Sommatore ieee 754

Implementazione di un sommatore floating point single precision

Politecnico di Milano, sede di Cremona

Beccari Giulio, Cuzzocrea Luca

Indice

[1 Introduzione 3](#_Toc19045764)

[1.1 Descrizione del progetto 3](#_Toc19045765)

[1.2 Formato IEEE 754 3](#_Toc19045766)

[1.3 Somma 3](#_Toc19045767)

[2 Moduli 5](#_Toc19045768)

[2.1 SpecialCasesCheck 5](#_Toc19045769)

[2.1.1 NaNCheck 6](#_Toc19045770)

[2.1.2 ZeroCheck 7](#_Toc19045771)

[2.1.3 TypeCheck 8](#_Toc19045772)

[2.1.4 EqualCheck 8](#_Toc19045773)

[2.1.5 Test bench 9](#_Toc19045774)

[2.2 PrepareForShift 10](#_Toc19045775)

[2.2.1 Comparator 11](#_Toc19045776)

[2.2.2 AddSub 11](#_Toc19045777)

[2.2.3 Test bench 12](#_Toc19045778)

[2.3 Swap 13](#_Toc19045779)

[2.3.1 Test bench 14](#_Toc19045780)

[2.4 TwoComplement 15](#_Toc19045781)

[2.4.1 Test bench 15](#_Toc19045782)

[2.5 SumDataAdapter 16](#_Toc19045783)

[2.5.1 ShiftRight48 17](#_Toc19045784)

[2.5.2 Test bench 17](#_Toc19045785)

[2.6 OperationCheck 18](#_Toc19045786)

[2.6.1 Test bench 19](#_Toc19045787)

[2.7 CarryLookAhead 20](#_Toc19045788)

[2.7.1 Test bench 21](#_Toc19045789)

[2.8 Normalizer 22](#_Toc19045790)

[2.8.1 ZeroCounter 22](#_Toc19045791)

[2.8.2 ShiftLeft48 23](#_Toc19045792)

[2.8.3 Test bench 24](#_Toc19045793)

[2.9 OutputSelector 25](#_Toc19045794)

[2.9.1 Test bench 25](#_Toc19045795)

[3 Pipeline 26](#_Toc19045796)

[3.1 Analisi dei ritardi combinatori 26](#_Toc19045797)

[3.2 RTL Stage 27](#_Toc19045798)

[3.3 Struttura della pipeline 28](#_Toc19045799)

[3.4 Test architettura completa 29](#_Toc19045800)

# Introduzione

## Descrizione del progetto

Lo scopo di questo progetto è la realizzazione in linguaggio VHDL di un sommatore floating point secondo lo standard IEEE 754. Il sommatore è in grado di trattare operandi normalizzati, non normalizzati ed operandi speciali (NaN e ∞).

## Formato IEEE 754

Lo standard IEEE 754 prevede che gli operandi (single precision) abbiano la seguente struttura:

|  |  |  |
| --- | --- | --- |
| S | G | T |

* S: segno, 1 bit;
* G: esponente, 8 bit;
* T: mantissa, 23 bit.

A seconda dei valori di G e T, possiamo identificare i seguenti casi:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S | G | T | Caso | Valore |
| - | 11111111 | 00000000000000000000000 | **Infinito** | ± ∞ |
| - | 11111111 | Diverso da tutti 0 | **Not a number** | NaN |
| - | 00000000 | Qualunque valore | **Numeri piccoli** | (-1)S × 2-127  × (0 + 2-23 × T) |
| - | Altro | Qualunque valore | **Normale** | (-1)S × 2G-127 × (1 + 2-23 × T) |

## Somma

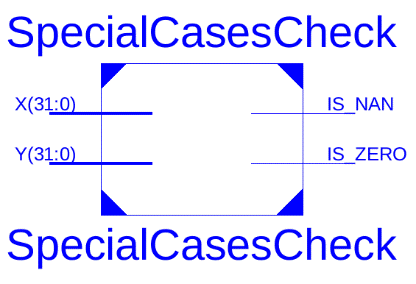
Il procedimento della somma tra 2 numeri floating point single precision è il seguente:

1. **Controllo dei casi particolari**: Ci sono alcuni casi particolari che non seguono il flusso normale della somma. Questi sono gli input che hanno come risultato: NaN, ± ∞ oppure 0.
2. **Controllo del maggiore e scambio degli operandi**: L’operando con valore assoluto maggiore deve essere messo a sinistra. In questo modo, le operazioni seguenti possono essere semplificate.
3. **Calcolo del segno finale e dell’operazione**: La somma/sottrazione viene eseguita sul valore assoluto degli operandi.  
   Il segno finale sarà il segno dell’operatore più grande.  
   L’operazione da fare sui valori assoluti sarà:
   * una somma, se gli operandi hanno segno concorde;
   * una sottrazione, se gli operatori hanno segno discorde.
4. **Preparazione delle mantisse**: Le mantisse hanno il primo bit rappresentato implicitamente nell’esponente. Questo va aggiunto esplicitamente al numero, portandolo su 24 bit. Il valore è 0 se G è tutto a 0, 1 nel caso normale.  
   Le mantisse vanno poi rappresentate con la stessa potenza di 2 (quella dell’esponente più grande). Per fare questo, la mantissa del numero più piccolo viene a destra della differenza tra gli esponenti.  
   Le mantisse devono anche essere estese a 48 bit per evitare perdite di informazione.
5. **Somma/Sottrazione mantisse**: Le mantisse estese possono essere sommate o sottratte usando gli input generati nelle operazioni precedenti. Il risultato è a 48 bit + 1 bit di overflow.
6. **Normalizzazione del risultato**: A questo punto abbiamo il segno del risultato, il suo esponente (che equivale all’esponente dell’operando più grande) e la mantissa su 48 bit (+ overflow).  
   Il segno è quello calcolato precedentemente.  
   L’esponente va diminuito in base al numero di zeri iniziali della mantissa.  
   La mantissa va shiftata in modo da rimuovere gli zeri iniziali e il primo 1. (Attenzione: Se l’esponente finale è il minore possibile non vanno rimossi tutti gli zeri, ma sono quanti ne bastano per portare l’esponente a –127). Poi si scartano i bit meno significativi e si mantengono solo i primi 23 bit.  
   In questo modo si ottiene il risultato della somma in formato IEEE754 a 32 bit.

