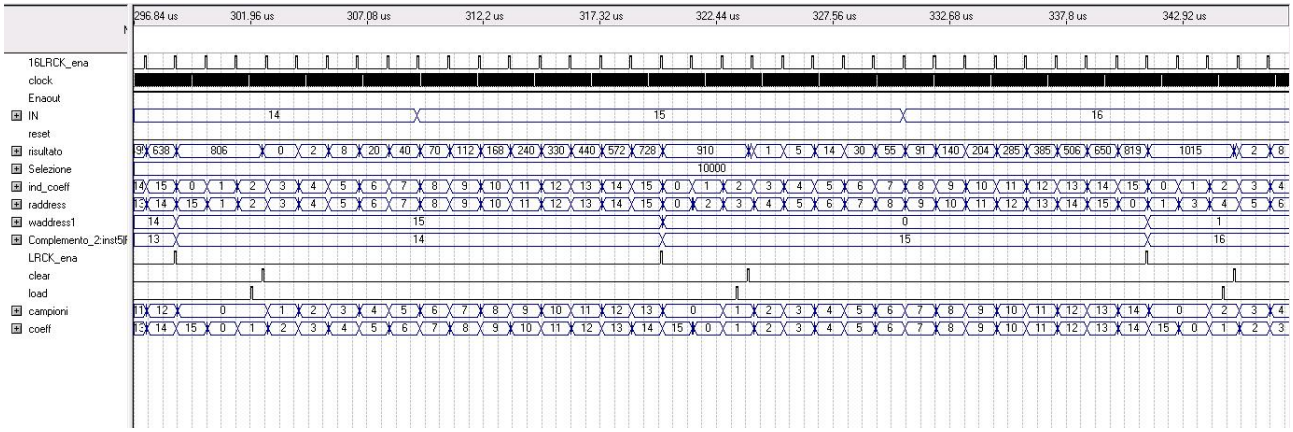


Fig. 3.9: a) Temporizzazione in lettura sul canale sinistro.

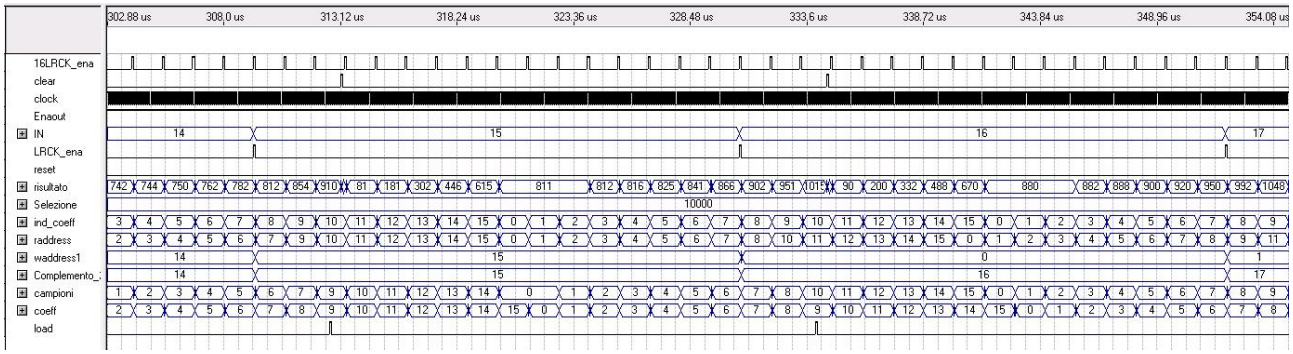


Fig. 3.9: b) Temporizzazione in lettura sul canale destro.

- *Elaborazione e filtraggio*

Questo blocco ha richiesto una fase di test molto più complessa degli altri, in quanto i segnali da portare in ingresso sono molti e complicati da generare, e, soprattutto, i segnali in uscita sono difficilmente interpretabili.

Sono stati controllati i contatori ed il sommatore adibiti alla generazione degli indirizzi, in maniera da verificare il corretto indirizzamento delle memorie. Successivamente sono state controllate le latenze tra i vari segnali in maniera da garantire l'arrivo contemporaneo al moltiplicatore sia del campione che del coefficiente giusto.

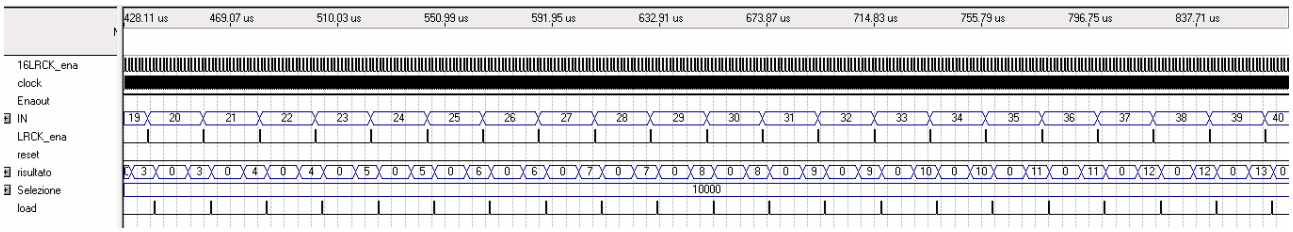


Fig. 3.10: Simulazione del blocco elaborazione con filtraggio disabilitato e senza il convertitore a complemento a due.

- *Selezione filtro*

La macchina a stati adibita alla selezione del filtro è stata verificata per garantirne il corretto funzionamento, sia nell'acquisizione dell'informazione sul tasto premuto, sia nella generazione dei segnali di controllo per gli altri blocchi.

- *Intero progetto*

Infine si è proceduto alla simulazione dell'intero sistema. In figura 3.11 b) è riportata la temporizzazione ottenuta utilizzando lo stesso segnale di ingresso utilizzato per la verifica dell'interfacciamento con ADC e DAC. Si noti che in tale simulazione (vedi figura 3.11 a) è stata selezionata la memoria coefficienti 5 (ovvero "No Filtro"), quindi in uscita si ottengono gli stessi dati in ingressi, a meno dei primi due bit di ogni parola, che vengono negati. La negazione del bit più significativo è conseguenza della trasformazione in complemento a due, mentre quella del secondo bit dipende dalla divisione per due effettuata dal "filtro passa tutto".

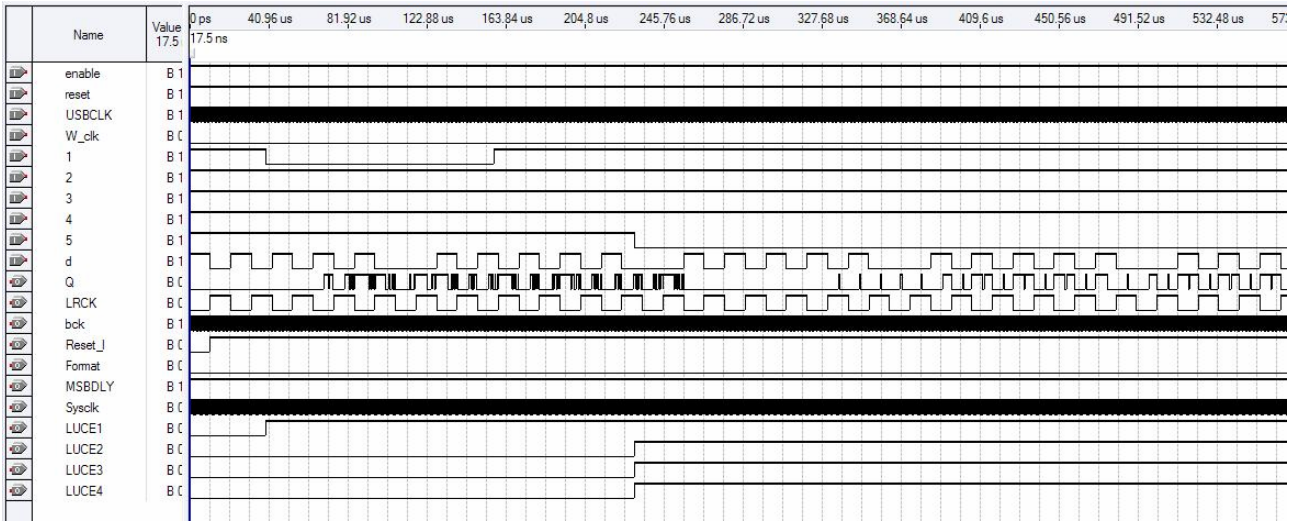


Fig. 3.11 a): Simulazione dell'intero sistema: particolare della selezione del filtro.

