POLITECNICO
MILANO 1863

Progetto di reti Logiche

Sommatore Floating Point IEEE754

Pesciotti A. e Pittari D.

Indice

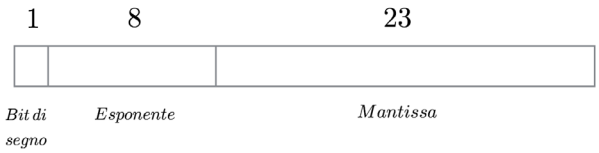
Introduzione

L'obiettivo centrale del progetto consiste nel progettare un sommatore floating point IEEE754 in VHDL tramite lo strumento software ISE di Xilinx

Standard IEEE 754

Lo standard corrente per la computazione a virgola mobile è IEEE 754. Esso fornisce il modello più usato universalmente per esprimere numeri a virgola mobile e include formati a precisione singola (32 bit), precisione doppia (64 bit), e le loro varianti. Nel nostro caso, l'attenzione è rivolta alla variante a 32 bit a precisione singola, che permette di esprimere numeri in un intervallo che va da $1{,}18 \cdot 10^{-38}$ a $3{,}40 \cdot 10^{38}$, inclusi casi speciali come ±0, ±∞ e NaN.

La rappresentazione di un numero a virgola mobile a precisione singola in conformità con l'IEEE 754 è composta da tre parti essenziali: il segno, l'esponente e la mantissa. Questi componenti sono strutturati come segue:



1. Il bit più significativo (31) è 0 per numeri positivi e 1 per quelli negativi.
2. I successivi 8 bit (dal bit 30 al bit 23) rappresentano l'esponente. Questo valore è calcolato aggiungendo 127 all'esponente reale del numero. Se indichiamo con "e" l'esponente effettivo e con "E" l'esponente del formato standard, il calcolo viene eseguito come segue:

$$E = e + 127$$

3. I rimanenti 23 bit (dal bit 22° al bit 0) costituiscono la mantissa, che rappresenta la parte frazionaria del numero.

In termini di notazione, chiamando "S" il segno, "E" l'esponente e "M" la mantissa, la rappresentazione numerica in accordo con queste disposizioni segue la seguente formula:

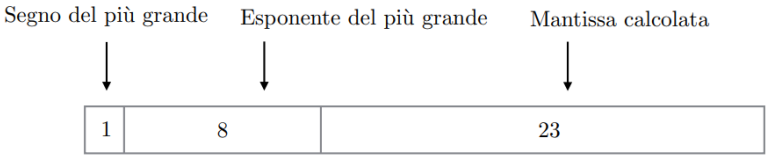
$$(-1)^{S} \cdot 1.M \cdot 2^{E}$$

È importante notare che l'uno a sinistra della virgola nella mantissa è implicito, dato che è già sottinteso dallo standard.

Sommatore a precisione singola

La procedura per sommare numeri a virgola mobile è fondamentalmente semplice e uniforme sia per precisione singola che doppia. Considerando due numeri normalizzati, etichettati come X e Y, le fasi da seguire includono:

1. Effettuare controlli per casi speciali come NaN o infiniti, dove il risultato è già determinato.
2. Se l'operazione è una sottrazione, invertire il segno del secondo numero.
3. Identificare il numero più grande tra i due: se necessario, fare uno scambio degli operandi per posizionare il numero più grande come primo numero.
4. Effettuare uno shift a destra della mantissa del numero più piccolo, della quantità pari alla differenza tra i due esponenti.
5. Procedere con l'addizione/sottrazione delle mantisse: se c'è un overflow della mantissa, eseguire un ulteriore shift a destra e incrementare l'esponente di uno.
6. A questo punto, il risultato parziale è stato ottenuto.
7. Terminare normalizzando il numero ottenuto e arrotondando secondo le regole di arrotondamento appropriate.



Gestione dei casi speciali

Oltre alle operazioni aritmetiche, lo standard IEEE 754 definisce anche regole per la gestione di casi speciali, come il trattamento di infinito, NaN (Not a Number) e zeri positivi e negativi.

