Master degree in Computer Engineering

Electronics and Communication Systems

UNIVERSITÀ DI PISA

Filtro CIC interpolatore

Dicembre 2019

Federico Cappellini, Alberto Lunghi

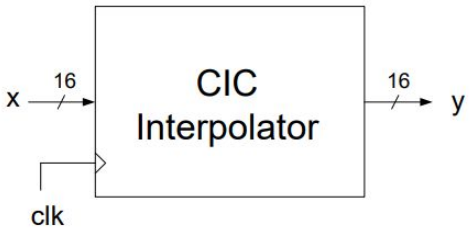
0 Specifiche di progetto

Progettare un circuito digitale che realizzi un filtro CIC (Cascated Integrator-Comb) passa basso interpolatore.
Le sue caratteristiche dovranno essere le seguenti:

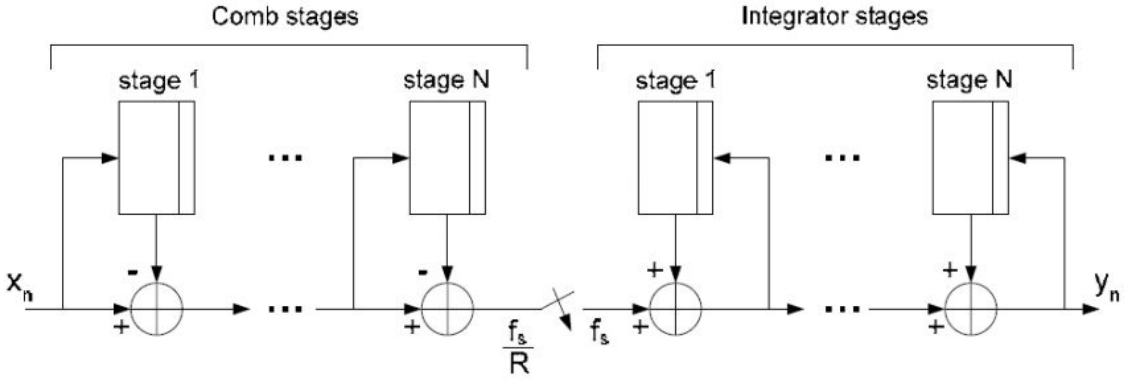
- Fattore di interpolazione pari a **R = 4**
- Ritardo degli stadi Comb pari a **M = 1**
- Numero di stadi del filtro pari a **N = 4**
- Inserzione di **R-1** zeri nel passaggio fra stadi Comb e stadi Integrator (Zero-Insertion)

$$y[n] = \left[\sum_{k=0}^{RM-1} x[n-k] \right]^N$$

Per ingressi e uscite utilizzare una rappresentazione su 16 bit.



Un’architettura di questo tipo è la seguente:



1 Introduzione

In questa relazione viene fornita una breve descrizione della classe di filtri Cascated Integrator-Comb (CIC), una classe di filtri proposta da Hogenauer nel 1981 per fornire una soluzione economica ai problemi dei filtri già in commercio per quanto riguarda le operazioni di decimazione e interpolazione.
Viene riportata ed analizzata l'architettura originale proposta da Hogenauer e ne viene mostrata una variante proposta da Losada e Lyons con lo scopo di migliorare le performance dell'originale e contemporaneamente ridurne la complessità.
Successivamente viene fornita una implementazione della versione originale tramite il linguaggio VHDL e vengono mostrati sia i codici che i test effettuati grazie al software di simulazione ModelSim, con annessi risultati.
Infine troviamo una sintesi logica effettuata con il software Xilinx Vivado e vediamo sia una descrizione di tale sintesi che i vari report, tra i quali path critica e consumo energetico.

2 Filtro CIC

Nell'ambito dei moderni sistemi di Digital Signal Processing vengono comunemente effettuate operazioni di decimazione e di interpolazione. La decimazione è una tecnica che consiste nel diminuire il numero di campioni necessari per la descrizione di un segnale trasformato in digitale, mentre l'interpolazione è una tecnica che consiste nell'aumentare il suddetto numero di campioni.
Una soluzione economica a queste operazioni viene fornita dai filtri CIC (Cascaded Integrator-Comb), una tipologia di filtri che rientra in quelli a risposta di impulso finita, detti anche FIR (Finite Impulse Response), dato che utilizzano ad ogni passo solo un numero finito di campioni per l'elaborazione.
La classe di filtri CIC venne proposta nel 1981 da Hogenauer e grazie alla loro struttura costituita da una sezione formata da integratori ed una formata da comb disposte a cascata, hanno notevoli peculiarità:

- Non sono necessari moltiplicatori. Questo permette di essere molto veloci nelle operazioni.
- Non ci sono coefficienti da memorizzare.
- La loro struttura è altamente regolare, infatti si può ottenere un intero filtro semplicemente disponendo in modo opportuno due blocchetti elementari.
- Non sono richieste unità di controllo o temporizzazioni particolarmente complicate.

2.1 Componenti del filtro CIC

Un filtro CIC è formato da tre componenti essenziali:

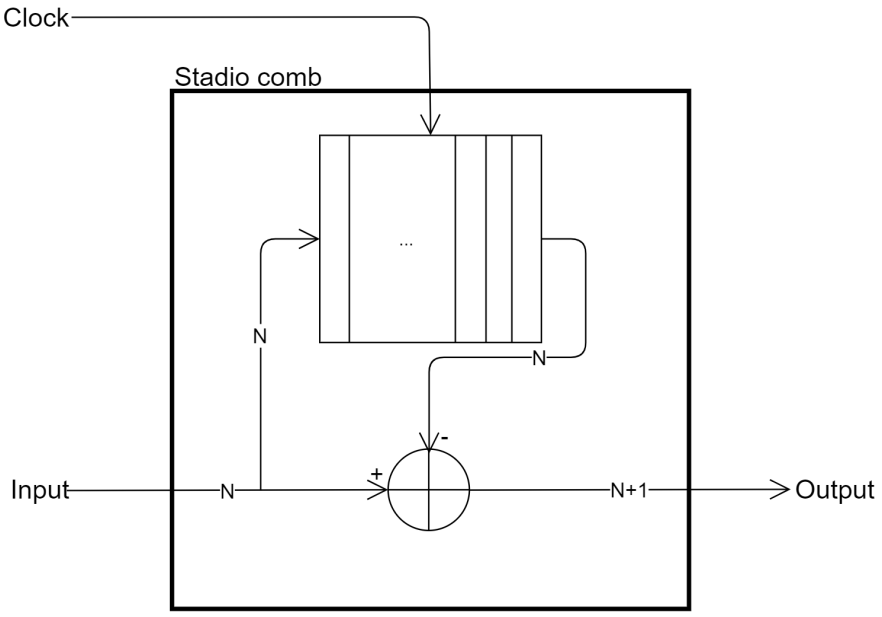
- Stadio Comb
- Zero Insertion
- Stadio Integrator

e per la variante

- Hold Insertion

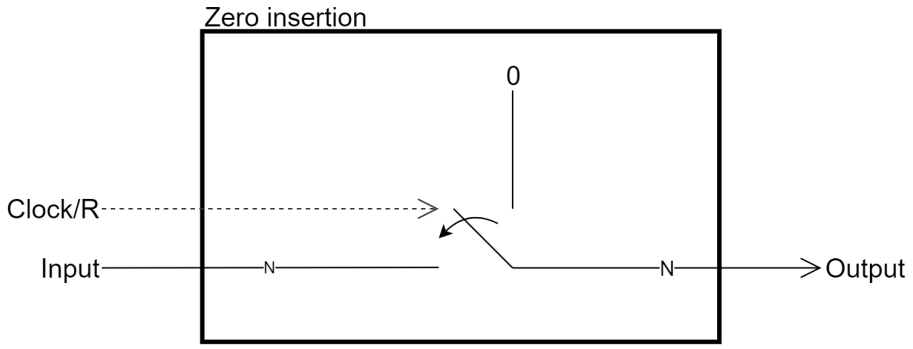
2.2.1 Stadio Comb

Uno dei componenti base nel filtro CIC è lo stadio Comb.
Esso è un sistema che prende in ingresso una parola di N bit e ne restituisce una a N+1 bit.
All'interno dello stadio Comb il dato rappresentato dalla parola da N bit in ingresso viene sommato al negato ritardato da un fattore M.