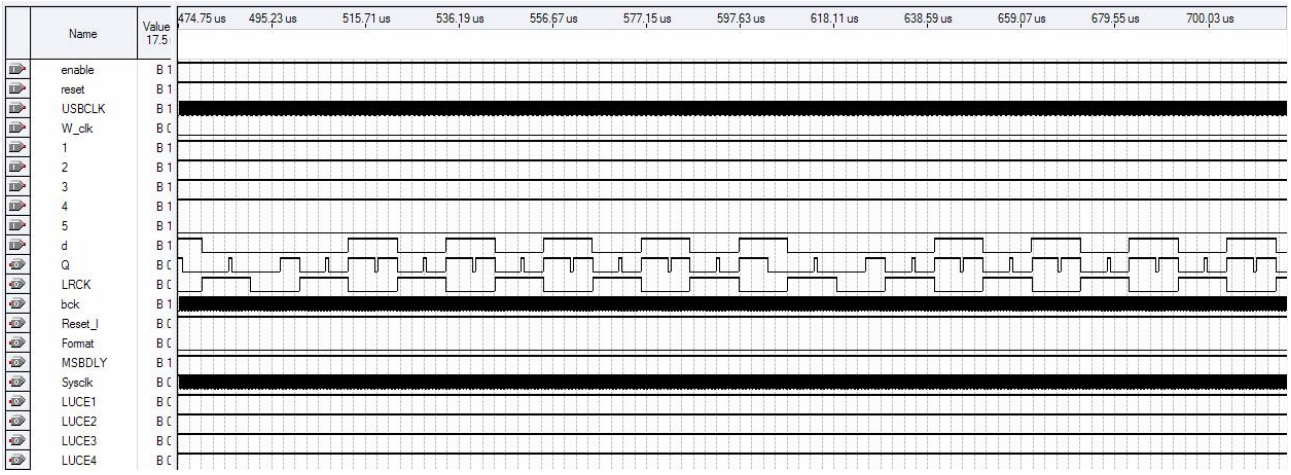


Fig. 3.11 b): Simulazione dell'intero sistema.

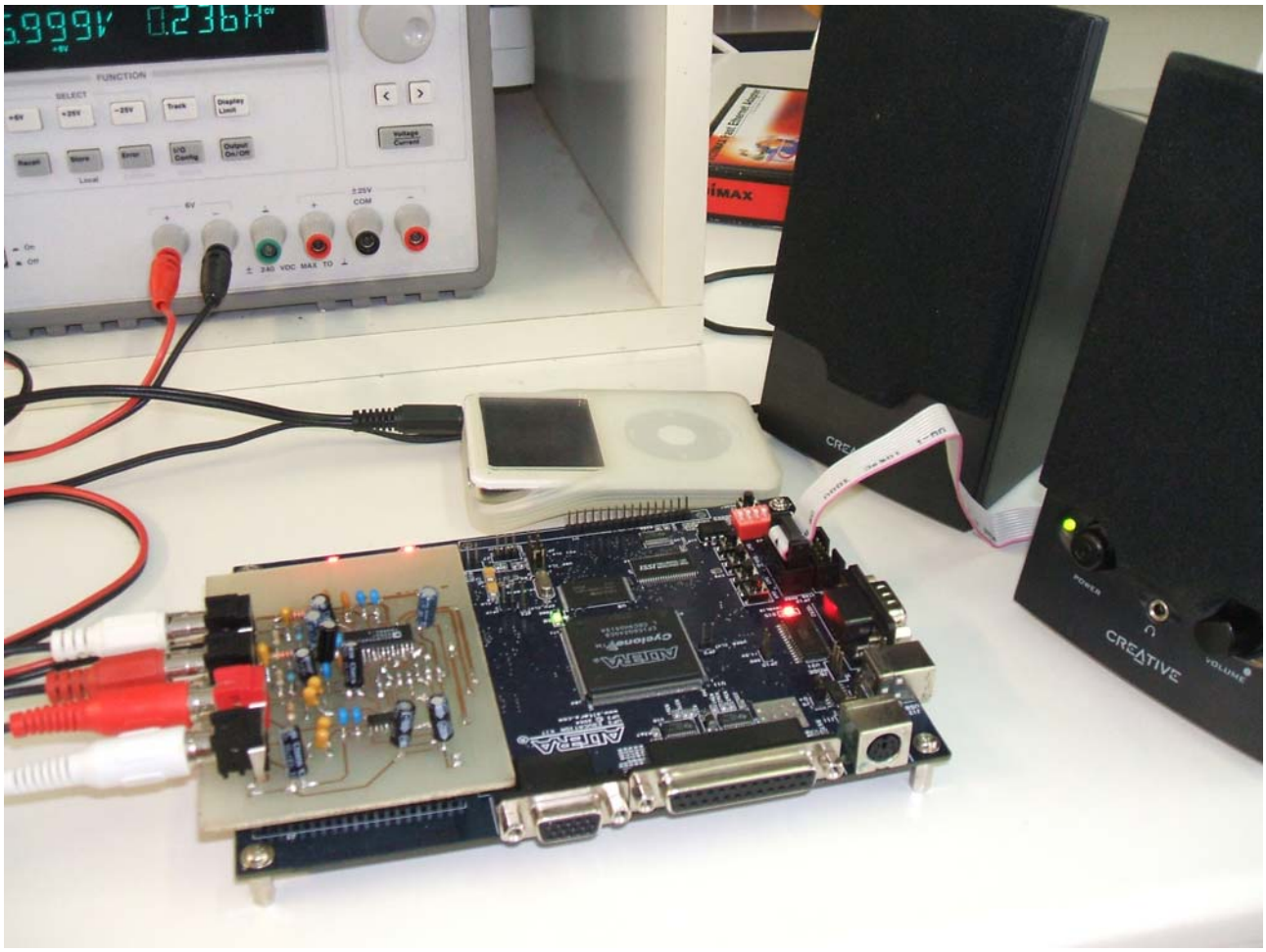
3.2 Verifiche sperimentali

- *Verifica strutturale scheda aggiuntiva e interfacciamento ADC-DAC*

La prima verifica sperimentale ha avuto come oggetto la scheda aggiuntiva. Subito dopo la sua realizzazione sono state controllare tutte le saldature e i collegamenti, successivamente è stato verificato il suo corretto inserimento sui connettori Santa Cruz della UP3 e poi si è passati al controllo dei segnali presenti su ogni nodo significativo attraverso l'oscilloscopio.

In particolare, dopo aver programmato l'FPGA in modo da implementare l'architettura di figura…., sono state controllate le frequenze, le fasi e le forme d'onda di ogni clock presente sulla scheda aggiuntiva e utile al funzionamento dei convertitori (LRCK, BCK, SYSCK) e di ogni piedino di controllo e di ogni ingresso o uscita.

Avendo avuto tutti i test descritti esito positivo si è passati al collaudo vero e proprio del sistema fino a quel punto progettato. Si sono collegati agli ingressi RCA dei riproduttore musicali (riproduttore mp3 e i-Pod) attraverso la loro uscita cuffie e in uscita delle casse amplificate. Il risultato dell'esperimento è stato quello di sentire con una buona qualità dalle casse (o dalle cuffie ad esse collegate) i brani musicali riprodotti dai music players in ingresso, quindi esattamente ciò che ci si aspettava.



- *Verifica elaborazione numerica*

La verifica sperimentale della corretta implementazione di ciascun filtro FIR è stata eseguita in due fasi.

La prima fase ha previsto l'utilizzo del generatore di forme d'onda HP33120A, con il quale sono stati inviati diversi toni puri, ovvero sinusoidi a diverse frequenze ed ampiezza fissata, in modo da verificarne in uscita, attraverso l'oscilloscopio, il giusto filtraggio (e quindi l'attenuazione prevista dallo specifico filtro). Inoltre è stata utilizzata la funzione SWEEP del generatore di segnali in modo da valutare in tempo reale l'attenuazione del segnale in ingresso su tutto lo spettro musicale. Le stesse verifiche sono state effettuate anche in maniera diretta, ascoltando in cuffia il segnale di uscita.

La seconda fase è consistita nel porre in ingresso un segnale musicale e nell'ascoltare attentamente l'uscita prodotta dal sistema, in modo da poter verificare "ad orecchio" l'effetto del filtraggio.

- *Verifica selezione filtro, luci e pulsanti di iclone o*

Infine è stata verificata la corretta selezione del filtro desiderato attraverso i pulsanti definiti in fase di progetto, l'accensione del LED o dei LEDs corrispondenti, il corretto funzionamento dei pulsanti di controllo ENABLE e RESET.

Conclusioni

Il dispositivo presentato in questa relazione ha raggiunto gli obiettivi prefissati.
Il sistema è in grado di acquisire un segnale audio stereo, convertirlo in un segnale digitale grazie all'utilizzo dell'integrato AD1870 di Analog Devices, filtrarlo utilizzando un filtro FIR implementato in hardware sul FPGA □iclone ed infine riconvertirlo in analogico utilizzando l'integrato PCM1725 di Texas Instruments. È Possibile, grazie alla pressione di determinati tasti presenti sulla scheda di sviluppo UP3 di variare il tipo di filtraggio scegliendo tra 4 possibili soluzioni. È anche possibile disabilitare il filtraggio e rendere il sistema trasparente ai segnali audio in ingresso.
L'utilizzo di convertitori a 16 bit e di una frequenza di campionamento di 46875 Hz, che risulta addirittura superiore a quella utilizzata nei dispositivi audio commerciali (44100 Hz), consentono di avere un segnale in uscita di buona qualità.
Il numero ridotto di TAP, però, non permette di realizzare un filtraggio molto selettivo; malgrado ciò si è riuscito ad ottenere un effetto filtrante abbastanza evidente all'udito.
Punto a favore del progetto è invece la flessibilità di utilizzo: è infatti possibile collegare in ingresso al sistema la stragrande maggioranza dei riproduttori musicali esistenti sul mercato (purché dispongano di un'uscita analogica); stessa cosa vale per i dispositivi di ascolto che è possibile collegare in uscita.
Si ritiene infine che possibili sviluppi futuri del sistema possono essere mirati a un aumento del numero di TAP dei filtri FIR e a una successiva accurata progettazione di essi, al fine di ottimizzare la loro risposta in frequenza.