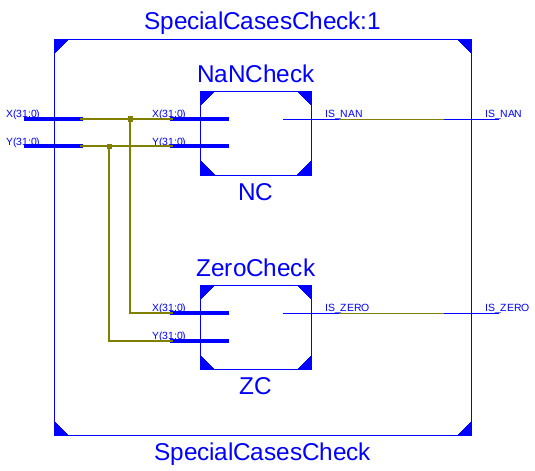
# Moduli

## SpecialCasesCheck

Il modulo SpecialCasesCheck riceve in input i due numeri da sommare/sottrarre, X e Y, e si occupa di verificare, attraverso i moduli NaNCheck e ZeroCheck, se questi danno origine ad un risultato speciale restituendo in uscita due flag IS\_NAN e IS\_ZERO.

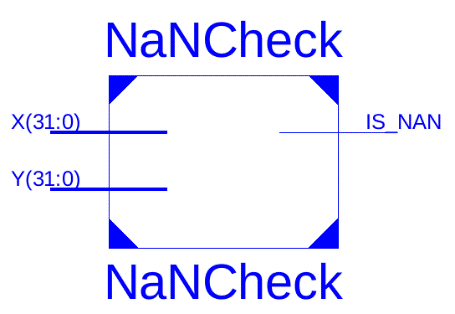


|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X | in | 32 bit | Primo operando della somma/sottrazione. |
| Y | in | 32 bit | Secondo operando della somma/sottrazione. |
| IS\_NAN | out | 1 bit | Flag: vale ‘1’ se X e Y generano NaN. |
| IS\_ZERO | out | 1 bit | Flag: vale ‘1’ se X e Y generano ‘0’. |

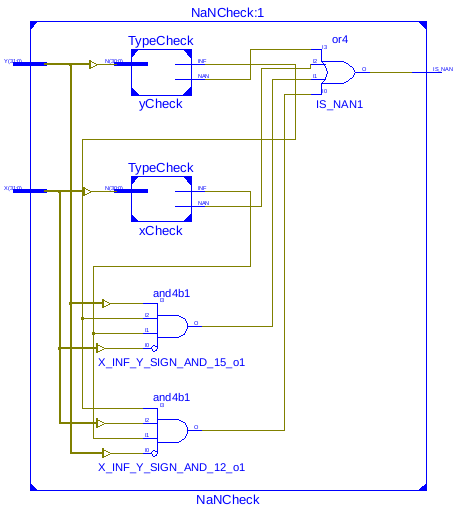


### NaNCheck

NaNCheck viene utilizzato dal modulo SpecialCasesCheck per verificare se gli operandi della somma/sottrazione danno origine al valore NaN. Per effettuare questo controllo si serve di TypeCheck.

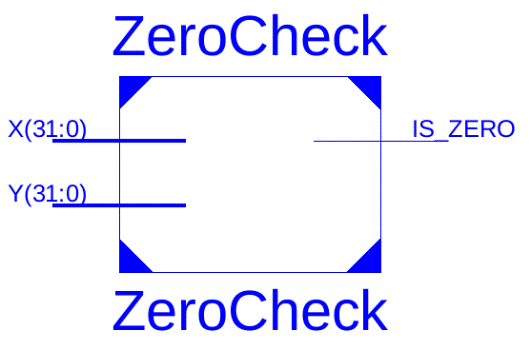


|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X | in | 32 bit | Primo operando della somma/sottrazione. |
| Y | in | 32 bit | Secondo operando della somma/sottrazione. |
| IS\_NAN | out | 1 bit | Flag: vale ‘1’ se X e Y generano NaN. |

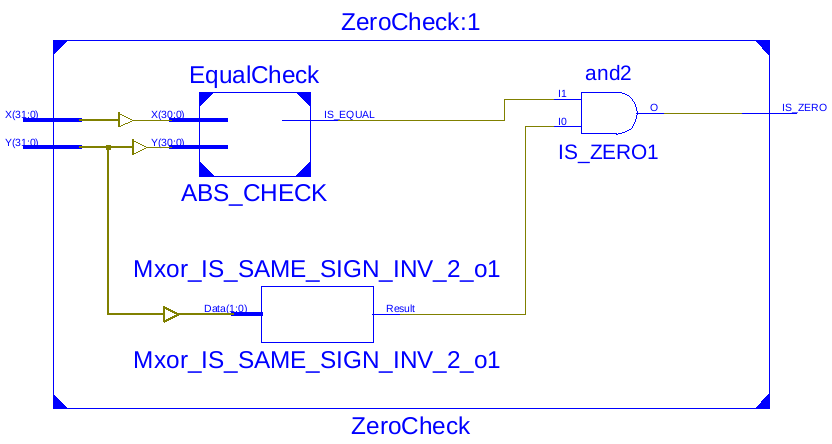


### ZeroCheck

ZeroCheck è anch’esso impiegato dal modulo SpecialCasesCheck e verifica se il risultato della somma è ‘0’.

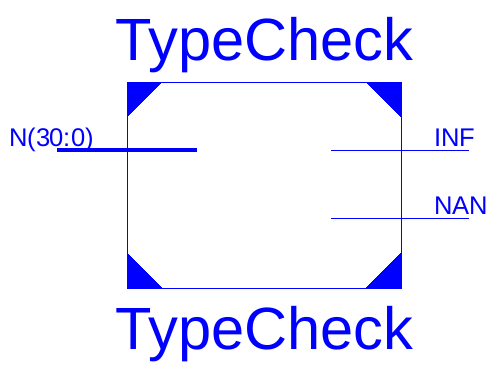


|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X | in | 32 bit | Primo operando della somma/sottrazione. |
| Y | in | 32 bit | Secondo operando della somma/sottrazione. |
| IS\_ZERO | out | 1 bit | Flag: vale ‘1’ se X e Y generano ‘0’. |



### TypeCheck

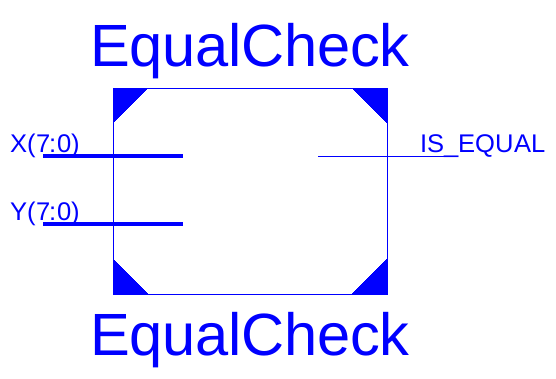
TypeCheck prende in input un numero N in 32 bit, controlla esponente e mantissa per verificare se N è NaN o ∞. Restituisce due flag per indicare se il valore di N è uno di questi.



