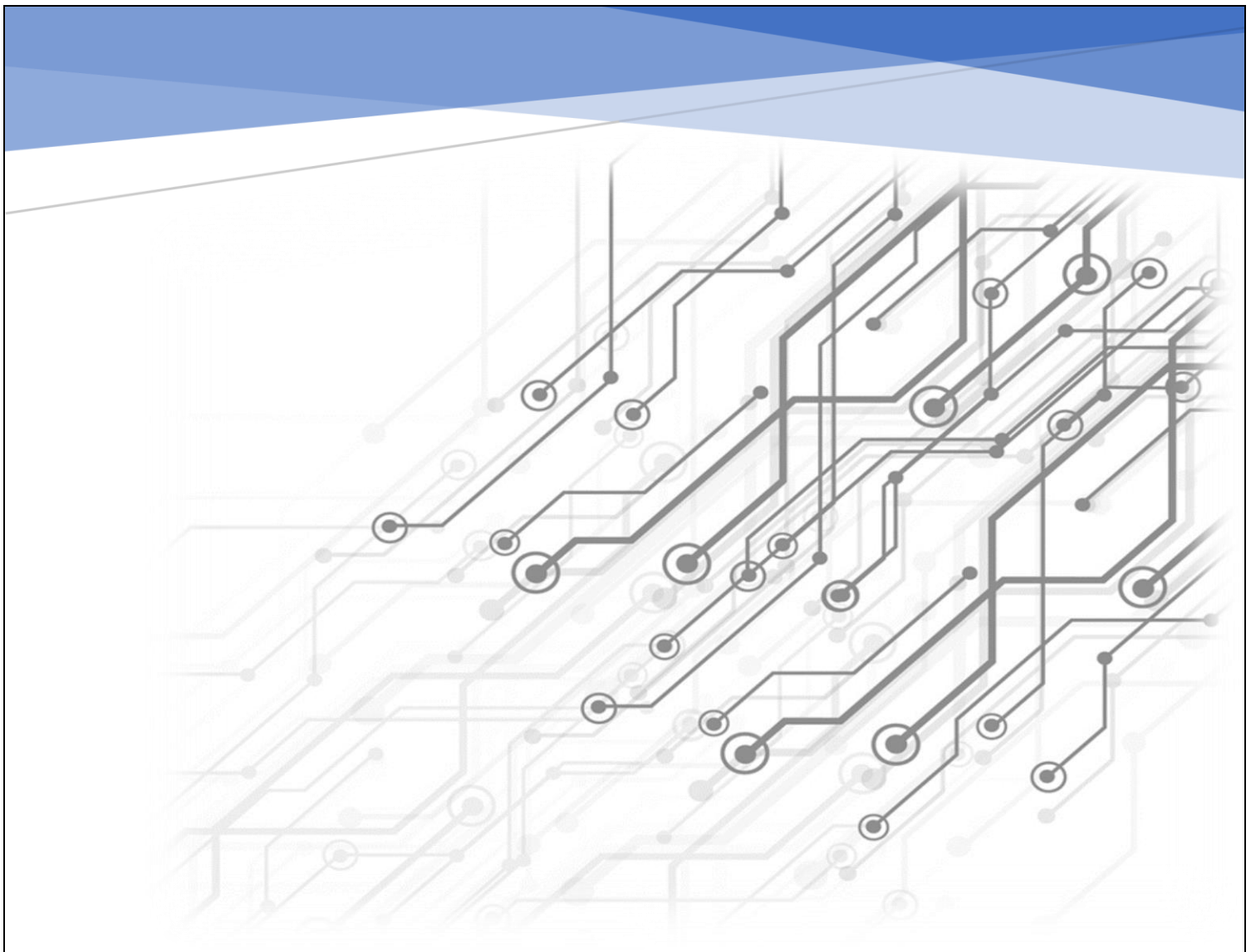


SOMMATORE FLOATING POINT IEEE 754

Documentazione

Sarzi Madidini Nicola, Scuttari Michele
[Politecnico di Milano – Ingegneria informatica]

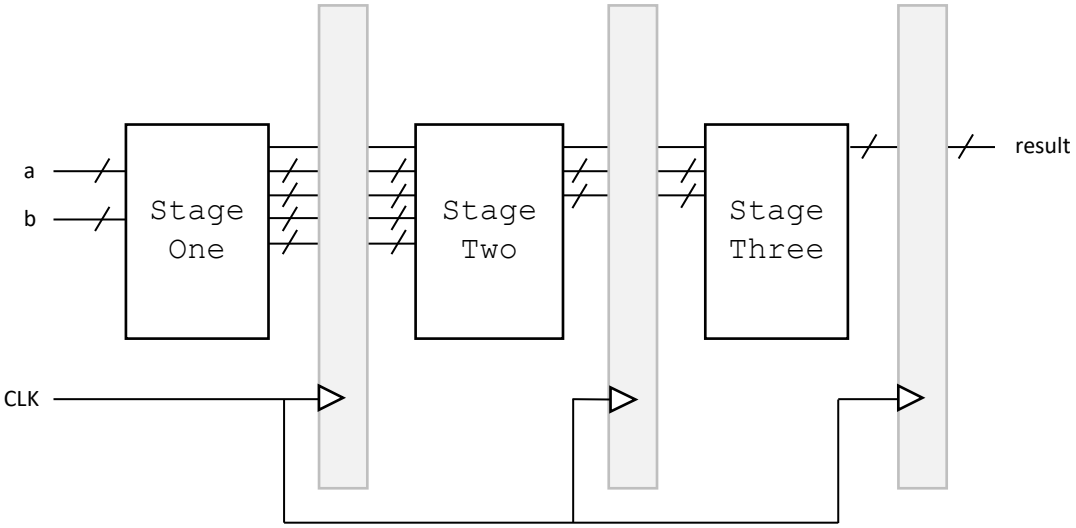
Indice

Entity top level

Descrizione del progetto

Lo scopo di questo progetto è la realizzazione in VHDL di un sommatore floating point a precisione singola (quindi operante a 32 bit) che rispetti lo standard IEEE 754. Il sommatore è in grado di trattare operandi normalizzati, operandi non normalizzati ed operandi speciali (come infinito e NaN). Inoltre, esegue la pura somma algebrica tra i valori forniti e non è quindi presente alcun segnale di controllo esterno volto alla selezione della somma o della sottrazione; sono solamente i segni degli operandi a determinare l'operazione da eseguire.

Struttura generale



Il sommatore implementa una pipeline a tre stadi (denominati StageOne, StageTwo e StageThree): il primo identifica l'operando con esponente maggiore, il secondo provvede ad effettuare la somma vera e propria tra le mantisse e il terzo esegue la normalizzazione del risultato.

Una volta forniti i due valori da sommare, il risultato è quindi presente al quarto successivo fronte di salita del clock. Il risultato è in realtà presente in maniera stabile anche al terzo fronte di discesa, ma si è scelto di campionare il segnale al fronte di salita successivo per uniformità del progetto.

Analisi dei timings

Per eseguire l'implementazione del design si è scelta la scheda Spartan3 XC3S1500L con package FG456, in quanto avente un numero di IOBs sufficiente per il progetto.

Dai report della sintesi per ogni stage emergono i seguenti ritardi combinatori:

STAGE	RITARDO MASSIMO
Stage One	25,288 ns
Stage Two	65,730 ns
Stage Three	64,477ns

Dall'analisi dei ritardi dei flip flop emerge un ritardo massimo CLK_TO_PAD di 13,727 ns.

E' stato quindi scelto un periodo di clock pari a 80 ns $(80ns > 65{,}730ns + 13{,}727ns)$, corrispondente a una frequenza di 12,5 MHz.

Descrizione dei test-bench

Ad ogni modulo in seguito analizzato è associato un test-bench volto ad analizzarne il corretto funzionamento durante l'analisi behavioral. In ogni test-bench si è deciso inoltre di inserire un segnale di "check" volto al confronto automatico tra i valori restituiti dal modulo e quelli previsti; in questo modo è possibile eseguire la simulazione e controllare rapidamente la validità dei risultati senza dover analizzare ogni segnale singolarmente.