Appendice A: File AHDL e schema a blocchi

Interfacciamento ADC/DAC

- *Interfaccia con ADC*

```
TITLE “Sipo_manager”;

INCLUDE “Sipo.inc”;
INCLUDE “32counter.inc”;
SUBDESIGN Sipo_manager
(LRCK, d, clk, enable, reset:INPUT;
QparL[15..0], QparR[15..0], eneout:OUTPUT;)

VARIABLE sip[1..0]:Sipo; count:32counter; ena:NODE;

BEGIN
sip[].d=d;
sip[0].enable=LRCK & ena & enable;
sip[1].enable=!LRCK & ena & enable;
sip[].reset=reset;

sip[].clk=clk;
QparL[]=sip[0].Qpar[];
QparR[]=sip[1].Qpar[];

count.clock=clk;
count.enable=enable;
count.reset=reset;

IF !count.count[4] THEN ena=VCC;
ELSE ena=GND;
END IF;
eneout=!ena;
END;
```

- *Serial in Parallel out*

```
TITLE “Sipo”;

SUBDESIGN Sipo
(d, clk, reset, enable: INPUT;
Qpar[15..0]: OUTPUT;)

VARIABLE FF[15..0]: DFFE;
BEGIN
FF[].clk=clk;
FF[].clrn=!reset;
FF[0].d=d;
FF[1].d=FF[0].q;
FF[2].d=FF[1].q;
FF[3].d=FF[2].q;
FF[4].d=FF[3].q;
FF[5].d=FF[4].q;
FF[6].d=FF[5].q;
FF[7].d=FF[6].q;
FF[8].d=FF[7].q;
FF[9].d=FF[8].q;
```

```
FF[10].d=FF[9].q;
FF[11].d=FF[10].q;
FF[12].d=FF[11].q;
FF[13].d=FF[12].q;
FF[14].d=FF[13].q;
FF[15].d=FF[14].q;
Qpar[]=FF[].q;
FF[].ena=enable;

END;
```

- *Interfaccia con DAC*

```
TITLE “Piso_manager”;

INCLUDE “Piso.inc”;

SUBDESIGN Piso_manager
(LRCK, DparL[15..0], DparR[15..0], clk, enable,enable_clk, reset:INPUT;
q:OUTPUT;)

VARIABLE pis[1..0]:Piso;

BEGIN

pis[0].enable=LRCK & enable;
pis[1].enable=!LRCK & enable;
pis[].reset=reset;
pis[].clk=clk;
pis[0].d[]=DparL[];
pis[1].d[]=DparR[];
pis[].enable_clk=enable_clk;


pis[0].load=LRCK & !enable;
pis[1].load=!LRCK & !enable;



IF LRCK THEN q=pis[0].q;
ELSE q=pis[1].q;
END IF;
END;
```

- *Parallel in serial out*

```
TITLE “Piso”;

SUBDESIGN Piso
(d[15..0], clk, reset, enable, enable_clk,load: INPUT;
q: OUTPUT;)

VARIABLE FF[15..0]: DFFE; Ffout:DFFE;
BEGIN
FF[].clk=clk;
FF[].clrn=!reset;
q=Ffout.q;
Ffout.clk=clk;
Ffout.clrn=!reset;
```

```
IF load THEN
FF[].d=d[];
FF[].ena=enable_clk;
Ffout.ena=GND;
ELSE
FF[].ena=enable & enable_clk;
FF[1].d=FF[0].q;
FF[2].d=FF[1].q;
FF[3].d=FF[2].q;
FF[4].d=FF[3].q;
FF[5].d=FF[4].q;
FF[6].d=FF[5].q;
FF[7].d=FF[6].q;
FF[8].d=FF[7].q;
FF[9].d=FF[8].q;
FF[10].d=FF[9].q;
FF[11].d=FF[10].q;
FF[12].d=FF[11].q;
FF[13].d=FF[12].q;
FF[14].d=FF[13].q;
FF[15].d=FF[14].q;
Ffout.d=FF[15].q;
Ffout.ena=enable & enable_clk;
END IF;
END;
```

Generazione sincronismi

- *Clocks*

```
TITLE “clocks”;

INCLUDE “256counter_enable2.inc”;
INCLUDE “4counter_enable2.inc”;
INCLUDE “16counter_enable2.inc”;
SUBDESIGN clocks
(sysck,enable, reset:INPUT;
lrck, 16lrckL_ena, bck, lrckR_ena, lrckL_ena, bck_ena:OUTPUT;)

VARIABLE
256count:256counter_enable2;4count:4counter_enable2;16count:16counter_enable2
;

BEGIN
4count.clock=sysck;
256count.clock=sysck;
16count.clock=sysck;
lrck=256count.count[7];
bck=!4count.count[1];
4count.enable=enable;
256count.enable=enable;
16count.enable=enable;
4count.reset=reset;
256count.reset=reset;
16count.reset=reset;


IF 16count.count[]==0 THEN
16lrckL_ena=VCC;
ELSE 16lrckL_ena=GND;
```

```
END IF;
IF 256count.count[]==128 THEN
lrckR_ena=VCC;
ELSE lrckR_ena=GND;
END IF;
IF 256count.count[]==0 THEN
lrckL_ena=VCC;
ELSE lrckL_ena=GND;
END IF;
IF 4count.count[]==0 THEN
bck_ena=VCC;
ELSE bck_ena=GND;
END IF;

END;
```

- *Generatore di reset*

```
TITLE “gen_reset”;

SUBDESIGN gen_reset
(clock, enable, reset_g:INPUT;
reset_l,dis_out:OUTPUT;)

VARIABLE ss:MACHINE WITH STATES (s0, s1, s2);

BEGIN
CASE ss IS

WHEN s0 => reset_l=VCC; dis_out=GND; ss=s1;
WHEN s1 => reset_l=GND; dis_out=GND; ss=s2;
WHEN s2 => reset_l=GND; dis_out=VCC; IF (reset_g==VCC) THEN ss=s0; END IF;
END CASE;

ss.clk=clock;
ss.ena=enable;
END;
```

- *Flip-Flop di protezione sul reset*

```
TITLE “RESET_globale”;

SUBDESIGN reset_globale
(reset, clock:INPUT;
reset_g:OUTPUT;)
VARIABLE  ff_sync:DFF;

BEGIN

ff_sync.d=reset;
ff_sync.clk=clock;
reset_g=ff_sync.q;

END;
```

- *Flip-Flop di protezione sull’enable*

```
TITLE “Enable_globale”;
```

```
SUBDESIGN enable_globale
(enable, clock:INPUT;
enable_g:OUTPUT;)
VARIABLE  ff_sync:DFF;

BEGIN
ff_sync.d=enable;
ff_sync.clk=clock;
enable_g=ff_sync.q;

END;
```

Selezione Filtro

- *Selezione dei filtri*

```
TITLE “MEM_STATE”;

SUBDESIGN MEM_STATE
(A,B,C,D,E,clock,reset,enable:INPUT;
enableout, selezione[4..0]:OUTPUT;)

VARIABLE ack:NODE; FF_sync[4..0]:DFF; FF_async[3..0]:DFF; ss:MACHINE WITH
STATES(s0,s1,s2,s3,s4,s5,s1idle,s2idle,s3idle,s4idle,s5idle);

BEGIN
ff_async[].d=VCC;
ff_async[0].clk=!A;
ff_async[1].clk=!B;
ff_async[2].clk=!C;
ff_async[3].clk=!D;
ff_sync[0].d=ff_async[0].q;
ff_sync[1].d=ff_async[1].q;
ff_sync[2].d=ff_async[2].q;
ff_sync[3].d=ff_async[3].q;
ff_sync[4].d=!E;
ff_sync[].clk=clock;
ff_async[].clrn=ack;

IF enable THEN
CASE ss IS
WHEN s0 => selezione[]=0; enableout=GND; ack=VCC; IF reset THEN ss=s0; ELSIF
ff_sync[4].q THEN ss=s0; ELSIF ff_sync[0].q THEN ss=s1; ELSIF ff_sync[1].q
THEN ss=s2; ELSIF ff_sync[2].q THEN ss=s3; ELSIF ff_sync[3].q THEN ss=s4; END
IF;
WHEN s1 => enableout=VCC; selezione[]=1; ack=GND; ss=s1idle;
WHEN s2 => enableout=VCC; selezione[]=2; ack=GND; ss=s2idle;
WHEN s3 => enableout=VCC; selezione[]=4; ack=GND; ss=s3idle;
WHEN s4 => enableout=VCC; selezione[]=8; ack=GND; ss=s4idle;
WHEN s5 => enableout=VCC; selezione[]=16; ack=GND; ss=s5idle;
WHEN s1idle => enableout=VCC; selezione[]=1; ack=VCC; IF reset THEN ss=s0;
ELSIF ff_sync[4].q THEN ss=s5; ELSIF ff_sync[1].q THEN ss=s2; ELSIF
ff_sync[2].q THEN ss=s3; ELSIF ff_sync[3].q THEN ss=s4; END IF;
WHEN S2idle => enableout=VCC; selezione[]=2; ack=VCC; IF reset THEN ss=s0;
ELSIF ff_sync[4].q THEN ss=s5; ELSIF ff_sync[0].q THEN ss=s1; ELSIF
ff_sync[2].q THEN ss=s3; ELSIF ff_sync[3].q THEN ss=s4; END IF;
WHEN s3idle => enableout=VCC; selezione[]=4; ack=VCC; IF reset THEN ss=s0;
ELSIF ff_sync[4].q THEN ss=s5; ELSIF ff_sync[0].q THEN ss=s1; ELSIF
ff_sync[1].q THEN ss=s2; ELSIF ff_sync[3].q THEN ss=s4; END IF;
```

```
WHEN s4idle => enableout=VCC; selezione[]=8; ack=VCC; IF reset THEN ss=s0;
ELSIF ff_sync[4].q THEN ss=s5; ELSIF ff_sync[1].q THEN ss=s2; ELSIF
ff_sync[2].q THEN ss=s3; ELSIF ff_sync[0].q THEN ss=s1; END IF;
WHEN s5idle => enableout=VCC; selezione[]=16; ack=VCC; IF reset THEN ss=s0;
ELSIF (ff_sync[0].q or ff_sync[1].q or ff_sync[2].q or ff_sync[3].q) &
!ff_sync[4].q THEN ss=s0; END IF;
END CASE;
END IF;


ss.clk=clock;
ss.reset=reset;
ss.ena=enable;
END;
```