|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| N | in | 32 bit | Numero da controllare. |
| NAN | out | 1 bit | Flag: vale ‘1’ se il numero è NaN. |
| INF | out | 1 bit | Flag: vale ‘1’ se il numero è ∞. |

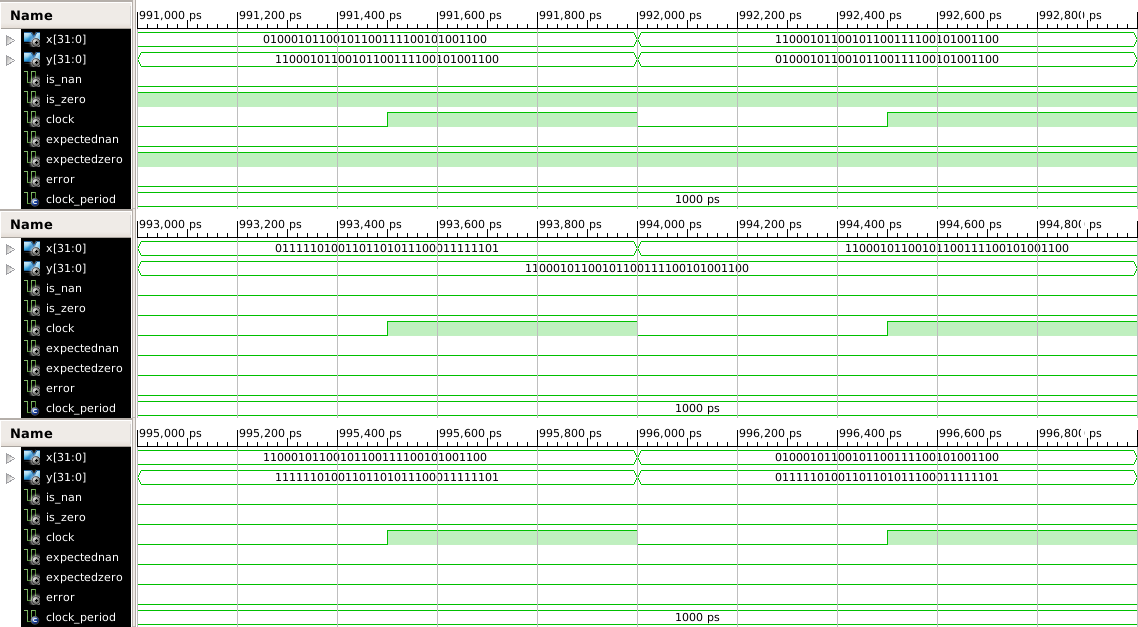
### EqualCheck

Il modulo EqualCheck si occupa di verificare se i valori assoluti dei due operandi, X e Y, sono uguali a meno del segno.



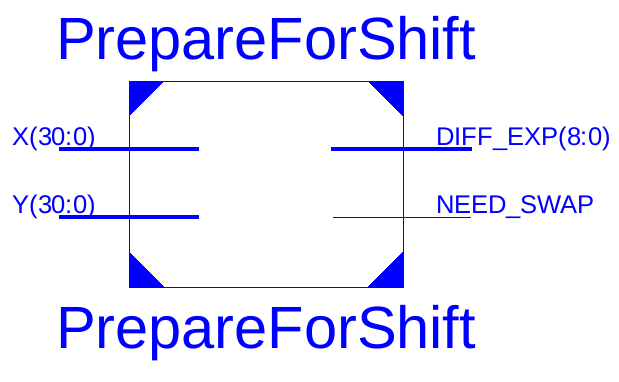
|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X | in | 31 bit | Valore assoluto del primo operando. |
| Y | in | 31 bit | Valore assoluto del secondo operando. |
| IS\_EQUAL | out | 1 bit | Flag: vale ‘1’ se X = Y. |

### Test bench

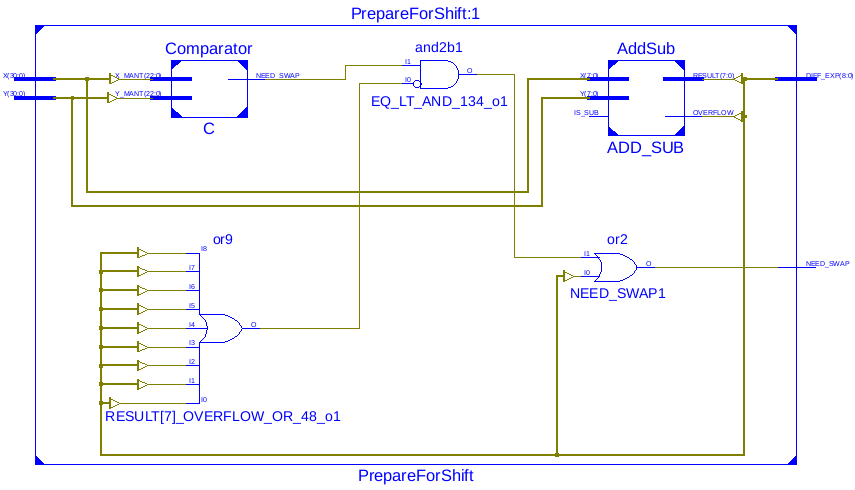


## PrepareForShift

PrepareForshift ha il compito di calcolare la differenza tra gli esponenti di X e Y, necessaria per fare lo shift della mantissa dell’addendo inferiore. Inoltre verifica se Y è maggiore di X e in tal caso abilita il segnale SW per fare lo swap dei due numeri (per convenzione X è l’addendo maggiore).

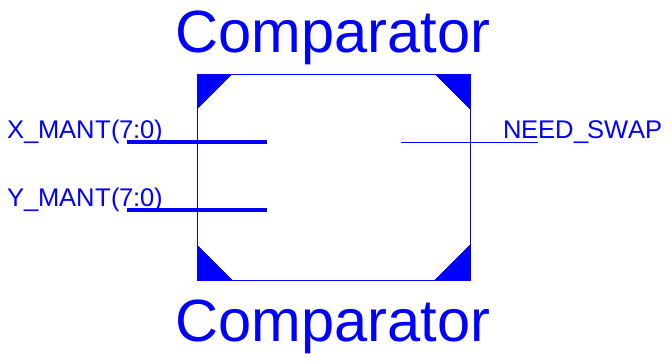


|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X | in | 31 bit | Valore assoluto del primo operando. |
| Y | in | 31 bit | Valore assoluto del secondo operando. |
| DIFF\_EXP | out | 9 bit | Differenza tra esponenti di X e Y in complemento a 2. |
| NEED\_SWAP | out | 1 bit | Flag: vale ‘1’ se Y > X. |