Per il modulo principale è stata realizzata una simulazione post place & route in modo da verificare il reale funzionamento di fronte alle reali caratteristiche dei componenti. Si riportano quindi di seguito i risultati:

CASO SPECIALE: ZERO + VALORE

a	00000000000000000000000000000000	0
b	01111110110101010101010101010101	$+1{,}4178432 \cdot 10^{38}$
result	01111110110101010101010101010101	$+1{,}4178432 \cdot 10^{38}$

CASO SPECIALE: VALORE $-\infty$

a	01111110101010101010101010101010	$2{,}8356863 \cdot 10^{38}$
b	11111111100000000000000000000000	$-\infty$
result	01111110110101010101010101010101	$-\infty$

CASO SPECIALE: $-\infty - \infty$

a	11111111100000000000000000000000	$-\infty$
b	11111111100000000000000000000000	$-\infty$
result	11111111100000000000000000000000	$-\infty$

CASO SPECIALE: $-\infty + \infty$

a	11111111100000000000000000000000	$-\infty$
b	01111110110101010101010101010101	$+\infty$
result	01111111100000000000000000000001	NaN

CASO SPECIALE: $-\infty + \mathbf{NaN}$

a	11111111100000000000000000000000	$-\infty$
b	01111111110101010101010101010101	NaN
result	01111111100000000000000000000001	NaN

SOMMA NORMALE

a	01111110110101010101010101010101	$+1{,}4178432 \cdot 10^{38}$
b	11111110110101010101010101010001	$-1{,}4178428 \cdot 10^{38}$
result	01110100000000000000000000000000	$+4{,}05682 \cdot 10^{31}$

SOMMA NORMALE

a	01111010101010100101010110101010	$+4{,}4221397 \cdot 10^{35}$
b	11111010110101010101010101010101	$-5{,}53845 \cdot 10^{35}$
result	11111001101010111111111010101100	$-1{,}1163102 \cdot 10^{35}$

SOMMA NORMALE

a	11111101010011001100110011001100	$-1{,}7014117 \cdot 10^{37}$
b	01111101001010100101010110101010	$+1{,}4150847 \cdot 10^{37}$
result	11111100000010011101110010001000	$-2{,}8632703 \cdot 10^{36}$

SOMMA NORMALE

a	11111000010111011101110111011101	$-1{,}7999961 \cdot 10^{34}$
b	11111000010111001101110011011100	$-1{,}7918514 \cdot 10^{34}$
result	11111000110111010101110101011101	$-3{,}5918476 \cdot 10^{34}$

SOMMA NORMALE

a	01111100110111011101110111011101	$+9{,}21598 \cdot 10^{36}$
b	01111011110111001101110011011100	$+2{,}2935697 \cdot 10^{36}$
result	01111101000010101000101010001010	$+1{,}150955 \cdot 10^{37}$

SOMMA TRA VALORI NON NORMALIZATI

a	00000000010101010101010101010001	$+7{,}836623 \cdot 10^{-39}$
b	00000000010101010101010101010101	$+7{,}836629 \cdot 10^{-39}$
result	00000000101010101010101010100110	$+1{,}5673251 \cdot 10^{-38}$

SOMMA TRA VALORI NON NORMALIZATI

a	00000000011111111111111111111111	$+1{,}1754942 \cdot 10^{-38}$
b	10000000001010100101010110101010	$-3{,}887821 \cdot 10^{-39}$
result	00000000010101011010101001010101	$+7{,}867121 \cdot 10^{-39}$

SOMMA TRA VALORI NON NORMALIZATI

a	00000000001010100101010110101010	$+3{,}887821 \cdot 10^{-39}$
b	10000000011111111111111111111111	$-1{,}1754942 \cdot 10^{-38}$
result	10000000010101011010101001010101	$-7{,}867121 \cdot 10^{-39}$

SOMMA TRA VALORI NON NORMALIZATI

a	10000000010111011101110111011101	$-8{,}62029 \cdot 10^{-39}$
b	00000000000111011101110111011101	$+2{,}742819 \cdot 10^{-39}$
result	10000000010000000000000000000000	$-5{,}877472 \cdot 10^{-39}$

SOMMA TRA VALORI NON NORMALIZATI

a	10000000010111001101110011011100	$-8{,}528095 \cdot 10^{-39}$
b	10000000010111011101110111011101	$-8{,}62029 \cdot 10^{-39}$
result	10000000101110101011101010111001	$-1{,}7148386 \cdot 10^{-38}$

SOMMA MISTE (NORMALIZZATO E NON)

a	00000010010101010101010101010001	$+1{,}5673253 \cdot 10^{-37}$
b	00000000010101010101010101010101	$+7{,}836629 \cdot 10^{-39}$
result	00000010010111111111111111111100	$+1{,}6456916 \cdot 10^{-37}$

SOMMA MISTE (NORMALIZZATO E NON)

a	10000100010101010101010101010101	$-2{,}5077212 \cdot 10^{-36}$
b	10000000010101010101010101010101	$-7{,}836629 \cdot 10^{-39}$
result	10000100010101100000000000000000	$-2{,}515558 \cdot 10^{-36}$

SOMMA MISTE (NORMALIZZATO E NON)

a	00000101001010100101010110101010	$+8{,}0090956 \cdot 10^{-36}$
b	00000000011111111111111111111111	$+1{,}1754942 \cdot 10^{-38}$
result	00000101001010101001010110101010	$+8{,}0208505 \cdot 10^{-36}$

SOMMA MISTE (NORMALIZZATO E NON)

a	10000000010101010101010101010101	$-7{,}836629 \cdot 10^{-39}$
b	00000101001110111011101110111101	$+2{,}319642 \cdot 10^{-37}$
result	00000101001100010001000100010001000	$+2{,}2412758 \cdot 10^{-37}$

SOMMA MISTE (NORMALIZZATO E NON)

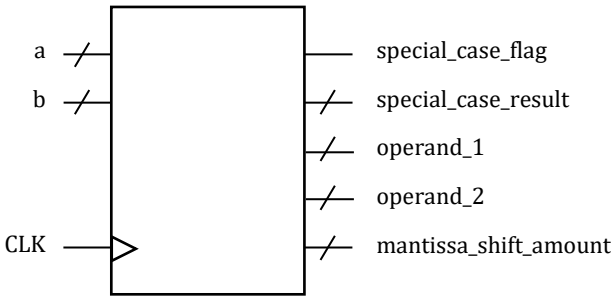
a	10000000010101010101010101010101	$-7{,}836629 \cdot 10^{-39}$
b	00000111101010101010101010101010	$+2{,}5679064 \cdot 10^{-34}$
result	00000111101010101010100101010101	$+2{,}5678281 \cdot 10^{-34}$

StageOne

Descrizione

È il primo stadio della pipeline e ha come scopo la determinazione dell'operando con esponente più grande. Provvede inoltre ad analizzare gli operandi al fine di riconoscere eventuali casi speciali il cui risultato è già noto; in caso di riscontro positivo, un apposito segnale indica agli stage successivi tale situazione e il risultato verrà semplicemente propagato nei vari stadi della pipeline, portando quindi ad ignorare i normali calcoli.

Segnali



SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
CLK	in	1	Clock
a	in	32	Primo operando
b	in	32	Secondo operando
special_case_flag	out	1	Indica se si è riscontrato un caso speciale
special_case_result	out	32	Risultato dell'eventuale caso speciale
operand_1	out	32	Operando con l'esponente minore
operand_2	out	32	Operando con l'esponente maggiore
mantissa_shift_amount	out	8	Numero di posizioni di cui shiftare a destra la mantissa dell'operando con esponente più piccolo

Funzionamento

- Gli operandi vengono forniti all'apposito modulo volto all'analisi dei casi speciali e il risultato viene propagato attraverso i vari stadi della pipeline. Se si è riconosciuto un caso speciale, vengono comunque eseguite le operazioni generali descritte in seguito ma vengono riportati in output dei don't care per quanto riguarda operand_1, operand_2 e mantissa_shift_amount.
- Viene effettuata la differenza tra l'esponente del primo operando e l'esponente del secondo e i due operandi vengono scambiati a seconda del segno del risultato. Se negativo, infatti, significa che il primo operando ha esponente minore del secondo.
- Viene effettuata la differenza tra l'esponente del primo operando e l'esponente del secondo, dove però ogni esponente, qualora non fosse normalizzato, viene trattato come 00000001, in quanto effettivamente rappresentante tale valore (come mostrato nella tabella, cambia infatti soltanto il numero che precede la virgola).

ESPONENTE IN BASE 2	ESPONENTE IN BASE 10	MANTISSA
00000000	-126	(0.)10000000000000000000000
00000001	-126	(1.)10000000000000000000000

In tal modo si ottiene il numero di posizioni di cui è necessario shiftare a destra la mantissa dell'operando con esponente più piccolo.