La rappresentazione di tali casi segue la tabella:

Categoria	Esponente	Mantissa
Zeri	0_{10}	0_{10}
Numeri denormalizzati	0_{10}	non zero
Numeri normalizzati	$1_{10} - 254_{10}$	qualunque
Infiniti	255_{10}	0_{10}
NaN	255_{10}	non zero

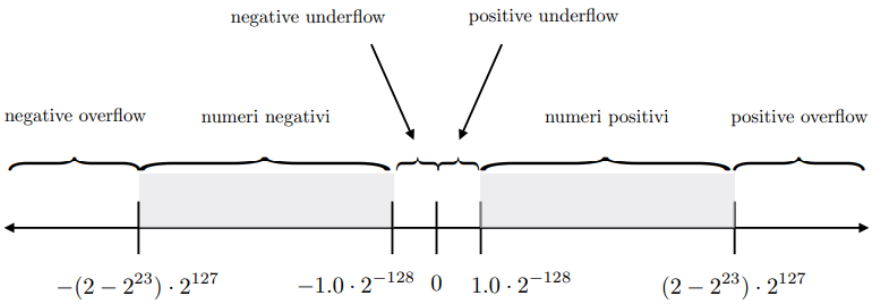
Per la codifica dei risultati NaN (Not a Number), è stato definito un formato di rappresentazione standard:

01111111111111111111111111111111

Per eseguire la somma tra casi speciali, lo standard definisce la seguente tabella:

Operando 1	Operando 2		Risultato
Non speciale	Zero	=	Operando 1
Zero	Non speciale	=	Operando 2
NaN	–	=	NaN
–	NaN	=	NaN
± Infinito	∓ Infinito	=	NaN
± Infinito	± Infinito	=	± Infinito
± Infinito	Non speciale	=	± Infinito
Non speciale	± Infinito	=	± Infinito

Arrotondamenti



In questa figura si evidenziano il range di valori che copre lo standard così come i range non coperti. Da questa figura risulta molto facile capire l'arrotondamento di default ovvero l'arrotondamento

al più vicino, ed è riassunto di seguito:

Negative overflow	→	$-\infty$		
Positive overflow	→	$+\infty$		
Negative underflow	→	$-denormalizzato$	→	**0**
Positive underflow	→	$+denormalizzato$	→	**0**

Abbiamo adottato determinati arrotondamenti per gestire casi particolari:

- Nel caso di un overflow, ossia quando l'esponente raggiunge 255 (rappresentato da 11111111) a causa della somma di due numeri dello stesso segno, il risultato diventa ±inf a seconda del segno dei valori coinvolti.

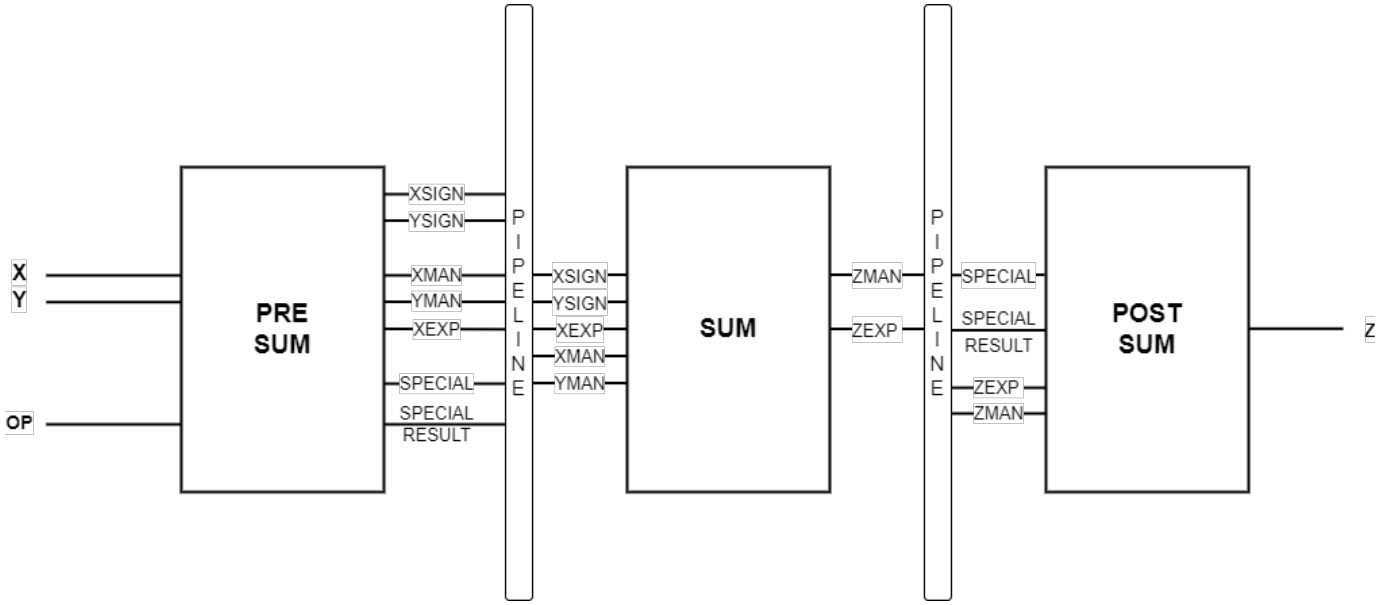
- Nel caso di un underflow, dove la differenza tra due numeri porterebbe a un valore compreso tra -2^{-128} e $2^{\wedge -128}$, avviene un arrotondamento a 0.