### Comparator

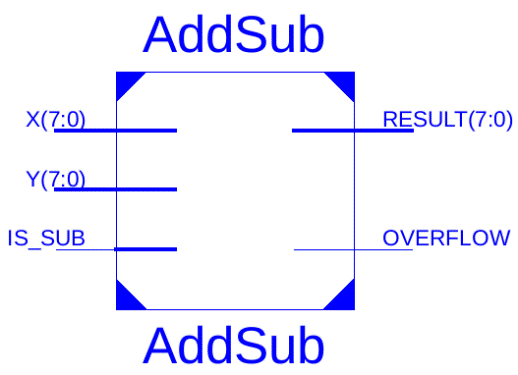
Comparator confronta le mantisse di X e Y per trovare la maggiore delle due in previsione dello swap. La logica nel modulo PrepareForShift si occuperà poi di controllare anche gli esponenti per capire se lo swap dovrà essere effettuato oppure no.



|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X\_MANT | in | generic | Mantissa di X. |
| Y\_MANT | in | generic | Mantissa di Y. |
| NEED\_SWAP | out | 1 bit | Flag: vale ‘1’ se Y\_MANT > X\_MANT. |

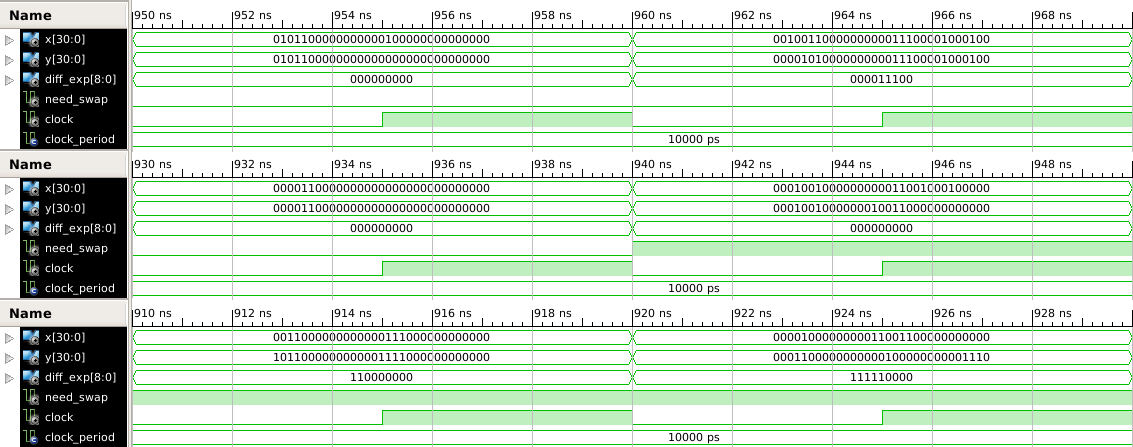
### AddSub

AddSub effettua la somma/sottrazione degli ingressi X e Y. Il flag IS\_SUB segnala al modulo l’operazione da effettuare. In output troviamo il risultato (RESULT) più l’eventuale overflow (OVERFLOW) del sommatore/sottrattore.



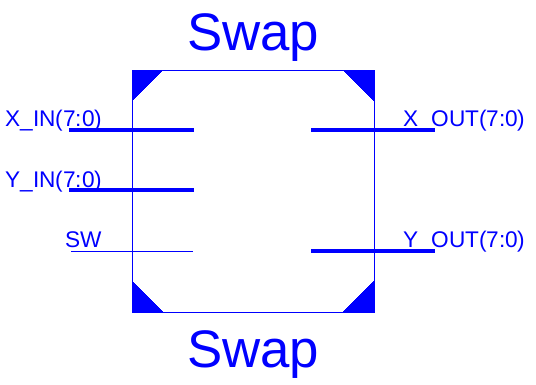
|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X | in | generic | Primo operando. |
| Y | in | generic | Secondo operando. |
| IS\_SUB | in | 1 bit | Flag: vale ‘1’ se l’operazione da effettuare è la differenza. |
| RESULT | out | generic | Risultato dell’operazione. |
| OVERFLOW | out | 1 bit | Bit di overflow dell’operazione. |

### Test bench

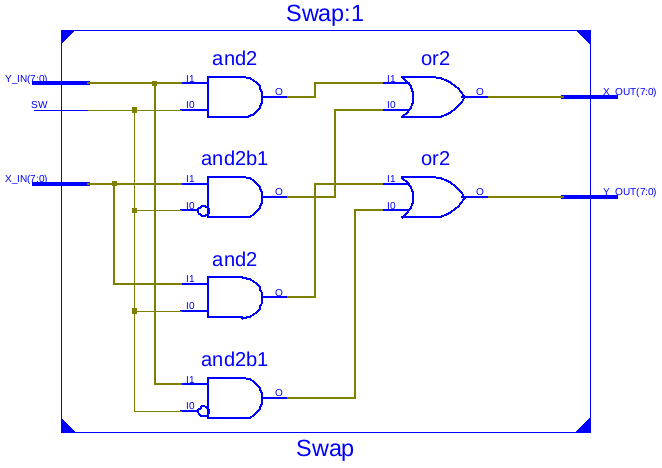


## Swap

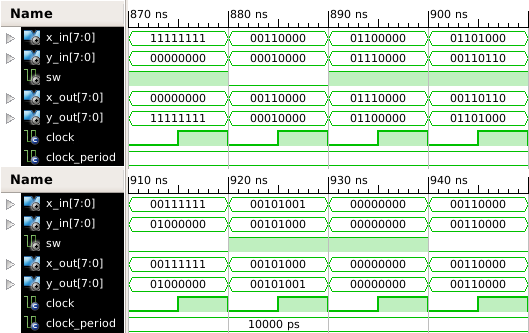
Il modulo Swap riceve in ingresso gli addendi X e Y per effettuare, se necessario, lo swap tra i due.



|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X\_IN | in | generic | Primo operando. |
| Y\_IN | in | generic | Secondo operando. |
| SW | in | 1 bit | Flag: vale ‘1’ se deve essere effettuato lo swap. |
| X\_OUT | out | generic | Nuovo primo operando. |
| Y\_OUT | out | generic | Nuovo secondo operando. |