Test-bench

PRIMO ESPONENTE PIÙ PICCOLO DEL SECONDO

INPUT	a	01000010010010010000000000000000
	b	11000010111011010000000000000000
OUTPUT	special_case_flag	0
	special_case_result	--------------------------------
	operand_1	01000010010010010000000000000000
	operand_2	11000010111011010000000000000000
	mantissa_shift_amount	00000001

PRIMO ESPONENTE PIÙ GRANDE DEL SECONDO

INPUT	a	11000010111011010000000000000000
	b	01000010010010010000000000000000
OUTPUT	special_case_flag	0
	special_case_result	--------------------------------
	operand_1	01000010010010010000000000000000
	operand_2	11000010111011010000000000000000
	mantissa_shift_amount	00000001

ESPONENTI UGUALI

INPUT	a	11000010111011010000000000000000
	b	01000010110010010000000000000000
OUTPUT	special_case_flag	0
	special_case_result	--------------------------------
	operand_1	01000010110010010000000000000000
	operand_2	11000010111011010000000000000000
	mantissa_shift_amount	00000000

PRIMO OPERANDO NON NORMALIZZATO

INPUT	a	10000000011011010000000000000000
	b	00000000110010010000000000000000
OUTPUT	special_case_flag	0
	special_case_result	--------------------------------
	operand_1	10000000011011010000000000000000
	operand_2	00000000110010010000000000000000
	mantissa_shift_amount	00000000

SECONDO OPERANDO NON NORMALIZZATO

INPUT	a	00000000110010010000000000000000
	b	10000000011011010000000000000000
OUTPUT	special_case_flag	0
	special_case_result	--------------------------------
	operand_1	10000000011011010000000000000000
	operand_2	00000000110010010000000000000000
	mantissa_shift_amount	00000000

ENTRAMBI GLI OPERANDI NON NORMALIZZATI

INPUT	a	10000000011011010000000000000000
	b	00000000010010010000000000000000
OUTPUT	special_case_flag	0
	special_case_result	--------------------------------
	operand_1	00000000010010010000000000000000
	operand_2	10000000011011010000000000000000
	mantissa_shift_amount	00000000

CASO SPECIALE

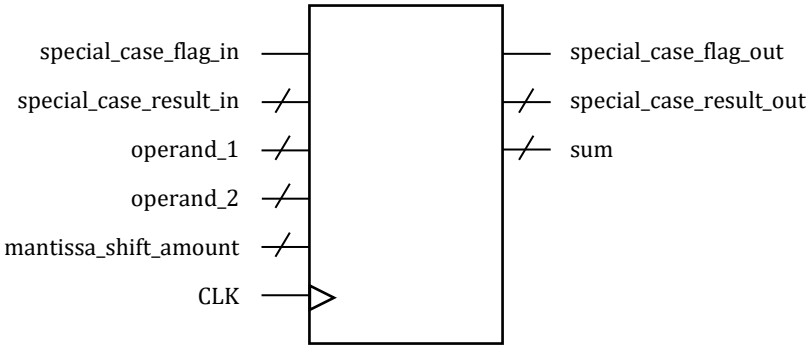
INPUT	a	11111111100000000000000000000000
	b	11101011001101001001001110100101
OUTPUT	special_case_flag	1
	special_case_result	11111111100000000000000000000000
	operand_1	----------
	operand_2	----------
	mantissa_shift_amount	--------

StageTwo

Descrizione

È il secondo stadio della pipeline e ha come scopo il calcolo della somma tra le due mantisse e la determinazione del segno del risultato. L'eventuale presenza di casi speciali viene semplicemente propagata senza alterazioni. Viene tuttavia generato un ulteriore caso speciale corrispondente al valore di infinito qualora la somma eccedesse il massimo valore rappresentabile.

Segnali



SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
CLK	in	1	Clock
special_case_flag_in	in	1	Indica se si è riscontrato un caso speciale
special_case_result_in	in	32	Risultato dell'eventuale caso speciale
operand_1	in	32	Operando con l'esponente minore
operand_2	in	32	Operando con l'esponente maggiore
mantissa_shift_amount	in	8	Numero di posizioni di cui shiftare a destra la mantissa dell'operando con esponente più piccolo
special_case_flag_out	out	1	Indica se si è riscontrato un caso speciale
special_case_result_out	out	32	Risultato dell'eventuale caso speciale
sum	out	37	Somma degli operandi

Funzionamento

- Le mantisse di entrambi gli operandi vengono estese a 28 bit: il primo bit viene settato a 1 se l'operando è normalizzato, 0 altrimenti; viene concatenata poi la mantissa originaria e vengono infine aggiunti quattro bit di guardia per gli arrotondamenti.

Esempio

	Segno	*Esponente*	*Mantissa*
Operando normalizzato:	0	01101011	10010010100101011001001
Risultato:	0	01101011	**1**10010010100101011001001**0000**
Operando non normalizzato:	0	00000000	10010010100101011001001
Risultato:	0	00000000	**0**10010010100101011001001**0000**

- L'operando con esponente maggiore viene portato direttamente in ingresso ai registri in quanto non vengono effettuate operazioni che comporterebbero la sua modifica.

- La mantissa dell'operando con esponente più piccolo viene shiftata a destra di un valore pari al relativo segnale in ingresso, indicato nella tabella precedente.

- Viene eseguita la somma algebrica tra le due mantisse. Per fare ciò, vengono eseguite contemporaneamente la somma e le due possibili differenze, e vengono scelti segno e mantissa del risultato in base ai segni degli operandi e al segno della prima differenza (la scelta della prima in favore della seconda è arbitraria; in caso di risultato positivo si conclude che la prima mantissa è maggiore della seconda e viceversa).

SEGNO OPERANDO 1	SEGNO OPERANDO 2	MANTISSA MAGGIORE	SEGNO DEL RISULTATO	SOMMA ALGEBRICA DELLE MANTISSE
$+$	$+$	$\mathrm{M1}$	$+$	$\mathrm{M1}+\mathrm{M2}$
$+$	$+$	$\mathrm{M2}$	$+$	$\mathrm{M1}+\mathrm{M2}$
$+$	$-$	$\mathrm{M1}$	$+$	$\mathrm{M1}-\mathrm{M2}$
$+$	$-$	$\mathrm{M2}$	$-$	$\mathrm{M2}-\mathrm{M1}$
$-$	$+$	$\mathrm{M1}$	$-$	$\mathrm{M1}-\mathrm{M2}$
$-$	$+$	$\mathrm{M2}$	$+$	$\mathrm{M2}-\mathrm{M1}$
$-$	$-$	$\mathrm{M1}$	$-$	$\mathrm{M1}+\mathrm{M2}$
$-$	$-$	$\mathrm{M2}$	$-$	$\mathrm{M1}+\mathrm{M2}$

- L'esponente del risultato viene preso dall'operando 2, in quanto si tratta di quello con esponente maggiore. Se tuttavia la somma tra le mantisse genera overflow e i due operandi sono concordi in segno, l'esponente viene incrementato di un'unità e alla mantissa, dopo essere stata shiftata a destra di una posizione, viene aggiunto un 1 iniziale.

- Se si ha la situazione di overflow descritta precedentemente e l'esponente è pari a 11111110, si ha un risultato che di per sé eccederebbe il valore massimo rappresentabile e viene pertanto generato un nuovo caso speciale (qualora non ne fosse già presente uno) avente come risultato un infinito con il segno appropriato. In tal caso, il segnale sum viene posto interamente a don't care.

Test-bench

SHIFT DI ZERO POSIZIONI, CON OVERFLOW E SENZA NUOVO CASO SPECIALE

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	01001011010010110110101001001011
	operand_2	01001011000101001010100101010110
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011110110000000010011101000010000

SHIFT DI ZERO POSIZIONI, SENZA OVERFLOW E SENZA NUOVO CASO SPECIALE

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	01000110010010110110101001001011
	operand_2	01001011000101001010100101010110
	mantissa_shift_amount	00001010
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011010010100110111000011000010001

SHIFT DI 23 POSIZIONI, SENZA OVERFLOW E SENZA NUOVO CASO SPECIALE

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	00111111110010110110101001001011
	operand_2	01001011000101001010100101010110
	mantissa_shift_amount	00010111
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011010010100101010010101111001

SHIFT DI 57 POSIZIONI, SENZA OVERFLOW E SENZA NUOVO CASO SPECIALE

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	00111111110010110110101001001011
	operand_2	01001011000101001010100101010110
	mantissa_shift_amount	00111001
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011010010100101010010101100000

$\boldsymbol{S1 = +, S2 = +, M1 > M2}$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	01001011010010110110101001001011
	operand_2	01001011000101001010100101010110
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011110110000000010011110100001000

$\boldsymbol{S1 = +, S2 = +, M1 < M2}$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	01001011000101001010100101010110
	operand_2	01001011010010110110101001001011
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011110110000000010011110100001000

$S1 = +, S2 = -, M1 > M2$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	01001011010010110110101001001011
	operand_2	11001011000101001010100101010110
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011000110110110000001111101010000

$S1 = +, S2 = -, M1 < M2$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	01001011000101001010100101010110
	operand_2	11001011010010110110101001001011
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	11001011000110110110000001111101010000

$S1 = -, S2 = +, M1 > M2$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	11001011010010110110101001001011
	operand_2	01001011000101001010100101010110
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	11001011000110110110000001111101010000

$S1 = -, S2 = +, M1 < M2$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	11001011000101001010100101010110
	operand_2	01001011010010110110101001001011
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	01001011000110110110000001111101010000

$S1 = -, S2 = -, M1 > M2$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	11001011010010110110101001001011
	operand_2	11001011000101001010100101010110
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	11001011110110000000010011110100001000

$S1 = -, S2 = -, M1 < M2$

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	11001011000101001010100101010110
	operand_2	11001011010010110110101001001011
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	0
	special_case_result_out	----------
	sum	11001011110110000000010011110100001000

NUOVO CASO SPECIALE

INPUT	special_case_flag_in	0
	special_case_result_in	----------
	operand_1	01111111000101001010100101010110
	operand_2	01111111010010110110101001001011
	mantissa_shift_amount	00000000
OUTPUT	special_case_flag_out	1
	special_case_result_out	01111111100000000000000000000000
	sum	----------

PROPAGAZIONE DI UN PRECEDENTE CASO SPECIALE

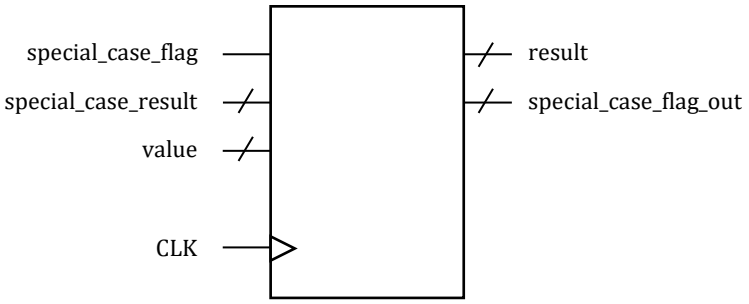
INPUT	special_case_flag_in	1
	special_case_result_in	11111111100000000000000000000000
	operand_1	----------
	operand_2	----------
	mantissa_shift_amount	--------
OUTPUT	special_case_flag_out	1
	special_case_result_out	01111111100000000000000000000000
	sum	----------

StageThree

Descrizione

È il terzo ed ultimo stadio della pipeline e ha come scopo la normalizzazione e l'arrotondamento del risultato ottenuto. Nel caso fosse stato riscontrato un caso speciale, viene restituito il relativo valore. Tuttavia, in modo analogo allo stage due, viene generato un ulteriore caso speciale corrispondente al valore di infinito qualora l'arrotondamento eccedesse il massimo valore rappresentabile.