Struttura del sommatore

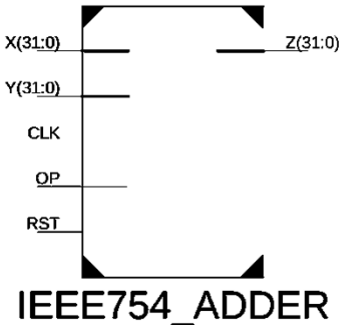
In questa sezione, saranno fornite informazioni di carattere generale riguardanti i componenti costituenti del sommatore. La scelta progettuale è stata orientata verso l'implementazione di un sommatore in virgola mobile che operi in modalità pipeline articolata in tre fasi distinte:

- una fase preliminare di preparazione, *pre-sum*
- una fase di somma effettiva, *sum*
- una fase di correzione, *post-sum*

Per gestire le tre fasi è stato scelto un ciclo di clock di 40 ns.



Top Level



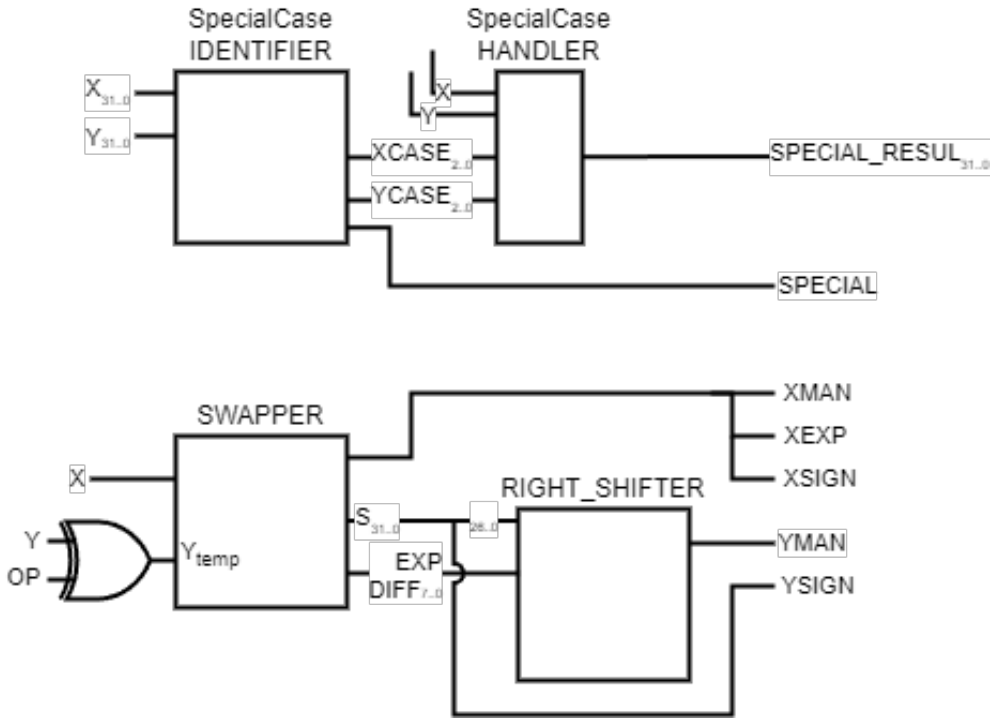

```
ENTITY IEEE754_ADDER IS
        PORT (
                X : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                Y : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
                OP : IN STD_LOGIC;
                Z : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
                CLK : IN STD_LOGIC;
                RST : IN STD_LOGIC
        );

END IEEE754_ADDER;
```

