Progetto di Reti Logiche

Filtro esponenziale fixed-point

Lorenzo Pigato

POLITECNICO
MILANO 1863

Indice

Introduzione

La specifica del progetto richiede l'implementazione di un filtro esponenziale fixed-point in cui i primi 16 bit rappresentano la parte intera e i secondi 16 la parte decimale.

Utilizzando questa notazione ed estendendola ai numeri negativi mediante rappresentazione in complemento a due, il range di lavoro del filtro è:

[+32767.9999847412 ; -32768.000]

La funzione matematica con la quale avviene il calcolo dell'uscita è stata fornita nelle specifiche:

$$Y_t = \alpha \cdot X_t + \alpha \cdot (1 - \alpha) \cdot Y_{t-1} + (1 - \alpha)^2 \cdot Y_{t-1}$$

Considerando le semplificazioni proposte dalla specifica, il coefficiente di filtro *α* è esprimibile come:

$$\alpha = \frac{1}{2^k}$$

Dove il valore di *k* è compreso tra 0 e 7 inclusi.

Date la funzione di calcolo dell'uscita e le semplificazioni fornite nella specifica, è stato possibile rielaborare matematicamente la formula di rappresentazione dell'output per renderla più facilmente computabile mediante una rete logica:

$$Y_t = \alpha \cdot X_t + Y_{t-1} \cdot (\alpha - \alpha^2 + 1 - 2 \cdot \alpha + \alpha^2)$$

Si può quindi notare come i coefficienti di secondo grado si elidano. Eseguendo i conti, la formula finale per l'espressione dell'uscita è la seguente:

$$Y_t = \alpha \cdot X_t + Y_{t-1} \cdot (1 - \alpha) \rightarrow Y_t = \alpha \cdot X_t + Y_{t-1} - Y_{t-1} \cdot \alpha$$

Specifica

Il sistema progettato è una struttura gerarchica di moduli interconnessi che si basa su componenti di base che costituiscono poi moduli che svolgono funzioni più avanzate, culminando infine con l'implementazione del modulo principale.

Considerando la funzione d'uscita semplificata calcolata nell'introduzione e le semplificazioni enunciate nella specifica di progetto, è possibile pensare il coefficiente *α* come una divisione per la potenza di 2 di ordine *k*, ovvero come uno *shift right* di k posizioni, con k sempre compreso tra 0 e 7.

In questo modo la funzione d'uscita diventa semplicemente calcolabile mediante una rete logica che esegue lo *shift* dell'ingresso e del valore precedente dell'uscita, quest'ultimo deve poi essere complementato a due in modo da invertirne il segno, e somma infine, mediante un sommatore a 3 operandi, i risultati ottenuti da queste operazioni con il valore precedente dell'uscita.

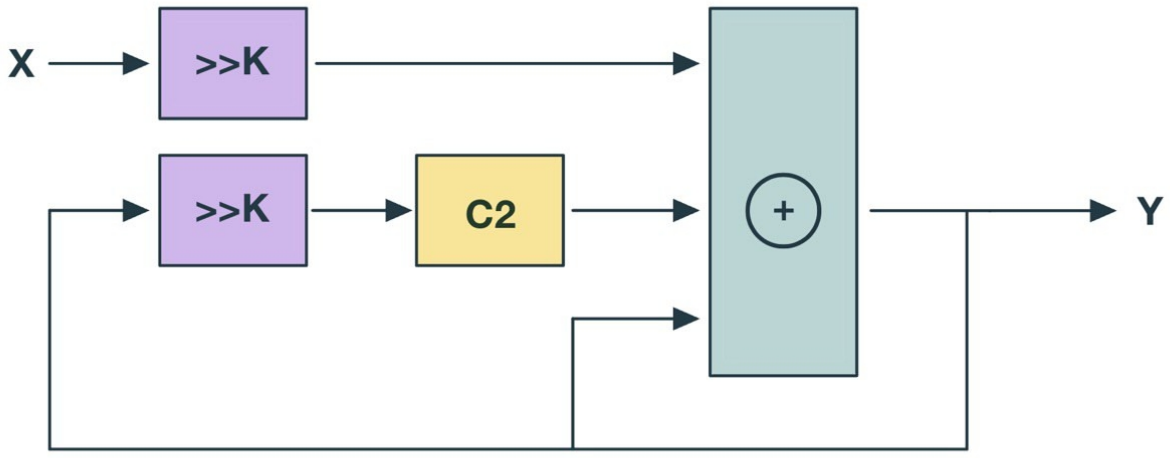
Per utilizzare il filtro è inoltre necessario inizializzarlo mediante l'opportuno segnale d'ingresso *Init,* che permette di impostare il valore di *k*. Il comportamento del filtro non è definito nel caso in cui si modifichi il valore di *k* durante la normale operatività del filtro senza poi reinizializzarlo.