Segnali



SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
CLK	in	1	Clock
special_case_flag	in	1	Indica se si è riscontrato un caso speciale
special_case_result	in	32	Risultato dell'eventuale caso speciale
value	in	37	Somma dei due operandi
result	out	32	Risultato finale
special_case_flag_out *	out	1	Indica se si è riscontrato un caso speciale in questo stage o nei precedenti

** segnale di debug*

Funzionamento

- Tramite il modulo ZeroCounter viene contato il numero di zeri antecedenti al primo 1 all'interno della mantissa estesa (comprensiva quindi del bit implicito e dei bit di guardia).
- Si calcola la differenza tra l'esponente e il numero di zeri, al fine di verificare se il valore è normalizzabile o meno. Se infatti la differenza ha segno negativo, l'esponente risulterebbe inferiore a -126 e non sarebbe pertanto rappresentabile nella versione a precisione singola; in tal caso, si avrà quindi un risultato non normalizzato, con esponente pari a -126 e mantissa shiftata il più possibile fino al raggiungimento del suddetto esponente.

  La seguente tabella riassume i vari casi:

  $\mathrm{exp} = \text{esponente}$ $\#\mathrm{z} = \text{n° di zeri iniziali della mantissa estesa}$

CONDIZIONE	SHIFT DELLA MANTISSA	NUOVO ESPONENTE (BASE 10)
$\mathrm{exp} = 0 \wedge \#\mathrm{z} = 0$	0 posizioni	1
$\mathrm{exp} = 0 \wedge \#\mathrm{z} \neq 0$	0 posizioni	0
$\mathrm{exp} \neq 0 \wedge \mathrm{exp} > \#\mathrm{z}$	$\#\mathrm{z}$ posizioni	$\mathrm{exp} - \#\mathrm{z}$
$\mathrm{exp} \neq 0 \wedge \mathrm{exp} \leq \#\mathrm{z}$	$\mathrm{exp} - 1$ posizioni	0

- Tramite il modulo Rounder si ricava l'arrotondamento dei bit di guardia e tale valore viene sommato alla mantissa, ora quindi costituita da 24 bit (bit implicito e mantissa vera e propria). In caso di overflow, si procede in maniera analoga al caso di overflow trattato nello stage due: la mantissa viene shiftata a destra di una posizione e l'esponente viene incrementato di un'unità. Se l'esponente risultante è composto da soli 1 si genera un nuovo caso speciale (qualora non ne fosse già presente uno) avente come risultato un infinito con il segno appropriato.
- Se si aveva un risultato speciale già dagli stage precedenti viene restituito quest'ultimo, ignorando quindi i risultati ottenuti negli altri punti di questo stage.

Test-bench

$\boldsymbol{exp = 0, \#z = 0}$

INPUT	special_case_flag	0
	special_case_result	----------
	value	00000000010110010001011010010011000000
OUTPUT	result	00000000101100100010110100100110
	special_case_flag_out	0

$exp = 0, \#z \neq 0$

INPUT	special_case_flag	0
	special_case_result	----------
	value	00000000000110010001011010010011000000
OUTPUT	result	00000000001100100010110100100110
	special_case_flag_out	0

$exp \neq 0, exp > \#z$

INPUT	special_case_flag	0
	special_case_result	----------
	value	01001011000110010001011010010011000000
OUTPUT	result	01001010010010001011010010011000
	special_case_flag_out	0

$exp \neq 0, exp = \#z$

INPUT	special_case_flag	0
	special_case_result	----------
	value	00000001000110010001011010010011000000
OUTPUT	result	00000000011001000101101001001100
	special_case_flag_out	0

$exp \neq 0, exp < \#z$

INPUT	special_case_flag	0
	special_case_result	----------
	value	00000001000011001000101101001001100000
OUTPUT	result	00000000001100100010110100100110
	special_case_flag_out	0

ARROTONDAMENTO PER DIFETTO

INPUT	special_case_flag	0
	special_case_result	----------
	value	01001011010110010001011010010011000000
OUTPUT	result	01001011001100100010110100100110
	special_case_flag_out	0

ARROTONDAMENTO PER ECCESSO SENZA OVERFLOW

INPUT	special_case_flag	0
	special_case_result	----------
	value	01001011010110010001011010010011101000
OUTPUT	result	01001011001100100010110100100111
	special_case_flag_out	0

ARROTONDAMENTO PER ECCESSO, CON OVERFLOW MA SENZA NUOVO CASO SPECIALE

INPUT	special_case_flag	0
	special_case_result	----------
	value	01001011011111111111111111111111111000
OUTPUT	result	01001011100000000000000000000000
	special_case_flag_out	0

ARROTONDAMENTO PER ECCESSO, CON OVERFLOW E NUOVO CASO SPECIALE

INPUT	special_case_flag	0
	special_case_result	----------
	value	01111111011111111111111111111111111000
OUTPUT	result	01111111100000000000000000000000
	special_case_flag_out	1

PROPAGAZIONE DI UN PRECEDENTE CASO SPECIALE

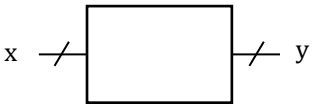
INPUT	special_case_flag	1
	special_case_result	11111111100000000000000000000000
	value	"----------";
OUTPUT	result	11111111100000000000000000000000
	special_case_flag_out	1

AbsoluteValue

Descrizione

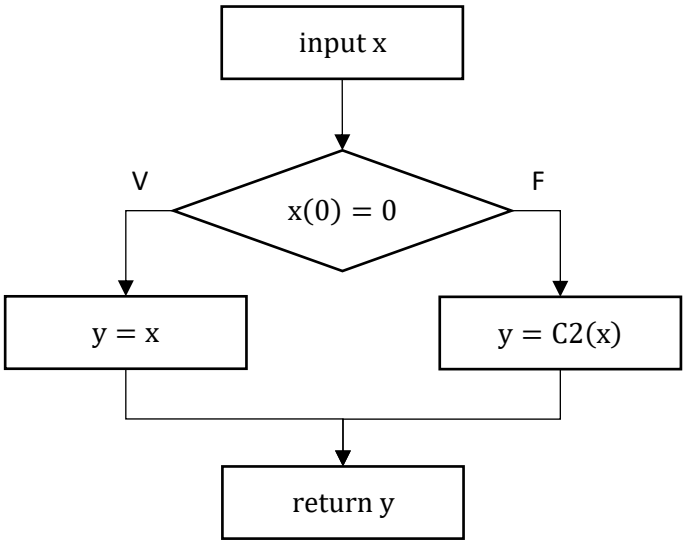
Restituisce la codifica binaria corrispondente al valore assoluto di un numero binario in complemento a due. Il risultato ha una lunghezza inferiore di uno rispetto all'input in quanto non comprensivo del segno.

Segnali



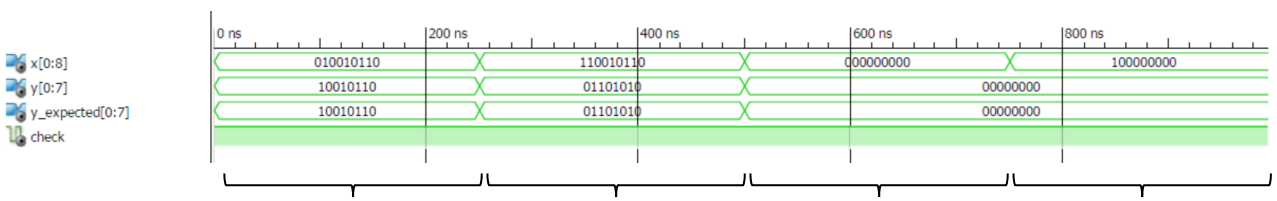
SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	n	Valore
y	out	$n-1$	Valore assoluto

Funzionamento



- Si calcola il complemento a due di x attraverso l'apposito modulo.
- Se il bit di segno (bit in posizione 0) di x è 0, viene restituito direttamente x in quanto il valore è già positivo.
- Se il bit di segno è 1, viene restituito il complemento a due calcolato in precedenza.