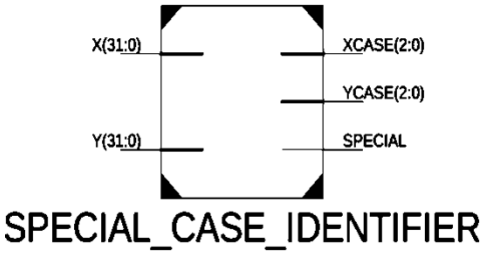
Il sommatore a top level richiede due operandi, definiti come X e Y (supponiamo che siano già stati normalizzati), l'indicazione dell'operazione da eseguire (0 per l'addizione, 1 per la sottrazione) e un segnale di clock per la gestione della pipeline.

Fase di Pre-Somma

Questa fase ha il compito di preparare i due operandi alla vera e propria fase di somma, ponendo in prima posizione l'operando più grande e shiftando opportunamente la mantissa dell'operando minore per una somma coerente.
Il modulo, inoltre, si occupa anche di identificare la presenza o meno di casi speciali e di calcolare il risultato della somma in presenza di essi.



Identificatore Casi Speciali



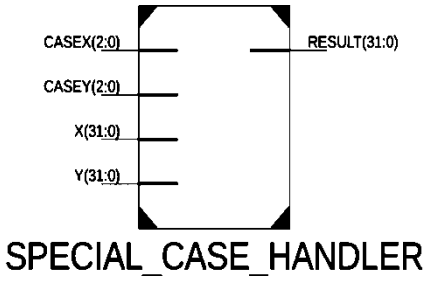

```
ENTITY SPECIAL_CASE_IDENTIFIER IS
    PORT (
        X     : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
        Y     : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
        SPECIAL : OUT STD_LOGIC;
        XCASE : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        YCASE : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)
    );
END SPECIAL_CASE_IDENTIFIER;
```

Il modulo prende in ingresso i due numeri normalizzati e assegna ad ognuno un vettore di 3 bit in base al caso a cui corrispondono. La codifica è dettata dalla tabella a lato.

Produce in uscita i due segnali che identificano il caso dei numeri e un bit che vale 0 se nessuno dei due numeri è un caso speciale, 1 altrimenti.

Caso	Codifica
Zero	000
+Inf	001
- Inf	010
NaN	011
Normal	100

Gestore di casi speciali

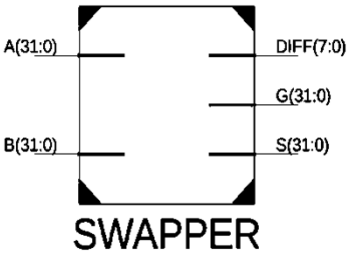



```
ENTITY SPECIAL_CASE_HANDLER IS
        PORT (
                CASEX  : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
                CASEY  : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
                X      : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
                Y      : IN  STD_LOGIC_VECTOR(31 DOWNTO 0);
                RESULT : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
        );
END SPECIAL_CASE_HANDLER;
```

Il modulo prende in input i due vettori di tre bit prodotti dal modulo precedente che identifica i casi per ogni singolo operando.

In base al valore dei due vettori calcola il risultato “speciale” seguendo le indicazioni dello standard.

Swapper

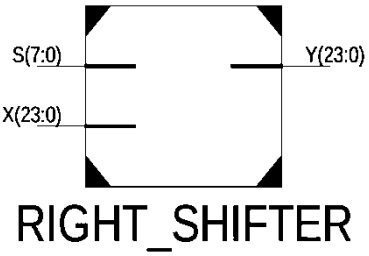



```
ENTITY SWAPPER IS
    PORT (
        A : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        B : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        G : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        S : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
        DIFF : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END SWAPPER;
```

Il modulo prendere in ingresso i due numeri normalizzati e ha lo scopo di invertire i due numeri, se necessario, per mettere in prima posizione quello più grande in valore assoluto.

Per fare ciò vengono utilizzati due comparatori, uno per esponente e uno per la mantissa, che funzionano come dei sottrattori per determinare quale numero è più grande.
Il comparatore utilizzato per l'esponente ha anche il compito di produrre in uscita la differenza tra di essi che verrà poi utilizzata per lo shifter.