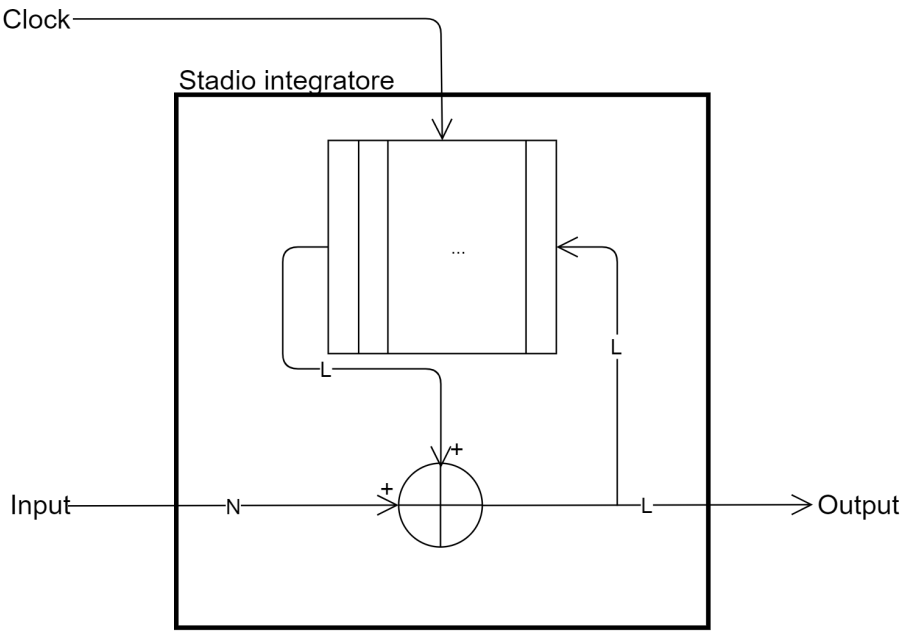
2.2.2 Zero Insertion

Per la realizzazione di un filtro CIC interpolatore è necessario l'inserimento di uno Zero Insertion tra lo stadio Comb ed lo stadio Integrator. La sua funzione è quella di inserire una sequenza di bit a 0 ad una frequenza maggiore rispetto a quella con cui i vengono restituiti i dati dallo stadio Comb.
In questo modo lo stadio integrator può compiere un'interpolazione riempiendo i cicli di clock dove i dati in input non sarebbero presenti con gli 0 inseriti dallo zero insertion.



2.2.3 Stadio Integrator

Uno dei componenti base nel filtro CIC è lo stadio Integrator. Esso è un sistema che prende in ingresso una parola a N bit e ne restituisce una a L bit dove L dipende dai parametri del filtro. All'interno dello stadio Integrator il dato rappresentato dalla parola da N bit in ingresso viene sommato al dato in output in precedenza. Il risultato sarà quindi la somma dell'input al tempo *t* e dell'output al tempo *t-1*.



2.2.4 Crescita dei blocchi

L'implementazione proposta da Hogenauer comporta un aumento dei bit necessari per rappresentare la parola dopo aver superato ogni blocco.
Infatti essendo il filtro CIC formato da una catena di sommatori, la somma di due parole a N bit può ritornare una parola che, per evitare overflow, è rappresentabile su N+1 bit.
Ogni input di ogni blocco è quindi sottoposto ad un fattore di crescita.

Definiamo il fattore di crescita $Growth_i$ (G_i) come:

$$G_i = \begin{cases} 2^j, & j = 1, 2, ..., N \\ \frac{2^{2N-j} RM^{j-N}}{R}, & j = N+1, ..., 2N \end{cases}$$

$$W_j = \lceil B_{in} + \log_2 G_j \rceil$$

Il fattore di crescita è regolato da due leggi differenti a seconda di considerare la catena degli stadi comb degli integratori.

Infatti nella catena dei comb vediamo che per ogni blocco il fattore di crescita segue la legge:

$G_i = 2^j$

dovuta esclusivamente alla presenza dei sommatori nella catena.

Nella catena degli integratori la legge è differente:

$G_i = \frac{2^{(2N-j)} (RM)^{J-N}}{R}$

Esiste una variazione da effettuare alla legge se ci troviamo nel caso in cui gli stadi Comb sono a delay unitario, ovvero che M equivalga ad 1.
Abbiamo infatti che con $j = N$ il fattore G_i non equivale a 2^N ma a 2^{N-1}.
Il risultato è che il j-esimo stadio comb e il ha lo stesso numero di bit sia in entrata che in uscita.

E' necessario porre l'attenzione anche sul fatto che la dimensione della parola in uscita dall'ultimo stadio integrator non è di B_{in} bit ma di $B_{in} + 6$ bit. E' necessario quindi effettuare un troncamento per rispettare il grado di precisione equivalente a B_{in} bit.

2.2.5 Hold Insertion

Uno dei componenti necessari per realizzare la variante suggerita da Losada e Lyons è l'Hold Insertion.
La funzione di questo componente è quella di mantenere costante l'output in uscita dallo stadio Comb e restituirlo come input allo stadio Integrator per un certo quantitativo di clock.
In questo modo è possibile ridurre il numero degli stadi comb e degli stadi integrator necessari per la realizzazione del filtro CIC.

3 Possibili architetture

In questo documento mostreremo due possibili architetture per il filtro CIC:

- Architettura di Hogenauer
- Architettura di Losada e Lyons

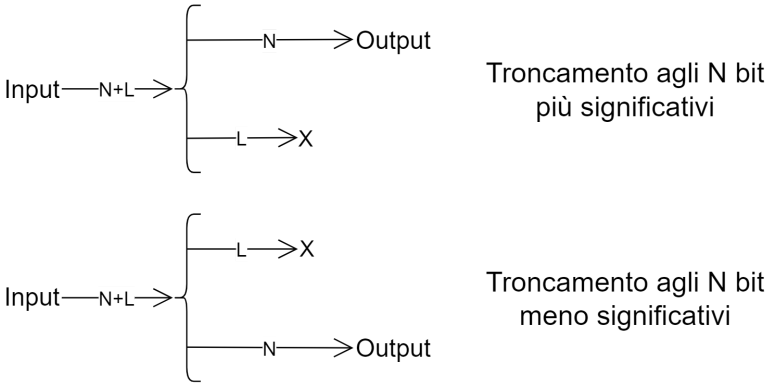
Le architetture verranno descritte con i parametri seguenti:

- N = 4
- M = 1
- R = 4

3.1 Architettura di Hogenauer

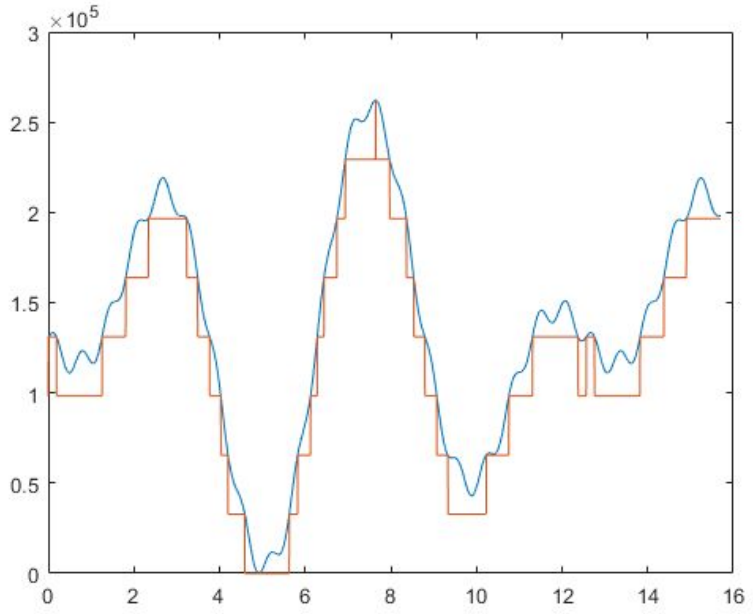
Questa architettura è composta da N stati comb, da N stadi integrator e da un campionatore ad una frequenza f/R.
Seguendo la legge che regola il dimensionamento delle parole in uscita dai blocchi abbiamo ricavato il numero dei bit necessari per evitare overflow durante il funzionamento del filtro.
Per mantenere la precisione di B_{in} bit, in uscita è necessario effettuare un troncamento della parola. Possiamo effettuarlo al MSB (Most Significant Bit) o al LSB (Less Significant Bit).

FOTO con dimensionamenti

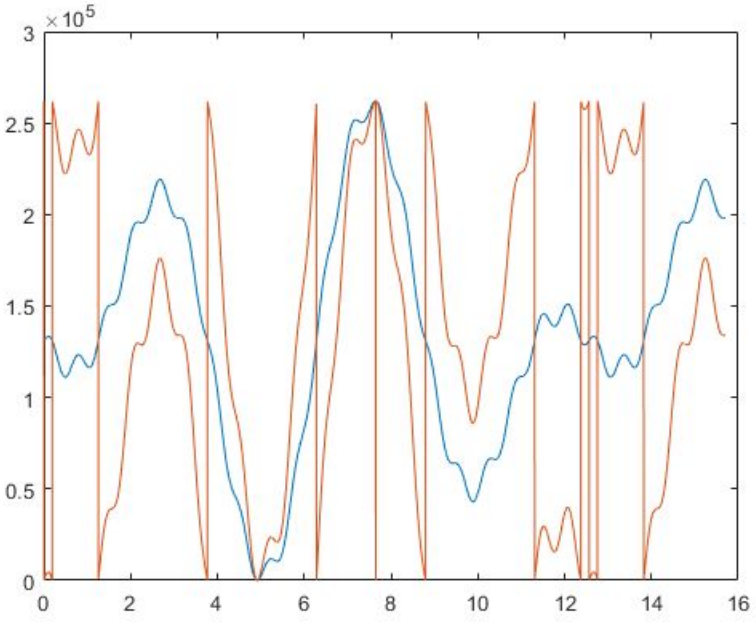


Quando viene eseguito un troncamento al LSB vengono scartati i bit più significativi della parola in uscita e quindi perdiamo sensibilità nei valori alti del segnale, ma a valori bassi l'uscita interpola perfettamente l'ingresso.
Al contrario, con un troncamento al MSB, vengono scartati i bit meno significativi, quindi perdiamo di precisione quando andiamo a considerare i valori più vicini allo zero. Con questo tipo di troncamento però l'uscita riesce a riprodurre perfettamente il segnale a valori alti, interpolando anche i picchi di input.