Test-bench



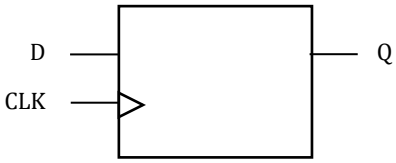
$|150| = 150$ $|-106| = 106$ $|0| = 0$ $|0| = 0$

DFF

Descrizione

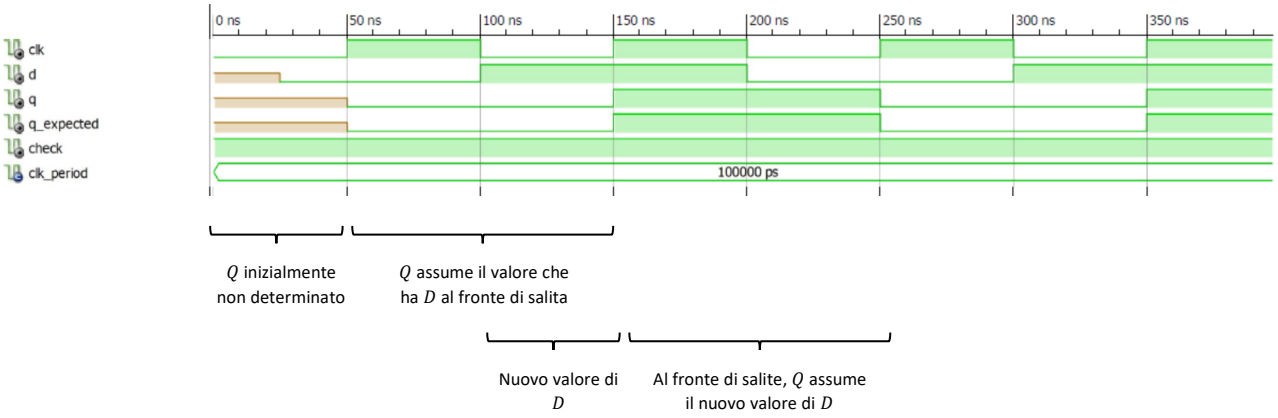
Implementazione di un flip flop di tipo D: dato un input D, questo viene riprodotto in output al successivo fronte di salita del clock. Viene utilizzato nel progetto per realizzare i registri della pipeline.

Segnali



SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
CLK	in	1	Clock
D	in	1	Segnale da salvare
Q	out	1	Segnale salvato precedentemente

Test-bench

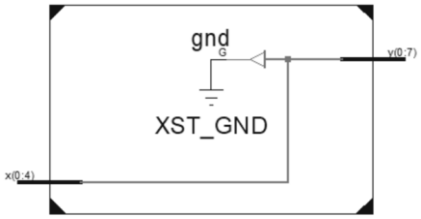


Extender (left / right)

Descrizione

Prende in input un vettore di lunghezza n e restituisce in output un vettore di lunghezza s, con $s > n$. Il risultato è ottenuto aggiungendo al vettore in input una quantità di zeri pari a $s - n$. Gli zeri vengono aggiunti in testa (LeftExtender) o in coda (RightExtender). I valori n e s sono generici e vanno pertanto specificati in fase di istanziazione del componente.

Segnali e struttura interna

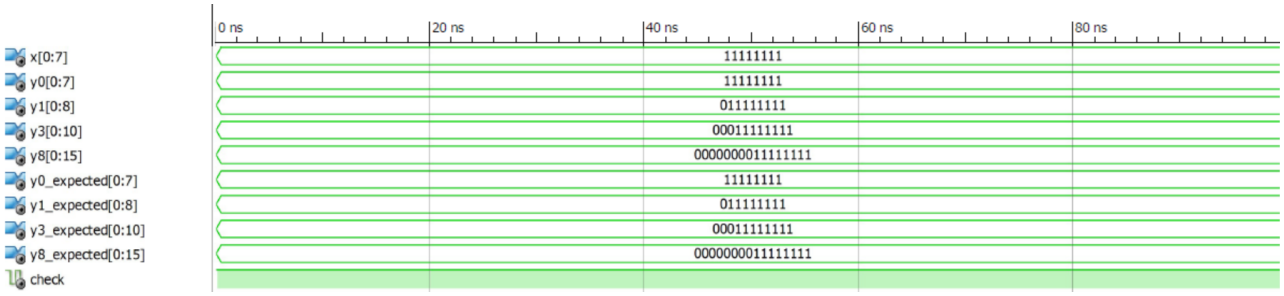


SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	n	Segnale da estendere
y	out	s	Segnale esteso

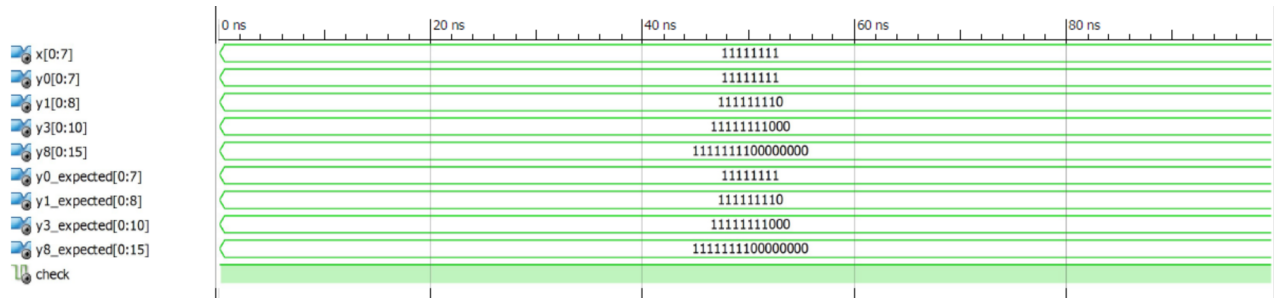
Test-bench

x: segnale da estendere
y_n: segnale esteso di n bit
y_n_expected: valore atteso di y_n

LeftExtender



RightExtender



MantissaExtender

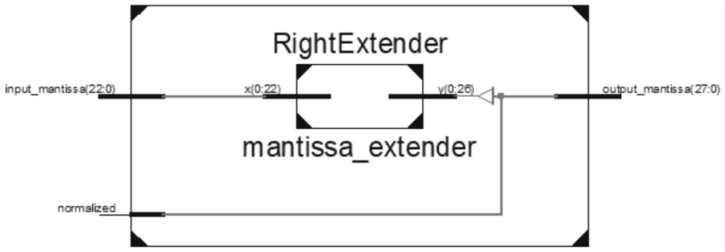
Descrizione

Effettua l'estensione della mantissa da 23 a 28 bit, aggiungendo il bit implicito iniziale (numero prima della virgola) e quattro bit di guardia finali per gli arrotondamenti.

Il bit implicito iniziale vale 1 se il numero è normalizzato o 0 se non normalizzato.

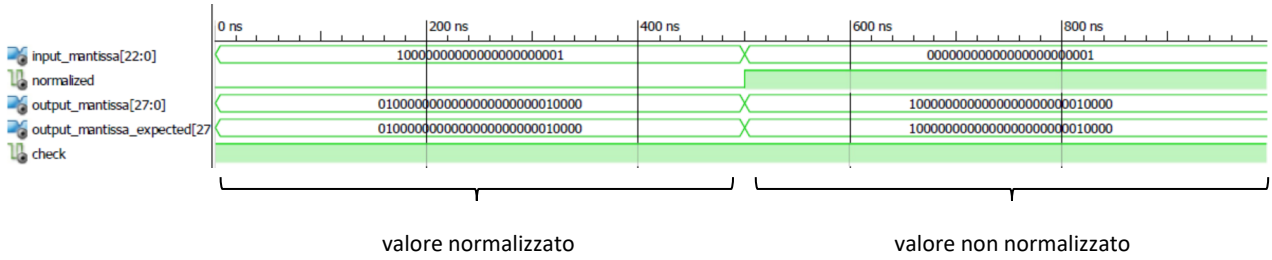
Segnali e struttura interna

SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
input_mantissa	in	23	Mantissa da 23 bit da estendere
normalized	in	1	Indica se il segnale in ingresso è normalizzato o meno
output_mantissa	out	28	Mantissa estesa a 28 bit



- La mantissa viene estesa a destra tramite l'apposito modulo; in questo modo vengono aggiunti i bit di guardia.
- Viene aggiunto in testa il segnale normalized, che corrisponde al bit implicito dell'operando.