È stato aggiunto a questo scopo un bit di *Valid,* che segnala nel ciclo seguente a un'inizializzazione che il segnale d'uscita non è attendibile.

Il funzionamento del filtro è esprimibile con la seguente formula:

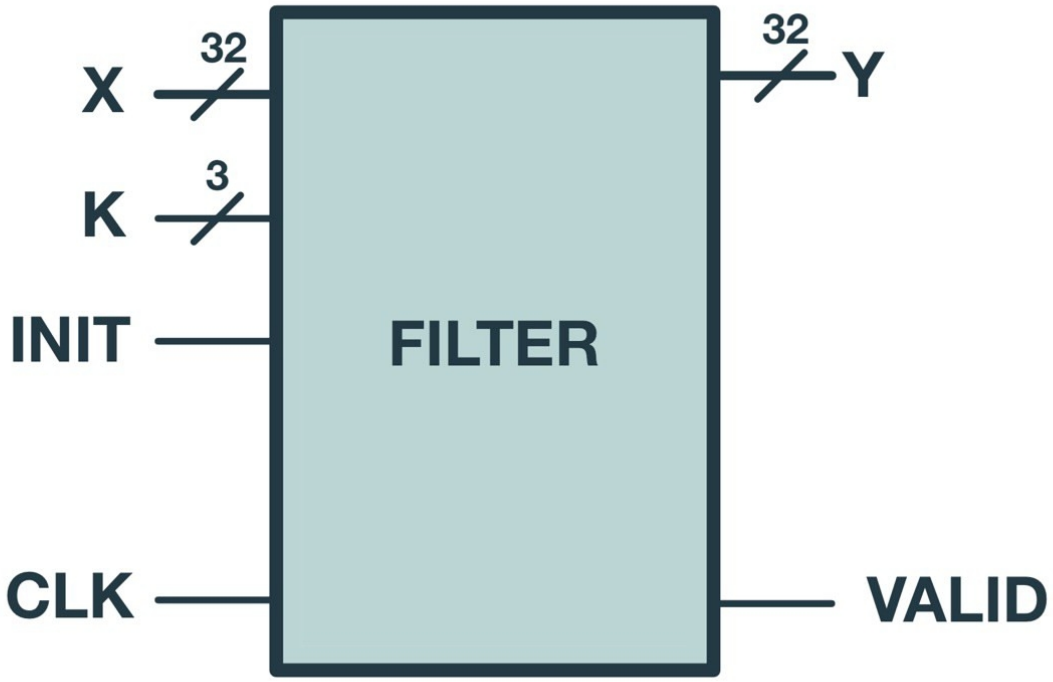
$$Y_t = \alpha \cdot X_t + Y_{t-1} - (Y_{t-1} \cdot \alpha) \rightarrow Y_t = [X_t \gg k] + Y_{t-1} + [\overline{(Y_{t-1} \gg k)} + 1]$$

I processi logici di funzionamento sono rappresentabili con il seguente diagramma di flusso:



Interfaccia

Complessivamente, l'interfaccia del *top module* è rappresentabile come segue:



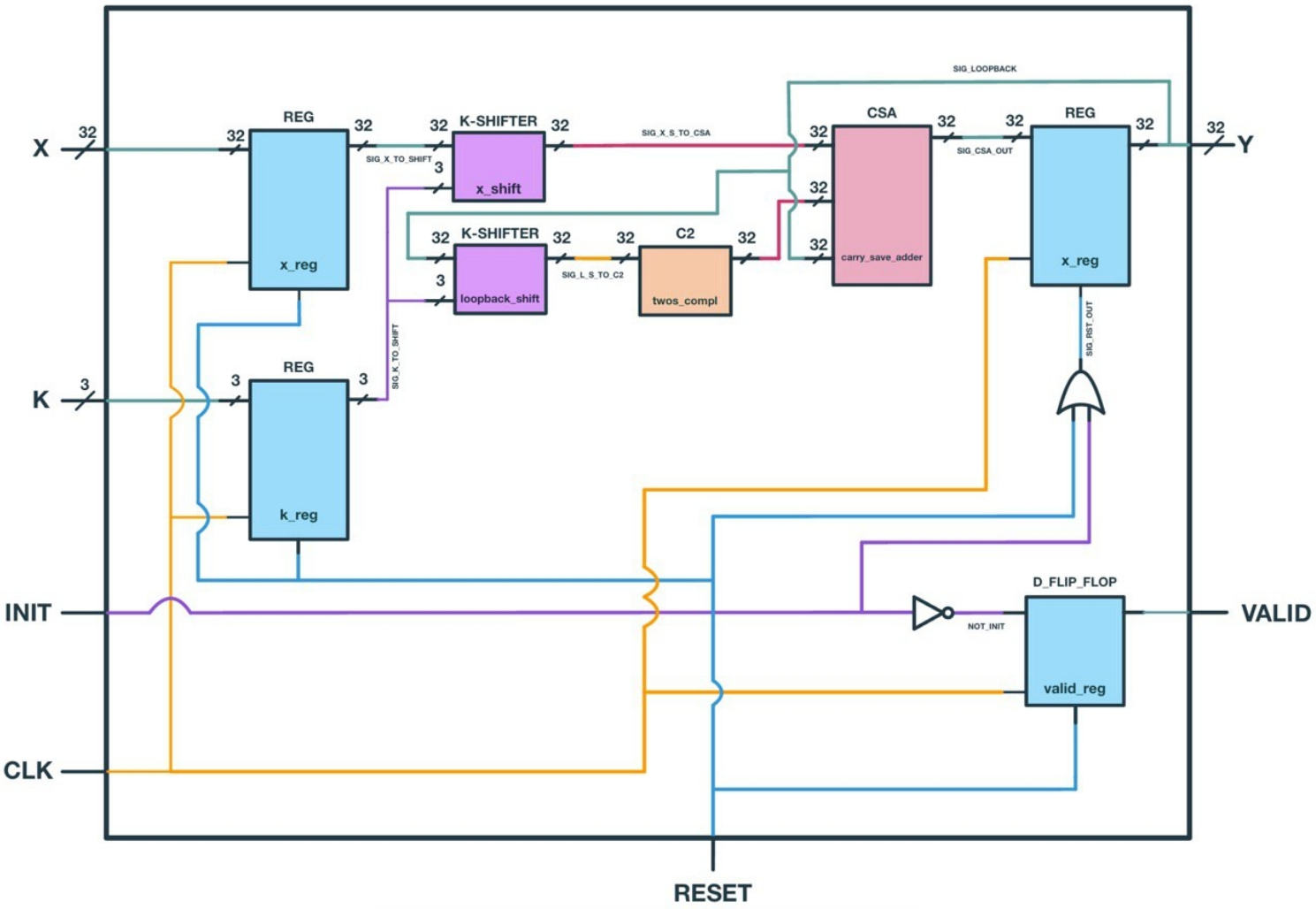
Il segnale *X* da 32 bit rappresenta il valore target posto in ingresso al circuito, mentre il segnale *K* da 3 bit rappresenta il coefficiente di filtro α.