Di seguito sono riportate due immagini che mettono a confronto un segnale troncato nei due modi. Il segnale è un insieme di valori naturali su 32 bit con picchi massimi di 2^{18}. Nella prima immagine è possibile osservare il segnale originale in blu ed il segnale troncato ai 17 bit più significativi in rosso, mentre nella seconda immagine è possibile osservare il segnale originale in blu ed il segnale troncato ai 17 bit meno significativi in rosso.



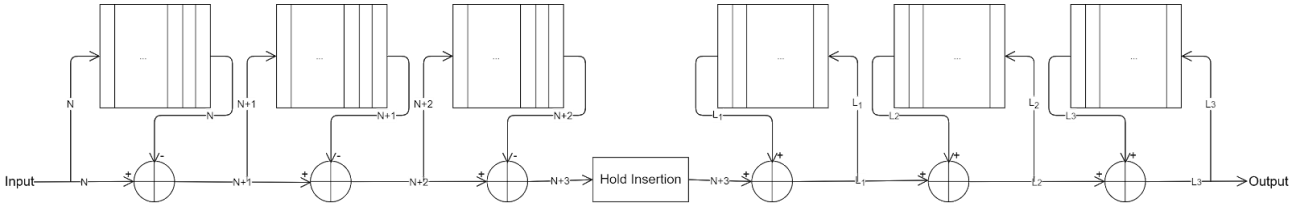
In blu il segnale originale, in rosso il segnale troncato al MSB



In blu il segnale originale, in rosso il segnale troncato al LSB

3.2 Architettura di Losada e Lyons

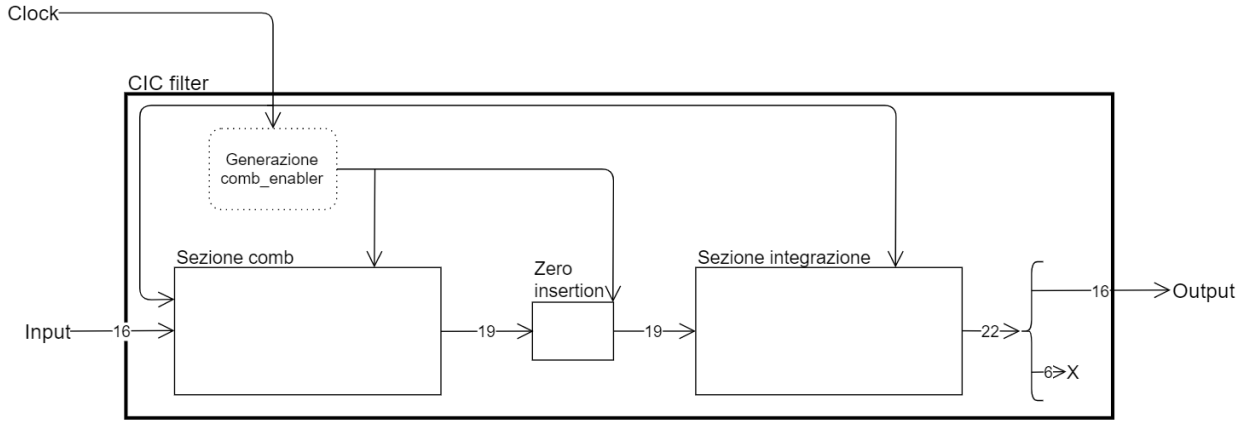
Questa variante, suggerita da Losada e Lyons, permette di ridurre la complessità del filtro CIC riducendo il numero degli stadi comb e degli stadi integrator. Vediamo infatti che questa architettura è priva dell'ultimo stadio Comb, del primo stadio Integrator e del campionatore e prevede la loro sostituzione con un hold-insertion.



4 Realizzazione del codice VHDL

4.1 Visione generale e logica per la divisione del clock

L'implementazione di seguito riportata fa riferimento all'architettura di Hogenauer. Gli stadi comb sono stati logicamente raggruppati sotto un'unica unità funzionale, la sezione comb, mentre gli stadi di integrazione sono stati logicamente raggruppati sotto un'unica unità funzionale che prende il nome di sezione integrazione.



Com'è possibile vedere dal diagramma, una parte della circuiteria del filtro CIC è responsabile della generazione del segnale *comb_enabler*. Questo segnale è di vitale importanza per la nostra implementazione poiché ogni filtro CIC prevede due sezioni operanti a due frequenze diverse:

- gli stadi di integrazione operano a frequenza *f*,
- gli stadi comb operano a frequenza ridotta: *f/R*.

Tuttavia è stata preferita una soluzione diversa alla realizzazione di una rete atta alla divisione del clock (o alla velocizzazione dello stesso) per evitare di dover gestire più di un segnale di clock: il software Vivado che sarà utilizzato nella fase di sintesi prevede l'utilizzo di un solo segnale di questo tipo per il quale calcolare le adeguate temporizzazioni e per il quale trovare una path ottimizzata.
La soluzione qua riportata prevede infatti che tutto il circuito sia pilotato da un'unica frequenza di clock, ma un segnale di enabler (*comb_enabler*) sia responsabile di attivare i registri e le reti in maniera opportuna (esattamente *1/R* volte rispetto al segnale di clock).

4.2 Filtro CIC

Di seguito si riporta il codice VHDL che descrive il modulo principale corrispondente alla figura sopra riportata.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity cic_filter is
```

```
    generic(
        Nbit : integer
    );
    port (
        clk     : in   std_logic;
        reset   : in   std_logic;
        input   : in   std_logic_vector(Nbit-1 downto 0);
        output  : out  std_logic_vector(Nbit-1 downto 0)
    );

end cic_filter;

architecture cic_filter_arch1 of cic_filter is

component comb_stages is
    generic(Nbit : integer);
    port (
        clk          : in   std_logic;
        comb_enabler : in   std_logic;
        reset        : in   std_logic;
        input        : in   std_logic_vector(Nbit-1 downto 0);
        output       : out  std_logic_vector(Nbit + 4 -2 downto 0)
    );
end component comb_stages;


component integrator_stages is
    generic(Nbit : integer);
    port (
        clk     : in   std_logic;
        reset   : in   std_logic;
        input   : in   std_logic_vector(Nbit+3-1 downto 0);
        output  : out  std_logic_vector(Nbit+6-1 downto 0)
    );
end component integrator_stages;


component zeroInsertionSampler is
    generic(Nbit : integer);
    port (
        enabler     : in  std_logic;
        inputData   : in  std_logic_vector(Nbit-1 downto 0);
        outputData  : out std_logic_vector(Nbit-1 downto 0)
    );
end component zeroInsertionSampler;

signal out_comb   : std_logic_vector (Nbit+2 downto 0);
signal out_z_i    : std_logic_vector (Nbit+2 downto 0);
signal out_integr : std_logic_vector (Nbit+5 downto 0);

signal comb_enabler : std_logic;
signal counter      : integer;


begin

    cicFilterProcess: process(clk, reset, counter, comb_enabler)
    begin
```

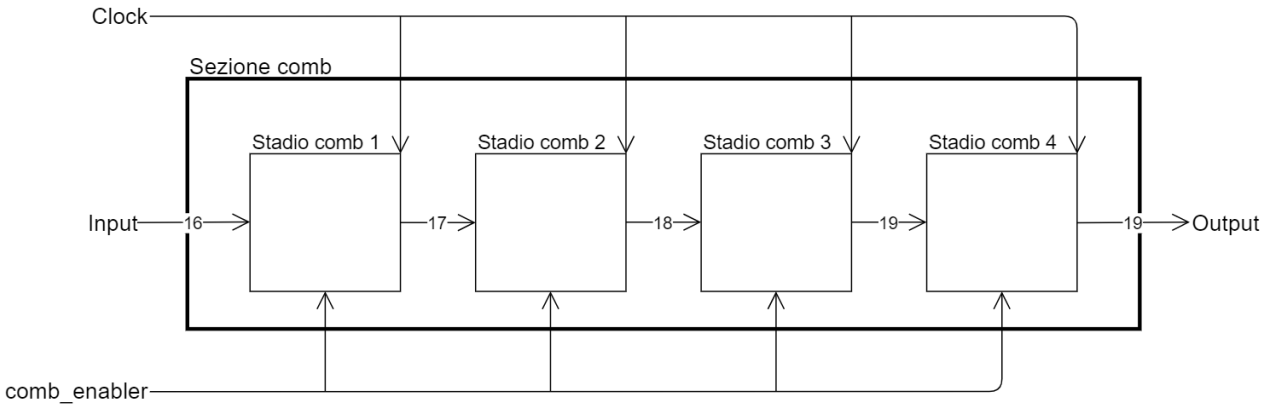
```
        if (clk'event and clk = '1') then
            if (reset = '1') then
                counter <= 0;
                comb_enabler <= '0';
            else
                if (counter = 3) then
                    comb_enabler <= '1';
                    counter <= 0;
                else
                    comb_enabler <= '0';
                    counter <= counter + 1;
                end if;
            end if;
        end if;

    end process cicFilterProcess;

    c_s: comb_stages generic map(Nbit => Nbit)
        port map (clk, comb_enabler, reset, input, out_comb);
    z_i: zeroInsertionSampler generic map(Nbit => Nbit+3)
        port map (comb_enabler, out_comb, out_z_i);
    i_s: integrator_stages generic map(Nbit => Nbit)
        port map (clk, reset, out_z_i, out_integr);

    output <= out_integr(Nbit+5 downto 6);