Shifter (Right or Left)




```
ENTITY SHIFTER IS
    PORT (
        X : IN STD_LOGIC_VECTOR(23 DOWNTO 0);
        S : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        Y : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
    );
END SHIFTER;
```

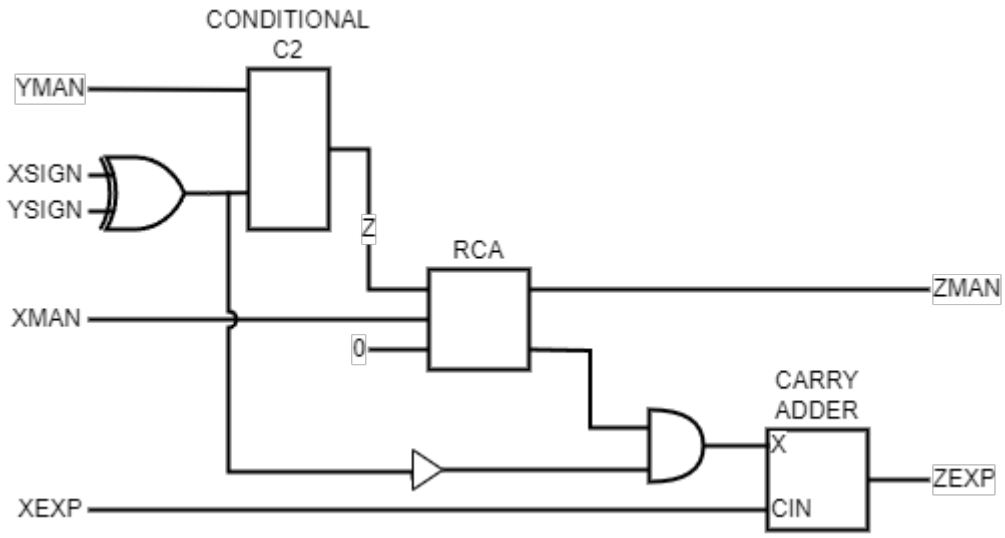
Questo modulo si occupa di shiftare la mantissa del numero più piccolo di un numero di posizioni determinata dalla differenza dei due esponenti calcolata in precedenza.
La mantissa precedentemente viene estesa aggiungendo il bit implicito in posizione più significativa e 3 bit di servizio infondo per gestire correttamente gli arrotondamenti.

Abbiamo utilizzato uno shifter di natura logaritmica, di conseguenza vengono eseguiti spostamenti di potenze del 2. Nel nostro caso, il segnale che indica il numero di spostamenti è rappresentato su 8 bit. Tuttavia, la strategia logaritmica è applicata esclusivamente ai 5 bit meno significativi, quindi se almeno uno dei restanti tre bit assume valore '1', la mantissa diventerà composta da soli '0'

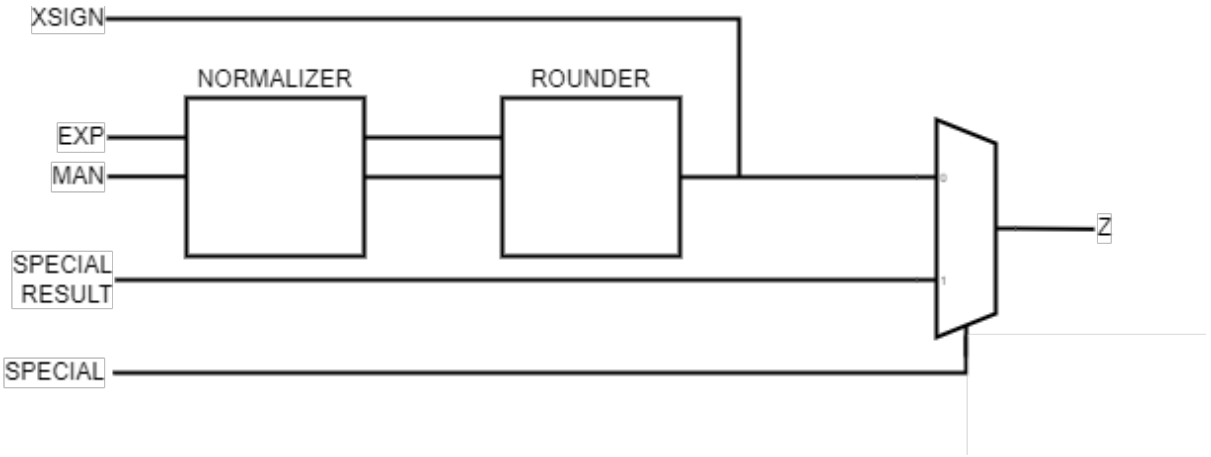
Fase di Somma

In questa fase viene eseguita la somma vera e propria delle mantisse o la differenza nel caso in cui i segni dei due operandi siano discordi.

