



# SISTEMI DIGITALI INTEGRATI

POLITECNICO DI TORINO

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI

---

## Progetto Audio Processor

Esercitazione finale di progetto sulle architetture integrate

---

*Autori:*

Palma Paolo 306046

Porcaro Mario 315888

Rinieri Filippo 317894

*Professore:*

Massimo Ruo Roch

**Gruppo 9**

Anno accademico: 2022-2023

# Contents

|          |                                 |           |
|----------|---------------------------------|-----------|
| <b>1</b> | <b>Trasmittitore</b>            | <b>4</b>  |
| 1.1      | Introduzione . . . . .          | 4         |
| 1.2      | Protocollo RS232 . . . . .      | 4         |
| 1.3      | Datapath . . . . .              | 4         |
| 1.4      | Timing Diagram . . . . .        | 5         |
| 1.5      | Diagramma degli stati . . . . . | 6         |
| 1.6      | Risultati . . . . .             | 6         |
| <b>2</b> | <b>Ricevitore</b>               | <b>8</b>  |
| 2.1      | Introduzione . . . . .          | 8         |
| 2.2      | Datapath . . . . .              | 8         |
| 2.3      | Timing Diagram . . . . .        | 9         |
| 2.4      | Diagramma degli stati . . . . . | 14        |
| 2.5      | Risultati . . . . .             | 14        |
| <b>3</b> | <b>Filtri</b>                   | <b>16</b> |
| 3.1      | Introduzione . . . . .          | 16        |
| 3.2      | Nozioni teoriche . . . . .      | 16        |
| 3.3      | Coefficienti . . . . .          | 17        |
| 3.4      | Datapath . . . . .              | 17        |
| 3.5      | Timing Diagram . . . . .        | 19        |
| 3.6      | Diagramma degli stati . . . . . | 19        |
| 3.7      | Risultati . . . . .             | 19        |
| <b>4</b> | <b>Audio processor</b>          | <b>22</b> |
| 4.1      | Diagramma a blocchi . . . . .   | 22        |
| 4.2      | Timing diagram . . . . .        | 22        |
| <b>5</b> | <b>Simulazioni</b>              | <b>24</b> |
| 5.1      | Script Python . . . . .         | 24        |
| 5.2      | Plot dei risultati . . . . .    | 27        |
| <b>6</b> | <b>Appendice</b>                | <b>31</b> |
| 6.1      | Trasmittitore . . . . .         | 31        |
| 6.1.1    | Counter . . . . .               | 31        |
| 6.1.2    | Parallel Register . . . . .     | 32        |
| 6.1.3    | Shift Register . . . . .        | 33        |
| 6.1.4    | Serial TX . . . . .             | 34        |
| 6.2      | Ricevitore . . . . .            | 37        |
| 6.2.1    | Counter . . . . .               | 37        |
| 6.2.2    | Counter 3 bit . . . . .         | 38        |
| 6.2.3    | Parallel Register . . . . .     | 39        |
| 6.2.4    | Shift register . . . . .        | 40        |
| 6.2.5    | SIPO Register . . . . .         | 41        |

|       |                                    |    |
|-------|------------------------------------|----|
| 6.2.6 | Serial RX . . . . .                | 42 |
| 6.3   | Audio Processor . . . . .          | 47 |
| 6.3.1 | Counter . . . . .                  | 47 |
| 6.3.2 | Saturated Adder . . . . .          | 48 |
| 6.3.3 | Saturated Multiplier . . . . .     | 50 |
| 6.3.4 | Audio Processor . . . . .          | 51 |
| 6.4   | TestAudioProcessor_HF.py . . . . . | 57 |
| 6.5   | TestAudioProcessor_LF.py . . . . . | 59 |
| 6.6   | compile.do . . . . .               | 61 |

### **Abstract**

Nella seguente relazione viene presentato il progetto finale del corso di Sistemi Digitali Integrati composto dalla realizzazione di un blocco trasmettitore e un blocco ricevitore con protocollo di comunicazione RS232, e un blocco di filtraggio digitale che implementa due filtri shelving.

Vengono descritti il flusso del progetto, le problematiche riscontrate e vengono motivate le soluzioni adottate. Il progetto viene poi caricato ed eseguito sulla FPGA della scheda VirtLab in dotazione e vengono commentati i risultati ottenuti.

# 1 Trasmettitore

## 1.1 Introduzione

Il blocco denominato Trasmettitore implementa una trasmissione di dati seguendo il protocollo di comunicazione RS232. La comunicazione avviene, per questo progetto, tra la FPGA lato user e il microcontrollore lato user della scheda Virtlab tramite la linea *lsas22*.

## 1.2 Protocollo RS232

Lo standard RS232 descrive le specifiche di una comunicazione seriale asincrona full-duplex di parole da 8 bit. Lo stato *idle* del canale di trasmissione è associato al valore logico alto.

È altresì necessario rispettare le seguenti specifiche:

- velocità di trasmissione = 9600 baud/s
- inizio della trasmissione segnalato da un bit di start (valore logico basso)
- fine della trasmissione segnalata da un bit di stop (valore logico alto)

Qualora una di queste specifiche non fosse rispettata il pacchetto viene considerato errato e viene scartato.

## 1.3 Datapath

Il datapath del progetto del trasmettitore seriale è mostrato in figura 1.

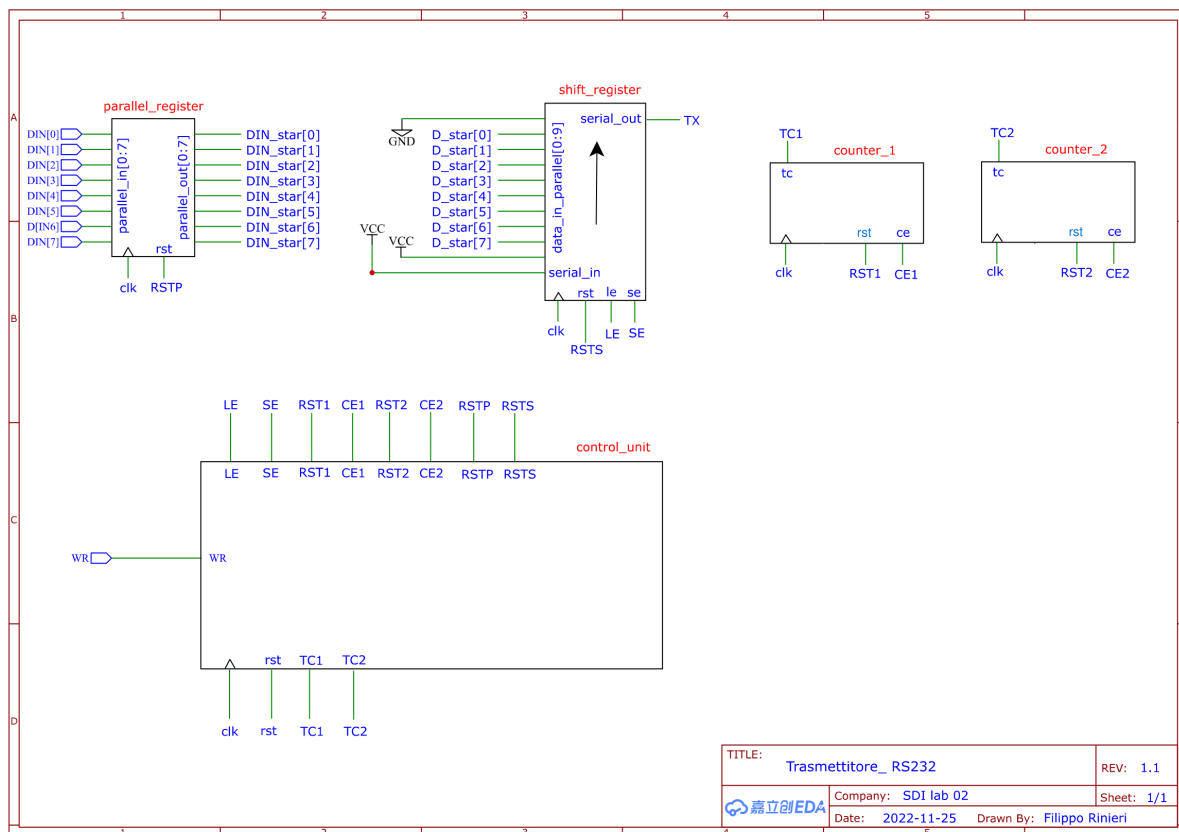
Per la realizzazione del modulo vengono utilizzati due shift register e due contatori.

**Parallel register** Il primo registro - di dimensione 8 bit - campiona in modo continuo il bus di ingresso dei dati DIN[0:7]. Ciò permette di mantenere valido il dato ricevuto per tutta la durata di un periodo di clock; il segnale DIN\_star[0:7] può essere così caricato nel registro di invio qualora venisse dato il segnale di start.

Il componente è sincrono e viene resettato dalla Control Unit.

**Shift register** Il secondo registro permette l'ingresso dei dati sia in parallelo sia in seriale ed ha un'unica uscita seriale; la dimensione di questo registro è 10 bit. L'uscita seriale è collegata al segnale TX mentre i 10 bit di ingresso sono così collegati: il primo bit (start bit) è posto a '0', i bit da 1 a 8 sono collegati al vettore DIN\_star e il decimo bit (stop bit) è collegato al potenziale di '1' logico. L'ingresso seriale è posto sempre a '1'.

Il componente è sincrono e resettato dalla CU. Sono presenti due segnali di controllo: 'le' (*load enable*) che carica il registro in parallelo, e 'se' (*shift enable*) che effettua lo shift di una posizione, inserendo come ingresso il segnale serial\_in.



**Figure 1:** Datapath del trasmettitore RS232

**Counter** Sono presenti due contatori.

Il counter1 viene utilizzato come divisore di frequenza: esso alza il segnale TC1 non appena raggiunge il valore di 1040 conteggi. Così la frequenza di 10 MHz del clock viene divisa per un fattore 1042 (questo contando anche un colpo di clock in più che viene aggiunto dalla macchina a stati) e si ottiene su TC1 il baud rete desiderato di 9600 baud/s.

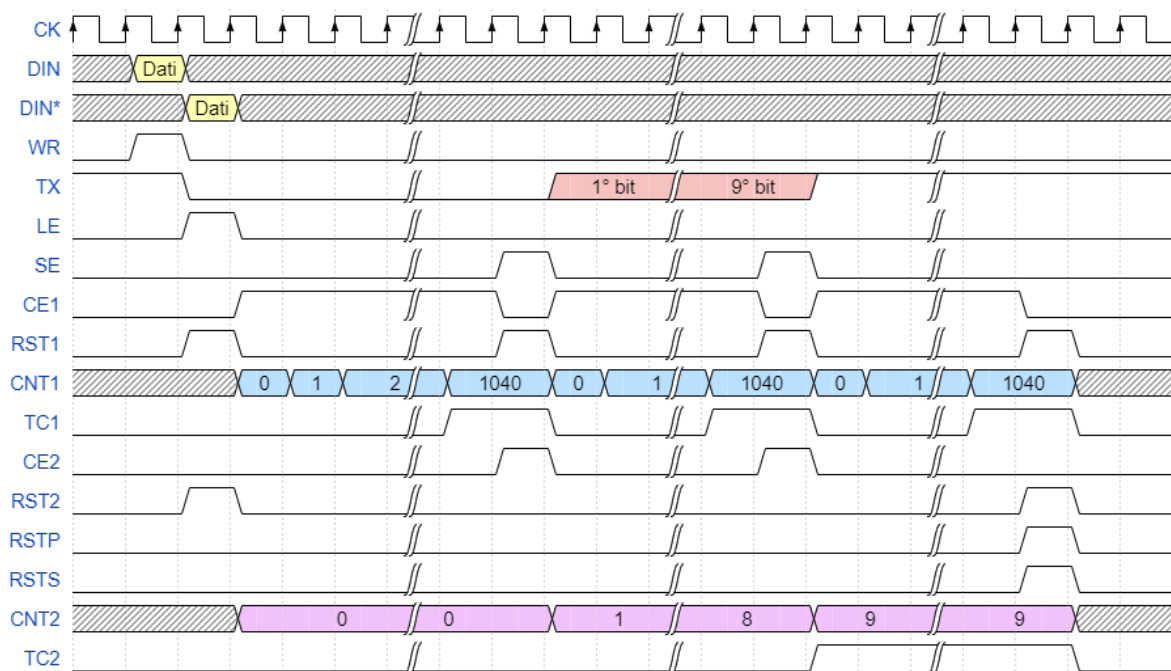
Il counter\_2 viene utilizzato per contare i 10 bit trasmessi e termina il conteggio a 9 alzando TC2.

**Control Unit** L'unico ingresso della Control Unit è il segnale di inizio trasmissione WR. La macchina è sincrona e possiede un segnale di reset asincrono; gli altri due segnali di controllo sono TC1 e TC2.

I segnali di stato che escono dalla CU sono: i segnali di controllo dello shift register (RSTS, LE e SE), i segnali di controllo dei due contatori (RST1, CE1, RST2 e CE2) e il reset del parallel register RSTP.

## 1.4 Timing Diagram

Il timing diagram del trasmettitore è mostrato in figura 2.



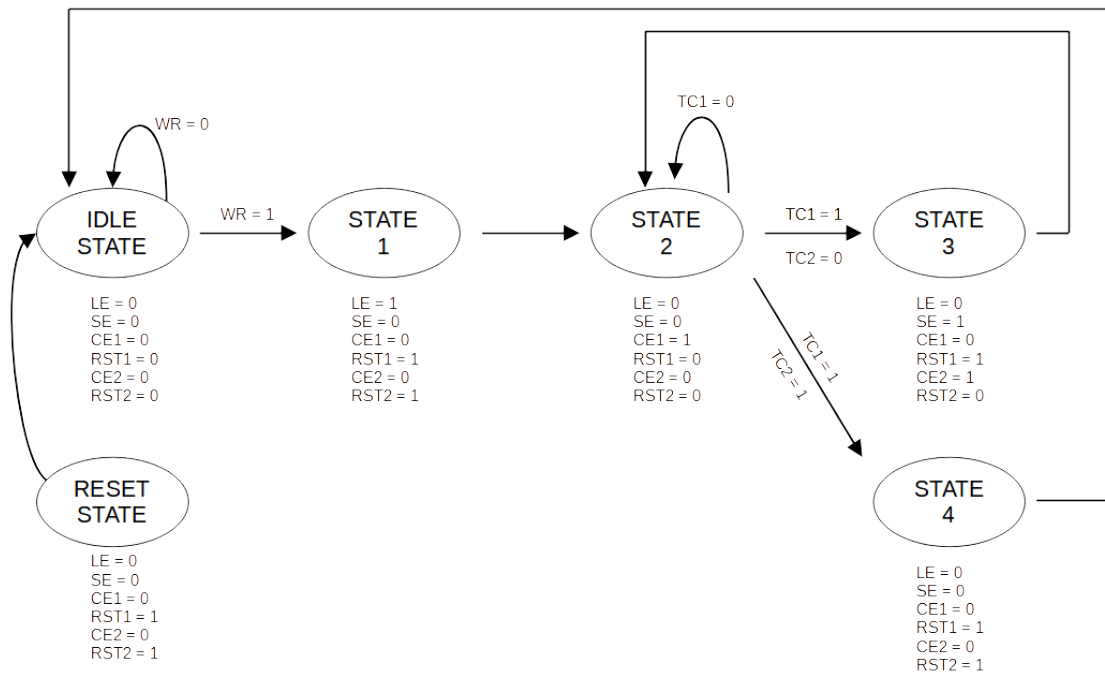
**Figure 2:** Timing diagram del trasmettitore

## 1.5 Diagramma degli stati

Il diagramma degli stati è riportato in figura 3.

## 1.6 Risultati

Il blocco viene testato utilizzando come ricevitore il microcontrollore user e inviando un byte alla volta, il cui valore è dettato dal valore degli switches SW[0:7]. Si rileva che i byte vengono trasmessi correttamente.



**Figure 3:** Diagramma degli stati del trasmettitore



## 2 Ricevitore

### 2.1 Introduzione

In questo capitolo viene presentata la progettazione di un blocco ricevitore su standard RS232.

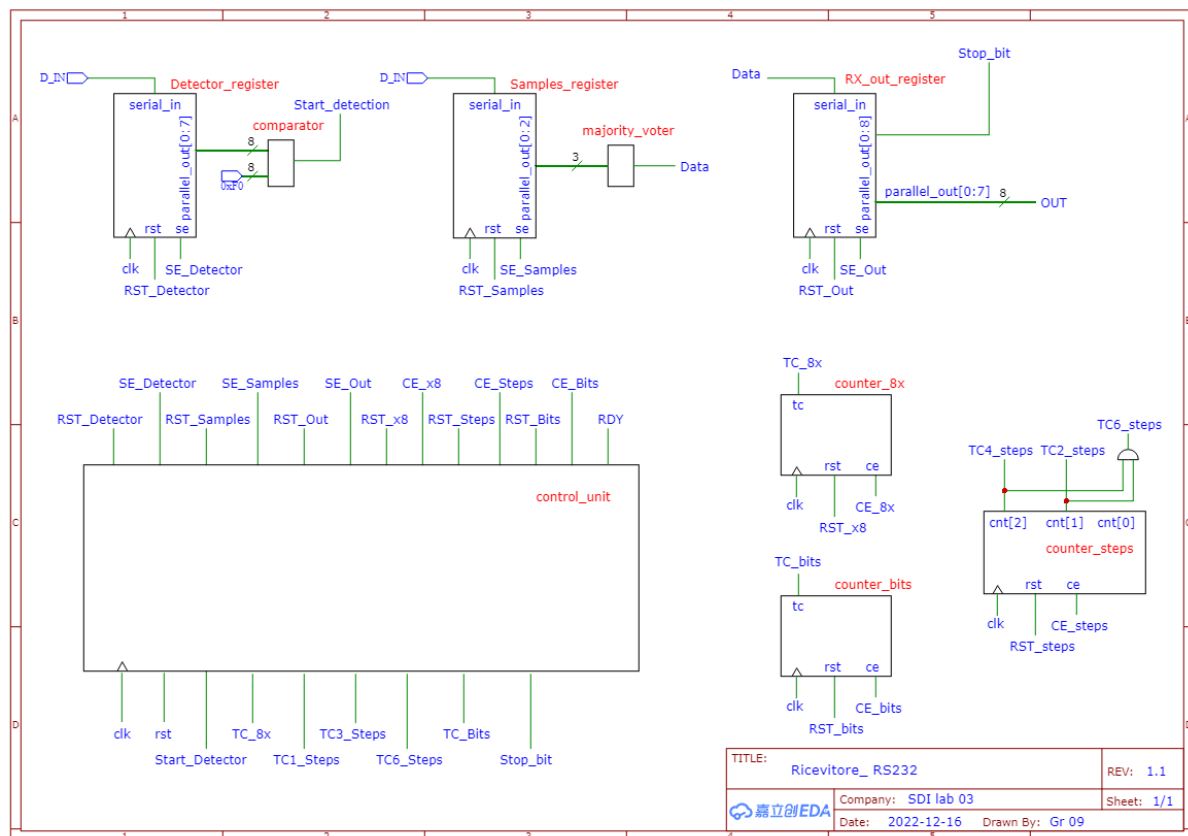
Il blocco riceve da seriale uno stream di 10 bit e verificata la correttezza del pacchetto restituisce in uscita gli 8 bit di dati. In particolare, il ricevitore verifica la presenza del bit di start e del bit di stop e, in caso di errore, scarta il pacchetto senza comunicare alcun messaggio di errore.

Per ovviare possibili disturbi della linea di trasmissione che potrebbero corrompere il valore di un dato, i bit vengono sovracampionati di un fattore 8 e vengono considerati i tre campioni centrali dai quali viene votato il valore del bit utilizzando una logica di maggioranza.

In questo progetto il ricevitore è collegato alla linea *lsas(21)*.

### 2.2 Datapath

Il datapath del ricevitore RS232 è mostrato in figura 4.



**Figure 4:** Datapath del ricevitore

Per implementare il ricevitore vengono istanziati tre registri e tre contatori.

**Detector register** Il primo registro di dimensione 8 bit riceve campioni seriali dalla linea e viene triggerato dal segnale di controllo SE\_Detector in modo da campionare alla frequenza richiesta di 9600 baud/s. L'uscita parallela del registro è comparata con un vettore di valore 0xF0 tramite un blocco comparatore e, in occorrenza di un match, viene alzato il segnale Start\_Detection.

**Samples register** Il registro riceve tre campioni seriali, pilotato dal segnale di controllo SE\_Samples, che corrispondono ai campioni centrali del bit nella trasmissione. I tre valori sono passati quindi ad un majority voter che elegge mediante una logica di maggioranza il valore più frequente, il quale viene trasmesso nel segnale Data.

**RX out register** Il registro riceve 9 bit dall'ingresso seriale Data e restituisce in uscita il byte ricevuto OUT[0:7] e l'ultimo bit ricevuto, il bit di stop. L'acquisizione dei campioni avviene mediante il segnale di controllo SE\_OUT.

**Counter 8x** Il contatore funziona come divisore della frequenza di clock per ottenere un segnale su TC\_8x a frequenza 8 volte maggiore della baud rate.

**Counter steps** Il contatore viene incrementato ogni volta che il counter\_8x termina il suo conteggio e tiene conto del numero del campione nella sequenza. Tre valori di terminal count sono considerati:

- TC\_steps = 2: è usato per contare l'acquisizione dei tre campioni centrali del dato;
- TC\_steps = 4: è usato per contare i campioni di attesa prima del campionamento del prossimo bit;
- TC\_steps = 6: è usato per scatenare la sequenza di campionamento del primo bit dopo aver ricevuto il segnale di start

**Counter bits** Il contatore tiene traccia dei bit ricevuti e il conteggio termina a 9, quando cioè sono presenti gli 8 bit di dati e il bit di stop. Il segnale di stato è TC\_bits e il segnale di incremento del conteggio è CE\_bits.

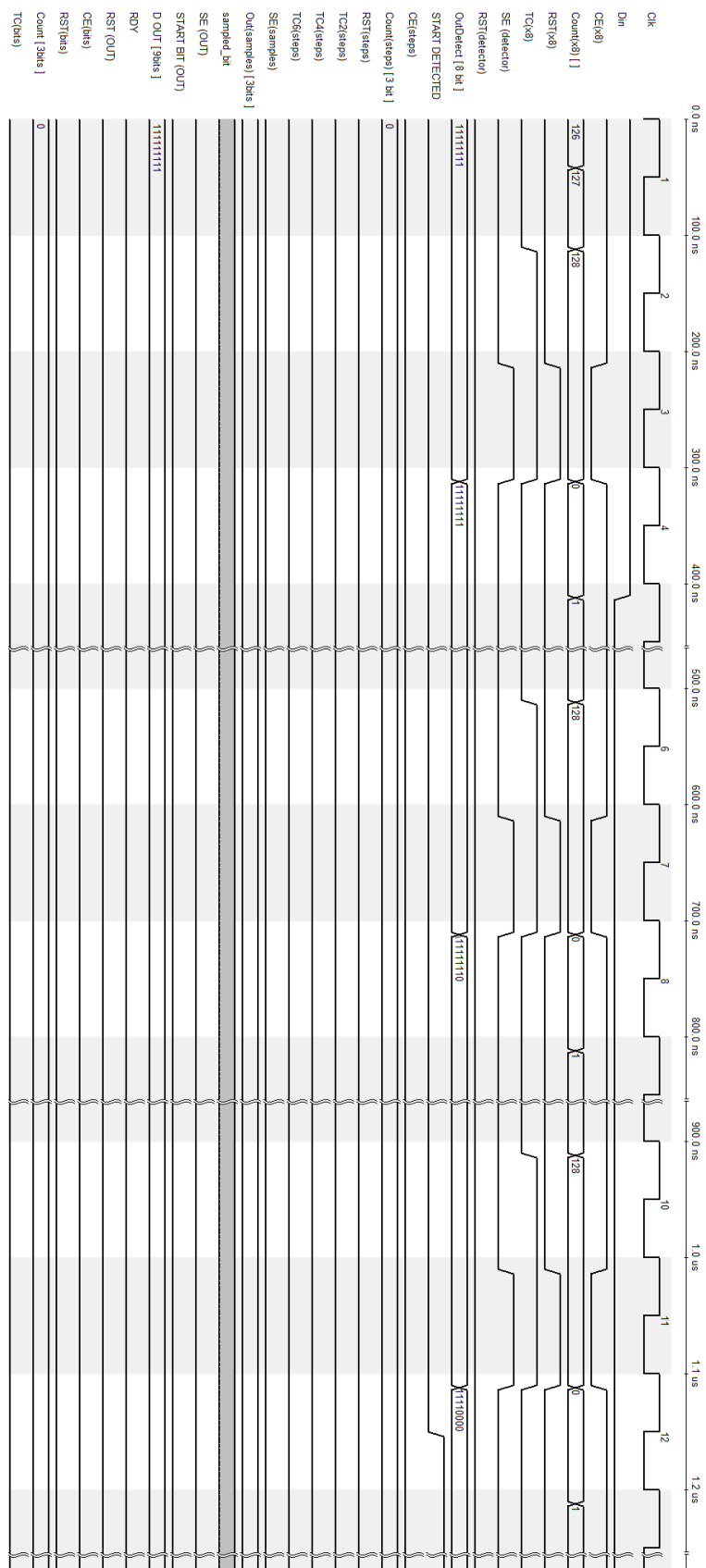
**NOTA:** Tutti i registri e tutti i contatori sono sincroni e possiedono un segnale di reset rispettivamente. Il segnale di reset è sincrono.

**Control Unit** La macchina (sincrona e con reset asincrono) riceve come segnali di controllo Start\_detect, i cinque segnali di stato TC dei counter e il segnale Stop\_bit.

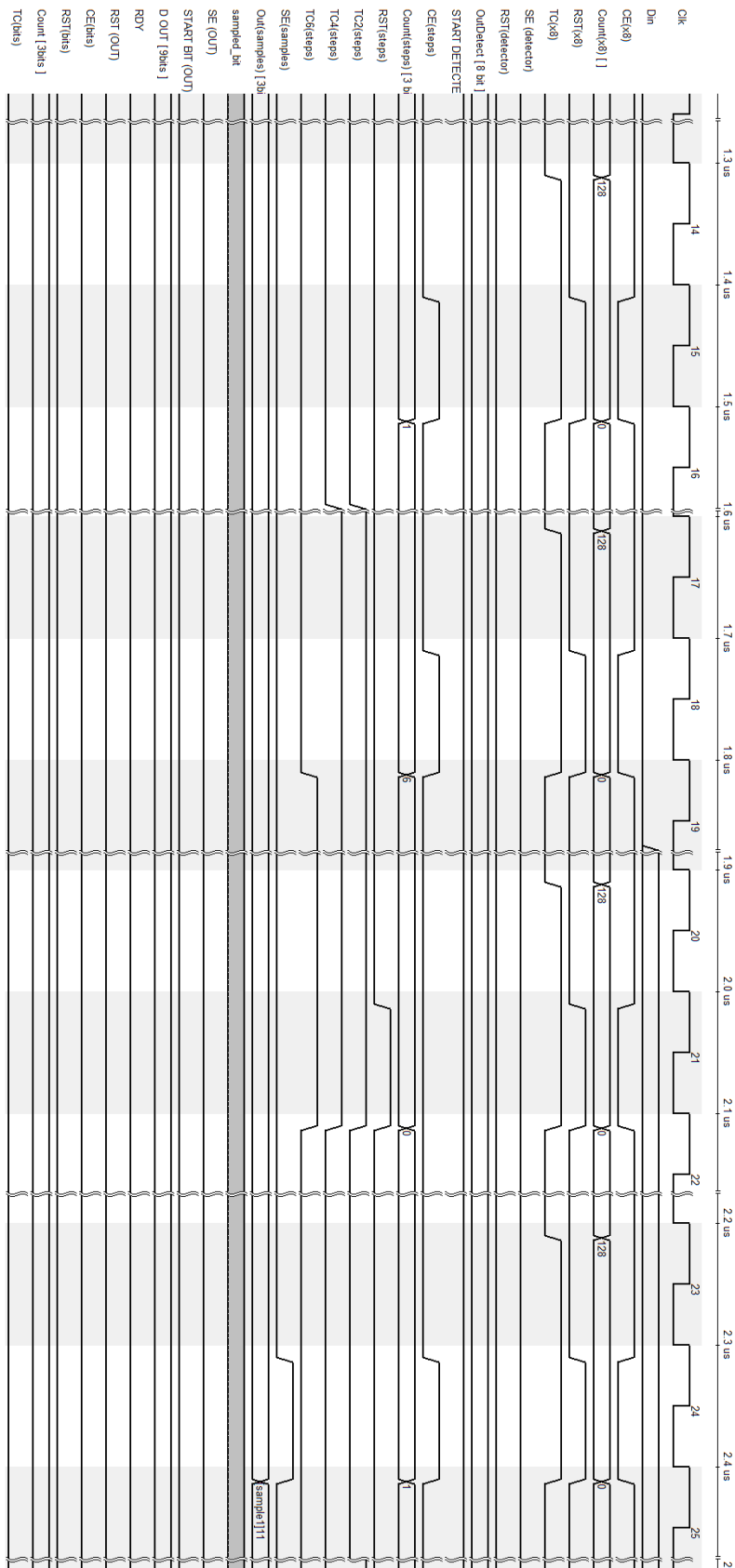
I segnali di stato della CU sono i reset dei sei componenti, i controlli di shift enable SE dei tre registri e i count enable CE dei tre contatori; infine il segnale RDY.

## 2.3 Timing Diagram

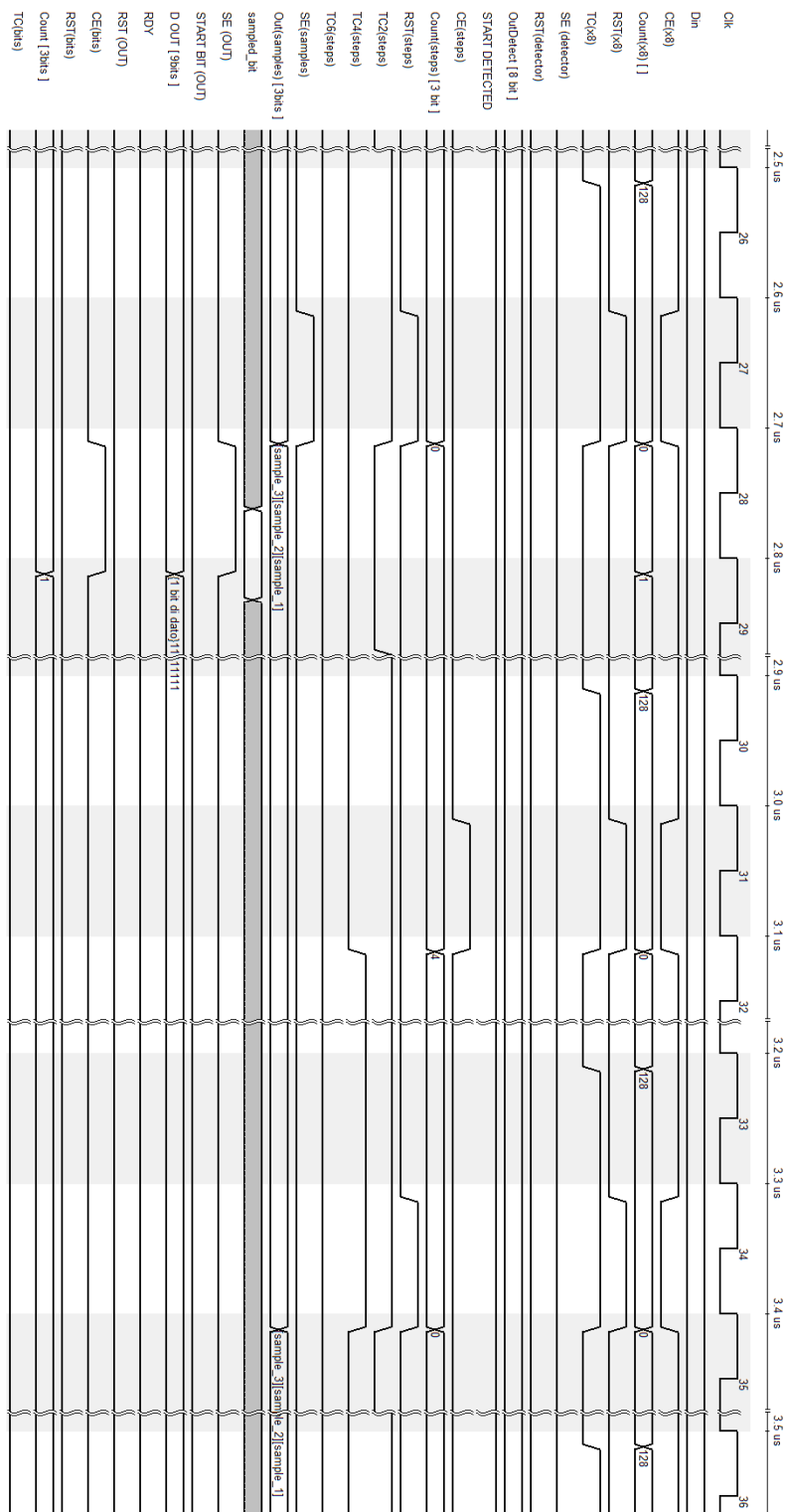
Il timing diagram del ricevitore è mostrato nelle figure 5, 6, 7 e 8.



**Figure 5:** Timing diagram ricevitore - parte 1



**Figure 6: Timing diagram ricevitore - parte 2**



**Figure 7:** Timing diagram ricevitore - parte 3



**Figure 8:** Timing diagram ricevitore - parte 4

## 2.4 Diagramma degli stati

Il diagramma degli stati è riportato in figura 9.

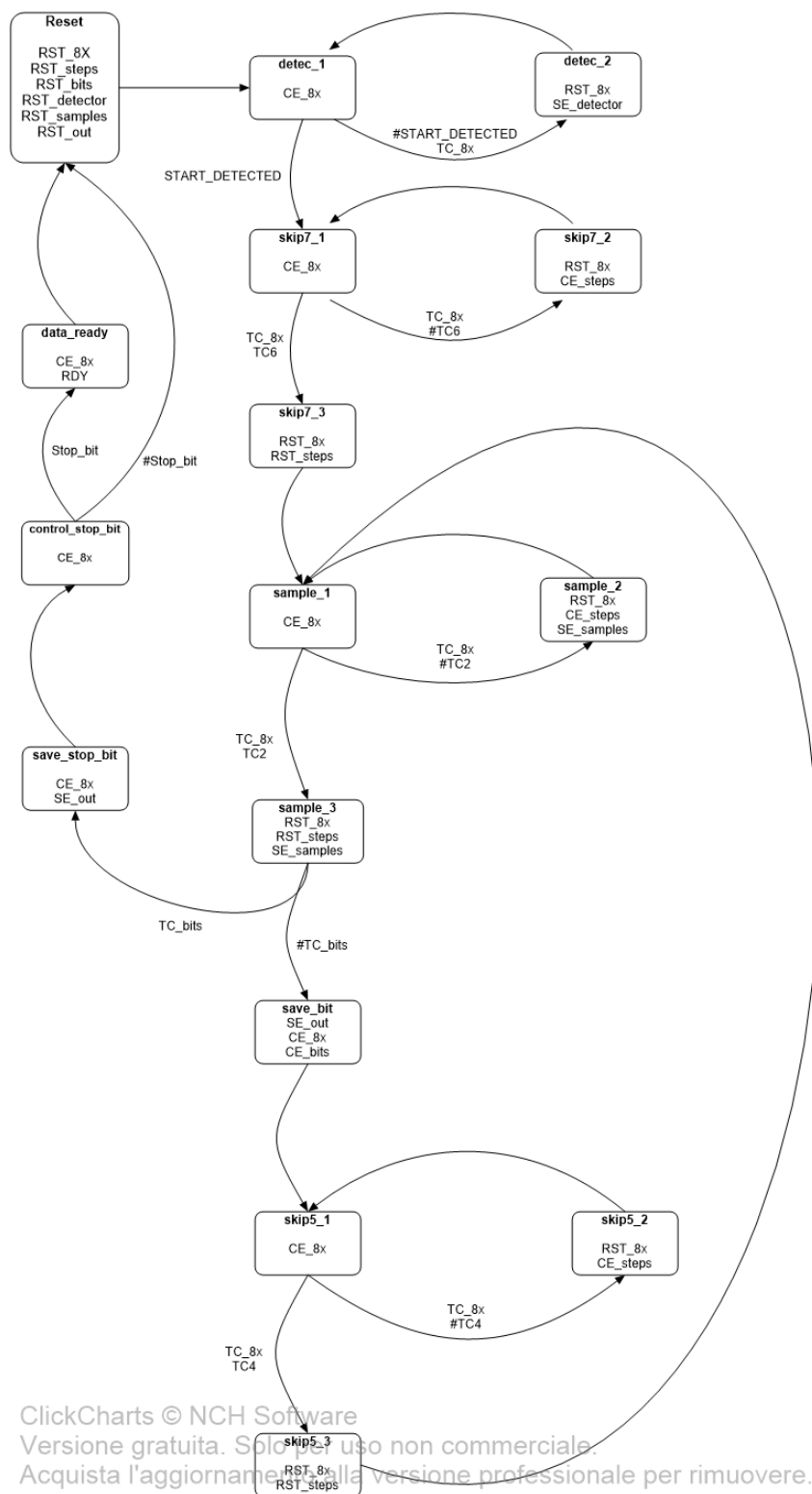
La FSM consta di 16 stati:

- reset: reset di tutti i componenti del datapath
- detect (1 e 2): rilevamento della *start condition*
- skip 7 (1, 2 e 3): attesa di 7 conteggi per campionare il primo dato a metà del suo periodo
- sample (1, 2 e 3): acquisizione di tre campioni del segnale
- save bit: salvataggio del dato rilevato
- skip 5 (1, 2 e 3): attesa di 5 conteggi per campionare il dato successivo
- save stop bit: campionamento del bit di stop
- control stop bit: controllo validità del bit di stop
- data ready: segnalazione validità del messaggio ricevuto

## 2.5 Risultati

Per testare il ricevitore viene utilizzato il trasmettitore RS232 - che a questo punto del progetto è già funzionante e collaudato - e il microcontrollore lato user; quest'ultimo invia byte casuali che vengono ricevuti e letti dalla macchina e re-inviati in loopback attraverso il trasmettitore.

Si rileva che il ricevitore così progettato è funzionante secondo le specifiche imposte.



**Figure 9:** Diagramma degli stati del ricevitore



## 3 Filtri

### 3.1 Introduzione

In questo capitolo viene descritta l'implementazione di un blocco di filtraggio digitale. In particolare vengono realizzati due filtri shelving del primo ordine seguendo le seguenti specifiche:

- codifica dei campioni audio PCM (*Pulse Code Modulation*)
- dimensione del dato di 8 bit
- formato RAW
- frequenza di campionamento di 44.1 kHz

È richiesto che i due filtri siano selezionabili dall'utente mediante due switches, che danno luogo a tre possibili configurazioni:

- no switches: loopback
- switch 0: filtro shelving cut low-pass
- switch 1: filtro shelving cut high-pass

### 3.2 Nozioni teoriche

I filtri Shelving vengono descritti dalle seguenti equazioni alle differenze:

$$y_1(n) = a_{B/C}x(n) + x(n-1) - a_{B/C}y_1(n-1)$$

$$y(n) = \frac{H_0}{2}(x(n) \pm y_1(n)) + x(n)$$

dove il coefficiente  $a_B$  (boost) oppure  $a_C$  (cut) indica la frequenza di taglio,  $H_0$  indica il guadagno della banda di frequenza considerata e il segno  $+$  o  $-$  nella seconda equazione indica se si tratta rispettivamente di un filtro low-pass o high-pass.

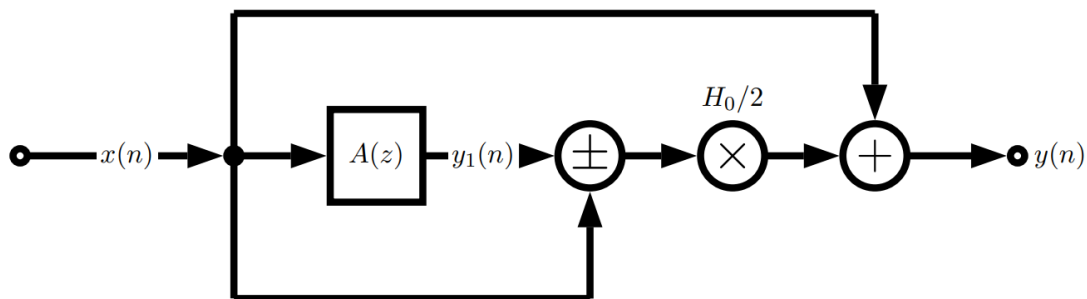
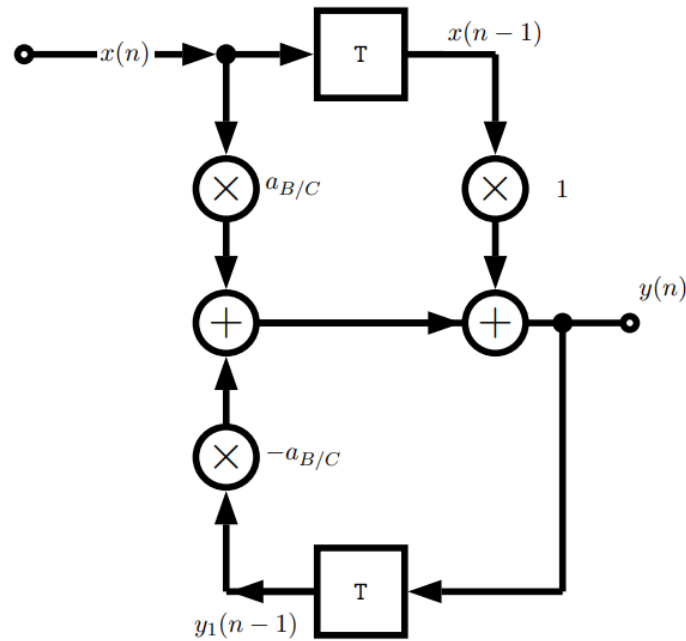


Figure 10



**Figure 11:** Implementazione filtro shelving

### 3.3 Coefficienti

I due filtri scelti sono i seguenti:

1. filtro low-pass cut con  $f_c = 1$  kHz e  $G = -20$  dB
2. filtro high-pass cut con  $f_C = 300$  Hz e  $G = -30$  dB

dove  $f_c$  è la frequenza di taglio.

I coefficienti calcolati quindi valgono rispettivamente:

1.  $a_C = 0.1785$ ,  $H_0 = -0.9$
2.  $a_C = 0.1498$ ,  $H_0 = -0.9180$

### 3.4 Datapath

Il datapath del blocco di filtraggio è mostrato in figura 12 e 13.

**Registri** Vengono utilizzati 5 registri sincroni, ognuno dotato del proprio segnale di *load enable* e di *reset*.

- sample: è utilizzato per campionare in continuazione il segnale d'ingresso e permette di avere un dato valido per l'elaborazione al momento dell'arrivo del segnale di start;
- Xn: memorizza il campione d'ingresso corrente della sequenza;
- Xp: memorizza il campione d'ingresso precedente della sequenza;

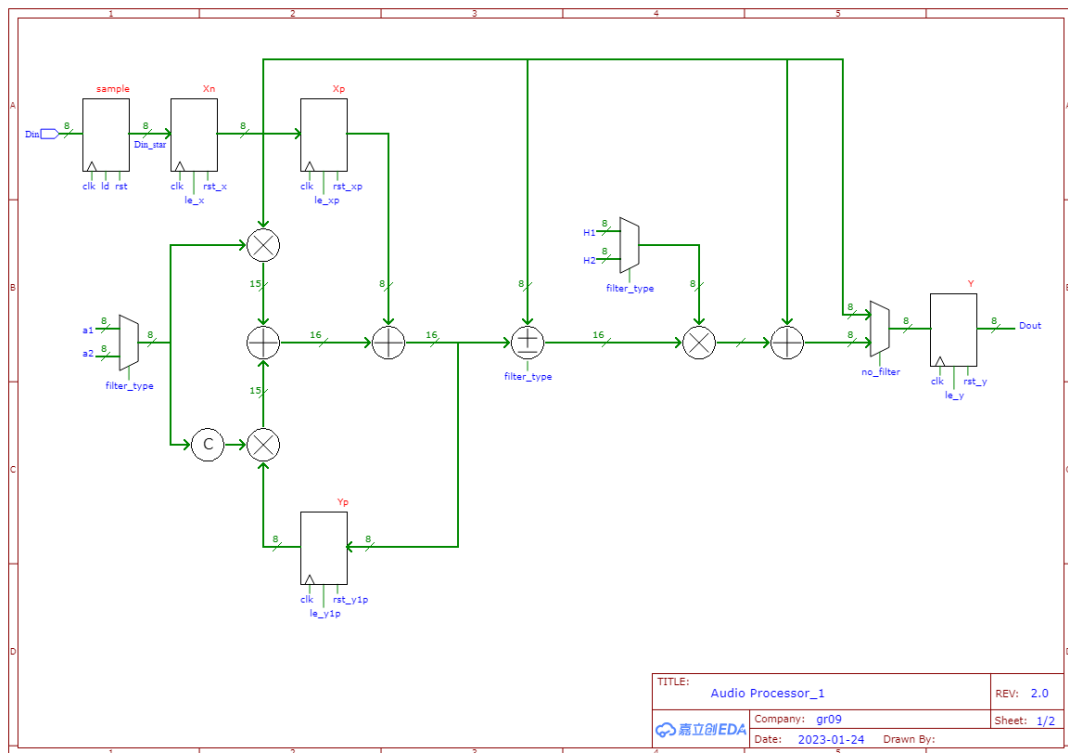


Figure 12

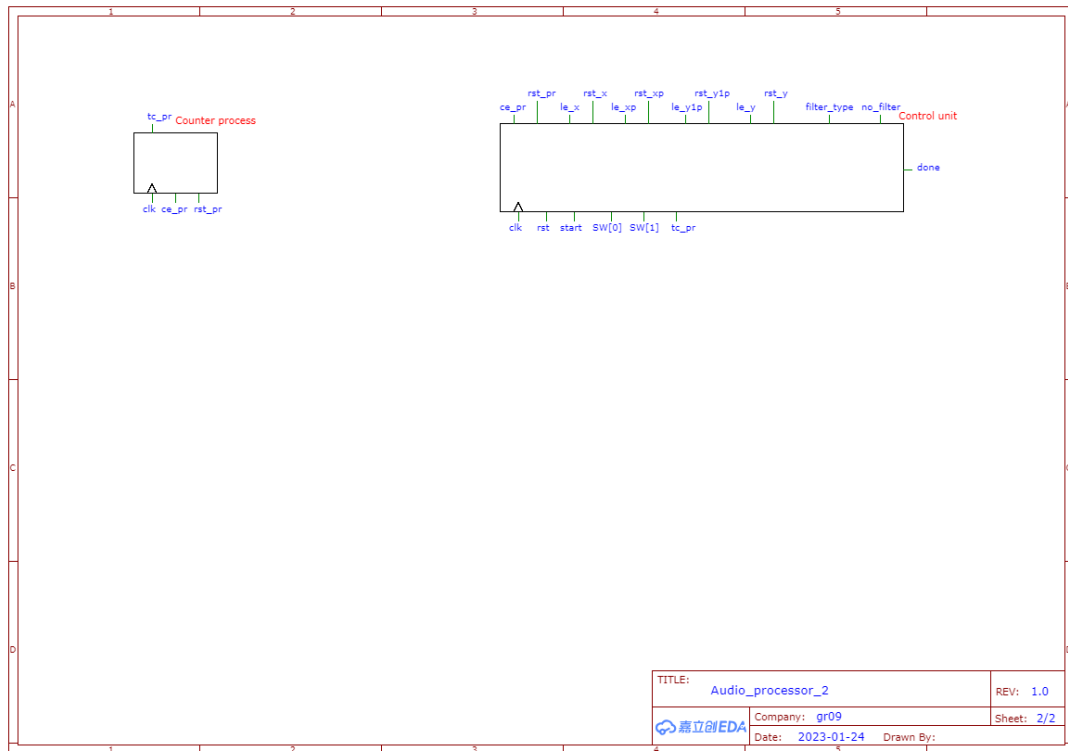


Figure 13: Datapath del blocco di filtraggio

- Y: memorizza il valore d'uscita corrente;
- Yp: memorizza il valore d'uscita precedente.

**Blocchi aritmetici** I componenti aritmetici utilizzati sono i seguenti:

- sommatore saturato
- sommatore/sottrattore controllato dal segnale *filter\_type*;
- moltiplicatore saturato
- blocco complemento a due

È da notare che, essendo i dati rappresentati su 8 bit, per non appesantire troppo l'implementazione del datapath si è scelto di mantenere un parallelismo interno non superiore a 16 bit. I dati vengono riportati su 8 bit mediante troncamento.

Inoltre il problema del possibile overflow delle operazioni di somma è stato risolto utilizzando la tecnica dell'aritmetica saturata.

**Counter process** Viene utilizzato un contatore per attendere il tempo necessario alla propagazione e stabilizzazione del valore corretto dell'uscita; il conteggio vale 1024 colpi di clock, cioè circa 100  $\mu$ s, un tempo sufficiente per l'elaborazione dei dati.

I due segnali di controllo che discriminano il comportamento del datapath sono *no\_filter* e *filter\_type*; a seconda del valore di questi segnali viene selezionata una delle possibili tre tipi di elaborazione:

- filtro shelving cut low pass
- filtro shelving cut high pass
- loopback dell'input

**Control unit** La control unit riceve come segnali di stato *start*, i valori degli switches SW[0] e SW[1] e tc del contatore; i segnali di controllo per il datapath sono ce del contatore, *filter\_type* e *no\_filter* per selezionare il percorso dei dati e i segnali di controllo dei registri.

### 3.5 Timing Diagram

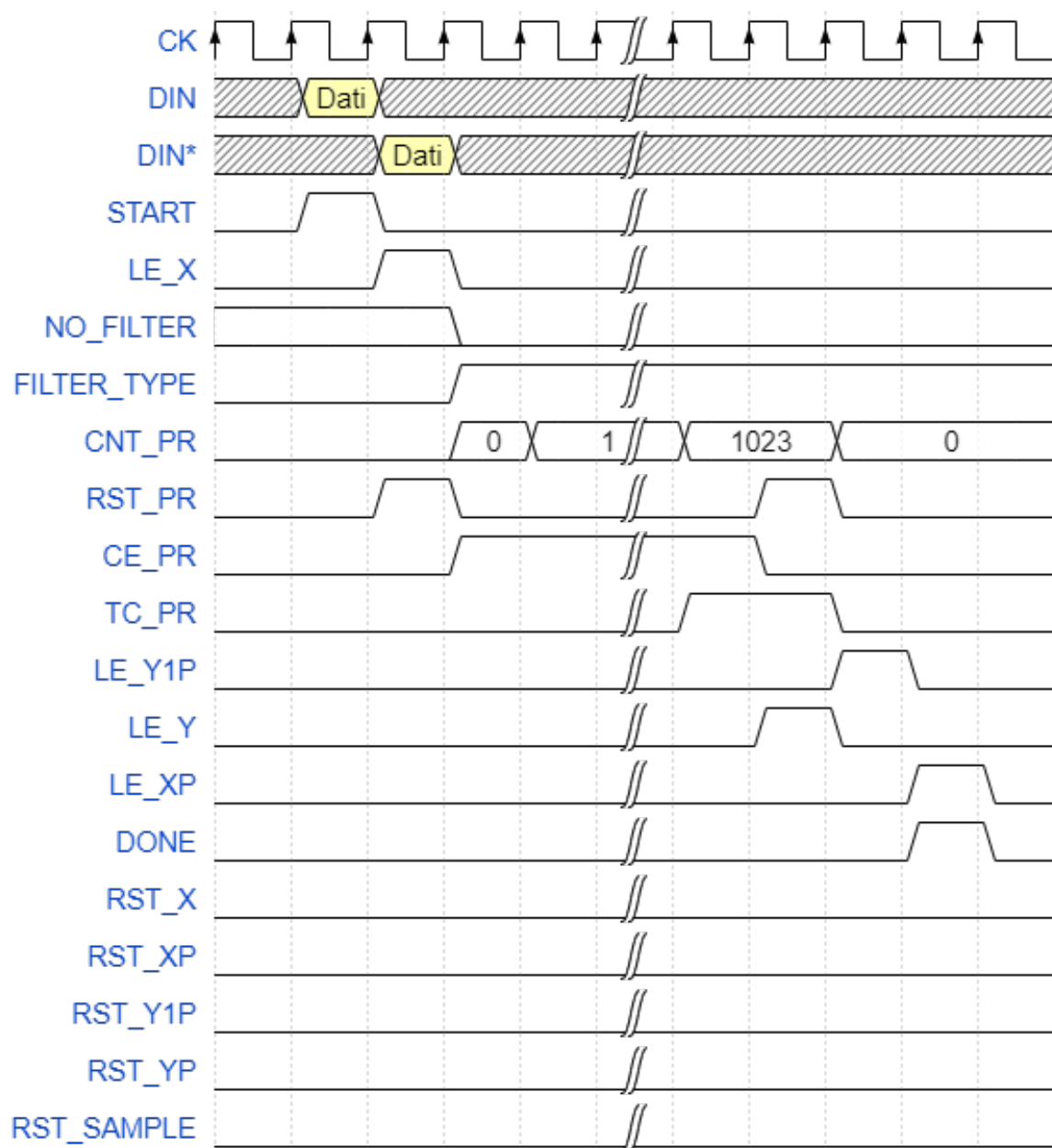
Il timing diagram del blocco di filtraggio è mostrato in figura 14.

### 3.6 Diagramma degli stati

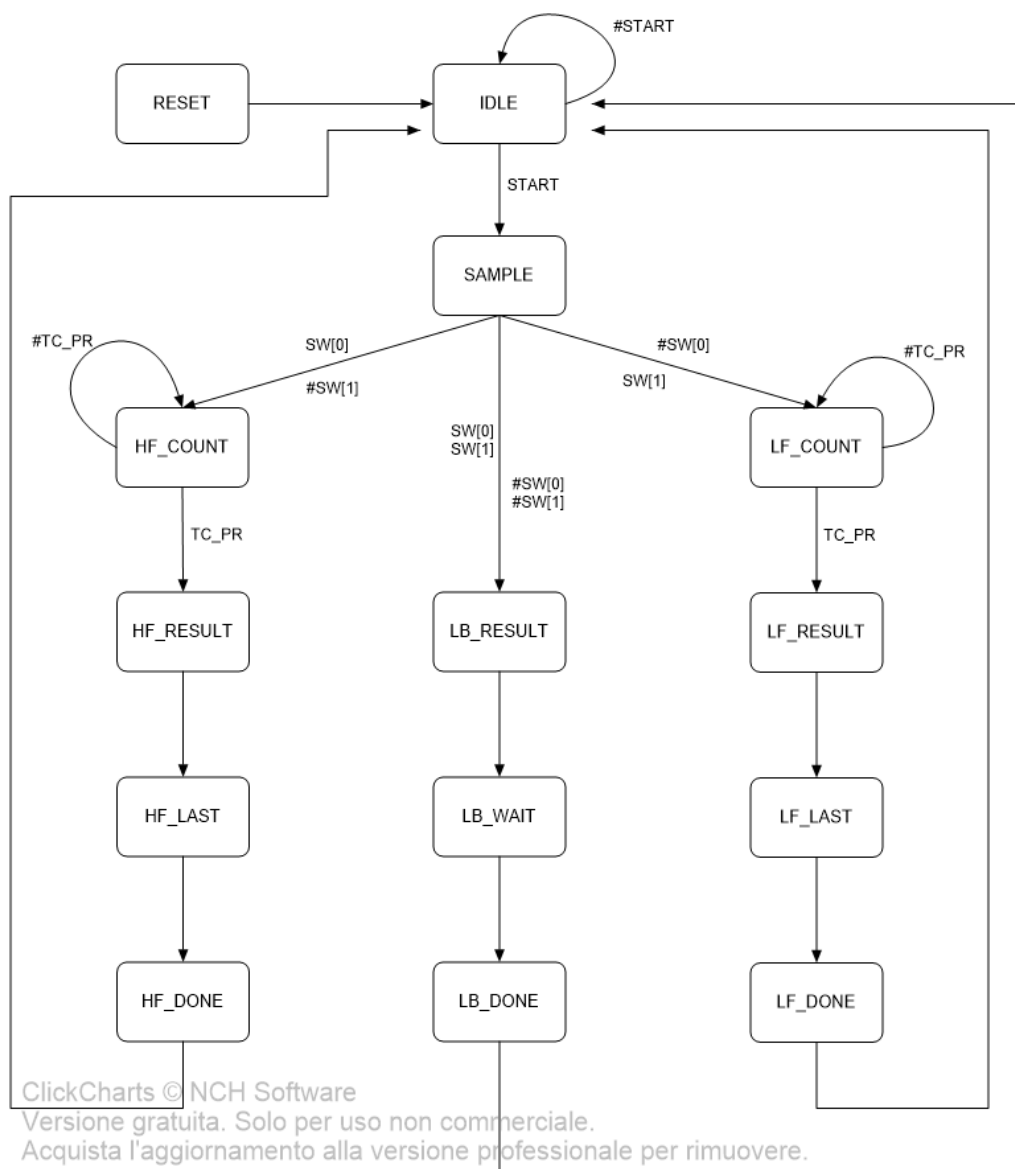
Il diagramma degli stati del filtro è mostrato in figura 15 e la tabella con i segnali associati a ogni stato è mostrata in figura 16.

### 3.7 Risultati

L'elaborazione di uno stream di byte mediante i due filtri viene verificata confrontandola con i medesimi algoritmi eseguiti da uno script Python.



**Figure 14:** Timing diagram del blocco di filtraggio



**Figure 15:** Diagramma degli stati del blocco di filtraggio

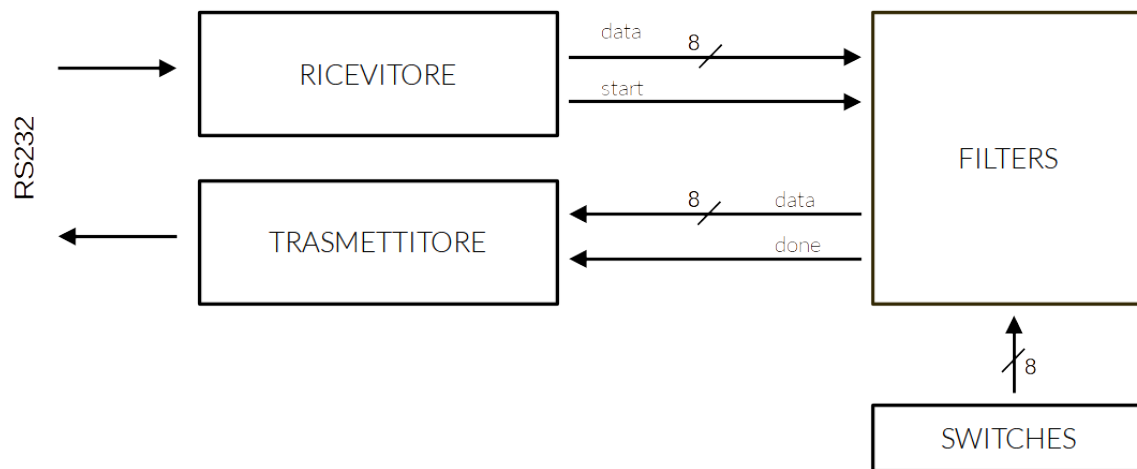
|    | A         | B    | C         | D         | E     | F      | G      | H    | I     | J    | K     | L     | M      | N       | O        |
|----|-----------|------|-----------|-----------|-------|--------|--------|------|-------|------|-------|-------|--------|---------|----------|
| 1  | Stato     | LE_X | NO_FILTER | FILTER_TY | CE_PR | RST_PR | LE_Y1P | LE_Y | LE_XP | DONE | RST_Y | RST_X | RST_XP | RST_Y1P | RST_samp |
| 2  | RESET     | 0    | 1         | 0         | 0     | 1      | 0      | 0    | 0     | 0    | 0     | 1     | 1      | 1       | 1        |
| 3  | IDLE      | 0    | 1         | 0         | 0     | 0      | 0      | 0    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 4  | Sample    | 1    | 1         | 0         | 0     | 1      | 0      | 0    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 5  | LF_Count  | 0    | 0         | 0         | 1     | 0      | 0      | 0    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 6  | LF_result | 0    | 0         | 0         | 0     | 1      | 0      | 1    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 7  | LF_last   | 0    | 0         | 0         | 0     | 0      | 1      | 0    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 8  | LF_done   | 0    | 0         | 0         | 0     | 0      | 0      | 0    | 1     | 1    | 0     | 0     | 0      | 0       | 0        |
| 9  | HF_Count  | 0    | 0         | 1         | 1     | 0      | 0      | 0    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 10 | HF_result | 0    | 0         | 1         | 0     | 1      | 0      | 1    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 11 | HF_last   | 0    | 0         | 1         | 0     | 0      | 1      | 0    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 12 | HF_done   | 0    | 0         | 1         | 0     | 0      | 0      | 0    | 1     | 1    | 0     | 0     | 0      | 0       | 0        |
| 13 | LB_result | 0    | 1         | 0         | 0     | 0      | 0      | 1    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 14 | LB_wait   | 0    | 1         | 0         | 0     | 0      | 0      | 0    | 0     | 0    | 0     | 0     | 0      | 0       | 0        |
| 15 | LB_done   | 0    | 1         | 0         | 0     | 0      | 0      | 0    | 0     | 1    | 0     | 0     | 0      | 0       | 0        |

**Figure 16:** Tabella dei segnali di controllo

## 4 Audio processor

### 4.1 Diagramma a blocchi

Il diagramma a blocchi del progetto Audio processor è mostrato in figura 17.

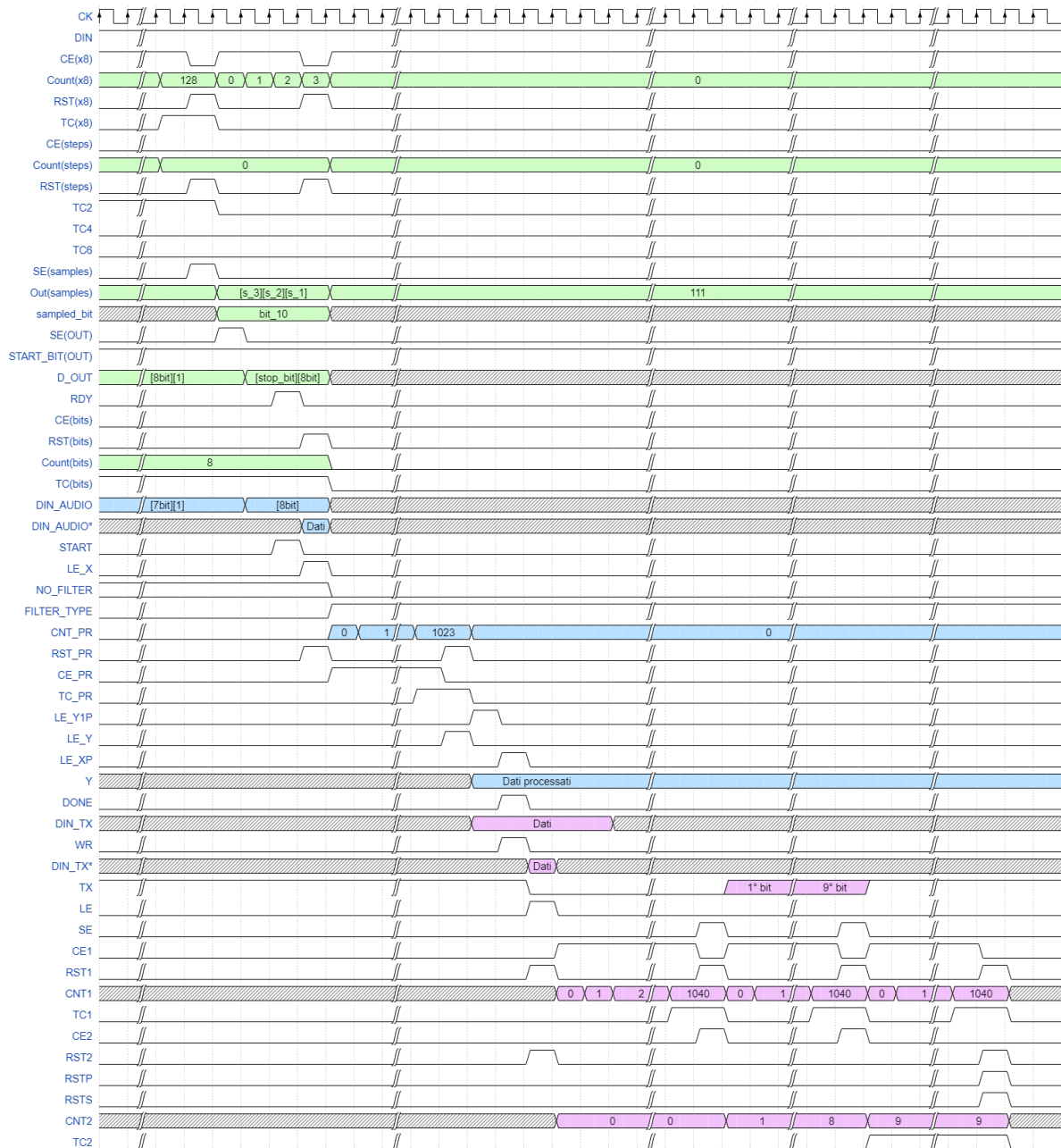


**Figure 17:** Diagramma a blocchi di audio processor

### 4.2 Timing diagram

Il timing diagram dell'audio processor è mostrato in figura 18.

Per l'analisi delle tempistiche dei singoli blocchi si rimanda ai capitoli 1.4 e 2.3.



**Figure 18:** Timing diagram di audio processor



## 5 Simulazioni

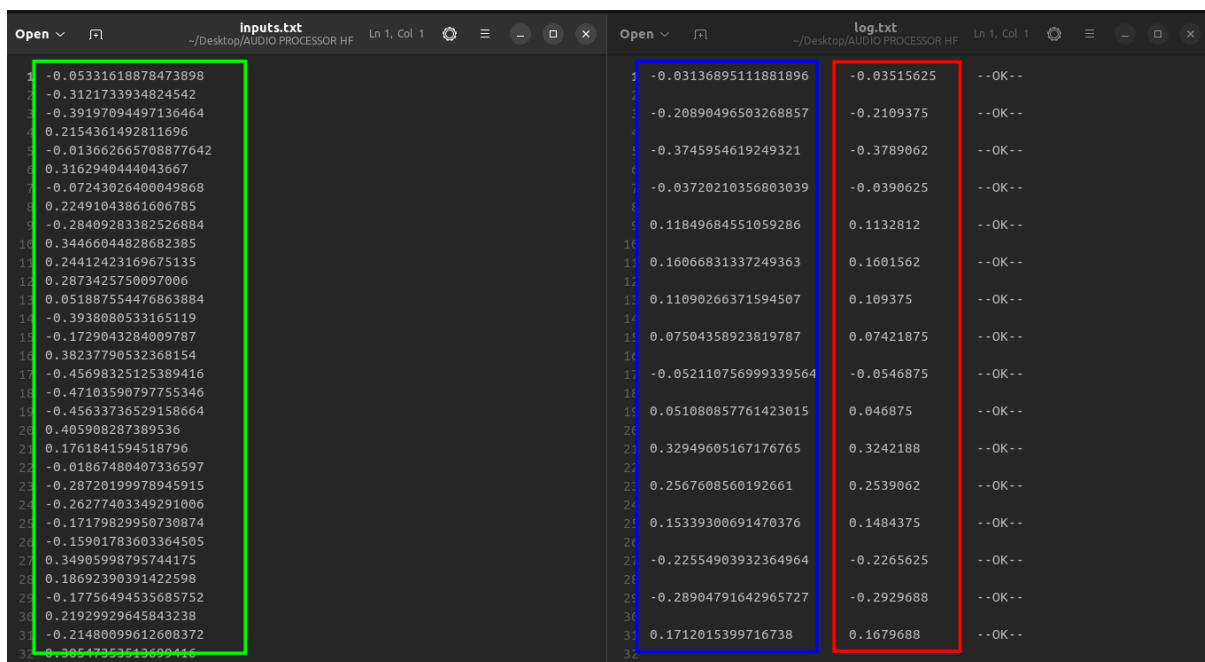
### 5.1 Script Python

Per testare l'architettura sviluppata per implementare i due filtri abbiamo utilizzato due script Python (consultabili in appendice) tra loro identici tranne per gli ovvi e opportuni aggiustamenti necessari a differenziare le due simulazioni. In questi script andiamo come prima cosa a generare un file contenente una serie di input randomici che verranno poi dati contemporaneamente in pasto a:

- un algoritmo scritto in Python per implementare il filtro, col fine di avere un vettore di output corretto con cui confrontare gli output della nostra architettura VHDL;
- l'architettura VHDL da noi sviluppata, lo script Python è infatti in grado di invocare l'esecuzione di ModelSim parametrizzata dal file compile.do (consultabile anch'esso in appendice).

Il file compile.do si occupa di compilare tutti i components necessari alla definizione della nostra architettura e di avviare la simulazione del testbench per il tempo desiderato, oltre che chiudere il programma conclusa la simulazione. Il testbench, d'altro canto, è scritto in maniera tale da "campionare" in maniera opportuna gli input generati dal Python e gli output in uscita dall'entity sotto esame, andandoli ad inserire in un opportuno file .txt.

Lo script si occupa inoltre di generare un file log.txt nel quale è possibile osservare i due output affiancati seguiti da un commento: "OK" oppure "ERROR".



The image shows two side-by-side text editors. The left editor, titled 'inputs.txt', contains a list of 32 numerical values, each on a new line, ranging from approximately -0.05 to 0.30. The right editor, titled 'log.txt', contains a table with three columns. The first column has 32 rows of numerical values, the second column has 32 rows of numerical values, and the third column contains the string '--OK--' repeated 32 times. The first column of the log file is highlighted with a blue border, and the second column is highlighted with a red border.

| inputs.txt            | log.txt (Col 1)       | log.txt (Col 2) | log.txt (Col 3) |
|-----------------------|-----------------------|-----------------|-----------------|
| -0.05331618878473898  | -0.03136895111881896  | -0.03515625     | --OK--          |
| -0.3121733934824542   | -0.20890496503268857  | -0.2109375      | --OK--          |
| -0.3919709497136464   | -0.3745954619249321   | -0.3789062      | --OK--          |
| 0.2154361492811696    | -0.03720210356803039  | -0.0390625      | --OK--          |
| -0.013662665708877642 | 0.11849684551059286   | 0.1132812       | --OK--          |
| 0.3162940444043667    | 0.16066831337249363   | 0.1601562       | --OK--          |
| -0.07243026400049868  | 0.11090266371594507   | 0.109375        | --OK--          |
| 0.22491043861606785   | 0.07504358923819787   | 0.07421875      | --OK--          |
| -0.28409283382526884  | -0.052110756999339564 | -0.0546875      | --OK--          |
| 0.34466044828682385   | 0.051080857761423015  | 0.046875        | --OK--          |
| 0.24412423169675135   | 0.32949605167176765   | 0.3242188       | --OK--          |
| 0.2873425750097006    | 0.2567608560192661    | 0.2539062       | --OK--          |
| 0.051887554476863884  | 0.15339300691470376   | 0.1484375       | --OK--          |
| -0.3938080533165119   | -0.2254903932364964   | -0.2265625      | --OK--          |
| -0.1729043284009787   | -0.28904791642965727  | -0.2929688      | --OK--          |
| 0.38237790532368154   | 0.1712015399716738    | 0.1679688       | --OK--          |
| -0.45698325125389416  |                       |                 |                 |
| -0.47103590797755346  |                       |                 |                 |
| -0.45633736529158664  |                       |                 |                 |
| 0.405908287389536     |                       |                 |                 |
| 0.1761841594518796    |                       |                 |                 |
| -0.01867480407336597  |                       |                 |                 |
| -0.28720199978945915  |                       |                 |                 |
| -0.26277403349291006  |                       |                 |                 |
| -0.17179829950730874  |                       |                 |                 |
| -0.15901783603364505  |                       |                 |                 |
| 0.34905998795744175   |                       |                 |                 |
| 0.18692390391422598   |                       |                 |                 |
| -0.17756494535685752  |                       |                 |                 |
| 0.21929929645843238   |                       |                 |                 |
| -0.21480099612608372  |                       |                 |                 |
| 0.30647365613699416   |                       |                 |                 |

Figure 19: Filtro high pass

Andando infatti a paragonare i due vettori di output ottenuti, col fine di calcolare l'errore commesso (in particolare come valore assoluto della differenza), lo script segnalerà con "ERROR" ogni coppia di output che genera un errore superiore ad una soglia impostata (in questo

```

inputs.txt
~/Desktop/AUDIO PROCESSOR LF Ln 1, Col 1
1 -0.25085426056815074
2 -0.22745727584021358
3 0.4574215422475191
4 0.3182361033196528
5 0.017938415856873502
6 0.05599918966673678
7 0.2735059756029372
8 -0.1091772341263173
9 0.14000577886168097
10 0.014144873641845246
11 0.13423408678094217
12 -0.06386257549200014
13 0.2143709343353386
14 -0.25256733680946164
15 0.11458779583846301
16 -0.43462261937204116
17 0.22147676618708523
18 -0.4164843239397692
19 0.44165590198549254
20 -0.10055657666651774
21 0.428293142430079
22 0.04652680896685035
23 0.1205218472435774
24 -0.33663411426306644
25 0.1683456925140926
26 -0.07814101912727434
27 0.32308028034386994
28 0.4303729772782413
29 0.015964948197702666
30 -0.07576136064190997
31 -0.22764220093447796
32 -0.0013972057396063957

log.txt
~/Desktop/AUDIO PROCESSOR LF Ln 3, Col 41
1 -0.11782351128010776 -0.1171875 --OK--
2 0.0024547254967842436 0.0 --OK--
3 0.2944368950678234 0.2929688 --OK--
4 -0.0640159491081756 -0.05078125 --ERRORE--
5 -0.09211858061514137 -0.09375 --OK--
6 0.036430997361662276 0.03515625 --OK--
7 0.1022581033628445 0.1015625 --OK--
8 -0.1657601769716086 -0.1640625 --OK--
9 0.13375534814905 0.1328125 --OK--
10 -0.06648739192148824 -0.06640625 --OK--
11 0.06993743973681757 0.06640625 --OK--
12 -0.0897064179362426 -0.08984375 --OK--
13 0.13916702698054387 0.140625 --OK--
14 -0.21888982667403234 -0.2226562 --OK--
15 0.18174947021220195 0.1796875 --OK--
16 -0.27689091814252775 -0.2773438 --OK--

```

**Figure 20:** Filtro low pass

caso  $2 \text{ LSB} = 2^{-7}$ ); il numero di errori totali riscontrati verrà inoltre utilizzato per calcolare la percentuale di errore sul totale degli output analizzati.

Le schermate dei terminali sono mostrate nelle figure 21, 22, 23 e 24.

```
paolo@paolo-Aspire-5742G: ~/Desktop/AUDIO PROCESSOR HF
paolo@paolo-Aspire-5742G:~/Desktop/AUDIO PROCESSOR HF$ python3 TestAudioProcessor\ _HF.py
Starting simulation...
Reading pref.tcl

# 2020.1

# do compile.do
# ** Warning: (vlib-34) Library already exists at "work".
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:21:59 on Feb 11,2023
# vcom Counter.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Compiling entity Counter
# -- Compiling architecture behavioral of Counter
# End time: 18:21:59 on Feb 11,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:21:59 on Feb 11,2023
# vcom saturated_adder.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Compiling entity saturated_adder
# -- Compiling architecture saturated_adder_arch of saturated_adder
# End time: 18:21:59 on Feb 11,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:21:59 on Feb 11,2023
# vcom saturated_multiplier.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Loading package instance fixed_pkg
```

**Figure 21:** Schermata del terminale per il test

```
paolo@paolo-Aspire-5742G: ~/Desktop/AUDIO PROCESSOR HF
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:22:00 on Feb 11,2023
# vcom tb_audio_processor.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Loading package instance fixed_pkg
# -- Loading package fixed_float_types
# -- Loading package fixed_generic_pkg
# -- Loading package MATH_REAL
# -- Loading package body fixed_generic_pkg
# -- Loading package std_logic_textio
# -- Compiling entity tb_audio_processor
# -- Compiling architecture behavioral of tb_audio_processor
# End time: 18:22:00 on Feb 11,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -c work.tb_audio_processor
# Start time: 18:22:00 on Feb 11,2023
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading ieee.numeric_std(body)
# Loading ieee.fixed_float_types
# Loading ieee.math_real(body)
# Loading ieee.fixed_generic_pkg(body)
# Loading ieee.fixed_pkg
# Loading ieee.std_logic_textio(body)
# Loading work.tb_audio_processor(behavioral)
# Loading work.audio_processor(behavioral)
# Loading work.counter(behavioral)
# Loading work.saturated_multiplier(saturated_multiplier_arch)
# Loading work.saturated_adder(saturated_adder_arch)
# End time: 18:22:02 on Feb 11,2023, Elapsed time: 0:00:02
# Errors: 0, Warnings: 0
Simulation completed
Percentuale output con errore > 2LSB = 6.281 %
paolo@paolo-Aspire-5742G:~/Desktop/AUDIO PROCESSOR HF$
```

**Figure 22:** Schermata del terminale per il test

```
paolo@paolo-Aspire-5742G: ~/Desktop/AUDIO PROCESSOR LF
paolo@paolo-Aspire-5742G:~/Desktop/AUDIO PROCESSOR LF$ python3 TestAudioProcessor_LF.py
Starting simulation...
Reading pref.tcl

# 2020.1

# do compile.do
# ** Warning: (vlib-34) Library already exists at "work".
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:08:10 on Feb 11,2023
# vcom Counter.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Compiling entity Counter
# -- Compiling architecture behavioral of Counter
# End time: 18:08:10 on Feb 11,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:08:10 on Feb 11,2023
# vcom saturated_adder.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Compiling entity saturated_adder
# -- Compiling architecture saturated_adder_arch of saturated_adder
# End time: 18:08:10 on Feb 11,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:08:10 on Feb 11,2023
# vcom saturated_multiplier.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Loading package instance fixed_pkg
```

Figure 23: Schermata del terminale per il test

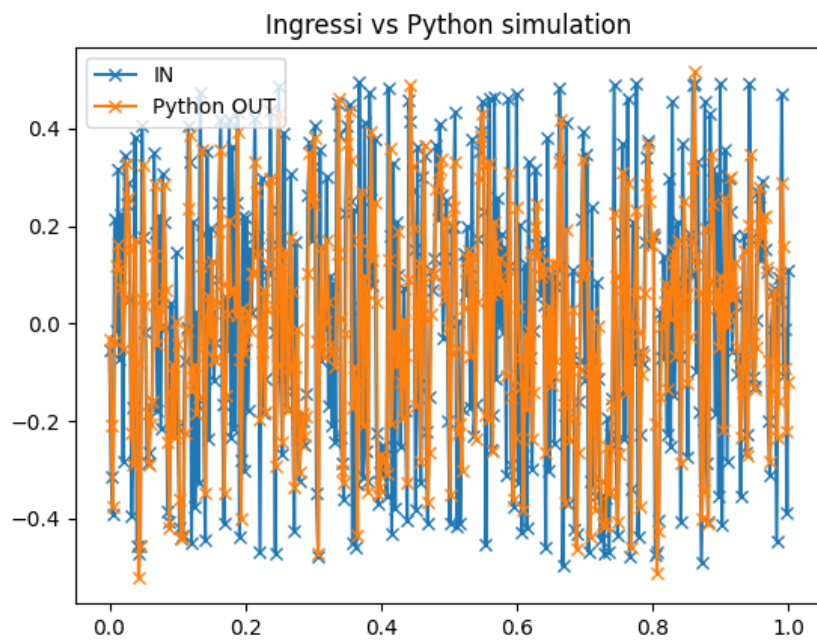
```
paolo@paolo-Aspire-5742G: ~/Desktop/AUDIO PROCESSOR LF
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 18:08:10 on Feb 11,2023
# vcom tb_audio_processor.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Loading package instance fixed_pkg
# -- Loading package fixed_float_types
# -- Loading package fixed_generic_pkg
# -- Loading package MATH_REAL
# -- Loading package body fixed_generic_pkg
# -- Loading package std_logic_textio
# -- Compiling entity tb_audio_processor
# -- Compiling architecture behavioral of tb_audio_processor
# End time: 18:08:10 on Feb 11,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# vsim -c work.tb_audio_processor
# Start time: 18:08:10 on Feb 11,2023
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading ieee.numeric_std(body)
# Loading ieee.fixed_float_types
# Loading ieee.math_real(body)
# Loading ieee.fixed_generic_pkg(body)
# Loading ieee.fixed_pkg
# Loading ieee.std_logic_textio(body)
# Loading work.tb_audio_processor(behavioral)
# Loading work.audio_processor(behavioral)
# Loading work.counter(behavioral)
# Loading work.saturated_multiplier(saturated_multiplier_arch)
# Loading work.saturated_adder(saturated_adder_arch)
# End time: 18:08:12 on Feb 11,2023, Elapsed time: 0:00:02
# Errors: 0, Warnings: 0
Simulation completed
Percentuale output con errore > 2LSB = 2.764 %
paolo@paolo-Aspire-5742G:~/Desktop/AUDIO PROCESSOR LF$
```

Figure 24: Schermata del terminale per il test

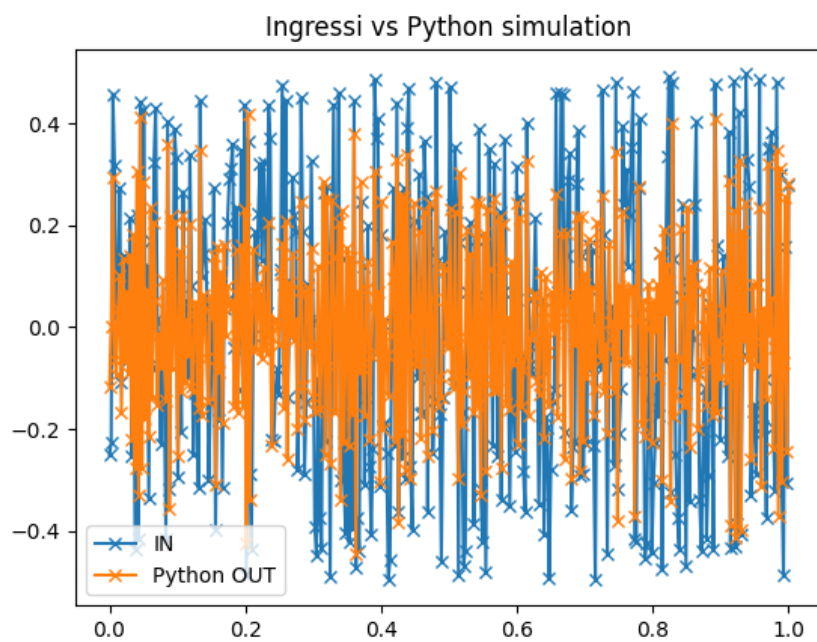
## 5.2 Plot dei risultati

Infine, con l'ausilio della libreria matplotlib di Python, andiamo a graficare i risultati ottenuti. In particolare avremo:

Confronto tra ingressi e output ottenuti da Python mostrato in figura 25 e 26.

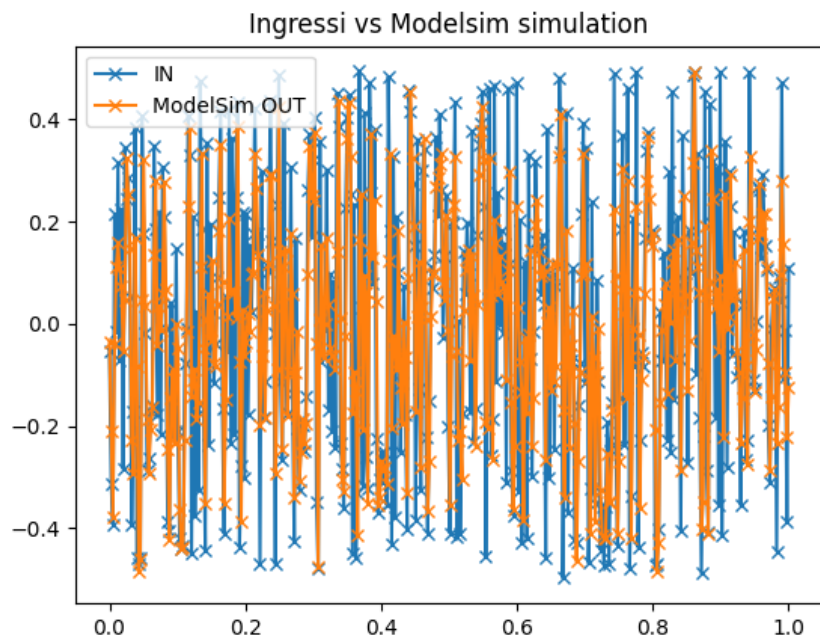


**Figure 25:** Confronto input-output su Python di high pass filter

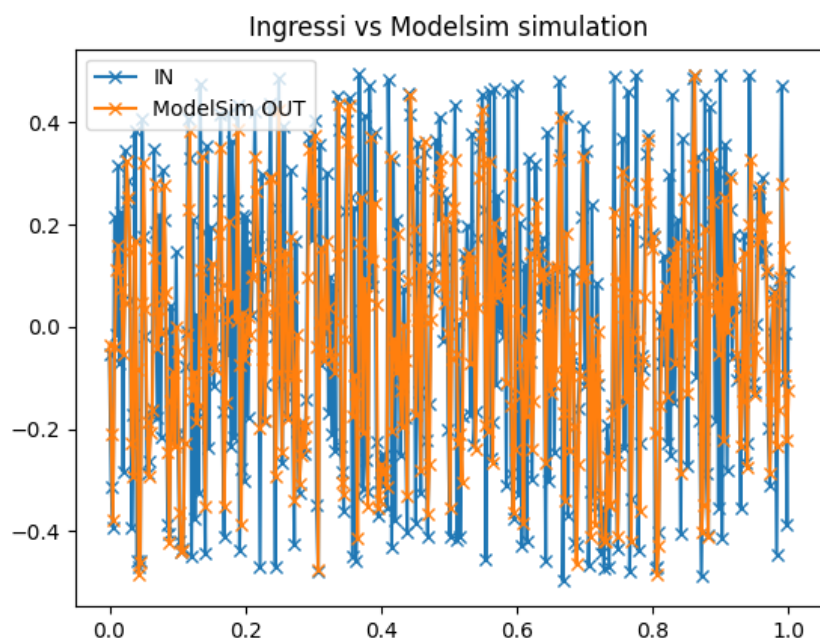


**Figure 26:** Confronto di input-output su Python di low pass filter

Confronto tra ingressi e output ottenuti da simulazione ModelSim mostrato in figure 28 e 28.



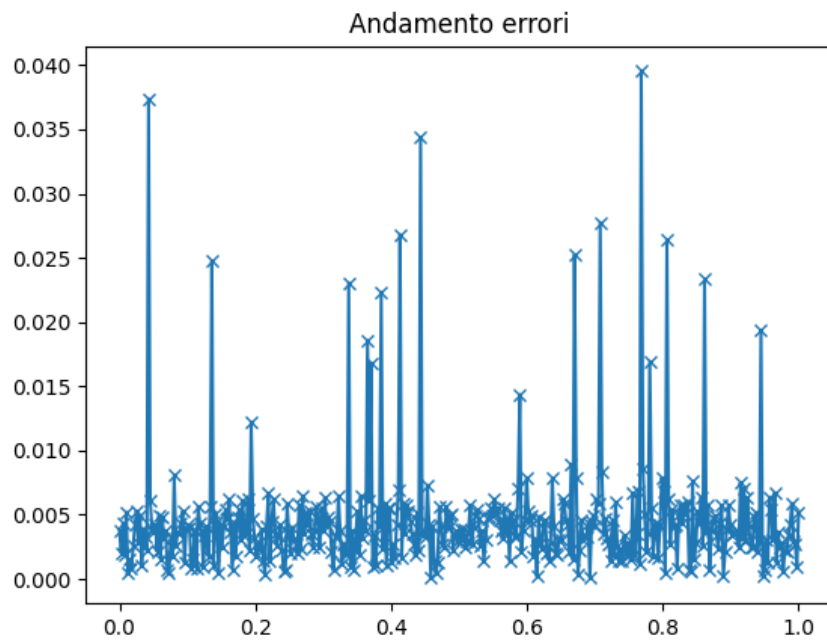
**Figure 27:** Confronto input-output su ModelSim di high pass filter



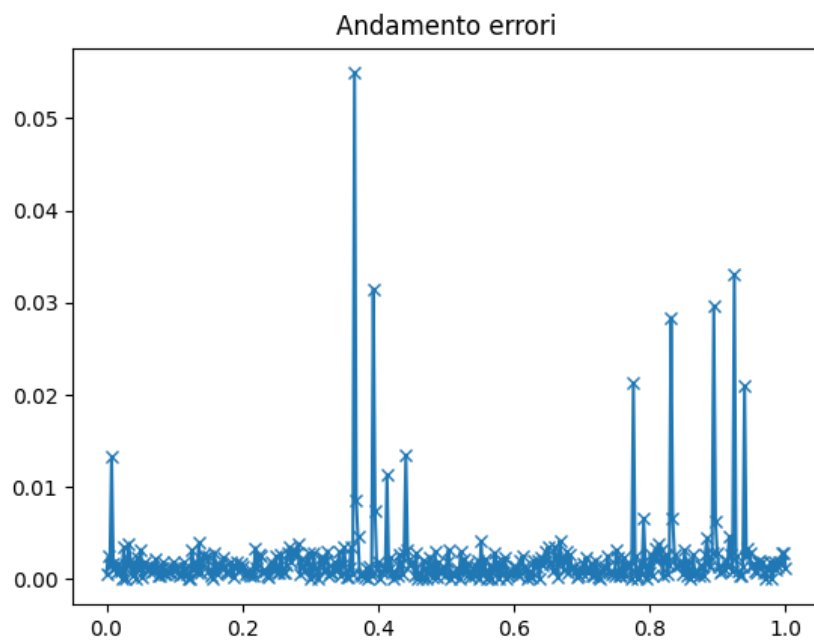
**Figure 28:** Confronto di input-output su ModelSim di low pass filter

Andamento degli errori, mostrato nelle figure 29 e 30.

Notiamo, come già anticipato dalle percentuali di errore totale, che per HF compiamo mediamente più errori.



**Figure 29:** Andamento errore di high pass filter



**Figure 30:** Andamento errore di low pass filter



## 6 Appendice

### 6.1 Trasmittitore

#### 6.1.1 Counter

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

entity Counter is

    generic
    (
        preset : integer := 255
    );

    port
    (
        clk      : in std_logic;
        async_rst : in std_logic;
        sync_rst  : in std_logic;
        ce       : in std_logic;
        tc       : out std_logic
    );

end entity;

architecture behavioral of Counter is
begin

    process (clk, async_rst)
        variable cnt : integer range 0 to preset;
    begin
        if async_rst='1' then
            cnt := 0;
        elsif (clk'event and clk='1') then
            if sync_rst = '1' then
                cnt := 0;
            elsif ce = '1' then
                if (cnt < preset) then
                    cnt := cnt + 1;
                end if;
            end if;
        end if;

        if (cnt < preset) then
            tc <= '0';
        elsif (cnt = preset) then
            tc <= '1';
        end if;

    end process;
end behavioral;
```



### 6.1.2 Parallel Register

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

entity Parallel_register is
generic(
    N: integer := 8
);
port(
    clk,rst: IN STD_LOGIC;
    parallel_in : IN STD_LOGIC_VECTOR(N-1 downto 0);
    parallel_out : OUT STD_LOGIC_VECTOR(N-1 downto 0)
);
end Parallel_register;

architecture behavioral of Parallel_register is
begin
    process(clk,rst)
    begin
        if(rst='1') then
            parallel_out<=(others=>'1');
        elsif(clk'event and clk='1') then
            parallel_out<=parallel_in;
        end if;
    end process;
end behavioral;
```

### 6.1.3 Shift Register

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

entity Shift_register is
generic(
    N: integer := 10
);
port(
    clk, le, se, rst: IN STD_LOGIC;
    data_in_parallel: IN std_logic_vector(N-1 downto 0);
    data_in_serial: IN std_logic;
    data_out_parallel: buffer std_logic_vector(N-1 downto 0)
);
end Shift_register;

architecture behavioral of Shift_register is
begin
    process(clk, rst)
    begin
        if(rst='1') then
            data_out_parallel<=(others=>'1');
        elsif(clk'event and clk='1') then
            if(le='1') then
                data_out_parallel<=data_in_parallel;
            elsif(se='1') then
                data_out_parallel((N-1) downto 1) <= data_out_parallel((N-2) downto 0);
                data_out_parallel(0)<=data_in_serial;
            else
                data_out_parallel<=data_out_parallel;
            end if;
        end if;
    end process;
end behavioral;
```

### 6.1.4 Serial TX

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

entity Serial_TX is
  port (
    clk : IN std_logic;
    rst : IN std_logic;
    WR: IN std_logic;
    DIN: IN std_logic_vector(7 downto 0);
    TX: OUT std_logic
  );
end Serial_TX;

architecture behavioral of Serial_TX is

  component Counter is
    generic(
      preset : integer := 255
    );
    port
    (
      clk      : in std_logic;
      rst      : in std_logic;
      ce       : in std_logic;
      tc       : out std_logic
    );
  end component;

  component Parallel_register is
    generic(
      N: integer := 8
    );
    port (
      clk,rst: IN STD_LOGIC;
      parallel_in : IN STD_LOGIC_VECTOR(N-1 downto 0);
      parallel_out : OUT STD_LOGIC_VECTOR(N-1 downto 0)
    );
  end component;

  component Shift_register is
    generic(
      N: integer := 10
    );
    port (
      clk,le,se,rst: IN STD_LOGIC;
      data_in_parallel: IN std_logic_vector(N-1 downto 0);
      data_in_serial: IN std_logic;
      data_out_serial: out std_logic
    );
  end component;
```

```

type state_type is (idle,load,bitTX,shift,reset);

signal current_state, next_state: state_type;
signal DIN_star: std_logic_vector(7 downto 0);
signal SE,LE: std_logic;
signal RST1,CE1: std_logic;
signal RST2,CE2: std_logic;
signal TC1,TC2: std_logic;
signal RSTP,RSTS: std_logic;

begin

Counter1: Counter
GENERIC MAP (preset=>1040)
PORT MAP (clk=>clk, rst=>RST1, ce=>CE1,tc=>TC1);

Counter2: Counter
GENERIC MAP (preset=>9)
PORT MAP (clk=>clk, rst=>RST2, ce=>CE2,tc=>TC2);

DelayRegister: Parallel_register
GENERIC MAP (N=>8)
PORT MAP (clk=>clk, rst=>RSTP, parallel_in=>DIN, parallel_out=>DIN_star);

ShiftRegister: Shift_register
GENERIC MAP (N=>10)
PORT MAP (clk=>clk, rst=>RSTS, le=>LE, se=>SE,data_in_parallel(0)=>'0',
          data_in_parallel(8 downto 1)=>
          DIN_star,
          data_in_parallel(9)=>'1',data_in_serial=>'1',data_out_serial=>TX);

cambio_stato: process(clk)
begin
    if(clk'event and clk='1') then
        case current_state is
            when idle => if (WR='1') then next_state<=load; end if;
            when load => next_state<=bitTX;
            when bitTX => if (TC1='1' and TC2='0') then next_state<=shift;
                          elsif (TC1='1' and TC2='1') then next_state<=reset;
                          end if;
            when shift=> next_state<=bitTX;
            when reset => next_state<=idle;
            when others => next_state<=reset;
        end case;
    end if;
end process;

aggiornamento_stato: process(next_state,rst)
begin
    if(rst='0') then
        current_state<=reset;
    else
        current_state<=next_state;
    end if;
end process;

```

```

end process;

cambio_output:process(current_state)
begin
    case current_state is
        when idle =>    LE<='0';SE<='0';CE1<='0';RST1<='0';CE2<='0';RST2<='0';
                        RSTP<='0';RSTS<='0';
        when load =>    LE<='1';SE<='0';CE1<='0';RST1<='1';CE2<='0';RST2<='1';
                        RSTP<='0';RSTS<='0';
        when bitTX =>    LE<='0';SE<='0';CE1<='1';RST1<='0';CE2<='0';RST2<='0';
                        RSTP<='0';RSTS<='0';
        when shift=>    LE<='0';SE<='1';CE1<='0';RST1<='1';CE2<='1';RST2<='0';
                        RSTP<='0';RSTS<='0';
        when reset =>    LE<='0';SE<='0';CE1<='0';RST1<='1';CE2<='0';RST2<='1'
                        ;RSTP<='1';RSTS<='1';

    end case;
end process;

end architecture;

```

## 6.2 Ricevitore

### 6.2.1 Counter

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

entity Counter is

    generic(
        preset : integer := 255
    );
    port(
        clk      : in std_logic;
        rst      : in std_logic;
        ce       : in std_logic;
        tc       : out std_logic
    );
end entity;

architecture behavioral of Counter is
begin

    process (clk)
        variable cnt      : integer range 0 to preset;
    begin
        if(clk'event and clk='1') then
            if rst = '1' then
                cnt := 0;
            elsif ce = '1' then
                if(cnt<preset) then
                    cnt := cnt + 1;
                end if;
            end if;
        end if;

        if(cnt<preset) then
            tc <= '0';
        elsif(cnt=preset) then
            tc <= '1';
        end if;

    end process;
end behavioral;
```

## 6.2.2 Counter 3 bit

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Counter_3bits is
port (
    clk    : in std_logic;
    rst    : in std_logic;
    ce     : in std_logic;
    q      : out std_logic_vector(2 downto 0)
);
end Counter_3bits ;

architecture behavioral of Counter_3bits is
begin

process(clk)
variable q_int : std_logic_vector(2 downto 0);
begin
    if(rising_edge(clk))then
        if rst = '1' then
            q_int := (others => '0');
        elsif ce = '1' and q_int < 7 then
            q_int := q_int+1;
        end if;
        q<=q_int;
    end if;
end process;

end behavioral;
```

### 6.2.3 Parallel Register

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

entity Parallel_register is
generic(
    N: integer := 8
);
port(
    clk,rst: IN STD_LOGIC;
    parallel_in : IN STD_LOGIC_VECTOR(N-1 downto 0);
    parallel_out : OUT STD_LOGIC_VECTOR(N-1 downto 0)
);
end Parallel_register;

architecture behavioral of Parallel_register is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            if (rst='1') then
                parallel_out<=(others=>'1');
            else
                parallel_out<=parallel_in;
            end if;
        end if;
    end process;
end behavioral;
```



## 6.2.4 Shift register

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

entity Shift_register is
generic(
    N: integer := 10
);
port(
    clk, le, se, rst: IN STD_LOGIC;
    data_in_parallel: IN std_logic_vector(N-1 downto 0);
    data_in_serial: IN std_logic;
    data_out_serial: out std_logic
);
end Shift_register;

architecture behavioral of Shift_register is
    signal data_out_parallel: std_logic_vector(N-1 downto 0);
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            if(rst='1') then
                data_out_parallel<=(others=>'1');
                data_out_serial<='1';
            elsif(le='1') then
                data_out_parallel<=data_in_parallel;
                data_out_serial<=data_in_parallel(0);
            elsif(se='1') then
                data_out_parallel((N-2) downto 0)<=data_out_parallel((N-1) downto 1);
                data_out_parallel(N-1)<=data_in_serial;
                data_out_serial<=data_out_parallel(1);
            else
                data_out_parallel<=data_out_parallel;
                data_out_serial<=data_out_parallel(0);
            end if;
        end if;
    end process;
end behavioral;
```

### 6.2.5 SIPO Register

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

entity SIPORegister is
generic(
    N: integer := 8
);
port(
    clk, se, rst: IN STD_LOGIC;
    data_in_serial: IN std_logic;
    data_out_parallel: buffer std_logic_vector(N-1 downto 0)
);
end SIPORegister;

architecture behavioral of SIPORegister is
begin
    process(clk)
    begin
        if(clk'event and clk='1') then
            if(rst='1') then
                data_out_parallel<=(others=>'1');
            elsif(se='1') then
                data_out_parallel((N-1) downto 1)<=data_out_parallel((N-2) downto 0);
                data_out_parallel(0)<=data_in_serial;
            else
                data_out_parallel<=data_out_parallel;
            end if;
        end if;
    end process;
end behavioral;
```

## 6.2.6 Serial RX

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

entity Serial_RX is
  port (
    clk : IN std_logic;
    rst : IN std_logic;
    DIN: IN std_logic;
    DOUT: OUT std_logic_vector (7 downto 0);
    RDY: OUT std_logic
  );
end Serial_RX;

architecture behavioral of Serial_RX is

  component Counter is
    generic(
      preset : integer := 255
    );
    port
    (
      clk      : in std_logic;
      rst      : in std_logic;
      ce       : in std_logic;
      tc       : out std_logic
    );
  end component;

  component SIPORegister is
    generic(
      N: integer := 8
    );
    port (
      clk,se,rst: IN STD_LOGIC;
      data_in_serial: IN std_logic;
      data_out_parallel: buffer std_logic_vector(N-1 downto 0)
    );
  end component;

  component Counter_3bits is
    port (
      clk      : in std_logic;
      rst      : in std_logic;
      ce       : in std_logic;
      q        : out std_logic_vector(2 downto 0)
    );
  end component;

  type state_type is (detec_1,detec_2,skip7_1,skip7_2,skip7_3,sample_1,
                      sample_2,sample_3,save_bit,skip5_1,
                      skip5_2,skip5_3,save_stop_bit,
                      control_stop_bit,data_ready,reset);
```

```

signal current_state, next_state: state_type;

signal data_out_detector: std_logic_vector(7 downto 0);
signal data_out_samples: std_logic_vector(2 downto 0);
signal sampled_bit: std_logic;
signal stop_bit: std_logic;
signal RST_Detector, SE_Detector: std_logic;
signal RST_Samples, SE_Samples: std_logic;
signal RST_Out, SE_Out: std_logic;
signal START_DETECTED: std_logic;
signal RST_x8, CE_x8, RST_Steps, CE_Steps, RST_Bits, CE_Bits: std_logic;
signal TC_steps: std_logic_vector(2 downto 0);
signal TC_x8, TC2_steps, TC4_steps, TC6_steps, TC_bits: std_logic;

begin

Counter_x8: Counter
  GENERIC MAP (preset=>128)
  PORT MAP (clk=>clk, rst=>RST_x8, ce=>CE_x8, tc=>TC_x8);

Counter_Steps: Counter_3bits
  PORT MAP (clk=>clk, rst=>RST_Steps, ce=>CE_Steps, q=>TC_Steps);

Counter_Bits: Counter
  GENERIC MAP (preset=>8)
  PORT MAP (clk=>clk, rst=>RST_Bits, ce=>CE_Bits, tc=>TC_Bits);

Detector: SIPORegister
  GENERIC MAP (N=>8)
  PORT MAP (clk=>clk, rst=>RST_Detector, se=>SE_Detector, data_in_serial=>
    DIN, data_out_parallel=>
    data_out_detector);

Samples: SIPORegister
  GENERIC MAP (N=>3)
  PORT MAP (clk=>clk, rst=>RST_Samples, se=>SE_Samples, data_in_serial=>DIN
    , data_out_parallel=>data_out_samples
    );

RX_OUT: SIPORegister
  GENERIC MAP (N=>9)
  PORT MAP (clk=>clk, rst=>RST_Out, se=>SE_Out, data_in_serial=>sampled_bit

    data_out_parallel(8)> DOUT(0), data_out_parallel(7)> DOUT(1),
    data_out_parallel(6)> DOUT(2),
    data_out_parallel(5)> DOUT(3), data_out_parallel(4)> DOUT(4),
    data_out_parallel(3)> DOUT(5),
    data_out_parallel(2)> DOUT(6), data_out_parallel(1)> DOUT(7),
    data_out_parallel(0) => stop_bit);

sampled_bit<= (data_out_samples(0) and data_out_samples(1)) or (
    data_out_samples(0) and
    data_out_samples(2)) or (
    data_out_samples(1) and
    data_out_samples(2));

```

```

START_DETECTED<= data_out_detector(7) and data_out_detector(6) and
                  data_out_detector(5) and
                  data_out_detector(4) and (not(
                  data_out_detector(3))) and (not(
                  data_out_detector(2))) and (not(
                  data_out_detector(1))) and (not(
                  data_out_detector(0)));

TC2_steps<=TC_steps(1);
TC4_steps<=TC_steps(2);
TC6_steps<=TC_steps(1) and TC_steps(2);

cambio_stato: process(clk)
begin
    if(clk'event and clk='1') then
        case current_state is
            when detec_1 => if (TC_x8='1' and START_DETECTED='0' ) then
                            next_state<=detec_2;
                            elsif(START_DETECTED='1') then next_state<=skip7_1; end if;
            when detec_2 => next_state<=detec_1;

            when skip7_1 => if (TC_x8='1' and TC6_steps='0') then next_state<=
                            skip7_2;
                            elsif(TC_x8='1' and TC6_steps='1') then next_state<=
                            skip7_3; end if;
            when skip7_2 => next_state<=skip7_1;
            when skip7_3 => next_state<=sample_1;

            when sample_1 => if (TC_x8='1' and TC2_steps='0' ) then next_state<=
                            sample_2;
                            elsif(TC_x8='1' and TC2_steps='1') then next_state<=
                            sample_3; end if;
            when sample_2 => next_state<=sample_1;
            when sample_3 => if (TC_bits='1') then next_state<=save_stop_bit;
                            elsif(TC_bits='0') then next_state<=save_bit; end if;

            when save_bit => next_state<=skip5_1;

            when skip5_1 => if (TC_x8='1' and TC4_steps='0') then next_state<=
                            skip5_2;
                            elsif(TC_x8='1' and TC4_steps='1') then next_state<=
                            skip5_3; end if;
            when skip5_2 => next_state<=skip5_1;
            when skip5_3 => next_state<=sample_1;

            when save_stop_bit => next_state<=control_stop_bit;
            when control_stop_bit => if (stop_bit='1') then next_state<=
                            data_ready;
                            elsif(stop_bit='0') then next_state<=reset; end if;

            when data_ready => next_state<=reset;

            when reset => next_state<=detec_1;
            when others => next_state<=reset;

        end case;
    end if;
end process;

```

```

    end if;
end process;

aggiornamento_stato: process(next_state,rst)
begin
    if(rst='0') then
        current_state<=reset;
    else
        current_state<=next_state;
    end if;
end process;

cambio_output:process(current_state)
begin
    case current_state is
        when detec_1 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='1';RST_x8<='0';
                        CE_Steps<='0';RST_Steps<='0'; CE_Bits
                        <='0';RST_Bits<='0'; RDY<='0';
        when detec_2 => SE_Detector<='1';RST_Detector<='0'; SE_Samples<='0';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='0';RST_x8<='1';
                        CE_Steps<='0';RST_Steps<='0'; CE_Bits
                        <='0';RST_Bits<='0'; RDY<='0';

        when skip7_1 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='1';RST_x8<='0';
                        CE_Steps<='0';RST_Steps<='0'; CE_Bits
                        <='0';RST_Bits<='0'; RDY<='0';
        when skip7_2 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='0';RST_x8<='1';
                        CE_Steps<='1';RST_Steps<='0'; CE_Bits
                        <='0';RST_Bits<='0'; RDY<='0';
        when skip7_3 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='0';RST_x8<='1';
                        CE_Steps<='0';RST_Steps<='1'; CE_Bits
                        <='0';RST_Bits<='0'; RDY<='0';

        when sample_1 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='1';RST_x8<='0';
                        CE_Steps<='0';RST_Steps<='0'; CE_Bits
                        <='0';RST_Bits<='0'; RDY<='0';
        when sample_2 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='1';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='0';RST_x8<='1';
                        CE_Steps<='1';RST_Steps<='0'; CE_Bits
                        <='0';RST_Bits<='0'; RDY<='0';
        when sample_3 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='1';
                        RST_Samples<='0';SE_Out<='0';RST_Out
                        <='0'; CE_x8<='0';RST_x8<='1';

```

```

CE_Steps<='0';RST_Steps<='1'; CE_Bits
<='0';RST_Bits<='0'; RDY<='0';

when save_bit => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
RST_Samples<='0';SE_Out<='1';RST_Out
<='0'; CE_x8<='1';RST_x8<='0';
CE_Steps<='0';RST_Steps<='0'; CE_Bits
<='1';RST_Bits<='0'; RDY<='0';

when skip5_1 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
RST_Samples<='0';SE_Out<='0';RST_Out
<='0'; CE_x8<='1';RST_x8<='0';
CE_Steps<='0';RST_Steps<='0'; CE_Bits
<='0';RST_Bits<='0'; RDY<='0';
when skip5_2 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
RST_Samples<='0';SE_Out<='0';RST_Out
<='0'; CE_x8<='0';RST_x8<='1';
CE_Steps<='1';RST_Steps<='0'; CE_Bits
<='0';RST_Bits<='0'; RDY<='0';
when skip5_3 => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
RST_Samples<='0';SE_Out<='0';RST_Out
<='0'; CE_x8<='0';RST_x8<='1';
CE_Steps<='0';RST_Steps<='1'; CE_Bits
<='0';RST_Bits<='0'; RDY<='0';

when save_stop_bit => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='
0';RST_Samples<='0';SE_Out<='1';
RST_Out<='0'; CE_x8<='1';RST_x8<='0';
CE_Steps<='0';RST_Steps<='0';
CE_Bits<='0';RST_Bits<='0'; RDY<='0';
when control_stop_bit => SE_Detector<='0';RST_Detector<='0'; SE_Samples
<='0';RST_Samples<='0';SE_Out<='0';
RST_Out<='0'; CE_x8<='1';RST_x8<='0';
CE_Steps<='0';RST_Steps<='0';
CE_Bits<='0';RST_Bits<='0'; RDY<='0';

when data_ready => SE_Detector<='0';RST_Detector<='0'; SE_Samples<='0';
RST_Samples<='0';SE_Out<='0';RST_Out
<='0'; CE_x8<='1';RST_x8<='0';
CE_Steps<='0';RST_Steps<='0'; CE_Bits
<='0';RST_Bits<='0'; RDY<='1';

when reset => SE_Detector<='0';RST_Detector<='1'; SE_Samples<='0';
RST_Samples<='1';SE_Out<='0';RST_Out
<='1'; CE_x8<='0';RST_x8<='1';
CE_Steps<='0';RST_Steps<='1';
CE_Bits<='0';RST_Bits<='1'; RDY<='0';

end case;
end process;
end architecture;

```

## 6.3 Audio Processor

### 6.3.1 Counter

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

entity Counter is

    generic(
        preset : integer := 255
    );
    port(
        clk      : in std_logic;
        rst      : in std_logic;
        ce       : in std_logic;
        tc       : out std_logic
    );
end entity;

architecture behavioral of Counter is
begin

    process (clk)
        variable cnt      : integer range 0 to preset;
    begin
        if(clk'event and clk='1') then
            if rst = '1' then
                cnt := 0;
            elsif ce = '1' then
                if(cnt<preset) then
                    cnt := cnt + 1;
                end if;
            end if;
        end if;

        if(cnt<preset) then
            tc <= '0';
        elsif(cnt=preset) then
            tc <= '1';
        end if;

    end process;
end behavioral;
```



### 6.3.2 Saturated Adder

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

entity saturated_adder is
generic (
  n, p, m : integer -- m <= max(n,p)+1
);
port (
  x_std : in std_logic_vector(n-1 downto 0); -- [n,q]
  y_std : in std_logic_vector(p-1 downto 0); -- [p,q]
  z_std : out std_logic_vector(m-1 downto 0); -- [m,q]
  op : in std_logic
);
end saturated_adder;

architecture saturated_adder_arch of saturated_adder is

function MAX(LEFT, RIGHT: INTEGER) return INTEGER is
begin
  if LEFT > RIGHT then return LEFT;
  else return RIGHT;
end if;
end;

signal x : signed(n-1 downto 0); -- [n,q]
signal y : signed(p-1 downto 0); -- [p,q]
signal z : signed(m-1 downto 0); -- [m,q]

signal zx : signed(max(n,p) downto 0);
signal OVi : std_logic;
constant wmax : signed(m-2 downto 0) := (others=>'1');
constant wmin : signed(m-2 downto 0) := (others=>'0');

begin

x<=signed(x_std);
y<=signed(y_std);
z_std<=std_logic_vector(z);

zx <= (x(n-1)&x) + (y(p-1)&y) when op='0' else
      (x(n-1)&x) - (y(p-1)&y);

overflow_detect : process(zx)
variable temp : std_logic;
begin
temp := '0';
for I in m to max(n,p) loop
  if ((zx(I) xor zx(m-1))='1') then
    temp := '1';
  end if;
end loop;
end loop;
```

```
OVi <= temp;
end process;

z <= ('0' & wmax) when OVi='1' AND zx(max(n,p))='0' else
    ('1' & wmin) when OVi='1' AND zx(max(n,p))='1' else
    zx(m-1 downto 0);

end saturated_adder_arch;
```