- *gestore dei LED*

```
TITLE “LUCI”;

SUBDESIGN luci
(selezione[4..0],enable, reset:input;
a,b,c,d:output;)

BEGIN
IF reset THEN A=GND; B=GND; C=GND; D=GND;
ELSIF enable THEN
CASE selezione[] IS
WHEN 1 => A=VCC; B=GND; C=GND; D=GND;
WHEN 2 => A=GND; B=VCC; C=GND; D=GND;
WHEN 4 => A=GND; B=GND; C=VCC; D=GND;
WHEN 8 => A=GND; B=GND; C=GND; D=VCC;
WHEN 16 => A=VCC; B=VCC; C=VCC; D=VCC;
END CASE;
ELSE A=GND; B=GND; C=GND; D=GND;
END IF;
END;
```

Elaborazione

- *Memoria coefficienti 1*

```
TITLE “MEMORIA_COEFF”;

--FILTRO Passa Alto 10K—

SUBDESIGN Memoria_coeff
(ind[3..0], clock,enable:INPUT;
q[15..0]:OUTPUT;)

VARIABLE FF[15..0]:DFFE; uscita[15..0]:NODE;
BEGIN
FF[].clk=clock;
FF[].d=uscita[];
q[]=FF[].q;
FF[].ena=enable;
CASE ind[] IS

WHEN 0 => uscita[]=223;
WHEN 1 => uscita[]=B”1111110010011010”;
```

```
WHEN 2 => uscita[]=507;
WHEN 3 => uscita[]=1430;
WHEN 4 => uscita[]=B”1111111101001011”;
WHEN 5 => uscita[]=B”1111001000000011”;
WHEN 6 => uscita[]=B”1111010111101011”;
WHEN 7 => uscita[]=16432;
WHEN 8 => uscita[]=B”1011111111010000”;
WHEN 9 => uscita[]=2581;
WHEN 10 => uscita[]=3581;
WHEN 11 => uscita[]=181;
WHEN 12 => uscita[]=B”1111101001101010”;
WHEN 13 => uscita[]=B”1111111000000101”;
WHEN 14 => uscita[]=870;
WHEN 15 => uscita[]=B”1111111100100001”;

END CASE;
END;
```

- *Memoria coefficienti 2*

```
TITLE “MEMORIA_COEFF2”;

--FILTRO PASSA BASSO 100Hz—

SUBDESIGN Memoria_coeff2
(ind[3..0], clock,enable:INPUT;
q[15..0]:OUTPUT;)

VARIABLE FF[15..0]:DFFE; uscita[15..0]:NODE;
BEGIN
FF[].clk=clock;
FF[].d=uscita[];
q[]=FF[].q;
FF[].ena=enable;

CASE ind[] IS

WHEN 0 => uscita[]=14;
WHEN 1 => uscita[]=95;
WHEN 2 => uscita[]=351;
WHEN 3 => uscita[]=915;
WHEN 4 => uscita[]=1854;
WHEN 5 => uscita[]=3058;
WHEN 6 => uscita[]=4220;
WHEN 7 => uscita[]=4942;
WHEN 8 => uscita[]=4942;
WHEN 9 => uscita[]=4220;
WHEN 10 => uscita[]=3058;
WHEN 11 => uscita[]=1854;
WHEN 12 => uscita[]=915;
WHEN 13 => uscita[]=351;
WHEN 14 => uscita[]=95;
WHEN 15 => uscita[]=14;

END CASE;
END;
```

- *Memoria coefficienti 3*

```
TITLE “MEMORIA_COEFF3”;
```

```
--Filtro Taglia banda da 1K a 8K—

SUBDESIGN Memoria_coeff3
(ind[3..0], clock, enable:INPUT;
q[15..0]:OUTPUT;)

VARIABLE FF[15..0]:DFFE; uscita[15..0]:NODE;
BEGIN
FF[].clk=clock;
FF[].d=uscita[];
q[]=FF[].q;
FF[].ena=enable;
CASE ind[] IS

WHEN 0 => uscita[]=6790;
WHEN 1 => uscita[]=B”1110100000101010”;
WHEN 2 => uscita[]=5708;
WHEN 3 => uscita[]=B”1101110010011101”;
WHEN 4 => uscita[]=6321;
WHEN 5 => uscita[]=B”1101011011011000”;
WHEN 6 => uscita[]=B”1110011110010101”;
WHEN 7 => uscita[]=12480;
WHEN 8 => uscita[]=B”1110001100110111”;
WHEN 9 => uscita[]=B”1100100100001011”;
WHEN 10 => uscita[]=B”1101000011110011”;
WHEN 11 => uscita[]=723;
WHEN 12 => uscita[]=B”1111111110101100”;
WHEN 13 => uscita[]=B”1111100101111010”;
WHEN 14 => uscita[]=B”1110010011010111”;
WHEN 15 => uscita[]=B”1110101100010110”;


END CASE;
END;
```

- *Memoria coefficienti 4*

```
TITLE “MEMORIA_COEFF4”;

--Filtro Enfasi Sui Medi da 2K a 8K—


SUBDESIGN Memoria_coeff4
(ind[3..0], clock, enable:INPUT;
q[15..0]:OUTPUT;)

VARIABLE FF[15..0]:DFFE; uscita[15..0]:NODE;
BEGIN
FF[].clk=clock;
FF[].d=uscita[];
q[]=FF[].q;
FF[].ena=enable;
CASE ind[] IS

WHEN 0 => uscita[]=107;
WHEN 1 => uscita[]=B”1111110111001101”;
WHEN 2 => uscita[]=B”1111010111011000”;
WHEN 3 => uscita[]=B”1110111010001011”;
WHEN 4 => uscita[]=B”1110111111101101”;
WHEN 5 => uscita[]=B”1111110100100000”;
```

```
WHEN 6 => uscita[]=4227;
WHEN 7 => uscita[]=7869;
WHEN 8 => uscita[]=7869;
WHEN 9 => uscita[]=4227;
WHEN 10 => uscita[]=B”1111110100100000”;
WHEN 11 => uscita[]=B”1110111111101101”;
WHEN 12 => uscita[]=B”1110111010001011”;
WHEN 13 => uscita[]=B”1111010111011000”;
WHEN 14 => uscita[]=B”1111110111001101”;
WHEN 15 => uscita[]=107;

END CASE;
END;
```

- *Memoria coefficienti 5*

```
TITLE “MEMORIA_COEFF5”;

SUBDESIGN Memoria_coeff5
(ind[3..0], clock, enable:INPUT;
q[15..0]:OUTPUT;)

VARIABLE FF[15..0]:DFFE; uscita[15..0]:NODE;
BEGIN
FF[].clk=clock;
FF[].d=uscita[];
q[]=FF[].q;
FF[].ena=enable;
CASE ind[] IS

WHEN 1 => uscita[]=16384;
WHEN OTHERS => uscita[]=0;

END CASE;
END;
```

- *Multiplexer*

```
TITLE “MUX4x4”;

SUBDESIGN mux4x4
(datoA[15..0],datoB[15..0],datoC[15..0],datoD[15..0], datoE[15..0],
selezione[4..0],clock,enable, reset:INPUT;
uscita[15..0]:output;)
VARIABLE FF[4..0]:DFF;
BEGIN
ff[].clk=clock;
ff[].d=selezione[];

IF reset THEN uscita[]=0;
ELSIF enable THEN
CASE ff[].q IS
WHEN 1 => uscita[]=datoA[];
WHEN 2 => uscita[]=datoB[];
WHEN 4 => uscita[]=datoC[];
WHEN 8 => uscita[]=datoD[];
WHEN 16 => uscita[]=datoE[];
WHEN OTHERS => uscita[]=0;
END CASE;
ELSE uscita[]=0;
```

```
END IF;
END;
```