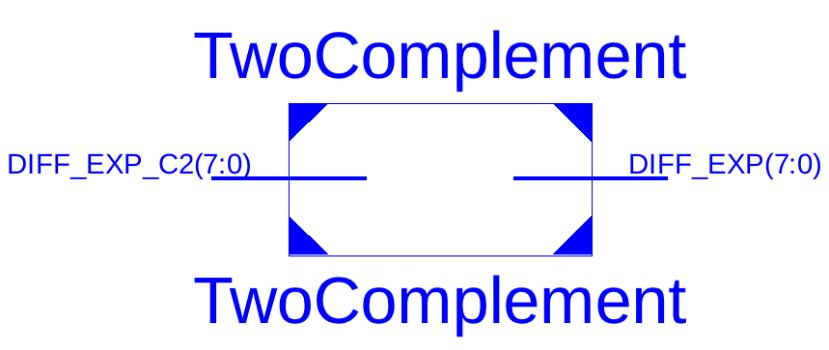


### Test bench



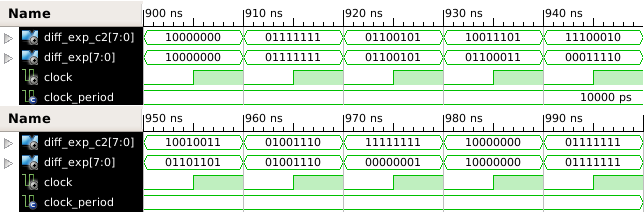
## TwoComplement

TwoComplement riceve in ingresso un numero in complemento a 2 e ne restituisce il valore assoluto in codifica binaria naturale.



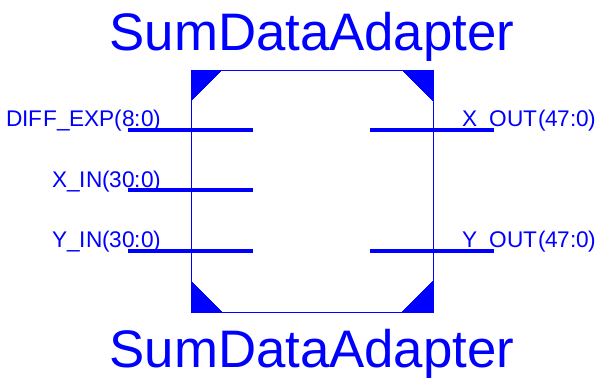
|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| DIFF\_EXP\_C2 | in | generic | Numero in complemento a 2. |
| DIFF\_EXP | out | generic | Valore assoluto del numero in input, in codifica binaria naturale. |

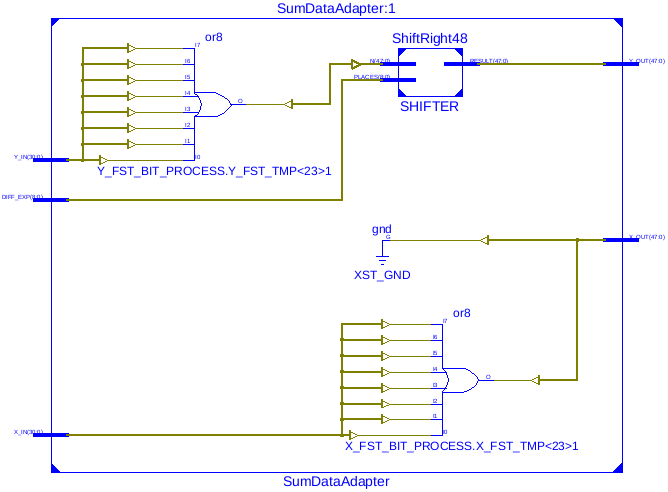
### Test bench



## SumDataAdapter

SumDataAdapter si occupa di preparare gli operandi X e Y per la somma/sottrazione. Ricevuti in ingresso X e Y dal modulo Swap, calcola, esaminando gli esponenti, il bit che precede la mantissa (0 o 1) attraverso una OR a 8 bit. Successivamente estende le mantisse da 23 a 47 bit e aggiunge in testa il bit calcolato precedentemente. Infine, attraverso il modulo ShiftRight48, effettua lo shift della mantissa di Y di un numero di posizioni pari a DIFF\_EXP (ricevuto da TwoComplement).

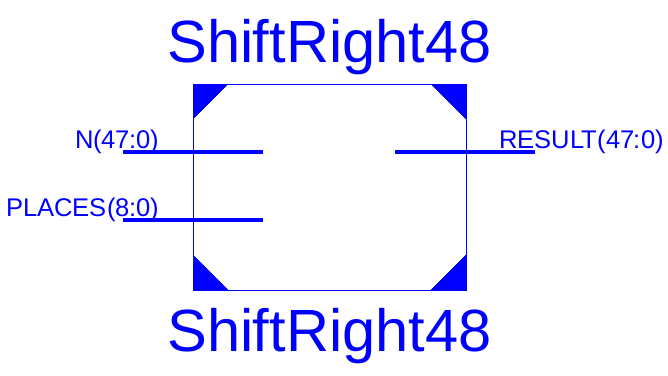




|  |  |  |  |
| --- | --- | --- | --- |
| Seganle | Tipo | Lunghezza | Descrizione |
| X\_IN | in | 31 bit | Valore assoluto del primo operando. |
| Y\_IN | in | 31 bit | Valore assoluto del secondo operando. |
| DIFF\_EXP | in | 9 bit | Numero di posizioni per lo shift della mantissa di Y. |
| X\_OUT | out | 48 bit | Mantissa estesa di X. |
| Y\_OUT | out | 48 bit | Mantissa estesa di Y. |

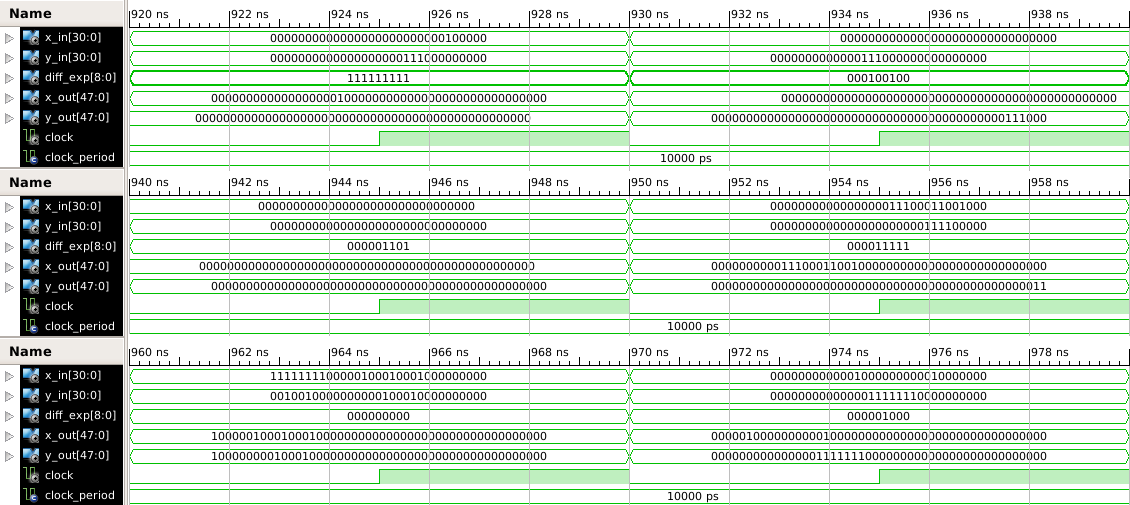
### ShiftRight48

ShiftRight48 riceve in ingresso un numero a 48 bit e ne effettua lo shift di un numero di posizioni pari a PLACES.