Il segnale d'ingresso *Init* da 1 bit è invece utilizzato per inizializzare il filtro e impostare nuovi valori di *k.*

In uscita vi sono il segnale di output *Y* contenente il valore d'uscita del circuito e il bit *Valid*, che sengala la validità del risultato ottenuto

Architettura

Di seguito è riportata un’analisi dettagliata di ogni singolo modulo in modo da permettere la completa comprensione del funzionamento del sistema.



C2 – Complemento a due

Questo modulo è utilizzato per invertire il segno del segnale di *loopback* in modo da effettuare la sottrazione necessaria al calcolo dell'uscita. Il complemento a due è per definizione calcolato come:

$$C_2 = \overline{X} + 1$$

Per rendere più rapida la computazione della somma, è stato utilizzato un principio analogo a quello di un sommatore *carry-lookahead*, ovvero sono state sfruttate una logica di generazione e una di propagazione del riporto.

In un normale *carry-lookahead adder* la logica di calcolo del riporto (*Carry)* di ciasuno stadio è la seguente:

$$C_{i+1} = X_i \cdot Y_i + C_i (X_i + Y_i)$$

è poi possibile separare la logica di generazione dei riporti da quella di propagazione degli stessi:

Generazione: $G = X_i \cdot Y_i$

Propagazione: $P_i = X_i + Y_i$

Considerando che il secondo operando è sempre 1 per definizione, è possibile semplificare le due logiche:

Generazione: corrisponde con il negato del bit meno significativo segnale di *loopback*

Propagazione: è equivalente al segnale di *loopback* eccetto per il bit meno significativo, che è invece posto sempre a 1

Specifiche del modulo:

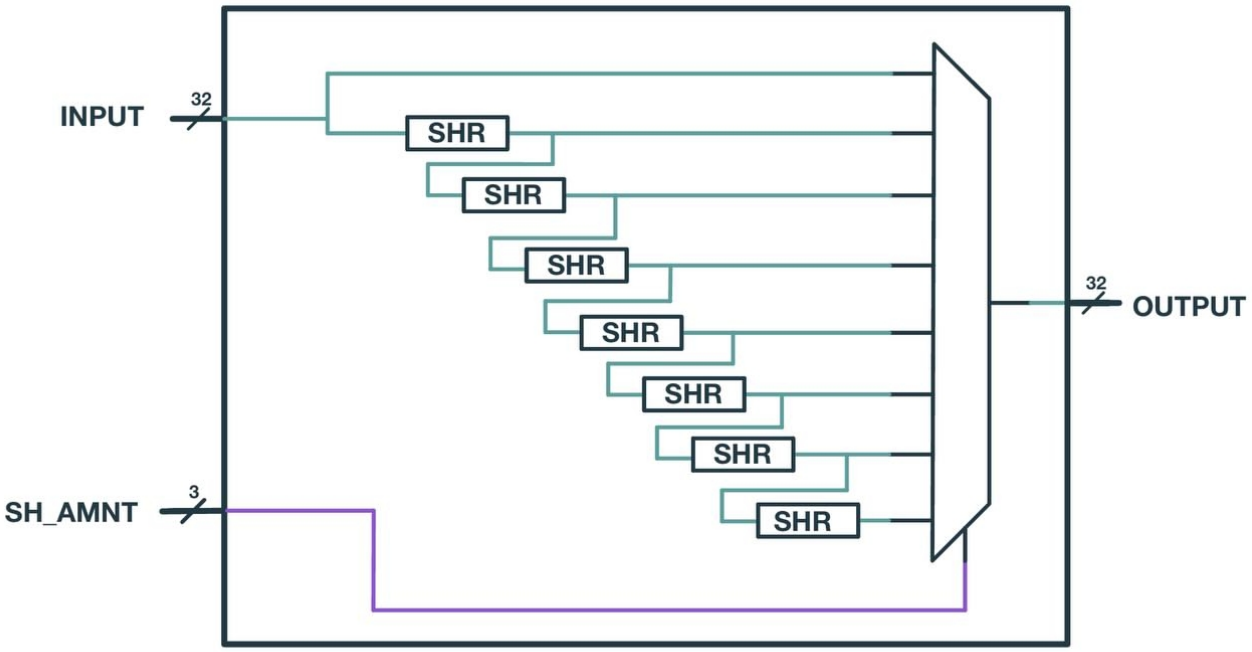
Interfaccia	C2	
Ingressi	A	32 bit
Uscite	C2_OUT	32 bit