Test-bench



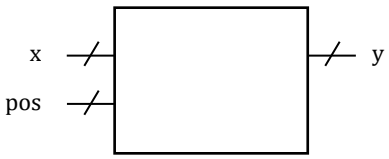
MantissaShifter (left / right)

Descrizione

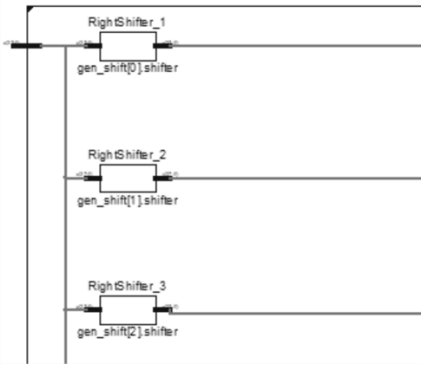
Sono previste due implementazioni del modulo. La prima prende in ingresso un vettore di 28 bit ed esegue lo shift a destra del numero di posizioni indicato tramite l'apposito segnale. La seconda implementazione è analoga alla prima ma esegue lo shift a sinistra.

La grandezza della mantissa è impostata a 28 invece di 23 in quanto comprensiva del bit implicito (precedente la virgola) e dei quattro bit di guardia utilizzati per gli arrotondamenti.

Segnali e struttura interna

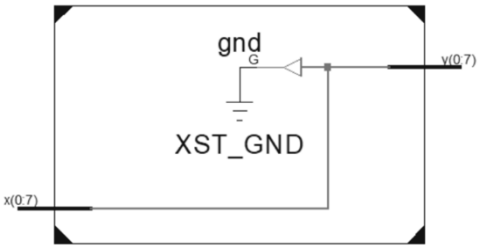


SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	28	Mantissa da shiftare
pos	in	8	Numero di posizioni
y	out	28	Mantissa shiftata



- I singoli shifter calcolano in modo parallelo tutte le possibili mantisse shiftate.
- Attraverso un multiplexer viene restituita la mantissa corrispondente al numero di spostamenti indicato in ingresso.

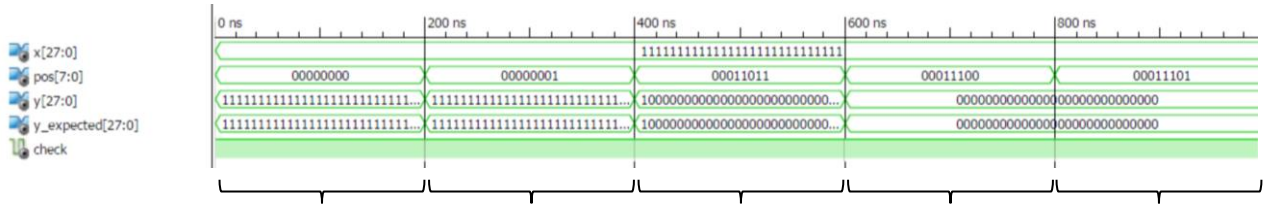
LeftShifter / RightShifter



Il singolo shifter trasla il vettore in ingresso, di grandezza n, di una quantità s maggiore o uguale a 0. I valori n e s sono generici e vanno pertanto specificati in fase di istanziazione del componente.

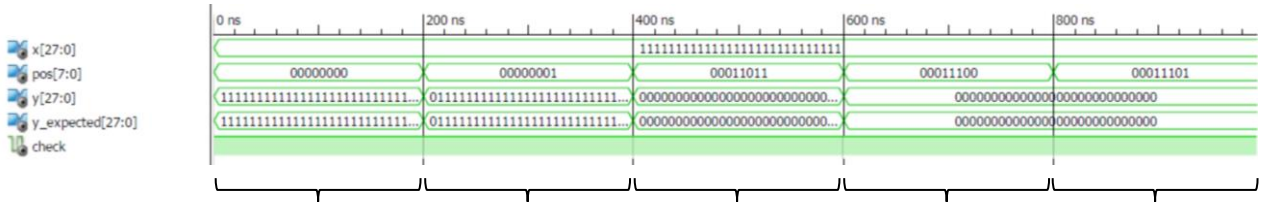
Test-bench

MantissaLeftShifter



Shift di 0 posizioni | Shift di 1 posizione | Shift di 27 posizioni | Shift di 28 posizioni | Shift di 29 posizioni

MantissaRightShifter



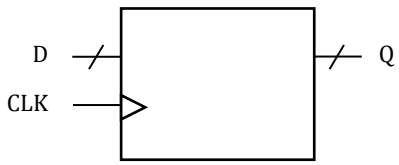
Shift di 0 posizioni | Shift di 1 posizione | Shift di 27 posizioni | Shift di 28 posizioni | Shift di 29 posizioni

Registers

Descrizione

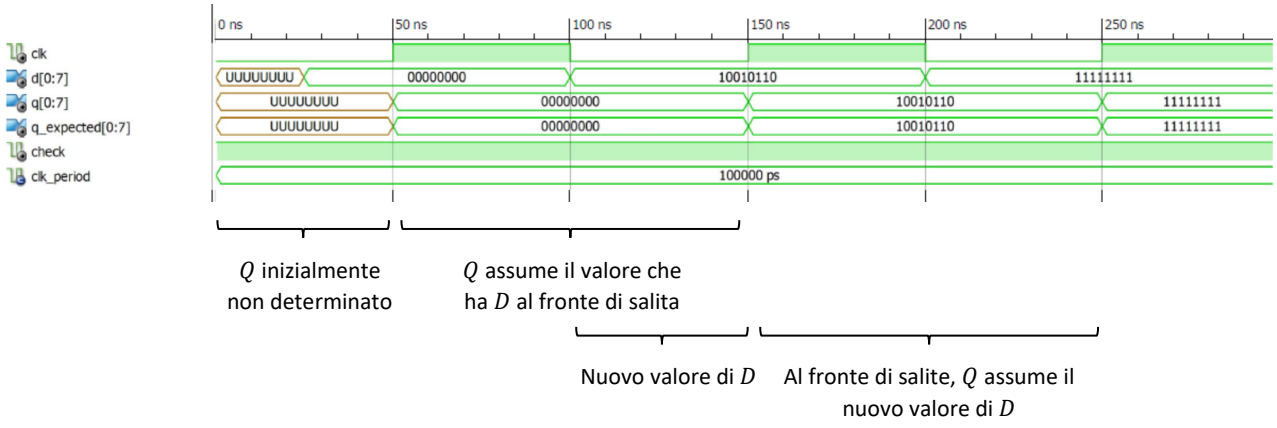
Parallelizzazione di più moduli DFF. Realizza di fatto i registri presenti tra uno stadio e l'altro della pipeline.

Segnali



SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
CLK	in	1	Clock
D	in	n	Segnale da salvare
Q	out	n	Segnale salvato precedentemente

Test-bench

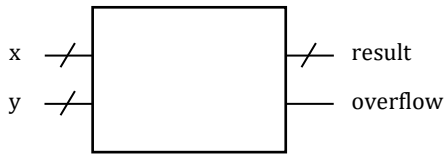


RippleCarryAdder

Descrizione

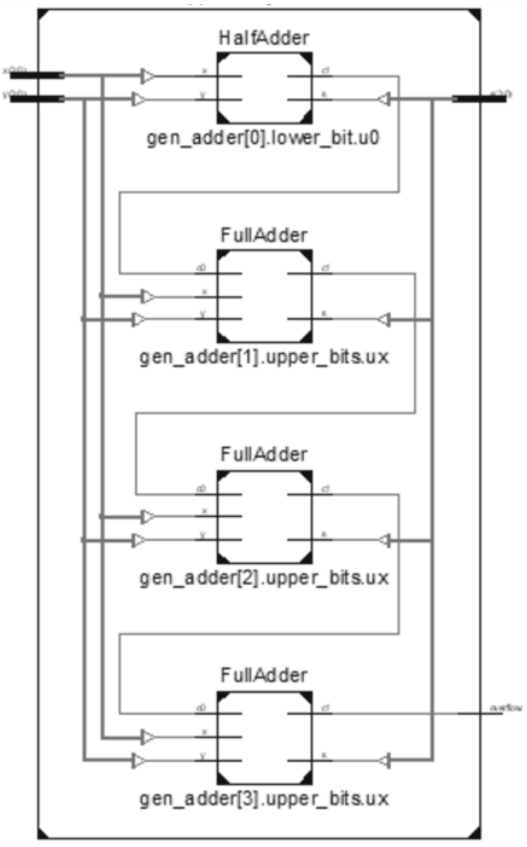
Effettua la somma algebrica tra due vettori di lunghezza n, riportando in output il risultato e l'eventuale overflow. Il valore n è generico e va pertanto specificato in fase di istanziazione del componente.

Segnali



SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	n	Primo operando
y	in	n	Secondo operando
result	out	n	Risultato
overflow	out	1	Overflow

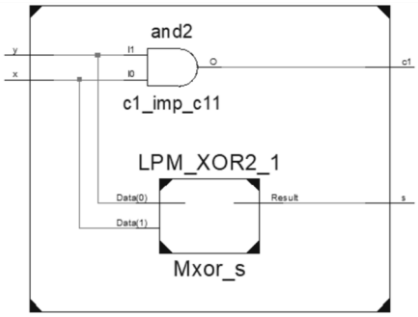
Struttura interna



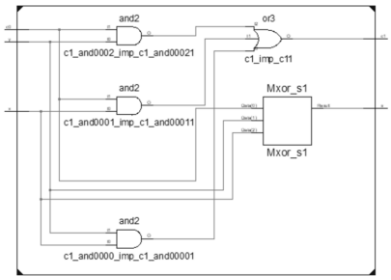
La realizzazione è effettuata attraverso una cascata di full adder, con l'eccezione dell'half adder di partenza in quanto non è previsto un riporto iniziale.