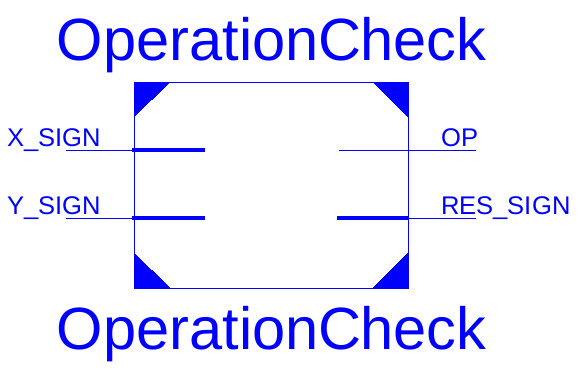
|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| N | in | 48 bit | Numero di cui fare lo shift. |
| PLACES | in | 9 bit | Numero di posizioni per lo shift di N. |
| RESULT | out | 48 bit | Risultato dello shift. |

### Test bench

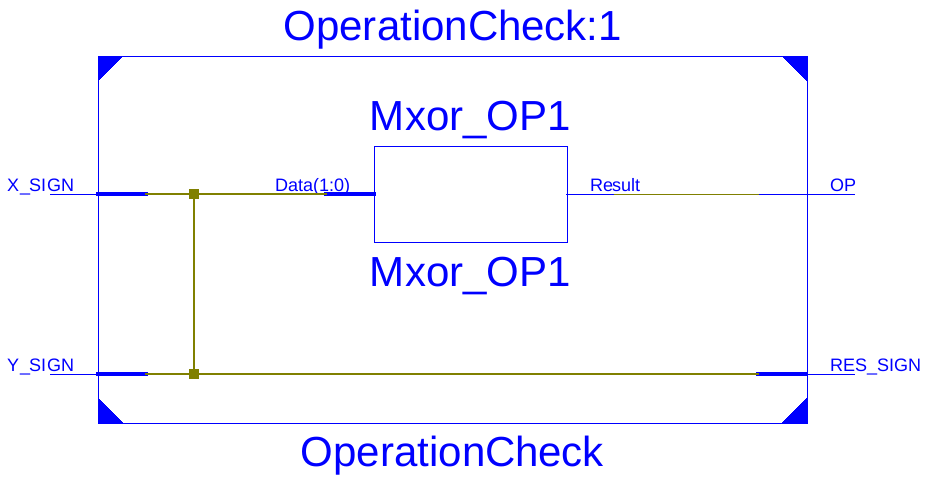


## OperationCheck

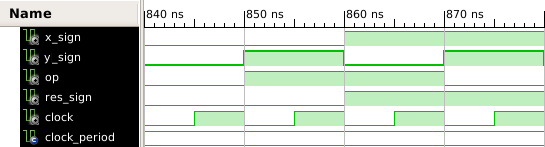
OperationCheck calcola, a partire dai segni dei due operandi, il segno del risultato finale e l’operazione da svolgere (somma o sottrazione).



|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X\_SIGN | in | 1 bit | Segno del primo operando. |
| Y\_SIGN | in | 1 bit | Segno del secondo operando. |
| OP | out | 1 bit | Flag: vale ‘1’ se deve essere effettuata la differenza. |
| RES\_SIGN | out | 1 bit | Segno del risultato finale. |

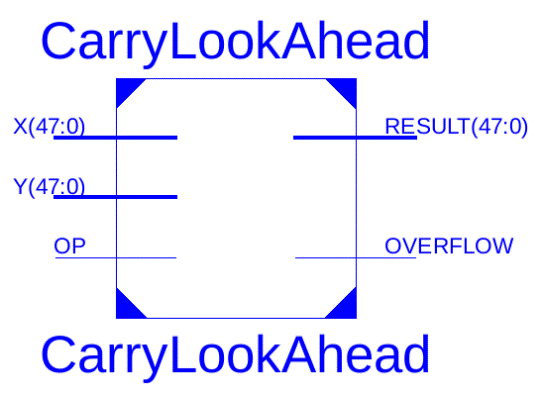


### Test bench

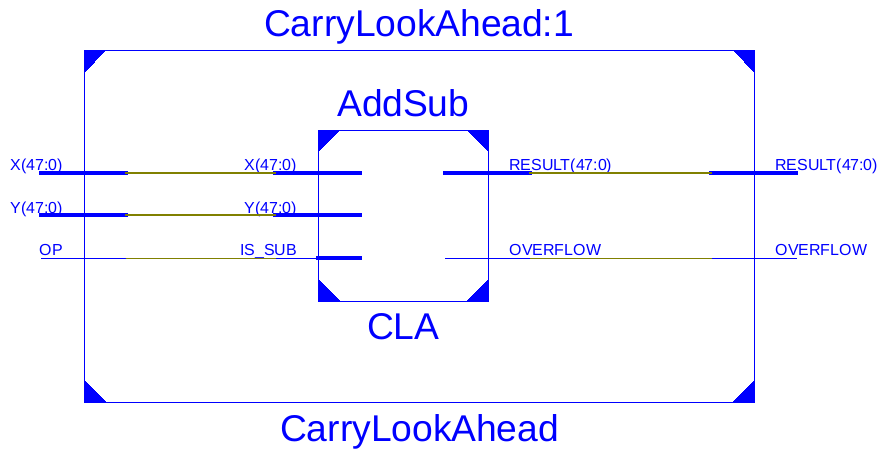


## CarryLookAhead

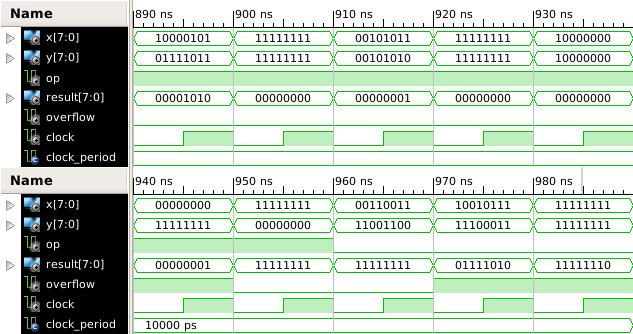
CarryLookAhead riceve le mantisse estese X e Y da SumDataAdapter e in base al valore del flag OP, ricevuto da OperationCheck, ne fa la somma o la differenza.