end cic_filter_arch1;
```

Com'è possibile notare tale modulo

- implementa tramite il processo VHDL *cicFilterProcess* la generazione del segnale *comb_enabler* precedentemente descritto,
- si compone di altri componenti la cui descrizione e il cui codice sorgente sono riportati nei paragrafi successivi, in particolar modo
    - il componente *comb_stages* realizza la sezione comb,
    - il componente *integrator_stages* realizza la sezione di integrazione,
    - il componente *zeroInsertionSampler* è il campionatore con inserimento di zeri
- si occupa di effettuare il troncamento dell'uscita della sezione di integrazione *out_integr* ai 16 bit più significativi per rimanere nelle specifiche in termini di dimensione dei bit.

4.3 Sezione Comb

La sezione comb si compone di 4 stadi comb in cascata come previsto dall'architettura di Hogenauer.



Il numero dei bit delle connessioni fra i vari stati rispetta la legge di bit grow precedentemente enunciata calcolata facendo uso dei parametri R, N e M in esame.
Il codice VHDL che implementa la sezione comb è riportato di seguito.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity comb_stages is

    generic(Nbit : integer);
    port (
        clk          : in   std_logic;
        comb_enabler : in   std_logic;
        reset        : in   std_logic;
        input        : in   std_logic_vector(Nbit-1 downto 0);
        output       : out  std_logic_vector(Nbit + 4 -2 downto 0)
    );

end comb_stages;

architecture comb_stages_arch1 of comb_stages is

    component comb_block is
        generic(Nbit : integer);
        port (
            clk          : in   std_logic;
            comb_enabler : in   std_logic;
            reset        : in   std_logic;
            input        : in   std_logic_vector(Nbit-1 downto 0);
            output       : out  std_logic_vector(Nbit downto 0)
        );
    end component comb_block;

    signal inside0 : std_logic_vector (Nbit downto 0);
    signal inside1 : std_logic_vector (Nbit+1 downto 0);
    signal inside2 : std_logic_vector (Nbit+2 downto 0);
    signal inside3 : std_logic_vector (Nbit+3 downto 0);

begin

    c_b0: comb_block generic map(Nbit => Nbit)
        port map (clk, comb_enabler, reset, input, inside0);
```

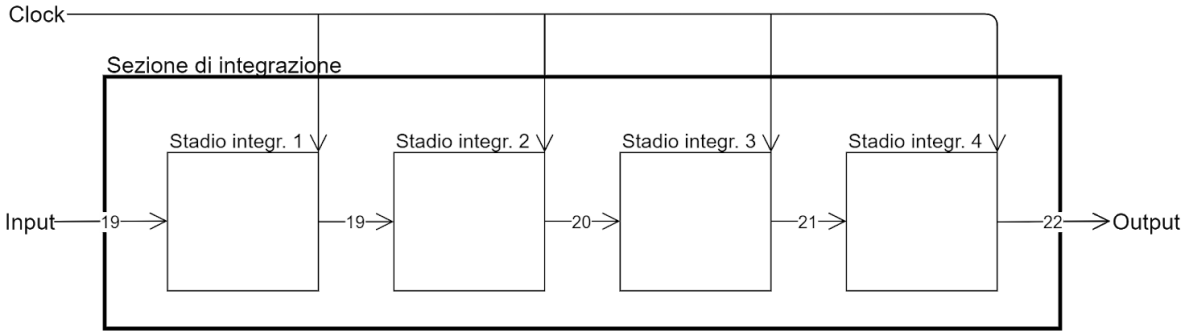
```
    c_b1: comb_block generic map(Nbit => Nbit+1
        port map (clk, comb_enabler, reset, inside0, inside1);
    c_b2: comb_block generic map(Nbit => Nbit+2)
        port map (clk, comb_enabler, reset, inside1, inside2);
    c_b3: comb_block generic map(Nbit => Nbit+3)
        port map (clk, comb_enabler, reset, inside2, inside3);
    output <= inside3(Nbit+2 downto 0);

end comb_stages_arch1;
```

Si noti che:

- Il componente che realizza lo stadio comb ha il nome di *comb_block*,
- i segnali *inside0*, *inside1*, *inside2* e *inside3* rappresentano le connessioni tra gli stadi comb,
- l'output di questa sezione subisce un troncamento per rispettare la legge di crescita dei bit dello specifico caso in cui il parametro M valga 1.

4.3 Sezione di integrazione

La sezione di integrazione di compone di 4 stadi integratori in cascata come previsto dall'architettura di Hogenauer.



Anche in questo caso il numero dei bit delle connessioni fra i vari stadi rispetta la legge di bit grow precedentemente enunciata calcolata facendo uso dei parametri R, N e M in esame.
Si noti che, in antitesi con la sezione comb, la sezione di integrazione non ha in input il segnale di *comb_enabler* poiché non si necessita di un riferimento ad una velocità di clock inferiore rispetto a quella del segnale di clock principale.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity integrator_stages is

    generic(Nbit : integer);
    port (
        clk     : in   std_logic;
        reset   : in   std_logic;
        input   : in   std_logic_vector(Nbit+3-1 downto 0);
        output  : out  std_logic_vector(Nbit+6-1 downto 0)
    );

end integrator_stages;
```

```
architecture integrator_stages_arch1 of integrator_stages is

    component integrator_block is
        generic(Nbit : integer; Mbit : integer);
        port (
            clk     : in   std_logic;
            reset   : in   std_logic;
            input   : in   std_logic_vector(Nbit-1 downto 0);
            output  : out  std_logic_vector(Mbit-1 downto 0)
        );
    end component integrator_block;

    signal inside0 : std_logic_vector (Nbit+3-1 downto 0);
    signal inside1 : std_logic_vector (Nbit+4-1 downto 0);
    signal inside2 : std_logic_vector (Nbit+5-1 downto 0);

begin

    integratorBlockComponent0: integrator_block
        generic map(Nbit => Nbit+3, Mbit => Nbit+3)
        port map (clk, reset, input, inside0);
    integratorBlockComponent1: integrator_block
        generic map(Nbit => Nbit+3, Mbit => Nbit+4)
        port map (clk, reset, inside0, inside1);
    integratorBlockComponent2: integrator_block
        generic map(Nbit => Nbit+4, Mbit => Nbit+5)
        port map (clk, reset, inside1, inside2);
    integratorBlockComponent3: integrator_block
        generic map(Nbit => Nbit+5, Mbit => Nbit+6)
        port map (clk, reset, inside2, output);

end integrator_stages_arch1;
```

Si noti che:

- Il componente che realizza lo stadio integratore ha il nome di *integrator_block*,
- i segnali *inside0*, *inside1* e *inside2* rappresentano le connessioni tra gli stadi integratori.

4.4 Zero Insertion

La fase di campionamento e zero insertion è realizzata da un modulo a parte che semplicemente connette la propria uscita all'ingresso ogni qualvolta che il segnale *comb_enabler* ha valore logico 1, mentre connette l'uscita al valore logico zero nel rimanente caso.
Facendo riferimento al segnale di clock, questo modulo fornisce quindi in ingresso alla sezione di integrazione

- l'uscita della sezione comb 1 volta ogni R,
- un segnale i cui bit hanno valore logico 0 R-1 volte ogni R.

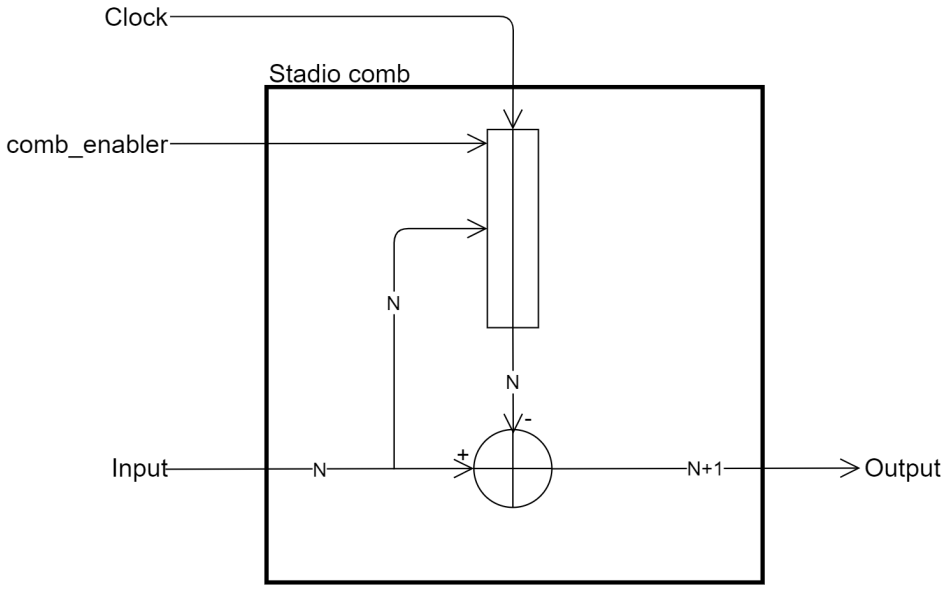
Così facendo viene rispettata l'architettura di Hogenauer che prevede l'inserimento di R-1 zeri come input agli stati integratori.

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
```

```vhdl
entity zeroInsertionSampler is

    generic(Nbit : integer);
    port (
        enabler     : in  std_logic;
        inputData   : in  std_logic_vector(Nbit-1 downto 0);
        outputData  : out std_logic_vector(Nbit-1 downto 0)
    );

end zeroInsertionSampler;

architecture arch of zeroInsertionSampler is

begin

    outputData <= inputData when enabler = '1' else (others => '0');

end arch;
```

4.5 Stadio comb

Lo stadio comb è stato realizzato in accordo con gli schemi precedentemente riportati e l'architettura di Hogenauer precedentemente descritta.