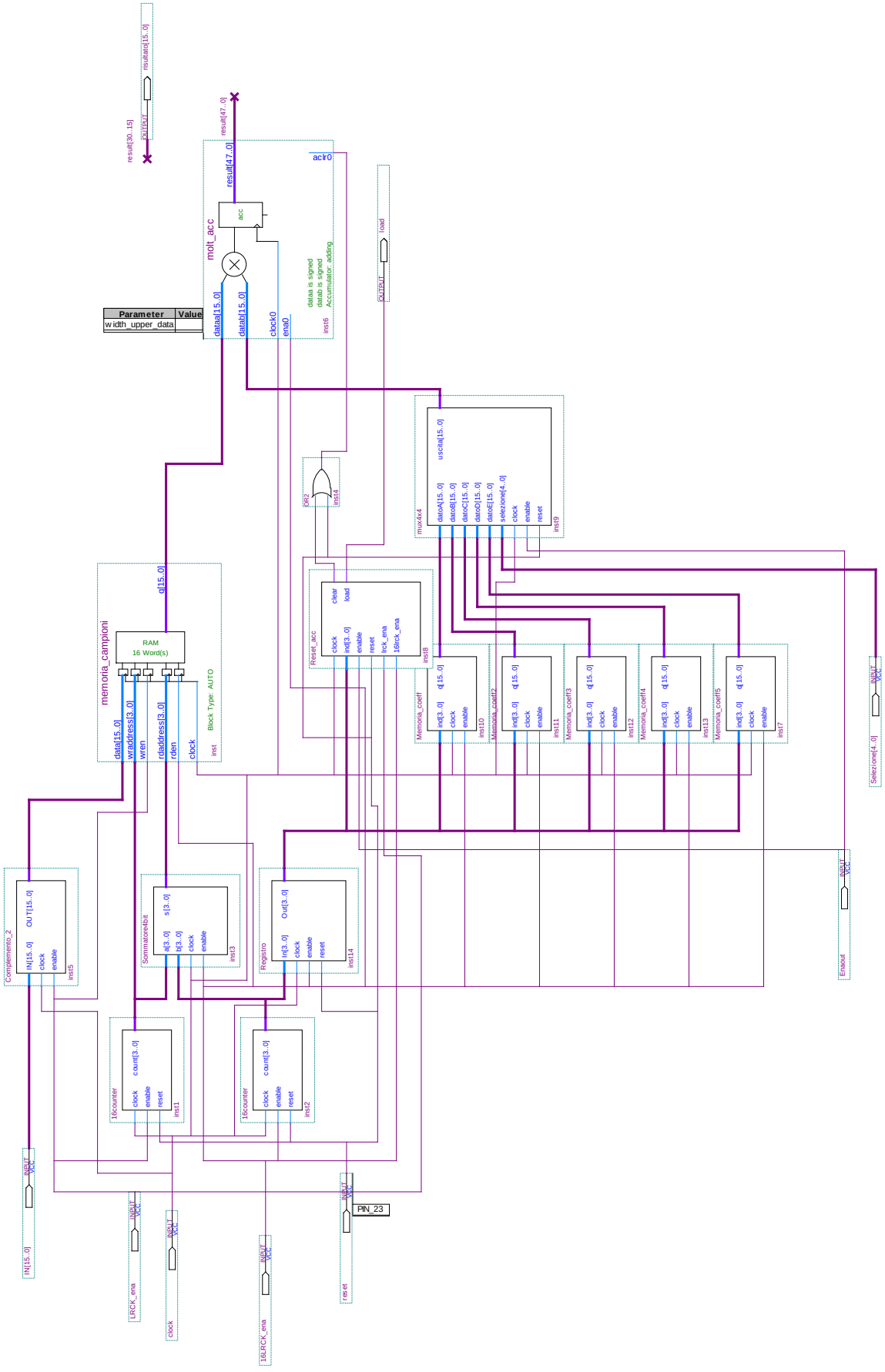
- *Generatore di clear e di load*

```
TITLE “Reset_acc”;

INCLUDE “16counter.inc”;
SUBDESIGN Reset_acc
(clock, ind[3..0], enable, reset,lrck_ena,16lrck_ena:INPUT;
clear, load:OUTPUT;)

VARIABLE  ss: MACHINE WITH STATES (s0, s1, s2, s3, s4, s5, s6,
s7);16count:16counter;

BEGIN
16count.clock=clock;
16count.enable=enable;


CASE ss IS
WHEN s0 => clear=GND; load=GND; 16count.reset=VCC; IF lrck_ena THEN ss=s1;
END IF;
WHEN s1 => clear=GND; load=GND; IF 16lrck_ena THEN ss=s2;END IF;
WHEN s2 => clear=GND; load=GND; IF 16lrck_ena THEN ss=s3; END IF;
WHEN s3 => clear=GND; load=GND; 16count.reset=VCC; 16count.enable=VCC; ss=s4;
WHEN s4 => clear=GND; load=GND; 16count.reset=GND; 16count.enable=VCC; IF
16count.count[]==6 THEN load=VCC; ss=s5; END IF;
WHEN s5 => clear=GND; 16count.enable=VCC; load=GND; ss=s6;
WHEN s6 => clear=GND; load=GND; 16count.enable=VCC; IF 16count.count[]==11
THEN ss=s7;END IF;
WHEN s7 => clear=VCC; load=GND; 16count.enable=VCC;  ss=s0;

END CASE;
ss.ena=enable;
ss.clk=clock;
ss.reset=reset;

END;
```

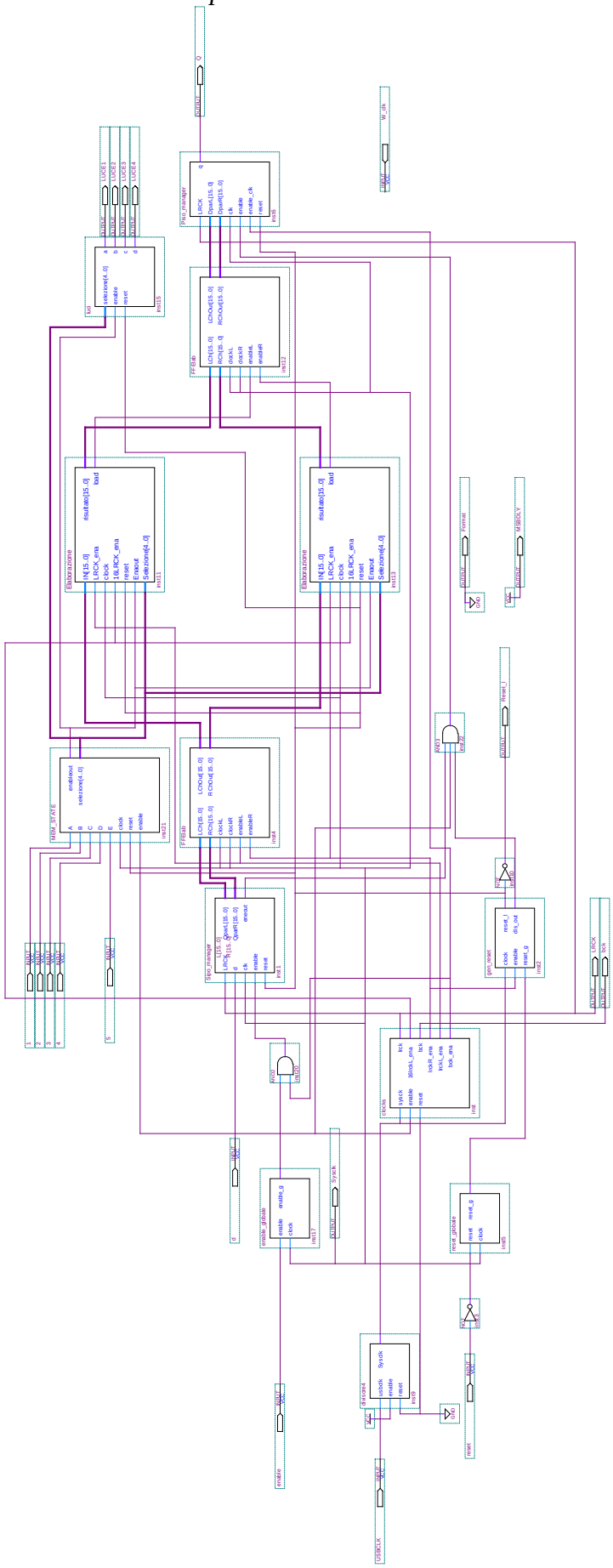
-Convertitore in complemento a 2

```
TITLE “Complemento_2”;

SUBDESIGN Complemento_2
(IN[15..0],clock,enable:INPUT;
OUT[15..0]:OUTPUT;)

VARIABLE FF[15..0]: DFFE;
BEGIN
FF[].clk=clock;
FF[].ena=enable;
FF[15].d=!IN[15];
FF[14..0].d=IN[14..0];

OUT[]=FF[].q;
END;
```

- *Blocco elaborazione*



Schema a blocchi a più alto livello



Appendice B: Temporizzazioni

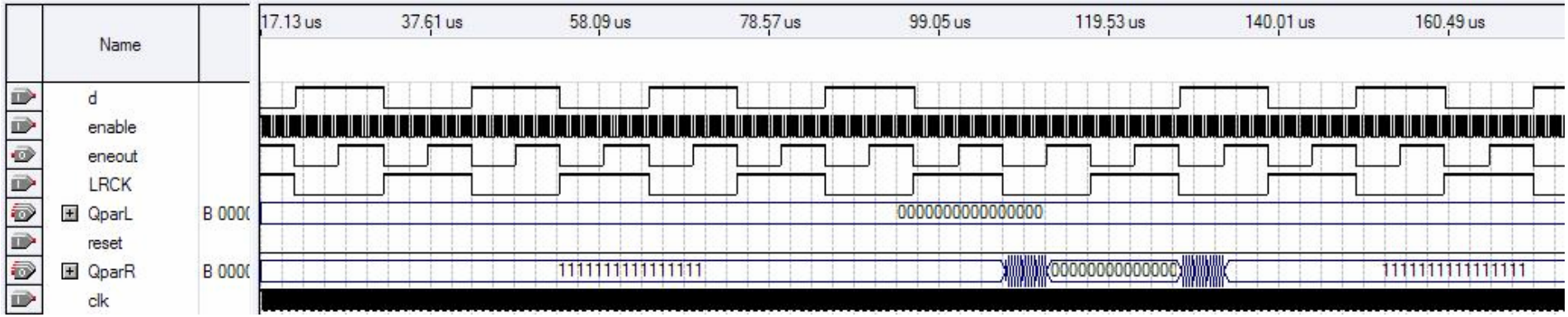


Fig. 3.1: Simulazione funzionale del Sipo_manager.