K-Shifter – Shift di *k* bit

Questo modulo è responsabile del calcolo del valore *α,* ovvero del coefficiente variabile di filtro espresso come:

$$\alpha = \frac{1}{2^k}$$

Per effettuare questo calcolo nel modo più semplice possibile e sfruttando al meglio i costrutti *generics*, sono stati impiegati due sotto moduli: un *multiplexer* con selettore a 3 bit e ingressi da 32 bit e 7 moduli di *shift* in grado di effettuare lo *shift right* di un bit per ingressi a 32 bit.
L'implementazione del modulo è rappresentabile come segue:



I singoli *shifter* sono progettati in modo da estendere il segno dell'ingresso in modo che non si verifichi perdita d'informazione.

Specifiche del modulo:

Interfaccia	K-SHIFTER	
Ingressi	SH_AMNT	3 bit
	INPUT	N = 32 bit
Uscite	OUTPUT	N = 32 bit

SH_AMNT è il valore *k* propagato dal registro d'ingresso

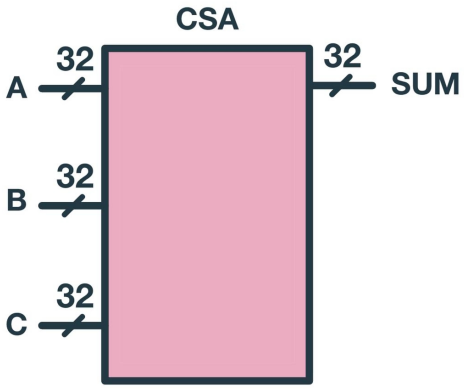
CSA – Carry Save Adder

Il modulo *Carry-Save Adder* implementa un sommatore per 3 operandi a 32 bit (variabile mediante *generic*) con output troncato su 32 bit, per mantenere la conformità alla specifica e in quanto non risulta possibile che la somma calcolata ecceda il numero di bit dell'ingresso siccome un filtro esponenziale è progettato per seguire e raggiungere il valore *target* che vi è posto in input.

Specifiche del modulo:

Interfaccia	K-SHIFTER	
Ingressi	A	N = 32 bit
	B	N = 32 bit
	C	N = 32 bit
Uscite	SUM	N = 32 bit

Al segnale d'ingresso *A* è assegnato il segnale *X* proveniente dal modulo *x_shift*, al segnale B corrisponde invece l'uscita del modulo *C2* che effettua l'inversione del segno del segnale di *loopba*ck, mentre *C* prende come input il segnale di *loopback* stesso proveniente dal registro di uscita.



REG – Registers

I registri costituiscono gli stadi di *pipeline* del circuito. Sono presenti due registri d'ingresso e un singolo registro d'uscita, dal quale viene propagato anche il segnale di *loopback.*

Ciascun registro sfrutta un costrutto *generic* che permette di scegliere il numero *N* di bit in ingresso e in uscita. Sono presenti un registro d'ingresso a 32 bit per il sengale *X* e un registro d'ingresso a 3 bit per il coefficiente di filtro *K.*

Ciascun registro è costituito da un numero variabile *N* di *flip-flop D* sensibili al fronte di salita del clock.

Specifiche del modulo:

Interfaccia	REG	
Ingressi	CLK	1 bit
	D	N bit
	RESET	1 bit
Uscite	Q	N bit

L'ingresso RESET è asincrono ed è utilizzato per azzerare il registro di uscita nel caso di inizializzazione del filtro mediante *Init* o nel caso di reset dell'intero circuito.