La struttura di half adder e full adder è riportata di seguito:

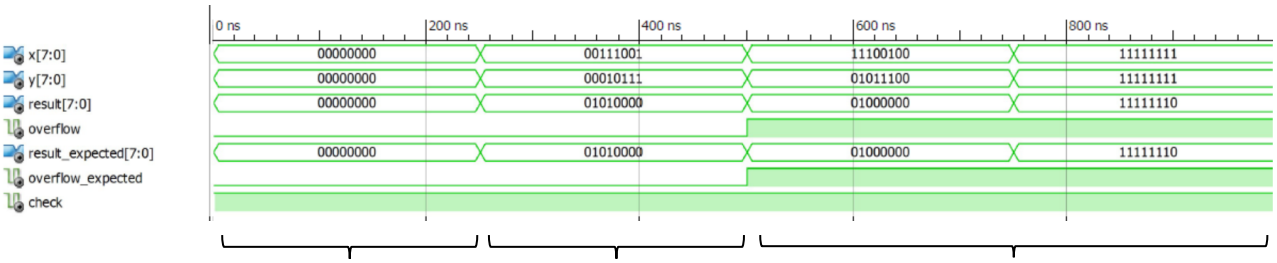
HalfAdder



FullAdder



Test-bench



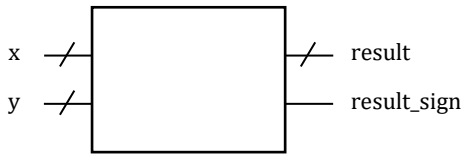
$0 + 0 = 0$ $57 + 23 = 80$ overflow

RippleCarrySubtractor

Descrizione

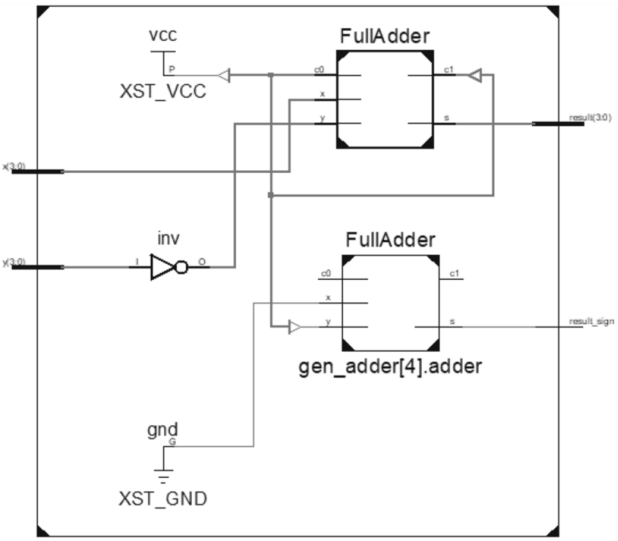
Effettua la differenza tra due vettori di lunghezza n, entrambi intesi come valori positivi. Oltre al risultato, riporta in output anche il segno. Il valore n è generico e va pertanto specificato in fase di istanziazione del componente.

Segnali



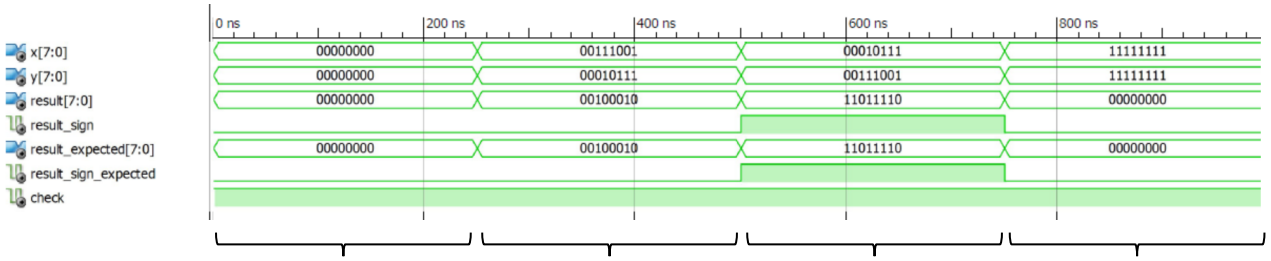
SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	n	Primo operando
y	in	n	Secondo operando
result	out	n	Risultato
result_sign	out	1	Segno del risultato

Funzionamento



La realizzazione è effettuata attraverso una cascata di full adder, in modo da poter specificare un riporto iniziale pari a uno. Il primo operando non viene modificato, mentre il secondo viene complementato a uno al fine di avere il suo complemento a due grazie al riporto sopra citato.

Test-bench



$0 - 0 = 0$ $57 - 23 = 34$ $57 - 23 = -34$ $255 - 255 = 0$

Rounder

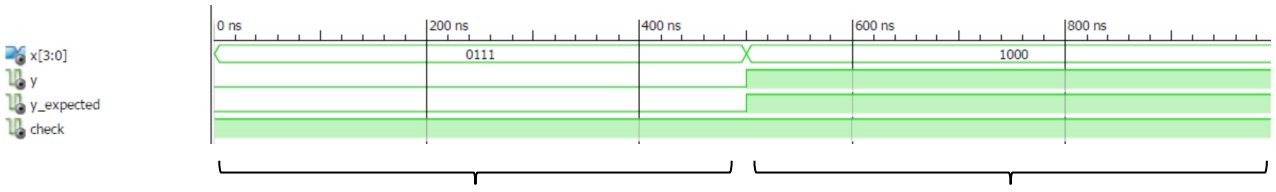
Descrizione

Prende in ingresso i quattro bit di guardia della mantissa ed effettua l'arrotondamento alla cifra più significativa, da sommare successivamente alla mantissa.

Segnali

SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	4	Bit di guardia
y	out	1	Arrotondamento

Test-bench



arrotondamento per difetto

arrotondamento per eccesso

SpecialCaseAssignation

Descrizione

Analizza gli operandi coinvolti nella somma e determina se si tratta di un caso particolare non normalmente gestibile attraverso il procedimento standard (come la somma tra infiniti) o il cui risultato è già noto (come la somma tra zero e un qualsiasi altro valore).

Segnali

SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
a	in	32	Primo operando
b	in	32	Secondo operando
enable	out	1	Indica se si è riscontrato un caso speciale
result	out	32	Risultato della somma, se si tratta di un caso speciale

Codifica dei valori

VALORE	SEGNO	ESPONENTE	MANTISSA
0	–	tutti 0	tutti 0
non normalizzato	–	tutti 0	non tutti 0
$+\infty$	0	tutti 1	tutti 0
$-\infty$	1	tutti 1	tutti 0
QNaN	–	tutti 1	1 & non tutti 0
SNaN	–	tutti 1	0 & non tutti 0

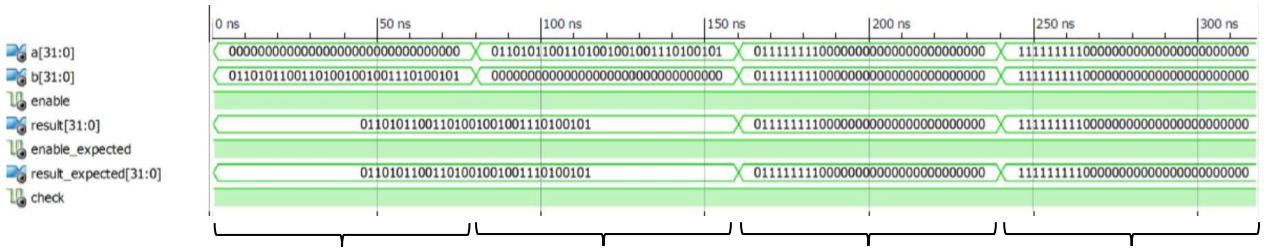
Operazioni con zeri

Qualsiasi operando sommato a uno zero genera come risultato l'operando stesso. Se invece vengono sommati due valori esattamente opposti, viene restituito zero come risultato.