|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| X | in | 48 bit | Mantissa estesa del primo operando. |
| Y | in | 48 bit | Mantissa estesa del secondo operando. |
| OP | in | 1 bit | Flag: vale ‘1’ se deve essere effettuata la differenza. |
| RESULT | out | 48 bit | Risultato della somma/differenza. |
| OVERFLOW | out | 1 bit | Bit di overflow della somma/differenza. |



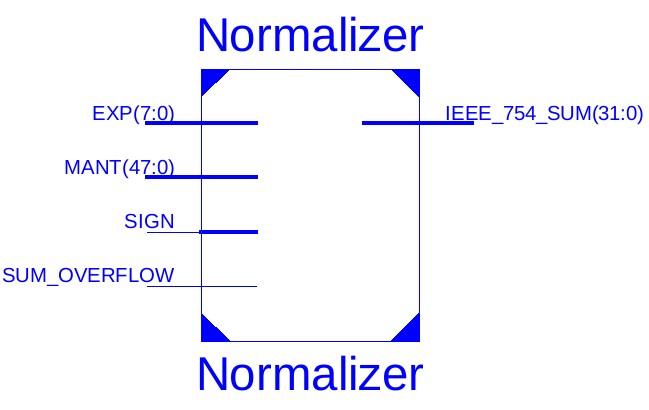
### Test bench



## Normalizer

Normalizer è il modulo che si occupa di generare il numero IEEE754 a 32 bit che rappresenta la somma degli operandi.

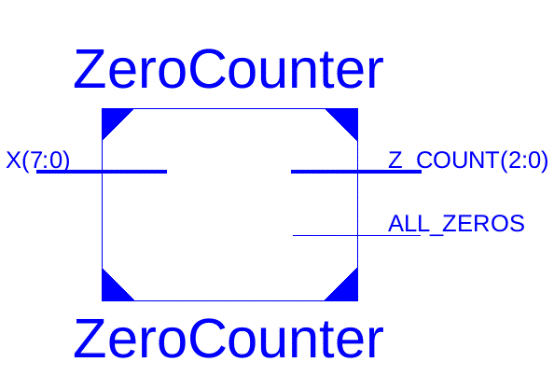
Si occupa di adattare l’esponente, aumentandolo o diminuendolo. Poi normalizza il risultato della somma/sottrazione (MANT) ricevuto da CarryLookAhead rimuovendo gli zeri iniziali.



|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | Lunghezza | Descrizione |
| SIGN | in | 1 bit | Segno del risultato della somma/sottrazione. |
| EXP | in | 8 bit | Esponente dell’ingresso più grande. |
| MANT | in | 48 bit | Mantissa del risultato della somma/sottrazione. |
| SUM\_OVERFLOW | in | 1 bit | Bit di overflow della somma delle mantisse. |
| IEEE\_754\_SUM | out | 32 bit | Somma dei due operandi X e Y in formato IEEE 754. |

### ZeroCounter

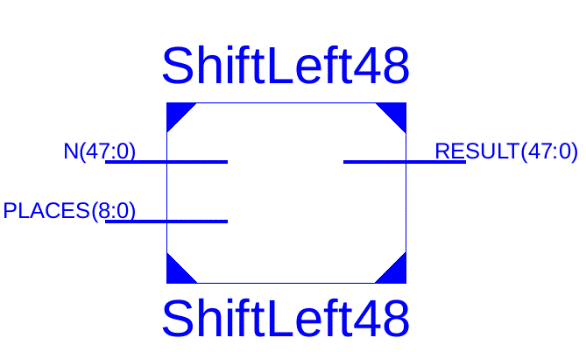
ZeroCounter conta il numero di ‘0’ che precedono il primo ‘1’ presenti nell’input X.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Segnale | Tipo | | Lunghezza | Descrizione |
| X | in | generic | | Numero su cui effettuare il conteggio. |
| Z\_COUNT | out | ⌈ log2 (generic) ⌉ | | Numero di ‘0’ che precedono il primo ‘1’. |
| ALL\_ZEROS | out | 1 bit | | Flag: vale ‘1’ se X non contiene ‘1’. |

### ShiftLeft48

ShiftLeft48 effettua lo shift a sinistra di N, di un numero di posizioni indicato da PLACES.

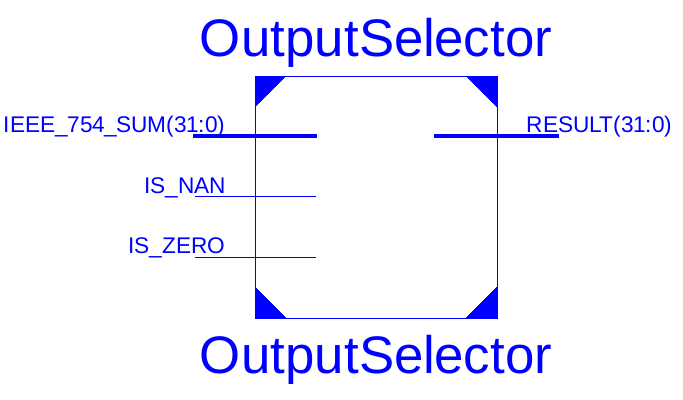


|  |  |  |  |
| --- | --- | --- | --- |
| segnale | tipo | lunghezza | descrizione |
| n | in | 48 bit | Numero di cui fare lo shift. |
| places | in | 9 bit | Numero di posizioni per lo shift di N. |
| result | out | 48 bit | Numero dopo lo shift. |

### Test bench

## OutputSelector

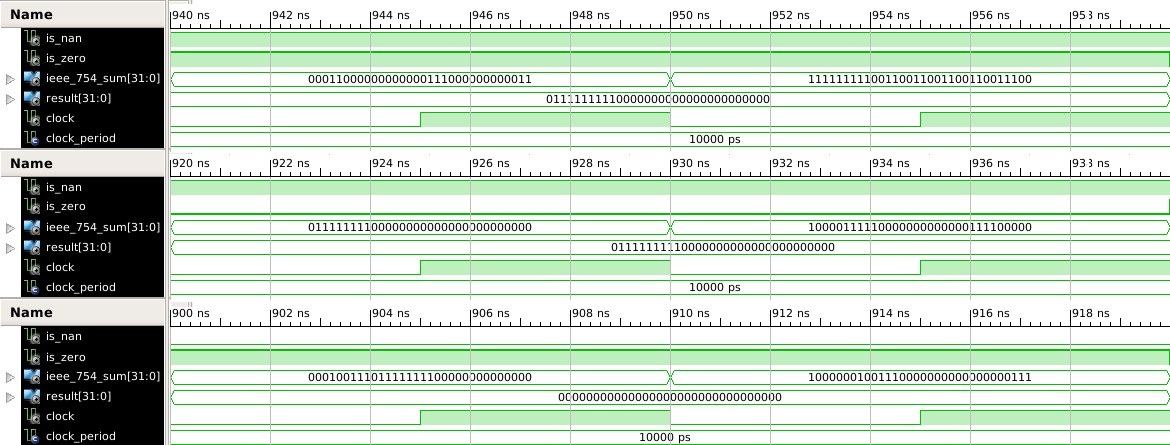
OutputSelector riceve da SpecialCasesCheck i flag IS\_NAN, IS\_ZERO e da Normalizer il risultato della somma/sottrazione normalizzato (IEEE\_754\_SUM). Quindi seleziona il risultato finale:



|  |  |
| --- | --- |
| Risultato | Valore |
| NaN | 0 11111111 10000000000000000000000 |
| Zero | 0 00000000 00000000000000000000000 |
| Somma calcolata | IEEE\_754\_SUM |

|  |  |  |  |
| --- | --- | --- | --- |
| Segnale | Tipo | lunghezza | descrizione |
| Is\_nan | in | 1 bit | Flag: vale ‘1’ se il risultato deve essere NaN. |
| is\_zero | in | 1 bit | Flag: vale ‘1’ se il risultato deve essere zero. |
| ieee\_754\_sum | in | 32 bit | Risultato della somma/sottrazione. |
| result | out | 32 bit | Risultato finale. |

### Test bench



# Pipeline

## Analisi dei ritardi combinatori

I report di sintesi dei diversi moduli hanno evidenziato i seguenti ritardi:

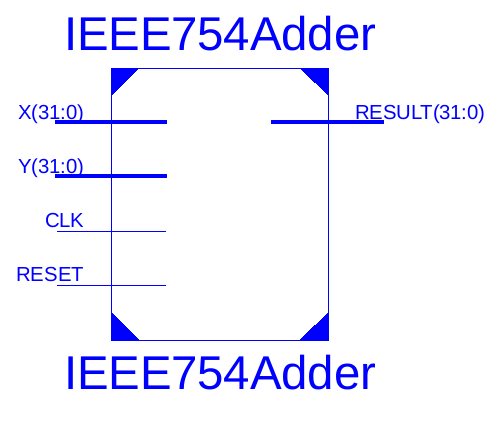
|  |  |
| --- | --- |
| Modulo | ritardo |
| Special Cases Check | 7,57 ns |
| prepare for shift | 11,894 ns |
| swap | 5,810 ns |
| two complement | 6,406 ns |
| operation check | 5,296 ns |
| sum data adapter | 9,801 ns |
| carry look ahead | 27,242 ns |
| normalizer | 28,233 ns |
| output selector | 5,971 ns |

Dall’analisi della tabella sopra riportata emerge che il ritardo più consistente è pari a 28,233 ns, quindi si è deciso di strutturare la pipeline in tre stage così composti:

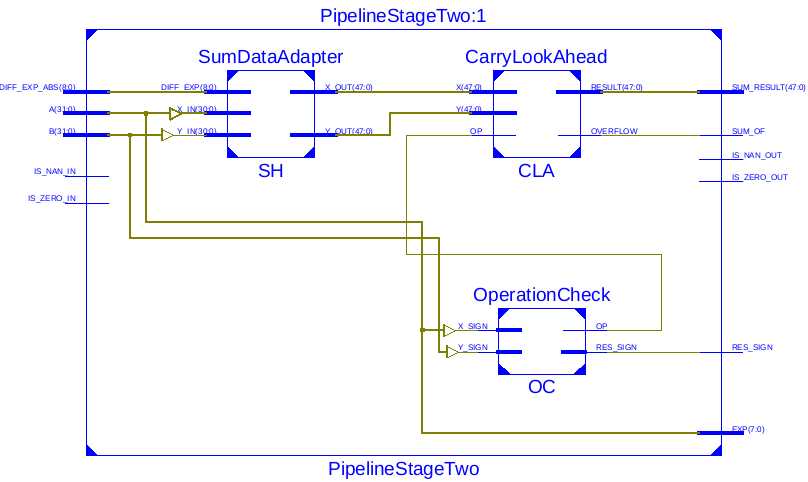
|  |  |  |
| --- | --- | --- |
| Stage | Moduli | ritardo |
| pipeline stage one | * SpecialCasesCheck; * OperationCheck; * PrepareForShift; * Swap; * TwoComplement. | 14,095 ns |
| Pipeline stage two | * SumDataAdapter; * CarryLookAhead. | 24,421 ns |
| pipeline stage three | * Normalizer; * OutputSelector. | 27,942 ns |

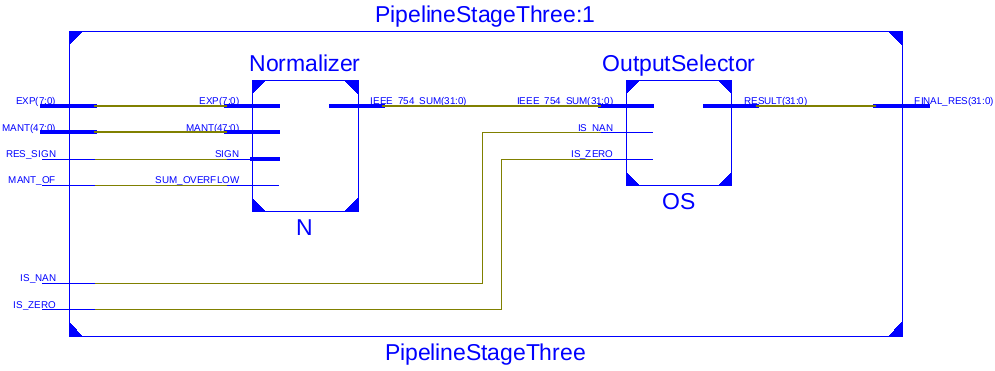
In risposta all’ottimizzazione nella sintesi dei tre stage sopra descritti, emerge che il ritardo maggiore è pari 27,942 ns (per giunta inferiore a quello del solo Normalizer!), mentre il ritardo introdotto dai Flip-Flop di tipo D risulta essere pari a 3,597 ns.

Si è quindi scelto un periodo di clock pari a 40 ns.



## RTL Stage





## Struttura della pipeline

La pipeline risulta infine così strutturata:

Pipeline

Stage

One

Pipeline

Stage

Two

Pipeline

Stage

Three

RESET

CLK

X

Y

RESULT

## Test architettura completa