Moduli Standard

Sono stati impiegati i seguenti moduli standard:

FA – Full Adder

È stato impiegato nei moduli *CSA* e *C2* per svolgere somme tra due operandi considerando i riporti

MUX2/MUX – Multiplexer a 2/8 ingressi

Il *MUX2* è il componente base che costituisce il *MUX8* utilizzato per selezionare il corretto valore di *shifting* degli operandi all'interno dei moduli *x_shift* e *loopback_shift*. Il *MUX2* è stato realizzato tramite *generics* nonostante la sua implementazione effettiva sia solamente su 32 bit.

SHR – Shift Right

Il modulo *SHR* è impiegato anch'esso nei moduli di tipologia *K_SHIFTER* nei quali è collegato in cascata per permettere di ottenere facilmente in uscita l'ingresso scalato di un numero di bit arbitrario verso destra. È stato tenuto conto dell'eventuale segno dell'ingresso aggiungendo a sinistra del bit più significativo una copia dello stesso, in modo da mantenere la rappresentazione in complemento a due.

D_FLIP_FLOP – Flip-Flop D

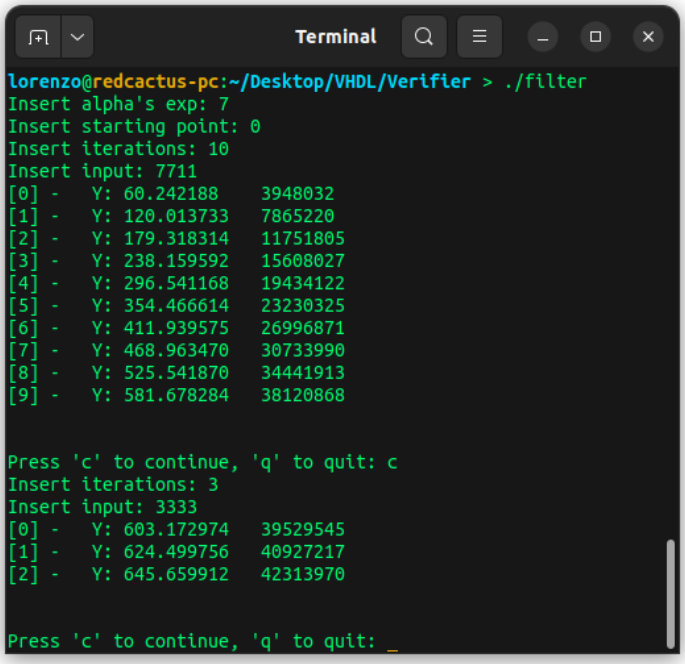
Il *flip-flop D* è il componente di base per la realizzazione di tutti i registri di memoria impiegati nel circuito. È stato realizzato tramite un costrutto *behavioral* che ha come segnali di riferimento il clock e il reset. Una interpretazione in pseudo-codice è la seguente:

```
if(RESET == 1)
     Q = 0;
else if(CLK_rising_edge)
     Q = D;
```