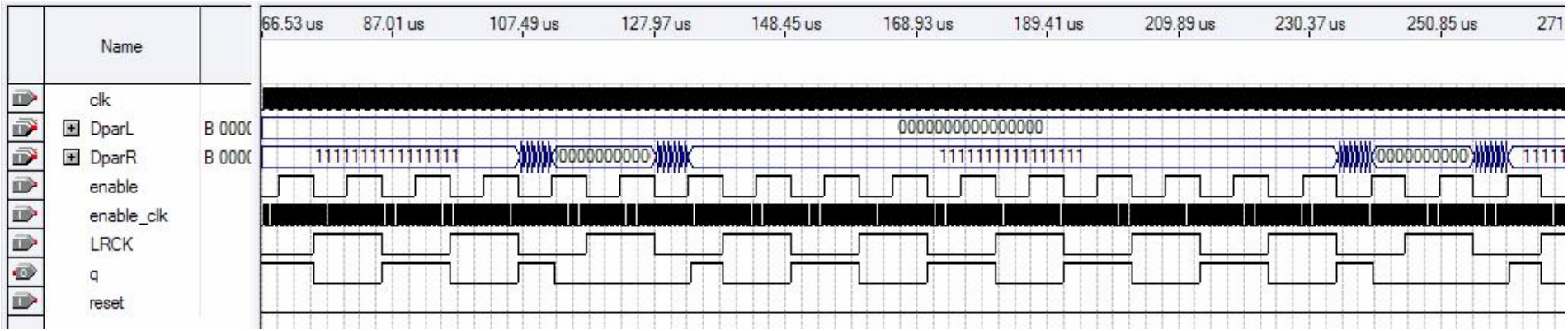


Fig. 3.3: Simulazione funzionale del Piso_manager.

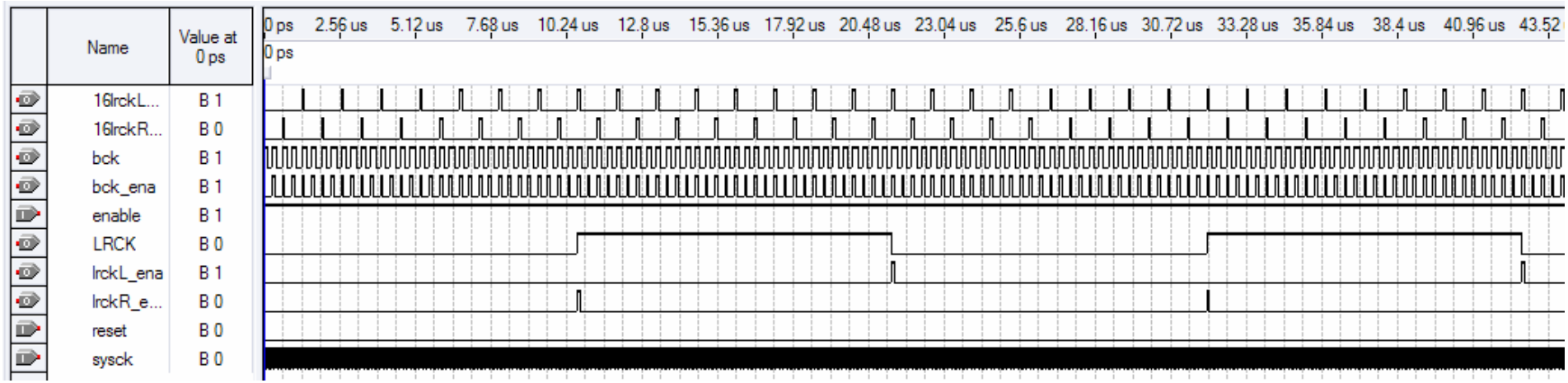


Fig. 3.5: Simulazione funzionale di Clocks.

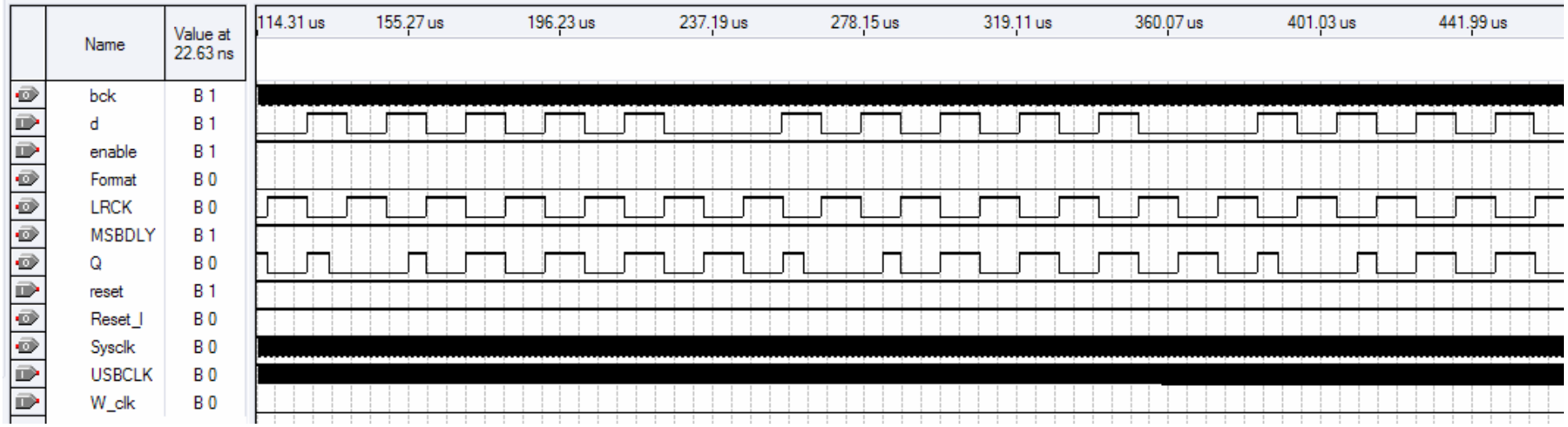


Fig. 3.7: Temporizzazione dell'interfacciamento ADC-DAC, relativo all'architettura di figura 3.6.

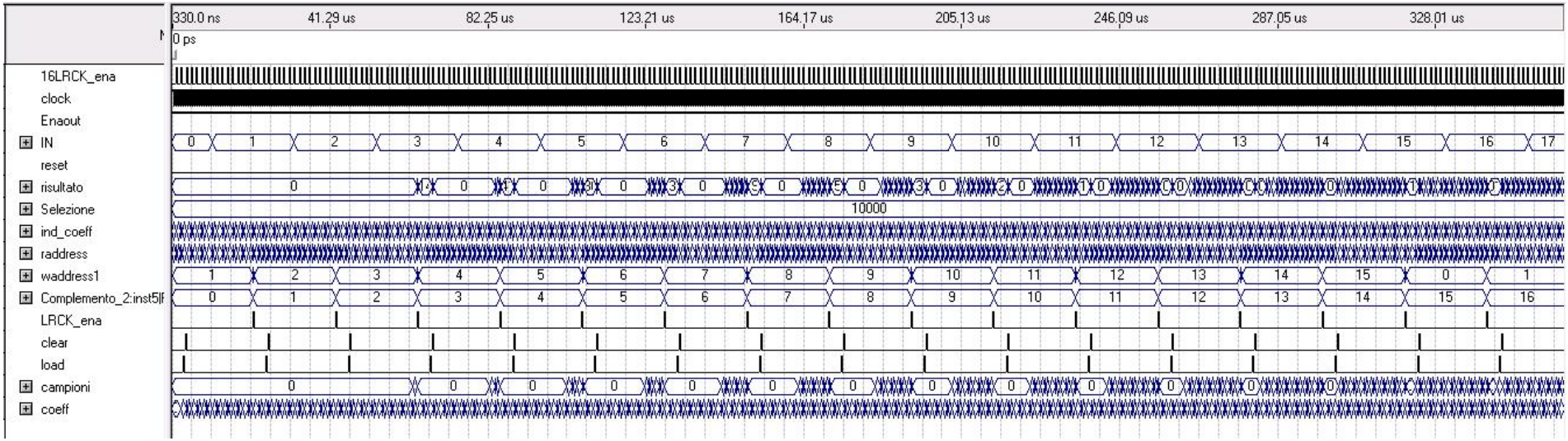


Fig. 3.8: a) Temporizzazione in scrittura sul canale sinistro.