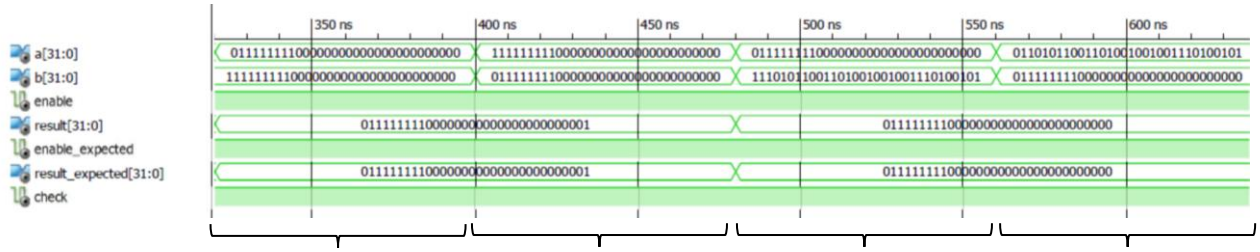
Operazioni con infiniti e NaN

OPERAZIONE	RISULTATO
qualsiasi valore (non NaN) $\pm\infty$	$\pm\infty$
$+\infty + \infty$	$+\infty$
$-\infty - \infty$	$-\infty$
$+\infty - \infty$	NaN
$-\infty + \infty$	NaN
qualsiasi valore + NaN	NaN

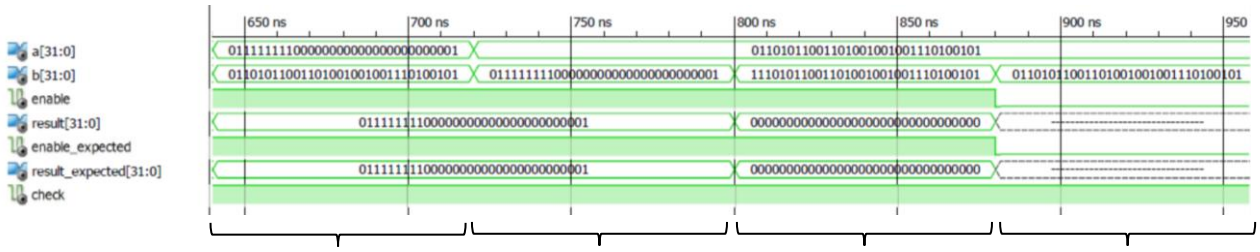
Test-bench



$0 + valore = valore$ $valore + 0 = valore$ $+\infty + \infty = +\infty$ $-\infty - \infty = -\infty$



$+\infty - \infty = NaN$ $-\infty + \infty = NaN$ $+\infty + valore\ (non\ Nan) = +\infty$ $valore\ (non\ Nan) + \infty = +\infty$



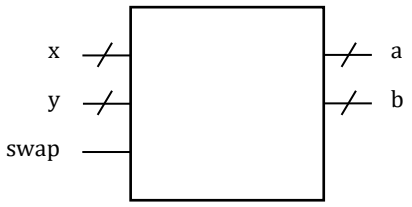
$NaN + valore = NaN$ $valore + NaN = NaN$ *valori opposti* *nessun caso speciale*

Swap

Descrizione

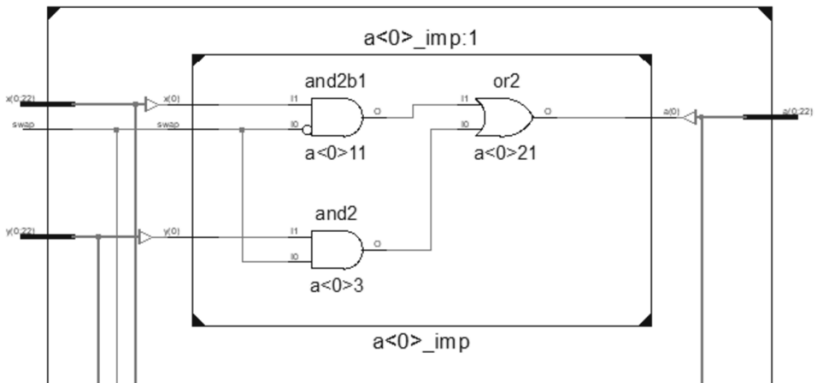
Effettua lo scambio di due vettori in ingresso in base a un segnale di controllo (i valori vengono scambiati se il segnale è pari a 1). Il valore n è generico e va pertanto specificato in fase di istanziazione del componente.

Segnali



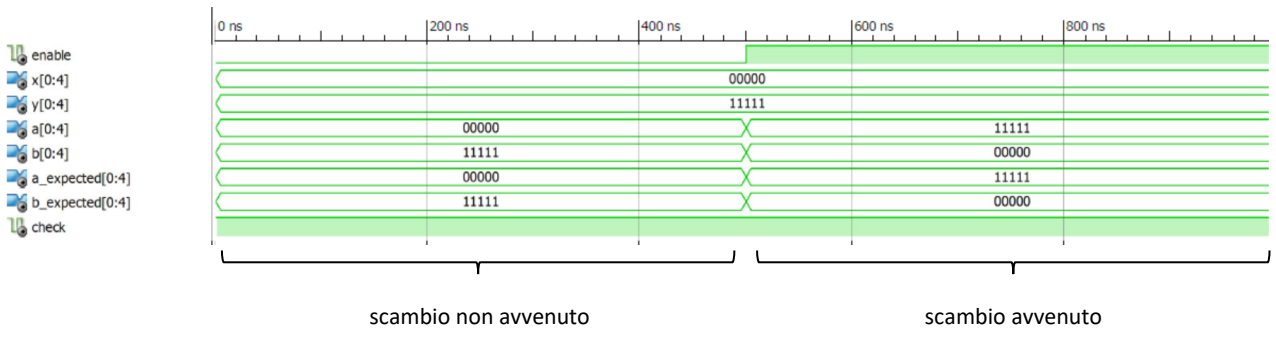
SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	n	Primo vettore
y	in	n	Secondo vettore
swap	in	1	Segnale di controllo
a	out	n	Primo vettore ($a = x \Leftrightarrow swap = 0$)
b	out	n	Secondo vettore ($b = y \Leftrightarrow swap = 0$)

Struttura interna



Ogni bit dei due segnali in ingresso viene fatto passare per un multiplexer che, a seconda del valore di swap, restituisce il bit opportuno.

Test-bench



TwoComplement

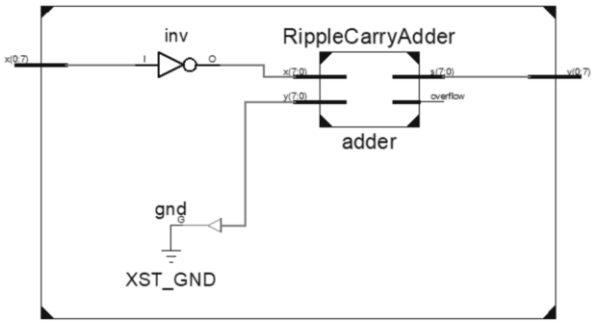
Descrizione

Restituisce il complemento a due di un vettore di n bit. Il valore n è generico e va pertanto specificato in fase di istanziazione del componente.

Segnali

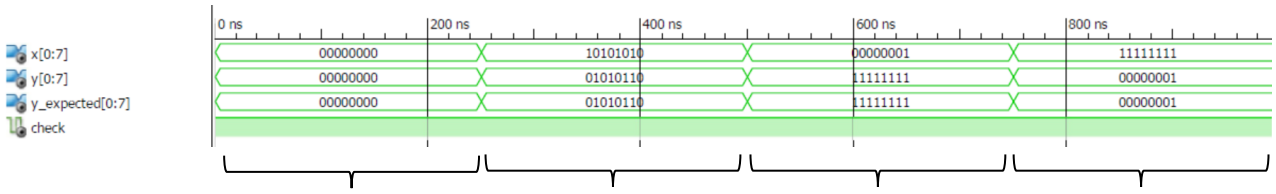
SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
x	in	n	Valore da complementare a due
y	out	n	Complemento a due

Struttura interna e funzionamento



- Il valore viene complementato a uno
- Il complemento a uno viene posto in ingresso a una serie di full adder (volta a realizzare un sommatore con riporto iniziale) assieme al valore zero e a un riporto iniziale pari a uno.

Test-bench



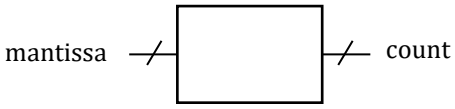
$0_{10} = 00000000_{C2}$ $170_{10} = 01010110_{C2}$ $1_{10} = 11111111_{C2}$ $255_{10} = 00000001_{C2}$

ZeroCounter

Descrizione

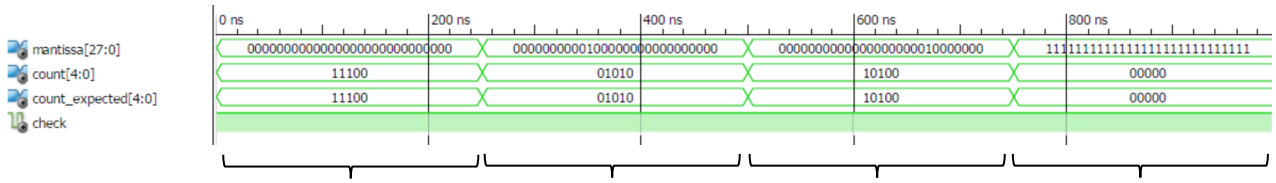
Restituisce il numero di zeri presenti prima del primo 1 della mantissa. Come per i moduli di shift, la grandezza della mantissa è impostata a 28 invece di 23 in quanto comprensiva del bit implicito e dei bit utilizzati per gli arrotondamenti.

Segnali



SEGNALE	TIPOLOGIA	LUNGHEZZA	DESCRIZIONE
mantissa	in	28	Mantissa
count	out	5	Numero di zeri

Test-bench



28 zeri iniziali | 10 zeri iniziali | 20 zeri iniziali | 0 zeri iniziali