### 6.3.3 Saturated Multiplier

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

entity saturated_multiplier is
generic (
    n, p, m : integer -- m<=n+p
);
port (
    x_std : in std_logic_vector(n-1 downto 0); -- [n,q]
    y_std : in std_logic_vector(p-1 downto 0); -- [p,l]
    z_std : out std_logic_vector(m-1 downto 0) -- [m,q+1]
);
end saturated_multiplier;

architecture saturated_multiplier_arch of saturated_multiplier is

    signal x : signed(n-1 downto 0); -- [n,q]
    signal y : signed(p-1 downto 0); -- [p,l]
    signal z : signed(m-1 downto 0); -- [m,q+1]

    signal zx : signed(n+p-1 downto 0);
    signal OVi : std_logic;
    constant wmax : signed(m-2 downto 0) := (others=>'1');
    constant wmin : signed(m-2 downto 0) := (others=>'0');

begin

    x<=signed(x_std);
    y<=signed(y_std);
    z_std<=std_logic_vector(z);

    zx <= x*y;

    overflow_detect : process(zx)
    variable temp : std_logic;
    begin
        temp := '0';
        for I in m to n+p-1 loop
            if ((zx(I) xor zx(m-1))='1') then
                temp := '1';
            end if;
        end loop;
        OVi <= temp;
    end process;

    z <= ('0'&wmax) when OVi='1' AND zx(n+p-1)='0' else
        ('1'&wmin) when OVi='1' AND zx(n+p-1)='1' else
        zx(m-1 downto 0);

end saturated_multiplier_arch;
```

### 6.3.4 Audio Processor

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

entity Audio_Processor is
    port (
        clk :IN std_logic;
        rst :IN std_logic;
        START: IN std_logic;
        SW_0: IN std_logic;
        SW_1: IN std_logic;
        DIN: IN std_logic_vector(7 downto 0);
        DOUT: buffer std_logic_vector(7 downto 0);
        DONE: OUT std_logic
    );
end Audio_Processor;

architecture behavioral of Audio_Processor is

    component Counter is
        generic(
            preset : integer := 255
        );
        port
        (
            clk      : in std_logic;
            rst      : in std_logic;
            ce       : in std_logic;
            tc       : out std_logic
        );
    end component;

    component saturated_adder is
        generic (
            n, p, m : integer
        );
        port (
            x_std : in std_logic_vector(n-1 downto 0);
            y_std : in std_logic_vector(p-1 downto 0);
            z_std : out std_logic_vector(m-1 downto 0);
            op : in std_logic
        );
    end component;

    component saturated_multiplier is
        generic (
            n, p, m : integer
        );
        port (
            x_std : in std_logic_vector(n-1 downto 0);
            y_std : in std_logic_vector(p-1 downto 0);
            z_std : out std_logic_vector(m-1 downto 0)
        );
    end component;
```

```

end component;

type state_type is (reset,idle,sample,LF_Count,LF_result,LF_last,LF_done,
                    HF_Count,HF_result,HF_last,HF_done,
                    LB_result,LB_wait,LB_done);

constant a_LF: std_logic_vector(7 downto 0):="00101110"; -- value 0.
                    1784 in sfixed(-1 downto -8) format
constant a_HF: std_logic_vector(7 downto 0):="00100110"; -- value 0.
                    1498 in sfixed(-1 downto -8) format
constant H02_LF: std_logic_vector(7 downto 0):="10001101"; --
                    value -0.45 in sfixed(-1 downto -8)
                    format
constant H02_HF: std_logic_vector(7 downto 0):="10000100"; -- value -0.
                    4841 in sfixed(-1 downto -8) format
constant inversion: std_logic_vector(7 downto 0):="00000001";
signal current_state, next_state: state_type;
signal DIN_star: std_logic_vector(7 downto 0);
signal X,XP,Y1P: std_logic_vector(7 downto 0);
signal outmux1,outmux2,outmux3,outadd1,outadd2,outadd4,outconv,inladd1,
                    in2add1,inadd4: std_logic_vector(7
                    downto 0);
signal outadd3 : std_logic_vector(8 downto 0);
signal outmult1,outmult2: std_logic_vector(15 downto 0);
signal outmult3: std_logic_vector(16 downto 0);
signal LE_X,NO_FILTER,FILTER_TYPE: std_logic;
signal CE_PR,RST_PR,TC_PR,LE_Y1P,LE_Y,LE_XP: std_logic;
signal RST_Y,RST_X,RST_XP,RST_Y1P,RST_SAMPLE: std_logic;
-----
begin

Counter_PROCESS: Counter
GENERIC MAP (preset=>1023)
PORT MAP (clk=>clk, rst=>RST_PR, ce=>CE_PR,tc=>TC_PR);

Sample_register: process(clk)
begin
if(clk'event and clk='1') then
    if(RST_SAMPLE='1') then
        DIN_star<=(others=>'0');
    else
        DIN_star<=DIN;
    end if;
end if;
end process;

X_register: process(clk)
begin
if(clk'event and clk='1') then
    if(RST_X='1') then
        X<=(others=>'0');
    elsif (LE_X='1') then
        X<=DIN_star;
    else

```

```

        X<=X;
    end if;
end if;
end process;

XP_register: process(clk)
begin
    if(clk'event and clk='1') then
        if(RST_XP='1') then
            XP<=(others=>'0');
        elsif (LE_XP='1') then
            XP<=X;
        else
            XP<=XP;
        end if;
    end if;
end process;

Y1P_register: process(clk)
begin
    if(clk'event and clk='1') then
        if(RST_Y1P='1') then
            Y1P<=(others=>'0');
        elsif (LE_Y1P='1') then
            Y1P<=outadd2;
        else
            Y1P<=Y1P;
        end if;
    end if;
end process;

Y_register: process(clk)
begin
    if(clk'event and clk='1') then
        if(RST_Y='1') then
            DOUT<=(others=>'0');
        elsif (LE_Y='1') then
            DOUT<=outmux3;
        else
            DOUT<=DOUT;
        end if;
    end if;
end process;

mux1: process(FILTER_TYPE)
begin
    if FILTER_TYPE='1' then
        outmux1<=a_HF;
    else
        outmux1<=a_LF;
    end if;
end process;

mux2: process(FILTER_TYPE)
begin

```

```

    if FILTER_TYPE='1' then
        outmux2<=H02_HF;
    else
        outmux2<=H02_LF;
    end if;
end process;

mux3: process (NO_FILTER,X,outadd4)
begin
    if NO_FILTER='1' then
        outmux3<=X;
    else
        outmux3<=outadd4;
    end if;
end process;

inverter: process(outmux1)
begin
    outconv<=std_logic_vector(unsigned(not(outmux1)) + unsigned(inversion))
                                ;
end process;

mult1: saturated_multiplier
generic map(8,8,16)
port map(outmux1,X,outmult1);

mult2: saturated_multiplier
generic map(8,8,16)
port map(outconv,Y1P,outmult2);

mult3: saturated_multiplier
generic map(9,8,17)
port map(outadd3,outmux2,outmult3);

in1add1<=outmult1(15 downto 8);
in2add1<=outmult2(15 downto 8);

add1: saturated_adder
generic map(8,8,8)
port map(in1add1,in2add1,outadd1,'0');

add2: saturated_adder
generic map(8,8,8)
port map(XP,outadd1,outadd2,'0');

add3: saturated_adder
generic map(8,8,9)
port map(X,outadd2,outadd3,FILTER_TYPE);

inadd4<=outmult3(16)&outmult3(14 downto 8);

add4: saturated_adder
generic map(8,8,8)
port map(inadd4,X,outadd4,'0');

```

```

cambio_stato: process(clk)
begin
    if(clk'event and clk='1') then
        case current_state is
            when idle => if (START='1') then next_state<=sample; end if;
            when sample => if (SW_0='1' and SW_1='0') then next_state<=HF_Count;
                           elsif (SW_0='0' and SW_1='1') then next_state<=LF_Count;
                           else next_state<=LB_result; end if;
            when LF_Count => if (TC_PR='1') then next_state<=LF_result; end if;
            when LF_result=> next_state<=LF_last;
            when LF_last=> next_state<=LF_done;
            when LF_done=> next_state<=idle;
            when HF_Count => if (TC_PR='1') then next_state<=HF_result; end if;
            when HF_result=> next_state<=HF_last;
            when HF_last=> next_state<=HF_done;
            when HF_done=> next_state<=idle;
            when LB_result=> next_state<=LB_wait;
            when LB_wait=> next_state<=LB_done;
            when LB_done=> next_state<=idle;
            when reset => next_state<=idle;
            when others => next_state<=reset;
        end case;
    end if;
end process;

aggiornamento_stato: process(next_state,rst)
begin
    if(rst='0') then
        current_state<=reset;
    else
        current_state<=next_state;
    end if;
end process;

cambio_output:process(current_state)
begin
    case current_state is
        when reset=> LE_X<='0'; NO_FILTER<='1'; FILTER_TYPE<='0'; CE_PR<='0';
                     RST_PR<='1';LE_Y1P<='0';LE_Y<='0';
                     LE_XP<='0';DONE<='0';RST_Y<='1';RST_X
                     <='1'; RST_XP<='1';RST_Y1P<='1';
                     RST_SAMPLE<='1';
        when idle=> LE_X<='0'; NO_FILTER<='1'; FILTER_TYPE<='0'; CE_PR<='0';
                     RST_PR<='0';LE_Y1P<='0';LE_Y<='0';
                     LE_XP<='0';DONE<='0';RST_Y<='0';RST_X
                     <='0'; RST_XP<='0';RST_Y1P<='0';
                     RST_SAMPLE<='0';
        when sample=> LE_X<='1'; NO_FILTER<='1'; FILTER_TYPE<='0'; CE_PR<='0';
                     ; RST_PR<='1';LE_Y1P<='0';LE_Y<='0';
                     LE_XP<='0';DONE<='0';RST_Y<='0';RST_X
                     <='0'; RST_XP<='0';RST_Y1P<='0';
                     RST_SAMPLE<='0';
    end case;
end process;

```



```

when LF_Count => LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='0'; CE_PR<=
    '1'; RST_PR<='0';LE_Y1P<='0';LE_Y<='0
    ';LE_XP<='0';DONE<='0';RST_Y<='0';
    RST_X<='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when LF_result=> LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='0'; CE_PR<=
    '0'; RST_PR<='1';LE_Y1P<='0';LE_Y<='1
    ';LE_XP<='0';DONE<='0';RST_Y<='0';
    RST_X<='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when LF_last=> LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='0'; CE_PR<='0
    '; RST_PR<='0';LE_Y1P<='1';LE_Y<='0';
    LE_XP<='0';DONE<='0';RST_Y<='0';RST_X
    <='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when LF_done=> LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='0'; CE_PR<='0
    '; RST_PR<='0';LE_Y1P<='0';LE_Y<='0';
    LE_XP<='1';DONE<='1';RST_Y<='0';RST_X
    <='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when HF_Count => LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='1'; CE_PR<=
    '1'; RST_PR<='0';LE_Y1P<='0';LE_Y<='0
    ';LE_XP<='0';DONE<='0';RST_Y<='0';
    RST_X<='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when HF_result=> LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='1'; CE_PR<=
    '0'; RST_PR<='1';LE_Y1P<='0';LE_Y<='1
    ';LE_XP<='0';DONE<='0';RST_Y<='0';
    RST_X<='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when HF_last=> LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='1'; CE_PR<='0
    '; RST_PR<='0';LE_Y1P<='1';LE_Y<='0';
    LE_XP<='0';DONE<='0';RST_Y<='0';RST_X
    <='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when HF_done=> LE_X<='0'; NO_FILTER<='0'; FILTER_TYPE<='1'; CE_PR<='0
    '; RST_PR<='0';LE_Y1P<='0';LE_Y<='0';
    LE_XP<='1';DONE<='1';RST_Y<='0';RST_X
    <='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when LB_result=> LE_X<='0'; NO_FILTER<='1'; FILTER_TYPE<='0'; CE_PR<=
    '0'; RST_PR<='0';LE_Y1P<='0';LE_Y<='1
    ';LE_XP<='0';DONE<='0';RST_Y<='0';
    RST_X<='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when LB_wait=> LE_X<='0'; NO_FILTER<='1'; FILTER_TYPE<='0'; CE_PR<='0
    '; RST_PR<='0';LE_Y1P<='0';LE_Y<='0';
    LE_XP<='0';DONE<='0';RST_Y<='0';RST_X
    <='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';
when LB_done=> LE_X<='0'; NO_FILTER<='1'; FILTER_TYPE<='0'; CE_PR<='0
    '; RST_PR<='0';LE_Y1P<='0';LE_Y<='0';
    LE_XP<='0';DONE<='1';RST_Y<='0';RST_X
    <='0'; RST_XP<='0';RST_Y1P<='0';
    RST_SAMPLE<='0';

```

```

        end case;
    end process;

end architecture;
```

## 6.4 TestAudioProcessor\_HF.py

[illegible]

```

file_output = "output_vectors.txt"
file_log = "log.txt"

#PER PLOT USCITA
py_plot=[]
sim_plot=[]
err_plot=[]
i=0

with open(file_sim, 'r') as f:
    with open(file_output, 'r') as f1:
        with open(file_log, 'w') as f2:
            for py, sim in zip(f1, f):
                #PER PLOT USCITA
                py_plot.append(float(py))
                sim_plot.append(float(sim))
                err_plot.append(abs(float(py)-float(sim)))
                if abs(float(py)-float(sim)) > pow(2,-7) :
                    print(float(py), '\t', float(sim), '\t', "--ERRORE--", '\n',
                        file=f2)
                    i+=1
                else:
                    print(float(py), '\t', float(sim), '\t', "--OK--", '\n',
                        file=f2)
            f2.close()
        f1.close()
    f.close()

print("Percentuale output con errore > 2LSB = %.3f %% " % (i*100/398))

#TIME BASE PER PLOT

time_base = np.linspace(0,1,398)

#PLOT INGRESSI

plt.figure(1)
plt.plot(time_base,vec[:len(vec)-1], "-x", label="IN")
plt.plot(time_base,py_plot, "-x", label="Python OUT")
plt.title("Ingressi vs Python simulation")
plt.legend()

plt.figure(2)
plt.plot(time_base,vec[:len(vec)-1], "-x", label="IN")
plt.plot(time_base,sim_plot, "-x", label="ModelSim OUT")
plt.title("Ingressi vs Modelsim simulation")
plt.legend()

plt.figure(3)
plt.plot(time_base,err_plot, "-x")
plt.title("Andamento errori")

plt.figure(4)
plt.plot(time_base,vec[:len(vec)-1], "-x", label="IN")
plt.plot(time_base,py_plot, "-x", label="Python OUT")

```

```
plt.plot(time_base,sim_plot,"-x",label="ModelSim OUT")
plt.title("Python vs Modelsim")
plt.legend()

plt.show()
```

## 6.5 TestAudioProcessor\_LF.py

```
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
from math import tan, pi
import os
import subprocess

G = -20
fs = 44100
fc = 1000
V0 = 10 ** (G / 20)
aB = (tan(2 * pi * fc / fs) - 1) / (tan(2 * pi * fc / fs) + 1)
aC = (tan(2 * pi * fc / fs) - V0) / (tan(2 * pi * fc / fs) + V0)
a = aC
H0 = V0 - 1
xp = 0
y1p = 0
vec = (random.rand(399) - 0.5)

file_input = "inputs.txt"
file_output = "output_vectors.txt"
with open(file_input, 'w') as f:
    with open(file_output, 'w') as fl:
        for X in vec:
            print(X, file=f)
            #process
            y1 = a * X + xp - a * y1p
            y = (H0 / 2) * (X + y1) + X
            y1p = y1
            xp = X
            #end process
            print(y, file=fl)
        fl.close()
f.close()

#####

# Launch Modelsim simulation
print ("Starting simulation...")
#DEVO ESEGUIRE DA DESKTOP!!!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
process = subprocess.call(["../..\\intelFPGA_lite\\20.1\\modelsim_ase\\bin\\vsim", "-c", "-do", "compile.do"])
print ("Simulation completed")

#####
```

```

file_sim = "outputs.txt"
file_output = "output_vectors.txt"
file_log = "log.txt"

#PER PLOT USCITA
py_plot=[]
sim_plot=[]
err_plot=[]
i=0

with open(file_sim, 'r') as f:
    with open(file_output, 'r') as f1:
        with open(file_log, 'w') as f2:
            for py, sim in zip(f1, f):
                #PER PLOT USCITA
                py_plot.append(float(py))
                sim_plot.append(float(sim))
                err_plot.append(abs(float(py)-float(sim)))
                if abs(float(py)-float(sim)) > pow(2,-7) :
                    print(float(py), '\t', float(sim), '\t', "--ERRORE--", '\n',
                        file=f2)
                    i+=1
                else:
                    print(float(py), '\t', float(sim), '\t', "--OK--", '\n',
                        file=f2)
            f2.close()
        f1.close()
    f.close()

print("Percentuale output con errore > 2LSB = %.3f %% " % (i*100/398))

#TIME BASE PER PLOT

time_base = np.linspace(0,1,398)

#PLOT INGRESSI

plt.figure(1)
plt.plot(time_base,vec[:len(vec)-1], "-x", label="IN")
plt.plot(time_base,py_plot, "-x", label="Python OUT")
plt.title("Ingressi vs Python simulation")
plt.legend()

plt.figure(2)
plt.plot(time_base,vec[:len(vec)-1], "-x", label="IN")
plt.plot(time_base,sim_plot, "-x", label="ModelSim OUT")
plt.title("Ingressi vs Modelsim simulation")
plt.legend()

plt.figure(3)
plt.plot(time_base,err_plot, "-x")
plt.title("Andamento errori")

plt.figure(4)
plt.plot(time_base,vec[:len(vec)-1], "-x", label="IN")

```

```
plt.plot(time_base,py_plot,"-x",label="Python OUT")
plt.plot(time_base,sim_plot,"-x",label="ModelSim OUT")
plt.title("Python vs Modelsim")
plt.legend()

plt.show()
```

## 6.6 compile.do

```
vlib work

vcom Counter.vhd
vcom saturated_adder.vhd
vcom saturated_multiplier.vhd
vcom Audio_Processor.vhd
vcom tb_audio_processor.vhd

vsim -c work.tb_audio_processor

#add list -decimal clk -notriggger a b c cout sum

run 0ns
run 10ms

#write list counter.lst
quit -f
```