In caso di differenza viene utilizzato un modulo che calcola il complemento a due dell'operando minore. Se siamo nel caso di una somma e viene prodotto un riporto dal RCA, l'esponente verrà incrementato di uno.

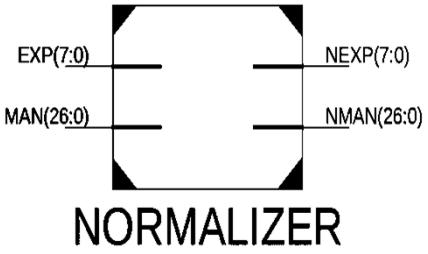


Fase Correttiva



Questa fase ha lo scopo di normalizzare i due numeri in seguito alla somma e arrotondare correttamente i due operandi, gestendo anche eventuali casi di underflow e overflow.

Normalizzatore

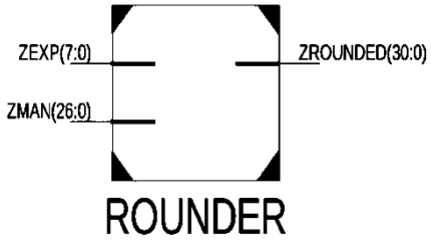



```
ENTITY NORMALIZER IS
  PORT (
          EXP : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          MAN : IN STD_LOGIC_VECTOR(26 DOWNTO 0);
          NEXP : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
          NMAN : OUT STD_LOGIC_VECTOR(26 DOWNTO 0)
  );
END NORMALIZER;
```

Il modulo ha il compito di normalizzare il risultato ottenuto dalla precedente fase di somma. All'interno di questo modulo, vengono elaborati l'esponente e la mantissa correnti e vengono prodotti in uscita normalizzati.

Per fare ciò viene utilizzato un priority-encoder che calcola di quanto la mantissa deve essere shiftata a sinistra in base alla posizione dell'uno più significativo. Dopo lo shift viene utilizzato un RCA per normalizzare anche l'esponente e se il riporto vale '1', si usano due multiplexer per porre il risultato a 0 poiché siamo di fronte ad un caso di underflow.

Arrotondamento




```
ENTITY ROUNDER IS
        PORT (
                ZEXP : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                ZMAN : IN STD_LOGIC_VECTOR(26 DOWNTO 0);
                ZROUNDED : OUT STD_LOGIC_VECTOR(30 DOWNTO 0)

        );
END ROUNDER;
```

Il modulo prende in input ZEXP e ZMAN che sono rispettivamente l'esponente e la mantissa dopo la normalizzazione, e produce in uscita l'esponente e mantissa finale che verranno posteriormente concatenati al segno del risultato calcolato in precedenza.

L'arrotondamento viene gestito seguendo lo standard, in particolare vengono presi gli ultimi 4 bit della mantissa, che comprendono il bit meno significativo e i 3 bit di servizio.

In base al valore di questi ultimi viene deciso se troncare la mantissa o se sommare prima un bit.

LSB	G	R	S	Incremento
0	0	-	-	0
0	1	0	0	0
0	1	0	1	1
0	1	1	-	1
1	0	-	-	0
1	1	-	-	1

Test-bench

All'interno di questo progetto, è stato scelto di adottare un approccio in cui i casi di test vengono definiti in base alle specifiche del**.**
Per ogni componente è presente un insieme limitato di casi di test che ne verificano il corretto funzionamento.
Per il componente finale invece si è scelta una vasta gamma di casi di test per coprire in maniera accurata tutte le possibili configurazioni e casi limite.

Casi d'uso

Questa sezione descrive brevemente i casi di test utilizzati per verificare il corretto funzionamento del top level sia in caso di componente con utilizzo di architettura pipelined che non.

Numeri normalizzati: è stato testato il corretto funzionamento del modulo in presenza di numeri normalizzati.

Limiti superiori: testati i limiti superiori che possono portare un arrotondamento a +Inf o -Inf

Limiti inferiori: testato il corretto arrotondamento a 0 di numeri non rappresentabili dallo standard IEEE754

Casi speciali: sono state testate tutte le possibili combinazioni nei casi in cui uno dei due o entrambi i numeri fossero dei casi speciali (NaN, +Inf, -Inf, Zero)