In questo caso il numero di bit non è espresso in termini numerici, ma sostituito da espressioni generiche perché ogni stadio comb ha profondità di bit differenti; ciò che è importante però è il rispetto della legge di crescita dei bit precedentemente descritta per cui l'uscita dello stadio comb presenta un bit in più rispetto all'ingresso.
Il codice sorgente VHDL è riportato di seguito.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

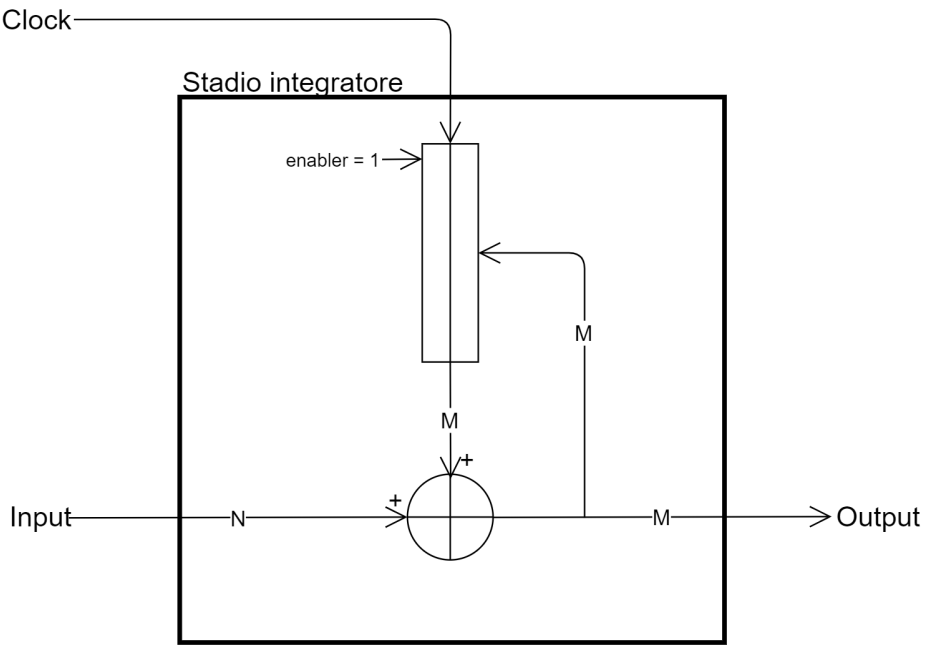
```
entity comb_block is

    generic(Nbit : integer);
    port (
        clk          : in   std_logic;
        comb_enabler : in   std_logic;
        reset        : in   std_logic;
        input        : in   std_logic_vector(Nbit-1 downto 0);
        output       : out  std_logic_vector(Nbit downto 0)
    );

end comb_block;


architecture comb_block_arch1 of comb_block is

    component flipflop
        generic(Nbit : integer := 4);
        port (
            clk         : in  std_logic;
            enabler     : in  std_logic;
            reset       : in  std_logic;
            inputData   : in  std_logic_vector(Nbit-1 downto 0);
            outputData  : out std_logic_vector(Nbit-1 downto 0)
        );
    end component flipflop;

    signal storedData  : std_logic_vector(Nbit-1 downto 0);

begin

    fpfp:  flipflop
        generic map(Nbit => Nbit
        port map (clk, comb_enabler, reset, input, storedData);

    output <= std_logic_vector(
        resize(signed(input),Nbit+1) - resize(signed(storedData),Nbit+1)
    );

end comb_block_arch1;
```

Si noti che

- il segnale *storedData* corrisponde alla connessione tra l'uscita del flipflop e l'ingresso del sommatore

4.6 Stadio integratore

Anche lo stadio integratore è stato realizzato in accordo con gli schemi precedentemente riportati e l'architettura di Hogenauer precedentemente descritta e sempre per lo stesso motivo (ogni stadio integratore ha profondità di bit differenti), il numero di bit non è espresso in termini numerici, ma sostituito da espressioni generiche.



Si noti che il segnale di *enabler* del flip flop, differentemente dal caso dello stadio comb, è collegato al valore logico costante 1. Il motivo, come precedentemente anticipato, deriva dal fatto che questa unità non funziona a clock ridotto (tramite le tempistiche dettate dal segnale *comb_enabler*), ma esegue un'operazione ogni ciclo di clock.
Il codice sorgente VHDL è riportato di seguito.

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity integrator_block is

    generic(Nbit : integer; Mbit : integer);
    port (
        clk     : in   std_logic;
        reset   : in   std_logic;
        input   : in   std_logic_vector(Nbit-1 downto 0);
        output  : out  std_logic_vector(Mbit-1 downto 0)
    );

end integrator_block;


architecture integrator_arch1 of integrator_block is

    component flipflop
        generic(Nbit : integer := 4);
        port (
            clk         : in  std_logic;
            enabler     : in  std_logic;
            reset       : in  std_logic;
            inputData   : in  std_logic_vector(Nbit-1 downto 0);
            outputData  : out std_logic_vector(Nbit-1 downto 0)
        );
    end component flipflop;
```

```
    signal savedData : std_logic_vector(Mbit-1 downto 0);
    signal outputRW  : std_logic_vector(Mbit-1 downto 0);

begin

    outputRW <= std_logic_vector(
        resize(signed(input),Mbit) + resize(signed(savedData),Mbit)
    );
    output <= outputRW;
    flipflopComponent: flipflop
        generic map (Nbit => Mbit)
        port map (
            clk => clk, enabler => '1', reset => reset,
            inputData => outputRW, outputData => savedData
        );

end integrator_arch1;
```

Si noti che

- il segnale *savedData* corrisponde alla connessione tra il flipflop e il sommatore
- il segnale *outputRW* serve come “segnale di appoggio” per permettere di far uscire il risultato dello stadio di integrazione come output e al contempo di poterlo utilizzare come ingresso del flipflop.

4.7 Flip flop con enabler

Gli stati comb e integratore basano la loro architettura su un sommatore (sottrattore nel caso dello stadio comb) ed un flipflop. Mentre il sommatore è stato descritto utilizzando la libreria matematica *ieee.numeric_std.all* offerta dal linguaggio VHDL, il flip flop è stato realizzato come un modulo a sé. Nell’implementazione realizzata il flip flop è dotato, oltre che degli ingressi clock, reset e dati e dell’uscita dati, di un segnale di *enabler* in ingresso.
Lo scopo di questo segnale è

- bloccare l’uscita del flipflop impedendo il campionamento dei dati al fronte di salita del clock quando ha valore logico 0
- permettere il normale funzionamento del flipflop quando ha valore 1

Grazie a questo segnale, come precedentemente descritto, è stato possibile realizzare la divisione delle frequenze operative delle due sezioni del filtro CIC.
Di seguito è riportato il codice sorgente VHDL.

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;


entity flipflop is

    generic(Nbit : integer := 4);
    port (
        clk         : in  std_logic;
        enabler     : in  std_logic;
```

```
        reset       : in  std_logic;
        inputData   : in  std_logic_vector(Nbit-1 downto 0);
        outputData  : out std_logic_vector(Nbit-1 downto 0)
    );

end flipflop;

architecture arch of flipflop is

begin

    registerProcess: process(clk,reset,inputData) begin
        if (clk'event and clk = '1') then
            if (reset = '1') then
                outputData <= (others => '0');
            else
                if (enabler = '1') then
                    outputData <= inputData;
                end if;
            end if;
        end if;
    end process registerProcess;

end arch;
```

5 Test e simulazioni

Qua sono riportati gli approcci di test e i codici VHDL delle testbench con i quali abbiamo effettuato i test in un approccio bottom-up, iniziando dalle prove dai singoli componenti fino ad arrivare alla prova del filtro completo.

5.1 Test del flipflop

Il flipflop è stato testato tramite una testbench che fornisce ogni ciclo di clock valori in ingresso differenti. Tramite una simulazione con il software ModelSim è stata accertata la capacità del flipflop di campionare e mantenere stabili in uscita i valori in ingresso ogni fronte di salita del clock; di conseguenza il flipflop risulta correttamente implementato.
Data l'estrema semplicità della testbench, il codice sorgente è omesso.

5.2 Test dello stadio comb

La testbench dello stadio comb si pone l'obiettivo di fornire in ingresso una sequenza ben precisa di valori al fine di permettere di verificare il corretto funzionamente del modulo tramite una simulazione con ModelSim. In particolar modo gli ingressi forniti rappresentano i punti di una retta di pendenza (coefficiente angolare / derivata) noti a priori.
Poiché il filtro comb (con numero di ritardi pari a 1) sostanzialmente è in grado di restituire la derivata della retta che interpola due input consecutivi o più semplicemente di restituire la differenza fra due input consecutivi, l'output atteso è costante (ad esclusione del breve periodo che segue il rilascio del reset) ed ha valore pari alla derivata della retta.
Il modulo dello stadio comb è così stato identificato come correttamente funzionante.
Il codice della testbench (con derivata della retta pari a 3) è riportato di seguito.

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;


entity comb_block_tb is
end comb_block_tb;

architecture arch of comb_block_tb is

    constant T_clk        : time      := 100 ns;
    signal clk_tb         : std_logic := '0';
    signal reset_tb       : std_logic := '1';
    constant N_tb         : integer   := 8;
    signal inputData_tb   : std_logic_vector(N_tb-1 downto 0) := "00000000";
    signal outputData_tb  : std_logic_vector(N_tb downto 0);

    component comb_block is
        generic(Nbit : integer);
        port (
            clk          : in   std_logic;
```

```
            comb_enabler : in   std_logic;
            reset        : in   std_logic;
            input        : in   std_logic_vector(Nbit-1 downto 0);
            output       : out  std_logic_vector(Nbit downto 0)
        );
    end component comb_block;

begin

    clk_tb <= not(clk_tb) after T_clk/2;
    reset_tb <= '0' after 2*T_clk;

    combBlockComponent: comb_block
        generic map (Nbit => N_tb)
        port map (
            clk => clk_tb, comb_enabler => '1', reset => reset_tb,
            input => inputData_tb, output => outputData_tb
        );

    process(clk_tb, inputData_tb) begin
        if (clk_tb'event and clk_tb = '1') then
            inputData_tb <= std_logic_vector( unsigned(inputData_tb) + 3 );
        end if;
    end process;

end arch;
```

Un secondo controllo è stato effettuato con funzioni non lineari durante il test dell'intero filtro CIC. SI faccia riferimento a tale paragrafo.

5.3 Test dello stadio integratore

Il test dello stadio integratore segue le modalità già riportate nel test dello stadio comb.
E' stato in particolar modo accertato che:

- fornito un input costante di valore K, l'uscita corrisponda ad un valore che cresce linearmente con step di K,
- fornito un input che cresce linearmente, l'uscita cresca quadraticamente

Data la somiglianza con la testbench dello stadio comb, il codice VHDL è omesso.
Un ulteriore controllo è stato effettuato con funzioni non lineari durante il test dell'intero filtro CIC. SI faccia riferimento a tale paragrafo.

5.4 Test delle sezioni comb e di integrazione

Appurato il corretto funzionamento dello stadio comb in condizioni di isolamento, si è comunque optato per eseguire un test anche per l'intera sezione comb.
La testbench utilizzata ha fornito input casuali o generati in maniera analoga al test dell'intero filtro CIC più in avanti riportato.
Tramite una simulazione in ambiente ModelSim i vari segnali interni sono stati messi a video ed esaminati per accertare che avessero ogni istante di clock il valore atteso.
L'intera procedura è stata ripetuta per la sezione di integrazione.

5.5 Test del campionatore con zero insertion

Il test del campionatore con zero insertion è stato eseguito per mezzo di una testbench analoga a quella usata per il test dello stadio comb. In input sono stati forniti i punti (ordinate) di una retta con pendenza nota ed un segnale di *enabler* attivo per un tempo pari ad un quarto del periodo di clock. E' stato accertato che

- unicamente durante i periodi di tempo in cui il valore del segnale di *enabler* è pari a 1 il circuito è perfettamente trasparente,
- il modulo restituisce tutti valori logici 0 al di fuori di questi periodi.

5.6 Test del filtro nella sua interezza

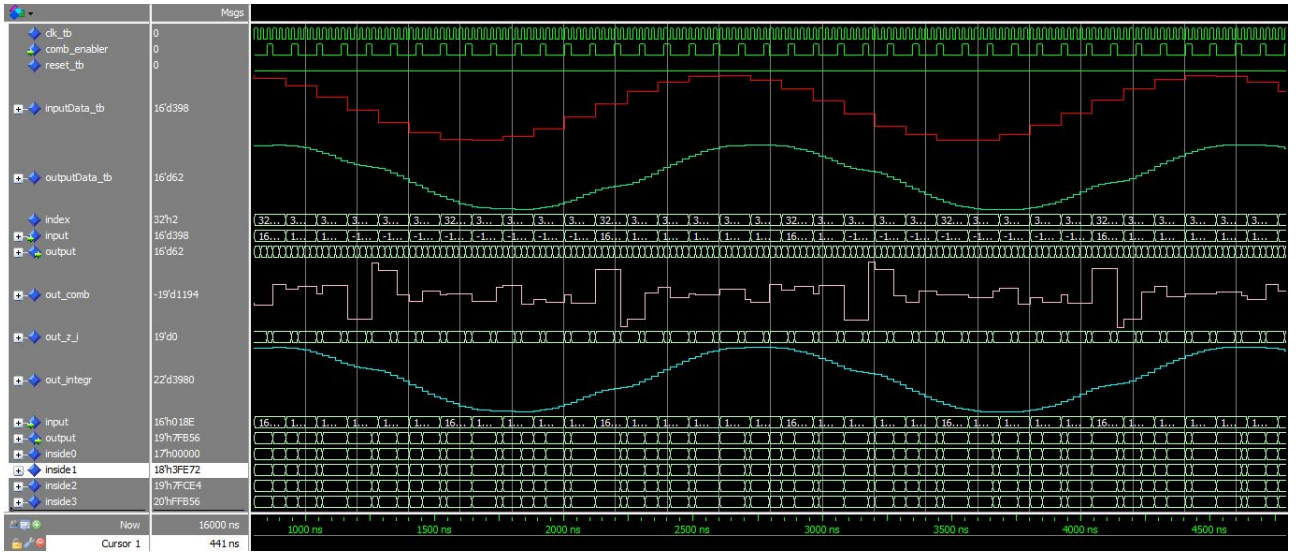
Come ultima fase di test è stata realizzata una testbench per accertarsi del corretto funzionamento dell'intero filtro CIC.
Tale testbench, per mezzo di una lookup table, fornisce in ingresso un segnale noto generato via software (Matlab) e permette di accertare visivamente il corretto funzionamento.
Di seguito è riportato il codice della testbench. In questo specifico caso i valori della lookup table corrispondono a 95 sample funzione *y=2^10*sin(x)* calcolata per x da 0 a 12π (le ordinate dei sample di differenziano per 0.4).

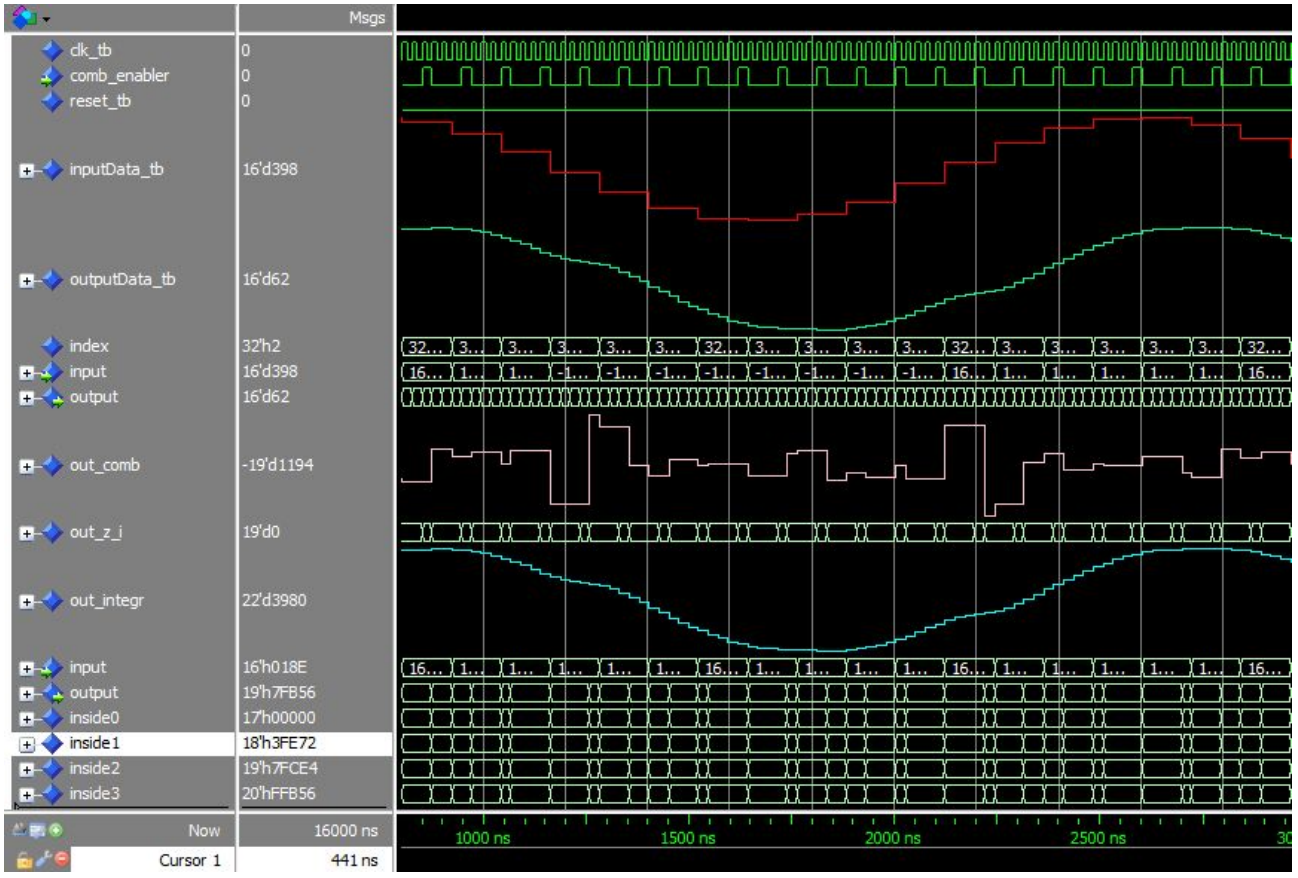
$$y_i = 2^{10} * \sin(x_i) : x_{i+1} - x_i = 0.4 \; \forall i = 0 \dots 95$$

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;


entity cic_filter_tb is
end cic_filter_tb;

architecture arch of cic_filter_tb is

    constant T_clk        : time      := 25 ns;
    signal clk_tb         : std_logic := '0';
    signal reset_tb       : std_logic := '1';
    constant N_tb         : integer   := 16;
    signal inputData_tb   : std_logic_vector(N_tb-1 downto 0) :=
        "0000000000000000";
    signal outputData_tb  : std_logic_vector(N_tb-1 downto 0);

    component cic_filter is
        generic(
            Nbit : integer
        );
        port (
            clk     : in   std_logic;
            reset   : in   std_logic;
            input   : in   std_logic_vector(Nbit-1 downto 0);
```

```
            output  : out  std_logic_vector(Nbit-1 downto 0)
        );
    end component cic_filter;

    type lut_t is array(natural range <>) of integer;
    constant lut : lut_t(0 to 94) := (
        0, 398, 734, 954, 1023, 931, 691, 343, -60, -454, -775, -975, -1021,
-905, -647, -287, 119, 505, 812, 991, 1013, 875, 598, 228, -179, -558, -848,
-1005, -1003, -843, -550, -170, 237, 606, 879, 1014, 988, 807, 498, 110, -295,
-654, -909, -1021, -972, -770, -445, -51, 351, 698, 934, 1023, 951, 728, 390,
-10, -408, -741, -958, -1024, -928, -685, -335, 68, 461, 780, 977, 1019, 900,
639, 277, -129, -514, -819, -994, -1012, -871, -592, -220, 187, 564, 852, 1006,
1000, 837, 541, 160, -246, -614, -885, -1016, -987, -802, -491, -102
    );

    signal index   : integer := 0;
    signal counter : integer := 0;

begin

    clk_tb <= not(clk_tb) after T_clk/2;
    reset_tb <= '0' after 4*T_clk;

    cicFilterComponent: cic_filter
        generic map (Nbit => N_tb)
        port map (
            clk => clk_tb, reset => reset_tb,
            input => inputData_tb, output => outputData_tb
        );

    process(clk_tb, inputData_tb) begin
        if (clk_tb'event and clk_tb = '1') then
            if (reset_tb = '1') then
                index <= 0;
                counter <= 0;
                inputData_tb <= (others => '0');
            else
                if(counter = 4) then
                    inputData_tb <= std_logic_vector(
                        to_signed(lut(index),inputData_tb'length)
                    );
                    if(index < 94) then
                        index <= index + 1;
                    else
                        index <= 0;
                    end if;
                    counter <= 0;
                else
                    counter <= counter + 1;
                end if;
            end if;
        end if;
    end process;

end arch;
```

La simulazione ha il risultato in figura





Si noti che

- il primo segnale corrisponde al segnale di clock,
- il secondo segnale è il segnale di *comb_enabler*,
- il segnale in rosso è il segnale di input,
- il segnale sottostante è il segnale di uscita dal filtro
- il segnale rosa è il segnale in uscita dalla sezione comb
- il segnale azzurro è il segnale in uscita dalla sezione di integrazione prima del troncamento ai 16 bit più significativi

E’ inoltre interessante notare che, in questo caso, il troncamento dei bit in uscita non comporta alcuna evidente perdita di precisione in quanto i segnali verde e azzurro hanno sostanzialmente la stessa forma e sono pressoché indistinguibili (nonostante la magnitudine sia nettamente diversa).

6 Sintesi logica

6.1 Vincoli

Per la fase di sintesi non sono stati identificati particolari vincoli ad eccezione di quello temporale dovuto alla frequenza di clock.
Immaginando l'applicazione del filtro CIC per interpolare un segnale audio analogico campionato a 44.1KHz, si ha una frequenza di clock richiesta di R volte 44,1KHz. Poiché il parametro R ha valore di 4, la frequenza di clock finale è di 176,4KHz.
E' stata scelta la frequenza base di 44.1KHz in quanto si tratta di uno dei più usati standard nel settore audio, per esempio le specifiche del Compact Disc Digital Audio prevedono una traccia audio digitale a 16 bit (pari dunque al numero di bit in ingresso al filtro) campionata proprio a questa frequenza.
Data la contenuta velocità del segnale di clock imposto e la intrinseca velocità del circuito del filtro CIC, non si prevedono situazioni di irrealizzabilità (clock slack negativo).

6.2 Sintesi

La fase di sintesi è stata effettuata con il software Xilinx Vivado 2019.2.
Come configurazione dei parametri per la sintesi sono stati lasciati quelli di default offerti dal programma.
La sintesi è stata completata senza errori o warning rilevanti.

6.2.1 Controllo dei vincoli

Il software Vivado permette di costruire un report sui timing dell'architettura in esame prima dell'effettiva implementazione su FPGA. In particolar modo permette di stimare il clock slack.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	5656,219 ns	Worst Hold Slack (WHS):	0,076 ns	Worst Pulse Width Slack (WPWS):	2834,000 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	239	Total Number of Endpoints:	239	Total Number of Endpoints:	186

All user specified timing constraints are met.

Come è possibile osservare dallo screen e come era stato previsto durante la decisione dei vincoli, il clock slack non è mai negativo e nel peggior caso è ampiamente positivo suggerendo, quindi, la possibilità di scegliere frequenze di clock molto maggiori rispetto a quella utilizzata.

6.2.2 Path critica

Grazie agli strumenti messi a disposizione dal software Vivado, è possibile andare a identificare tutte le data path ed in particolar modo la path critica ovvero quella che determina la frequenza massima di clock.

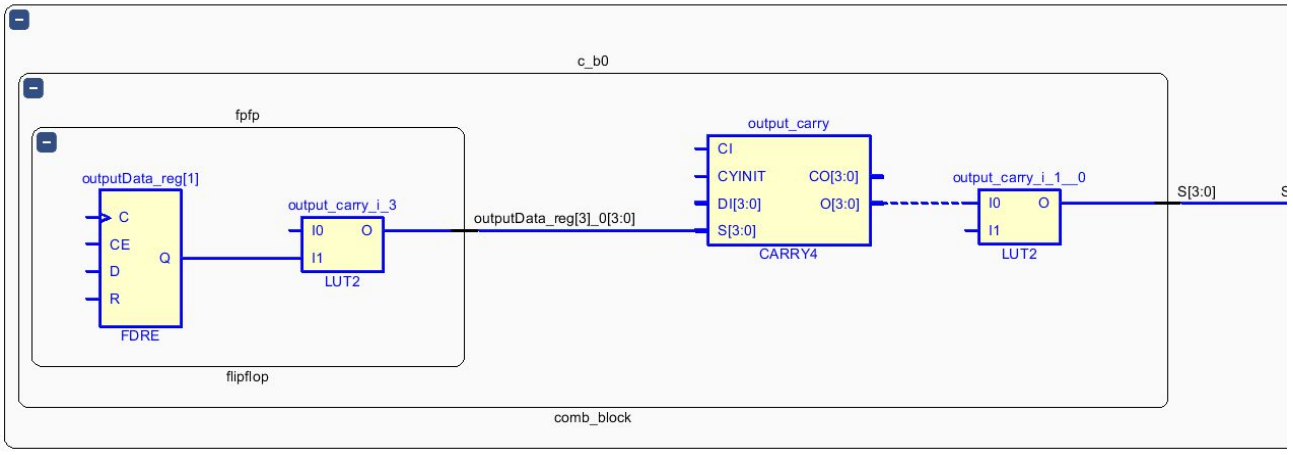
Summary

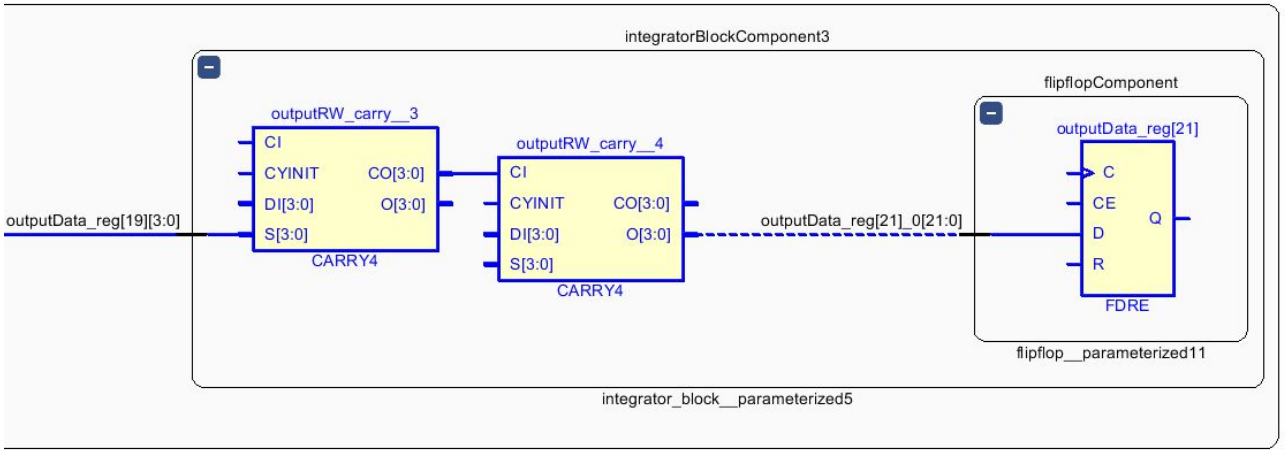
Name	Path 1
Slack	5656.219ns
Source	c_s/c_b0/fpfp/outputData_reg[1]/C (rising edge-triggered cell FDRE clocked by clock5669 {rise@0.000ns fall@2834.500ns period=5669.000ns})
Destination	i_s/integratorBlockComponent3/flipflopComponent/outputData_reg[21]/D (rising edge-triggered cell FDRE clocked by clock5669 {rise@0.000ns fall@2834.500ns period=5669.000ns})
Path Group	clock5669
Path Type	Setup (Max at Slow Process Corner)
Requirement	5669.000ns (clock5669 rise@5669.000ns - clock5669 rise@0.000ns)
Data Path Delay	12.662ns (logic 8.525ns (67.327%) route 4.137ns (32.673%))
Logic Levels	21 (CARRY4=13 LUT2=7 LUT3=1)
Clock Path Skew	-0.145ns
Clock Un...rtainty	0.035ns

Source Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clock5669 rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000	Site: L16	clk
net (fo=0)	0.000	0.000		clk
			Site: L16	clk_IBUF_inst/I
IBUF (Prop_ibuf_I_O)	(r) 0.976	0.976	Site: L16	clk_IBUF_inst/O
net (fo=1, unplaced)	0.800	1.776		clk_IBUF
				clk_IBUF_BUFG_inst/I
BUFG (Prop_bufg_I_O)	(r) 0.101	1.877		clk_IBUF_BUFG_inst/O
net (fo=185, unplaced)	0.584	2.461		c_s/c_b0/fpfp/CLK
FDRE				c_s/c_b0/fpfp/outputData_reg[1]/C

Vivado identifica come path critica la path che inizia dal registro del primo stadio di comb e termina con il registro dell'ultimo stadio integratore.
Come precedentemente affermato i parametri sono corretti: con un requirement di 5669ns (dovuti al clock a frequenza di 176,4KHz) si ha un path delay di soli 12,662ns.
L'inizio e la fine della path sono graficamente rappresentate di seguito





Data la struttura nota del filtro CIC, l'esito positivo di questa verifica era noto a priori a prescindere dalle modalità di sintesi utilizzate da Vivado.

6.2.3 Report dell’utilizzo

Il software Vivado permette inoltre di tener traccia del numero di componenti hardware derivati dalla sintesi e che dovranno essere implementati durante la fase di implementazione.
Il report dell’utilizzo è il seguente:

Name	Slice LUTs (17600)	Slice Registers (35200)	Bonded IOB (100)	BUFGCTRL (32)
cic_filter	214	185	34	1
c_s (comb_stages)	111	70	0	0
c_b0 (comb_block)	32	16	0	0
c_b1 (comb_block__parameterized1)	21	17	0	0
c_b2 (comb_block__parameterized3)	19	18	0	0
c_b3 (comb_block__parameterized5)	39	19	0	0
i_s (integrator_stages)	63	82	0	0
integratorBlockComponent0 (integrator_block)	0	19	0	0
integratorBlockComponent1 (integrator_block__parameterized1)	39	20	0	0
integratorBlockComponent2 (integrator_block__parameterized3)	22	21	0	0
integratorBlockComponent3 (integrator_block__parameterized5)	2	22	0	0

Dove:

- LUTs si riferisce al numero di lookup table
- Registers al numero di registri
- Bonded IOB al numero di IOB (pin) allocabili nella board
- BUFGCTRL al numero di global clock buffer necessari

Secondo questo report è quindi possibile implementare successivamente tale modello su piattaforma Xilinx FPGA Zynq. Non solo: è probabilmente possibile ripetere la sintesi (ed effettuare l’implementazione) per board più piccole ma più economiche.

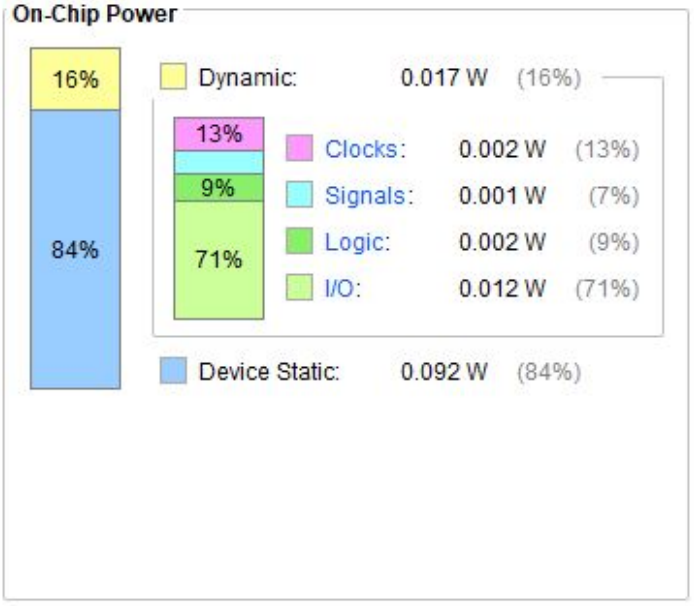
6.2.4 Report sul consumo energetico

Il software Vivado offre la possibilità di visualizzare un report sull’utilizzo della potenza costruendo quindi una stima dell’energia richiesta dalla board in fase di esecuzione.
Il report, come è possibile notare dall’immagine sotto, distingue tra potenza dinamica e statica mettendo in evidenza le cause di tali assorbimenti energetici.

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.109 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,3°C**
Thermal Margin:	58,7°C (5,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Si noti che la grande maggioranza della potenza è statica, solo il 16% è dinamica ed è spesa principalmente per il segnale di clock (che deve essere propagato a tutto il filtro CIC) e per le interfacce di I/O.
Il report è stato effettuato considerando i seguenti parametri ambientali di default:

Device Settings

Temp grade:	commercial	
Process:	typical	

Environment Settings

Output Load:	0	pF	[0 - 10000]
Junction temperature:	26.034	°C	
Ambient temperature:	25	°C	
Effective ϑJA:	11.533	°C/W	[0 - 100]
Airflow:	250	LFM	
Heat sink:	none		
ϑSA:	0	°C/W	[0 - 100]
Board selection:	medium (10"x10")		
Number of board layers:	8to11 (8 to 11 Layers		
ϑJB:	9.3	°C/W	[0 - 100]
Board temperature:	25	°C	[-55 - 85]

Anche tutti gli altri parametri sono stati lasciati come di default.

6.3 Conclusioni della fase di sintesi e massima frequenza di funzionamento

Dato l'elevato clock slack positivo in relazione al delay della worst path, si ha un ampio margine di modifica del segnale di clock.
La sintesi è stata ripetuta anche con frequenze di clock derivanti da altri standard audio:

Applicazione	**Frequenza campionamento audio**	**Frequenza di clock**
DVD-Audio, alcune tracce DVD LPCM, tracce audio del formato BD-ROM (Disco Blu-ray) e HD DVD (High-Definition DVD)	96KHz	384KHz
produzione di un audio CD (multipla di 44,100 Hz)	176,4KHz	705,6KHz
DVD-Audio, alcune tracce DVD LPCM, tracce audio del formato BD-ROM (Disco Blu-ray) e HD DVD (High-Definition | DVD). Registratori e sistemi di editing in Alta definizione	192KHz	768KHz

In tutti i casi la situazione di clock slack abbondantemente positivo non cambia di molto ed è quindi possibile affermare che il modello proposto e sviluppato può essere implementato su bord Xilinx FPGA Zynq per un utilizzo in ambito audio digitale.
Osservando la stima del delay della path critica (12.662ns) è possibile affermare che una stima approssimativa della massima frequenza di funzionamento sia di 78,97MHz, fin troppo elevata per qualsiasi applicazione audio, ma probabilmente utile in ambito scientifico e di elaborazione dati.

7 Conclusioni

I filtri CIC (Cascated Integrated Comb) rappresentano un’ottima soluzione per le operazioni di decimazione e interpolazione.
La loro struttura è composta da blocchi base (sommatori, registri) inseriti in cascata e questo fa di loro circuiti non complessi per quanto riguarda la produzione industriale.
In più vediamo che i blocchi basi di cui sono composti sono sommatori e registri, e questo permette loro di essere molto veloci nelle operazioni da effettuare, come mostrano i test con le varie frequenze di clock.
Infatti con la sintesi abbiamo visto che la stima della massima frequenza di clock è così elevata da poter impiegare il filtro CIC in ambito audio digitale, rispettando ampiamente tutti i più comuni standard, e in altri ambiti dove è richiesto un tempo di campionamento ridotto.
Sono presenti nella letteratura anche delle varianti con lo scopo di migliorare ulteriormente le performance e contemporaneamente ridurre il numero dei componenti necessari alla costruzione.
Dai test notiamo che il filtro CIC si comporta molto bene con il troncamento al MSB nonostante l’uscita non sia sensibile al segnale per valori bassi.
Tutti questi fattori fanno sì che il filtri CIC siano importanti componenti inseriti fin dalla loro invenzione in tutti i circuiti dove operazioni come interpolazione o decimazione sono necessarie.