Verifica

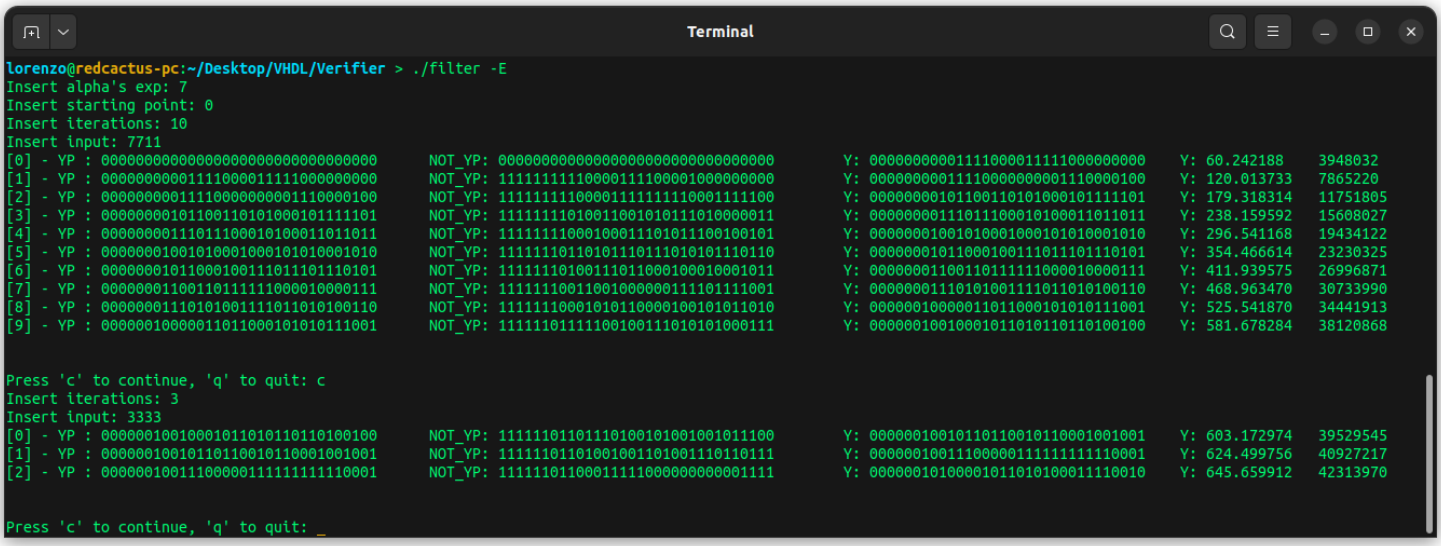
Al fine di verificare il corretto funzionamento del circuito, è stato realizzato un programma in linguaggio C che calcola il valore atteso per ogni iterazione, tenendo conto dello stato precedente del circuito e di eventuali cambiamenti dell'ingresso, ma mantenendo costante il valore di filtro *k* come definito dalle specifiche.

Un output utilizzato per verificare la correttezza dell'uscita del circuito logico è riportato in seguito:



Il parametro *apha's exp* è utilizzato per indicare in notazione decimale il parametro *k,* mentre *starting point* è utilizzato per impostare il valore iniziale contenuto nel registro di *loopback,* nel caso vi fosse.

L'output è scandito in base all'iterazione che lo produce ed è espresso sia in formato *floating point* che nella notazione decimale in modo da permettere un facile confronto con l'uscita prodotta dai test effettuati sul simulatore Xilinx.È inoltre possibile visualizzare i risultati espressi in notazione binaria utilizzando il flag "-E" o "--extended":



Test Bench

Behavioral test bench

Al fine di verificare il corretto funzionamento del circuito, sono stati predisposti appositi *test-bench* per ciascun modulo. I test bench sono tutti stati predisposti secondo una logica *behavioral*.

Sono stati inoltre definiti più test per il *top module* in modo da verificare la correttezza sia della risposta allo scalino, come richiesto da specifiche di progetto, sia all'eventuale variazione del valore target in ingresso durante la normale operatività del filtro. È di seguito riportato il solo output della risposta allo scalino per via della difficoltà della rappresentazione dei dati forniti dal simulatore Xilinx, è però possibile verificare la correttezza di qualsiasi altro test mediante il verificatore.

Post Place & Route test bench:

L'analisi di *timing post place & route* mira a trovare il percorso critico del sistema, ovvero la serie di istruzioni che richiede il tempo massimo d'esecuzione per passare dall'ingresso all'uscita.