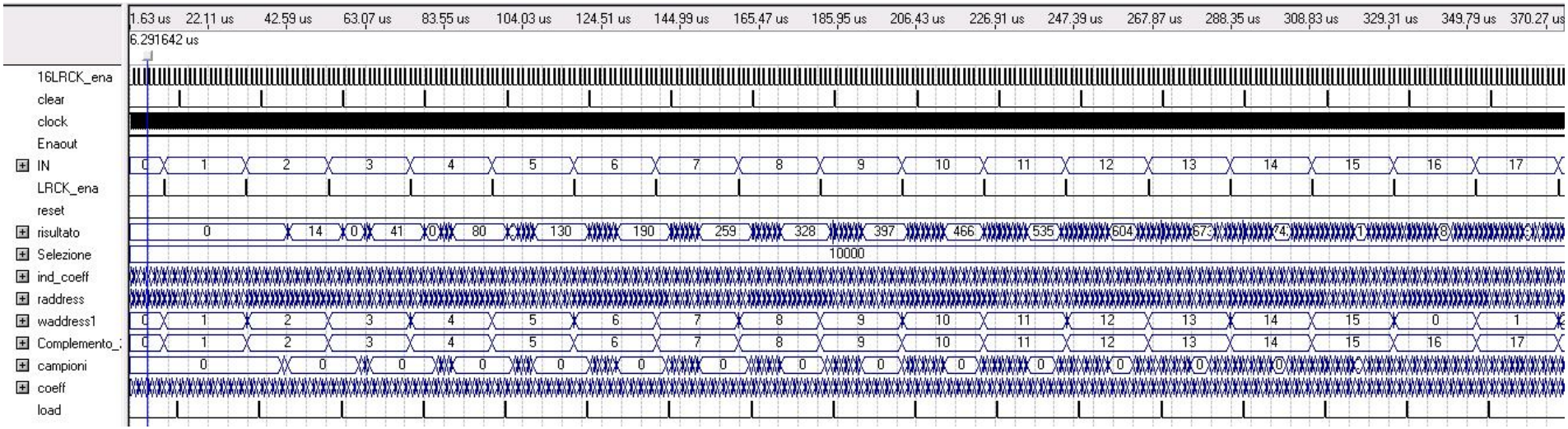


Fig. 3.8: b) Temporizzazione in scrittura sul canale destro.

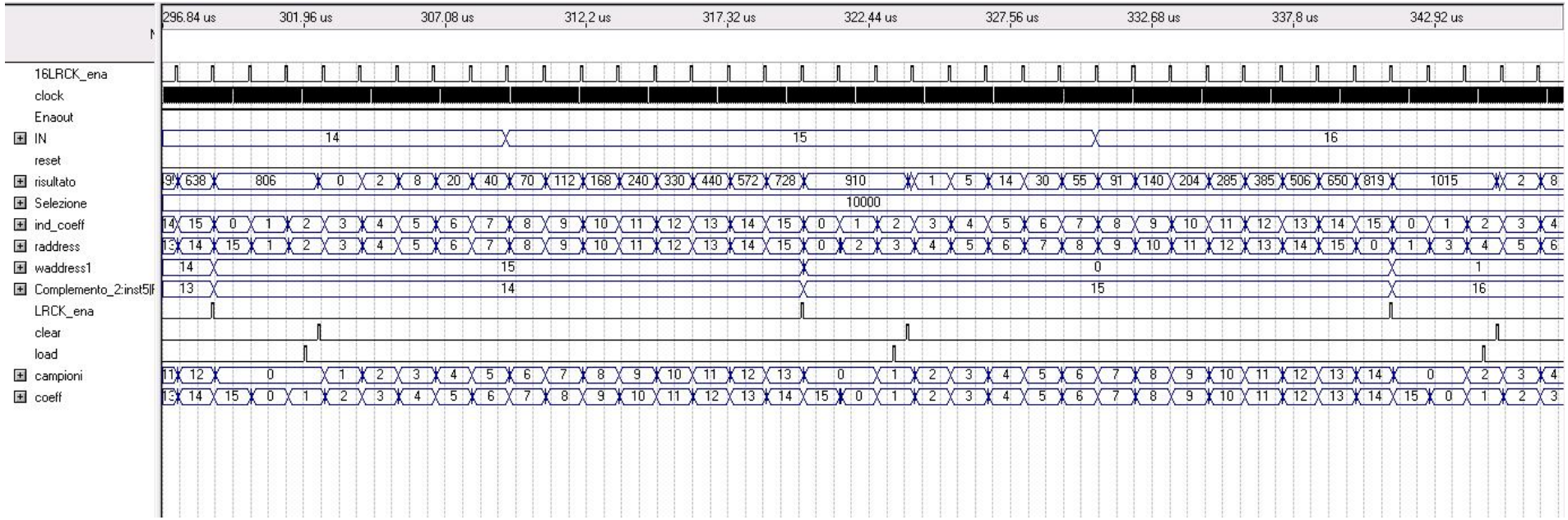


Fig. 3.9: a) Temporizzazione in lettura sul canale sinistro.

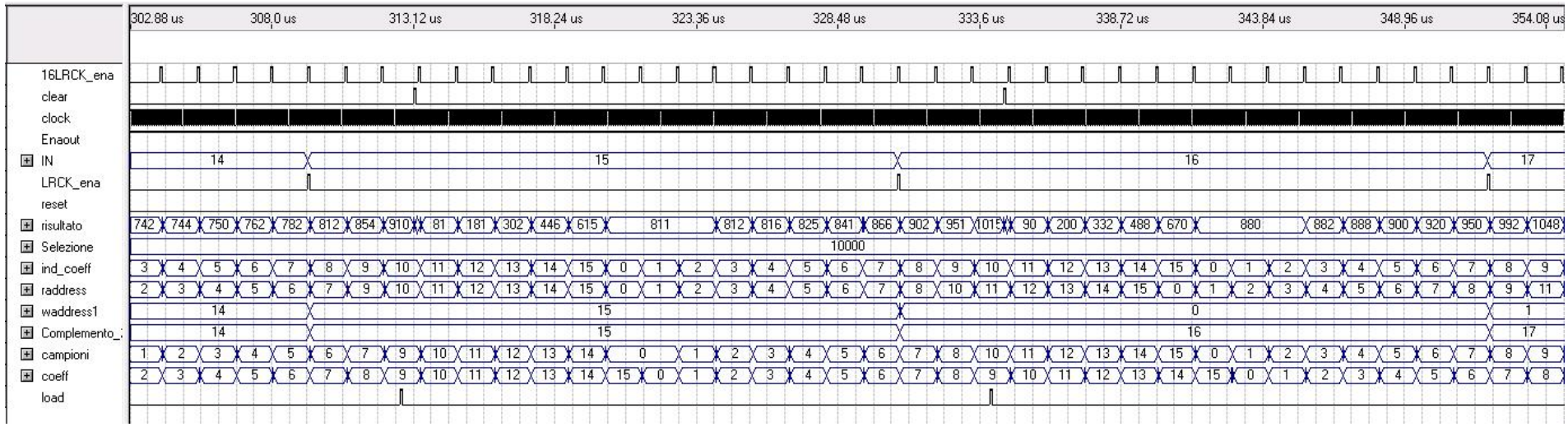


Fig. 3.9: b) Temporizzazione in lettura sul canale destro.

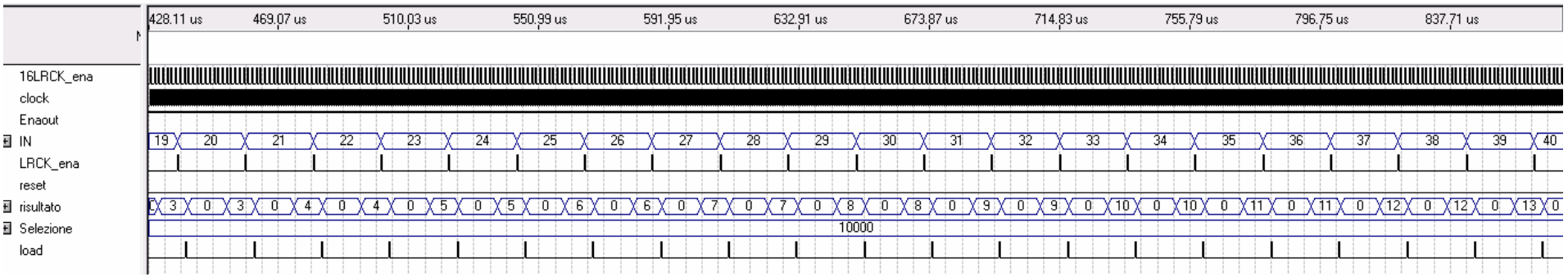


Fig. 3.10: Simulazione del blocco elaborazione senza il convertitore complemento a due.

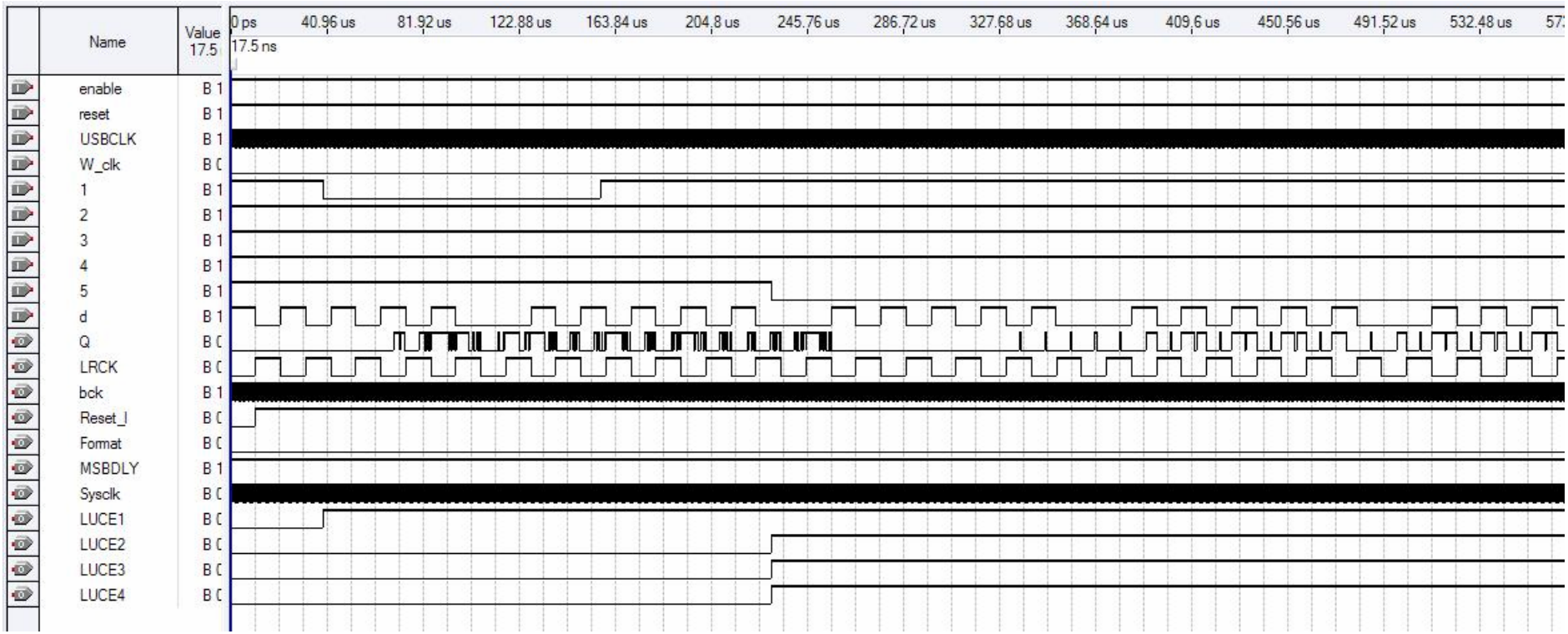


Fig. 3.11 a): Simulazione dell'intero sistema: particolare della selezione del filtro.

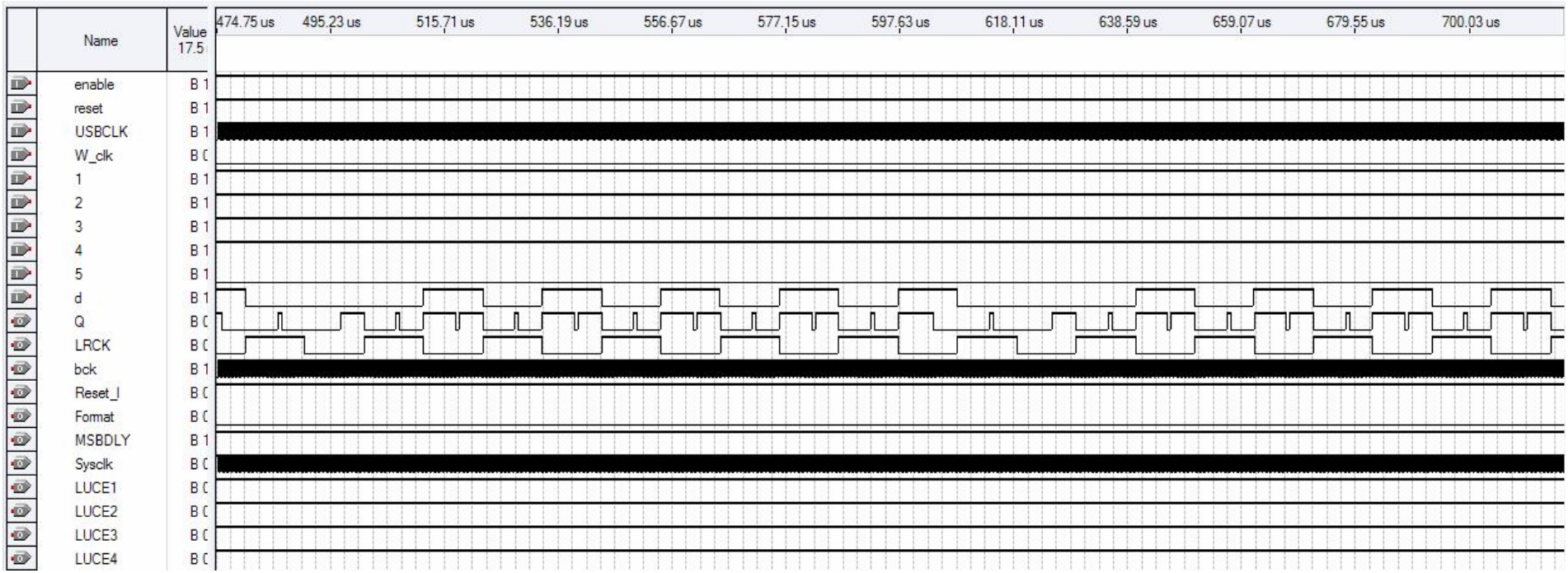
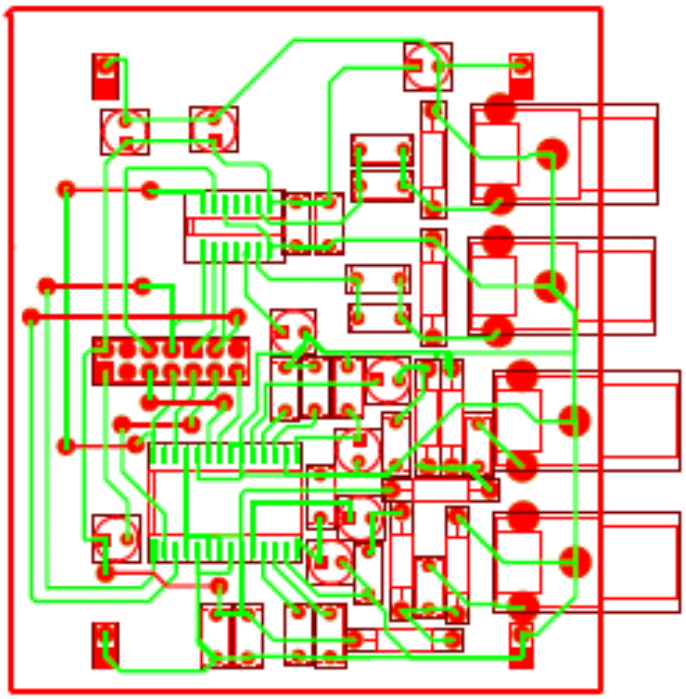
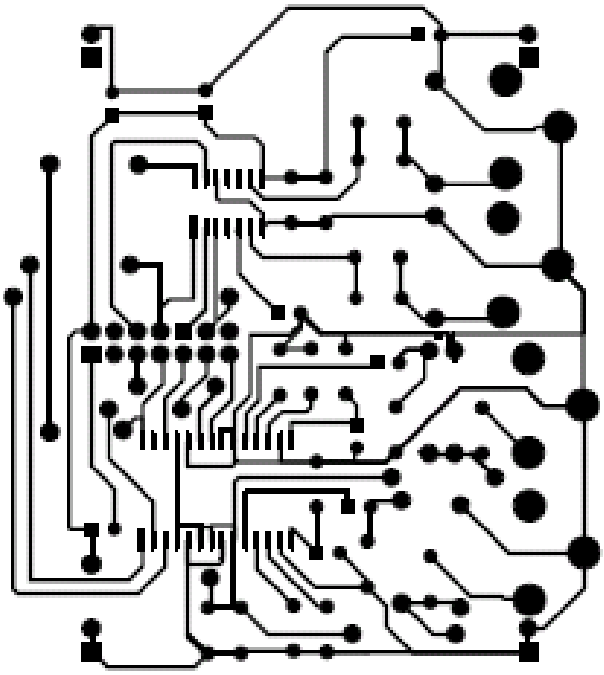


Fig. 3.11 b): Simulazione dell'intero sistema.

Appendice C: Layout scheda aggiuntiva





Bibliografia

[1] R.Roncella, R.Saletti, P.Terreni, D.Piatelli, *The aplication of a systolic macrocell-based vlsi design style to the design a single-chip high-performance fir filter*, Ets editrice, ottobre 1990.

[2] John Wiley & Sons,Ltd., *Real-Time Digital Signal Processing* , *2001*.

[3] SLS Corp., *UP3-1C6 Education Kit Reference Manual, Cyclone Edition* , *2005*.

[4] Analog Devices AD1870 datasheet.

[5] Burr-Brown PCM1725 datasheet.