È stato stimato dall'ambiente di sviluppo che il minimo tempo per un ciclo di clock sia di **10.955 [ns]**, ovvero che la massima frequenza ottenibile sia di **91.283 [MHz]**

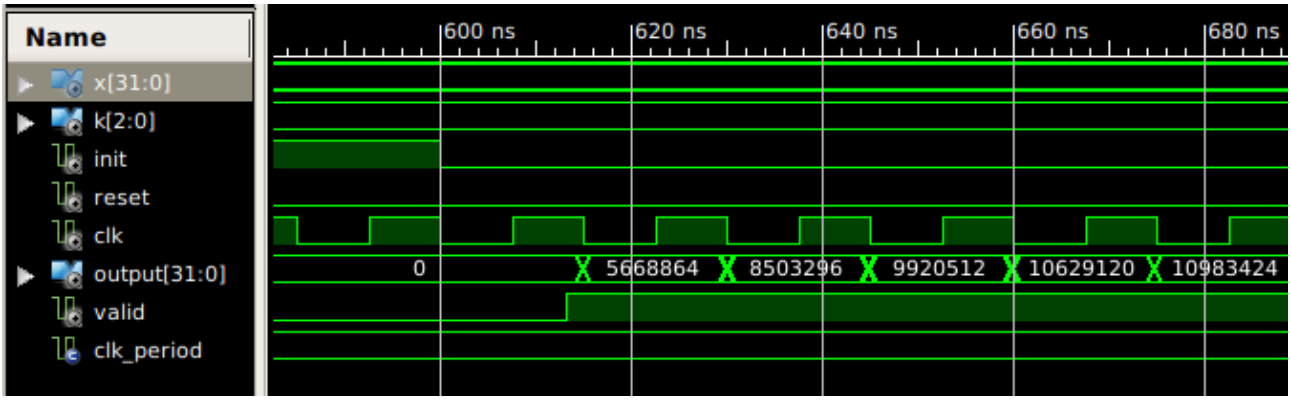
Frequenza di clock:

Considerando un margine per i tempi di setup dei registri in entrata e in uscita e per i tempi di propagazione del clock, è stato assegnato un tempo di ciclo pari a **15 [ns]**, ovvero una frequenza pari all'incirca a **66,67 [MHz]**.

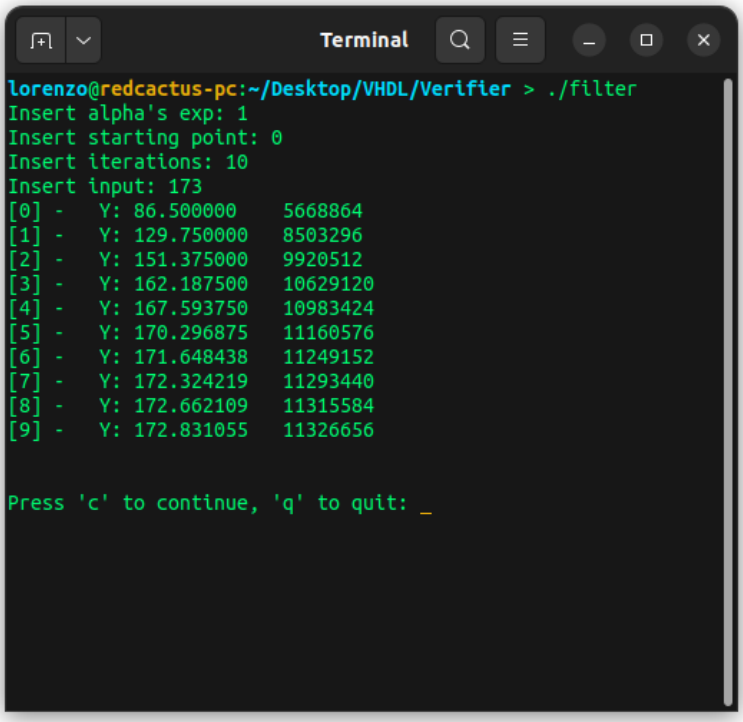
Il tempo di ciclo è stato assegnato a seguito di opportune verifiche tramite *Static Timing Analysis* e i risultati sono stati poi paragonati a quelli calcolati dal verificatore precedentemente descritto e dal *test behavioral.*

Test Case

È di seguito riportata una porzione di output del test riguardante la risposta allo scalino effettuata con un valore di ingresso pari a 173,0 e un α paria 2:



Si può verificare come i valori corrispondano con quelli riportati dal verificatore per gli stessi ingressi:



Si può anche constatare come il filtro raggiunga poi uno stato finale stabile: