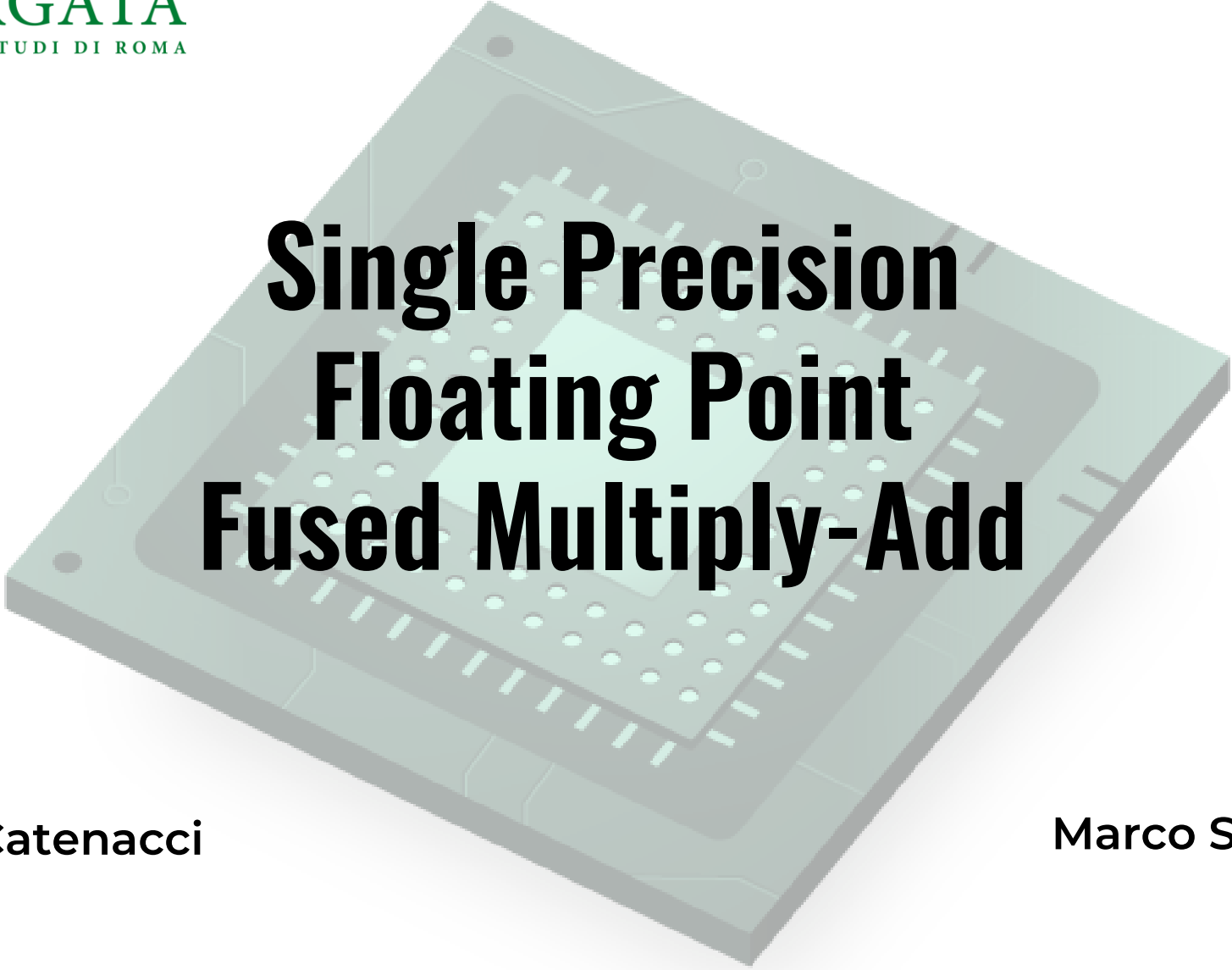




**TOR VERGATA**  
UNIVERSITÀ DEGLI STUDI DI ROMA

A 3D perspective illustration of a square microchip with a grid of pins and internal circuitry, rendered in a light green and grey color scheme.

# **Single Precision Floating Point Fused Multiply-Add**

Simone Catenacci

Marco Salvatori

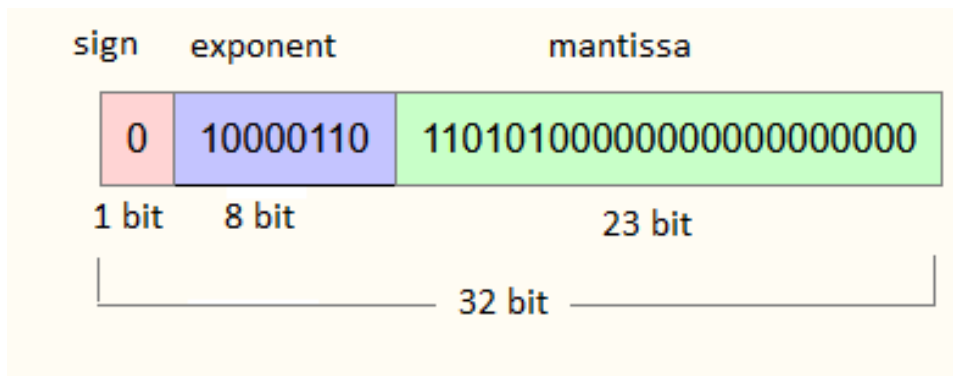
- ❑ Il progetto si concentra sull'analisi dell'architettura del *fused multiply-add* (FMA), una delle operazioni fondamentali nel calcolo *floating point*. L'FMA combina moltiplicazione e somma in un unico passaggio, riducendo errori di arrotondamento e migliorando l'efficienza computazionale

$$out = x \cdot y + w$$

- ❑ Le CPU e diversi acceleratori hardware progettati per il calcolo *floating point*, sfruttano l'FMA per eseguire calcoli intensivi con prestazioni elevate, riducendo il numero di cicli e il consumo energetico

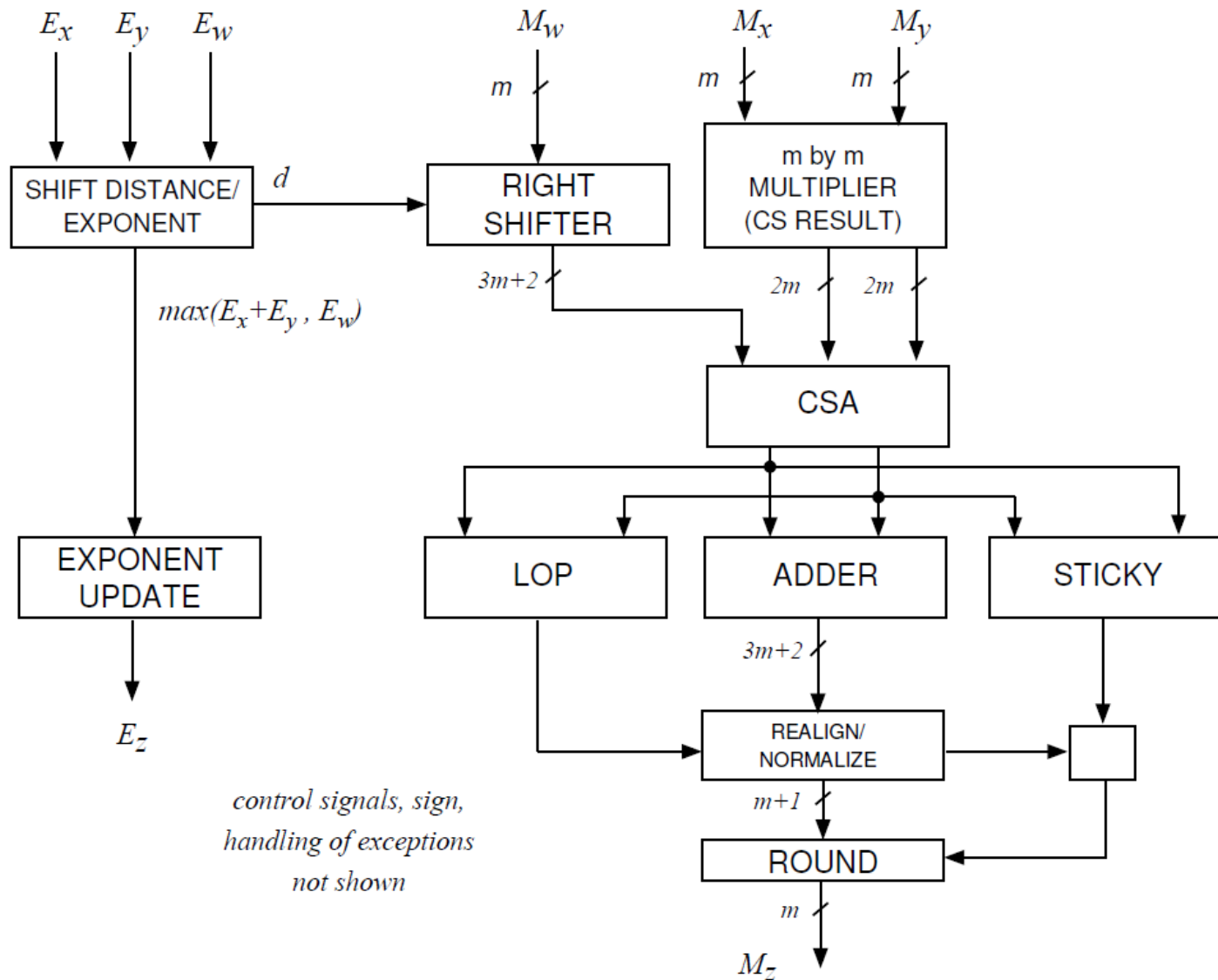
# Floating Point – Standard IEEE 754

- Rappresentazione binary32  $\rightarrow \begin{cases} (-1)^S \cdot (1.M) \cdot 2^{E-127}, & 0 < E < 255 \\ (-1)^S \cdot (0.M) \cdot 2^{-126}, & E = 0 \end{cases}$
- Uso di una rappresentazione normalizzata  $\rightarrow$  Riduzione dynamic range tramite denormals
- Special values:
  - $\pm$ Infinito  $\rightarrow E=2^e-1 \ F=0$
  - NaN  $\rightarrow E=2^e-1 \ F \neq 0$
  - Zero  $\rightarrow E=0 \ F=0$



$S_x$	$E_x$	$M_x$	$V$
0	00000000	0.000000000000000000000000	= 0
0	01111111	1.000000000000000000000000	= $2^{127-127} \cdot (1.0)_2 = 1$
0	01111110	1.000000000000000000000000	= $2^{126-127} \cdot (1.0)_2 = 0.5$
0	10000000	1.000000000000000000000000	= $2^{128-127} \cdot (1.0)_2 = 2$
0	10000001	1.101000000000000000000000	= $2^{129-127} \cdot (1.101)_2 = 6.5$
1	10000001	1.101000000000000000000000	= $-[2^{129-127} \cdot (1.101)_2] = -6.5$
0	00000001	1.000000000000000000000000	= $2^{1-127} \cdot (1.0)_2 = 2^{-126}$
0	00000000	0.100000000000000000000000	= $2^{-126} \cdot (0.1)_2 = 2^{-127}$
0	00000000	0.000000000000000000000001	= $2^{-126} \cdot (0.000000000000000000000001)_2$
			= $2^{-149}$ (Smallest positive value)
0	11111111	000000000000000000000000	= $\infty$
1	11111111	000000000000000000000000	= $-\infty$
1	11111111	100000000000000000000000	= $\sqrt{-1}$ (NaN)

# Fused Multiply-Add



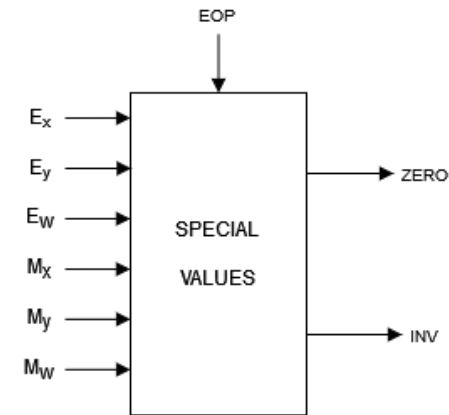
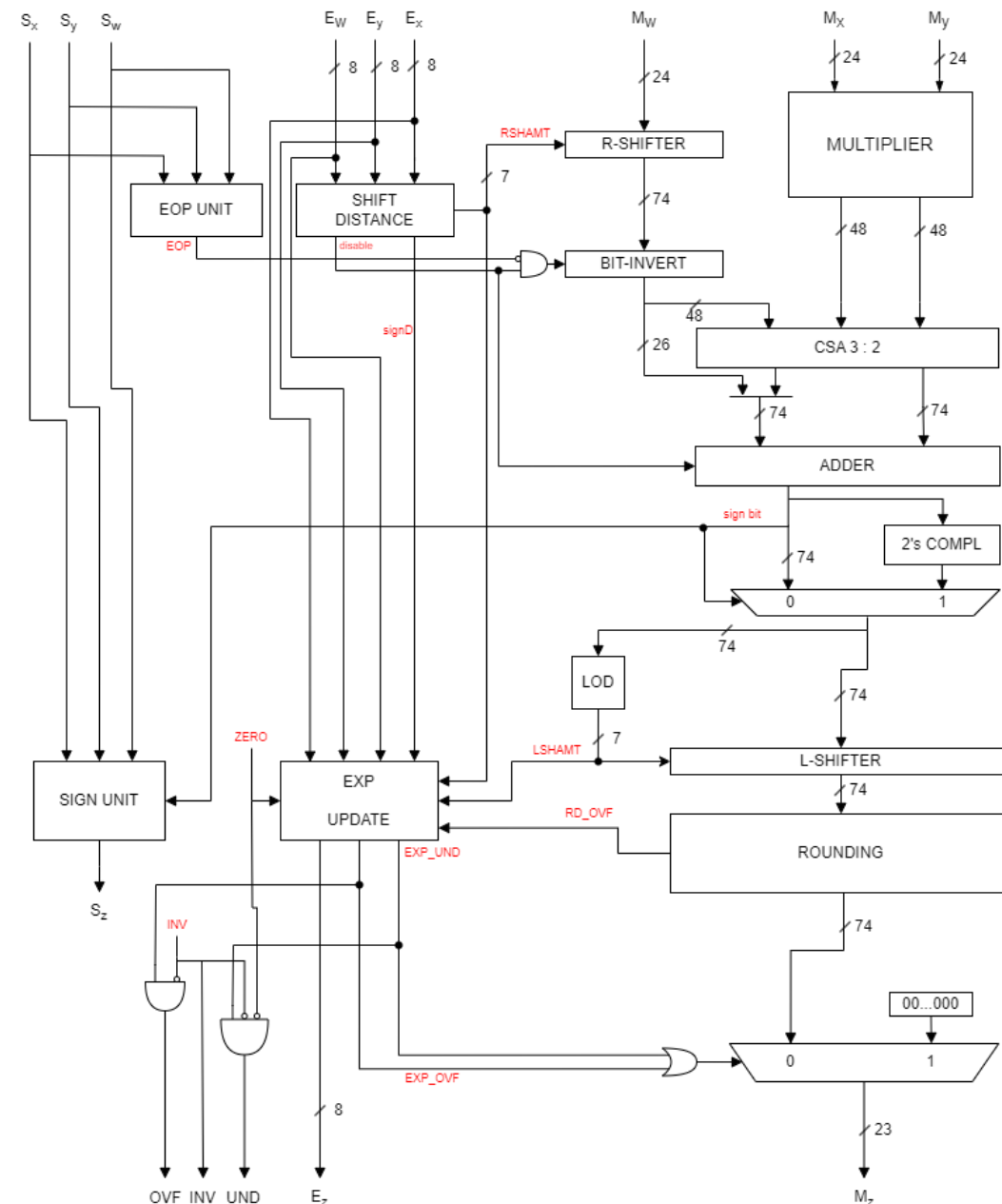
□  $Z = W + X * Y$

□ Step:

- 1)  $M_x * M_y$
- 2)  $E_x + E_y$
- 3) Determinare lo shift di allineamento e shifting di  $M_w$
- 4)  $E_z = \max(E_x + E_y)$
- 5) Somma tra il prodotto e  $M_w$  allineato
- 6) Normalizzare output adder e update dell'esponente
- 7) Rounding
- 8) Determinare flag e special values

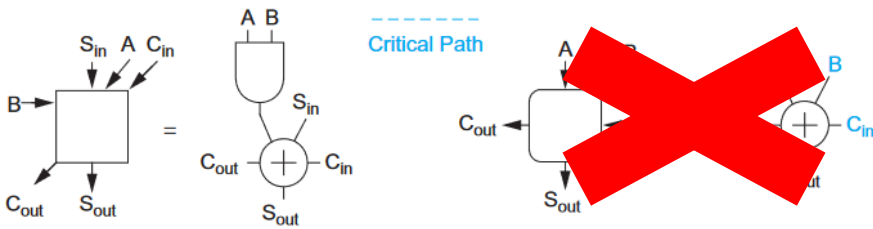
# Implementazione

- ❑ Basata su [2]
- ❑ FMA single precision (parametrizzabile in half precision)
- ❑ Supporto per tutti gli exception
- ❑ 4 Modalità di Rounding



[2] Alberto Nannarelli, Fused Multiply-Add for Variable Precision Floating Point

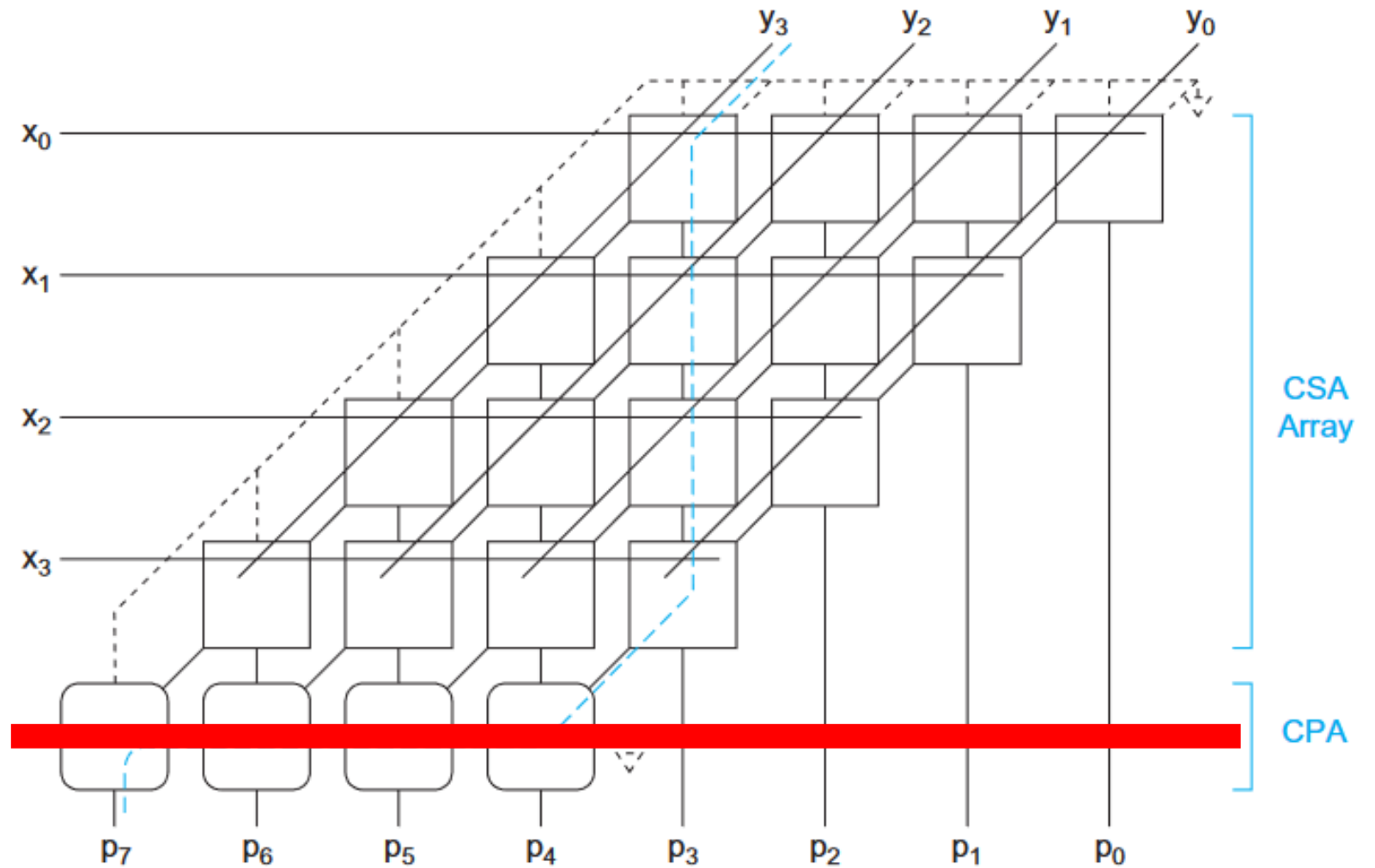
# Multiplier



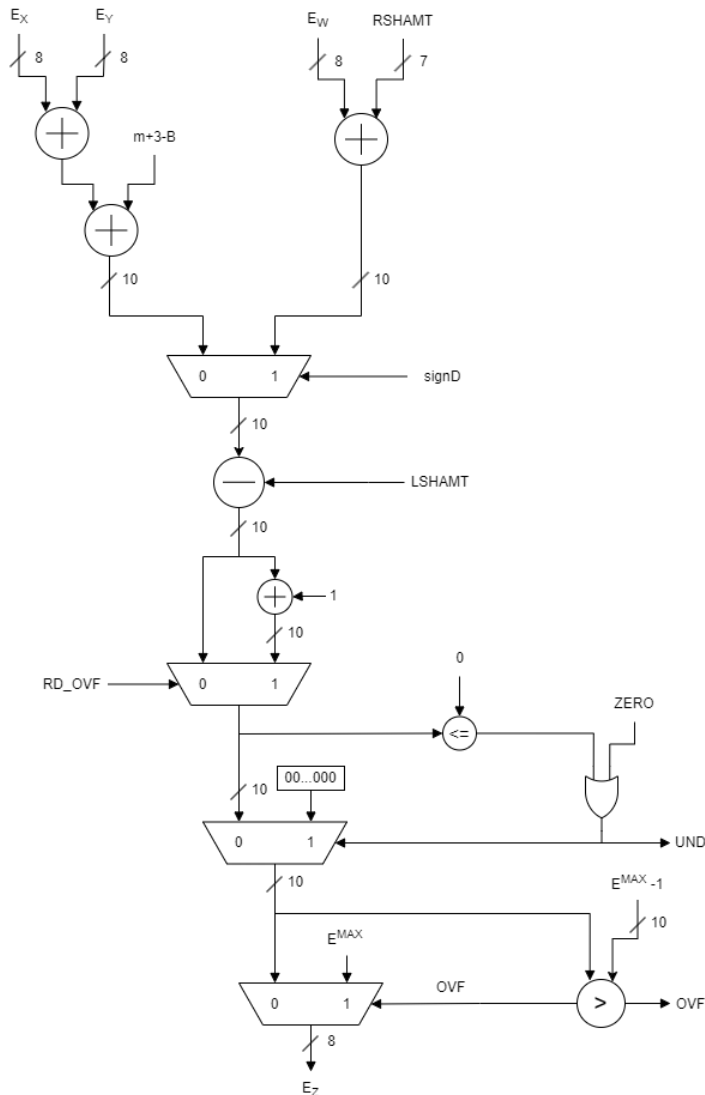
Multiplier con Carry-Save Result



No Carry Propagate Adder

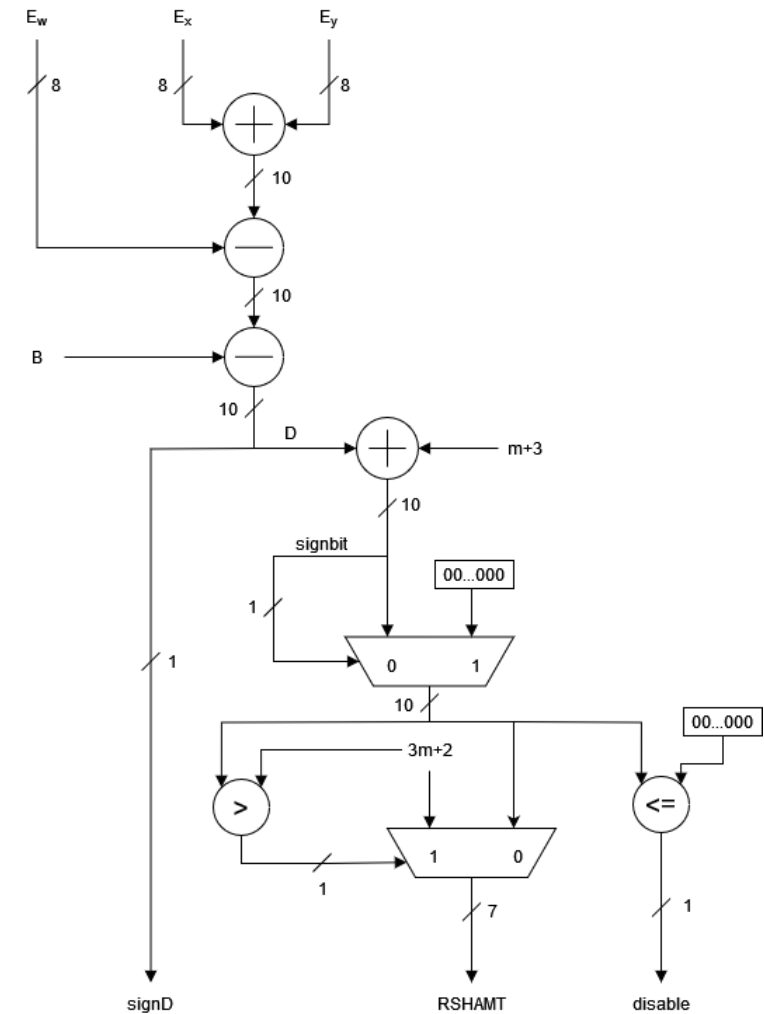


## Exp. Update & Shift Distance



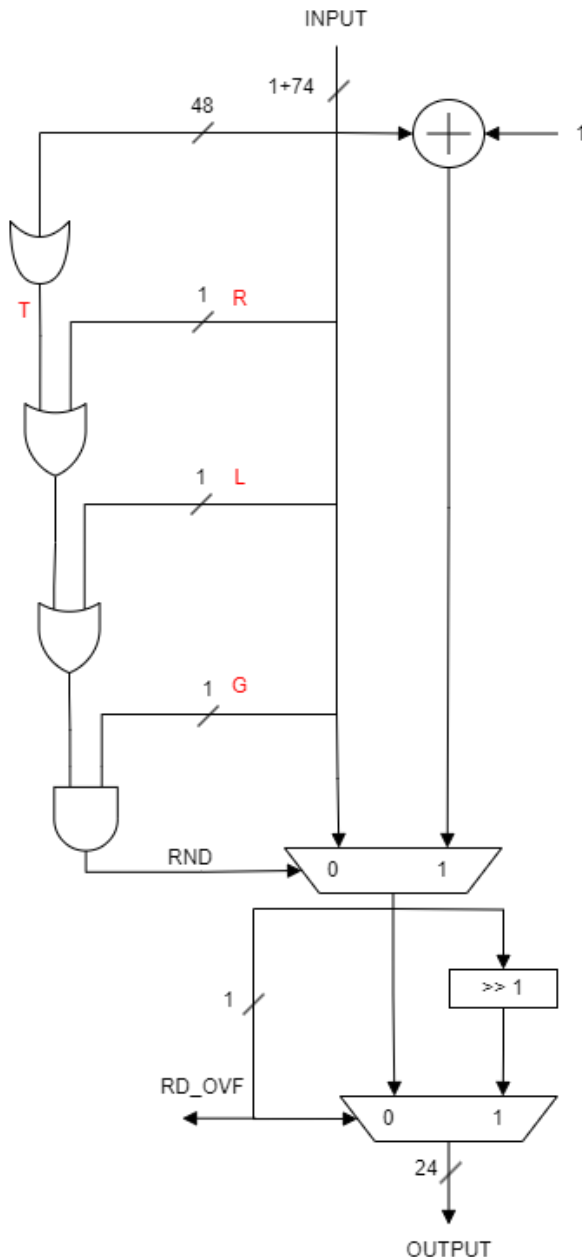
$$E_X + E_Y > E_W \rightarrow E_Z = E_X + E_Y - B - (LSHAMT - 27)$$

$$E_W > E_X + E_Y \rightarrow E_Z = E_W + RSHAMT - LSHAMT$$



$$D = (E_X + E_Y - B) - E_W \rightarrow RSHAMT = D + 27$$

# Rounding



## Modalità:

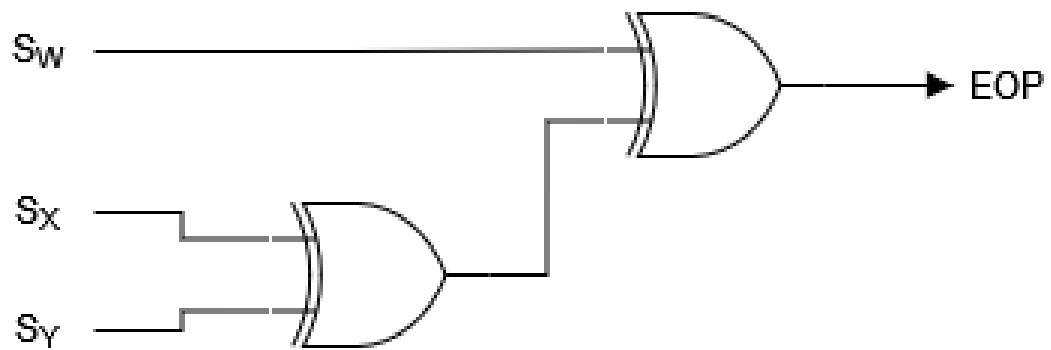
- ❖ Round to Nearest (Tie to Even)  $\rightarrow rnd = G \text{ AND } (R \text{ OR } T \text{ OR } L)$
- ❖ Round to  $+\infty$   $\rightarrow rnd = \bar{S}_z \text{ AND } (G \text{ OR } R \text{ OR } T)$
- ❖ Round to  $-\infty$   $\rightarrow rnd = S_z \text{ AND } (G \text{ OR } R \text{ OR } T)$
- ❖ Round toward zero  $\rightarrow \text{trunc}(\text{IN})$

## Bit per il calcolo:

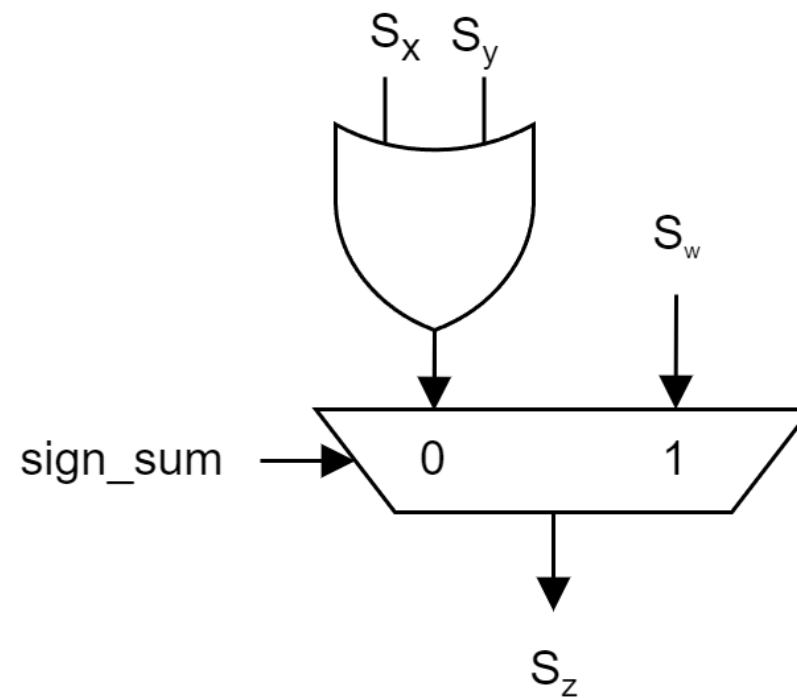
- ❖ Guard bit  $\rightarrow$  primo bit oltre l'LSB del risultato
- ❖ Round bit  $\rightarrow$  bit successivo al guard-bit
- ❖ Sticky bit  $\rightarrow$  indica se i bit dopo al round-bit sono  $\neq 0$
- ❖ LSB del risultato normalizzato  $\rightarrow$  Per decidere in caso di parità



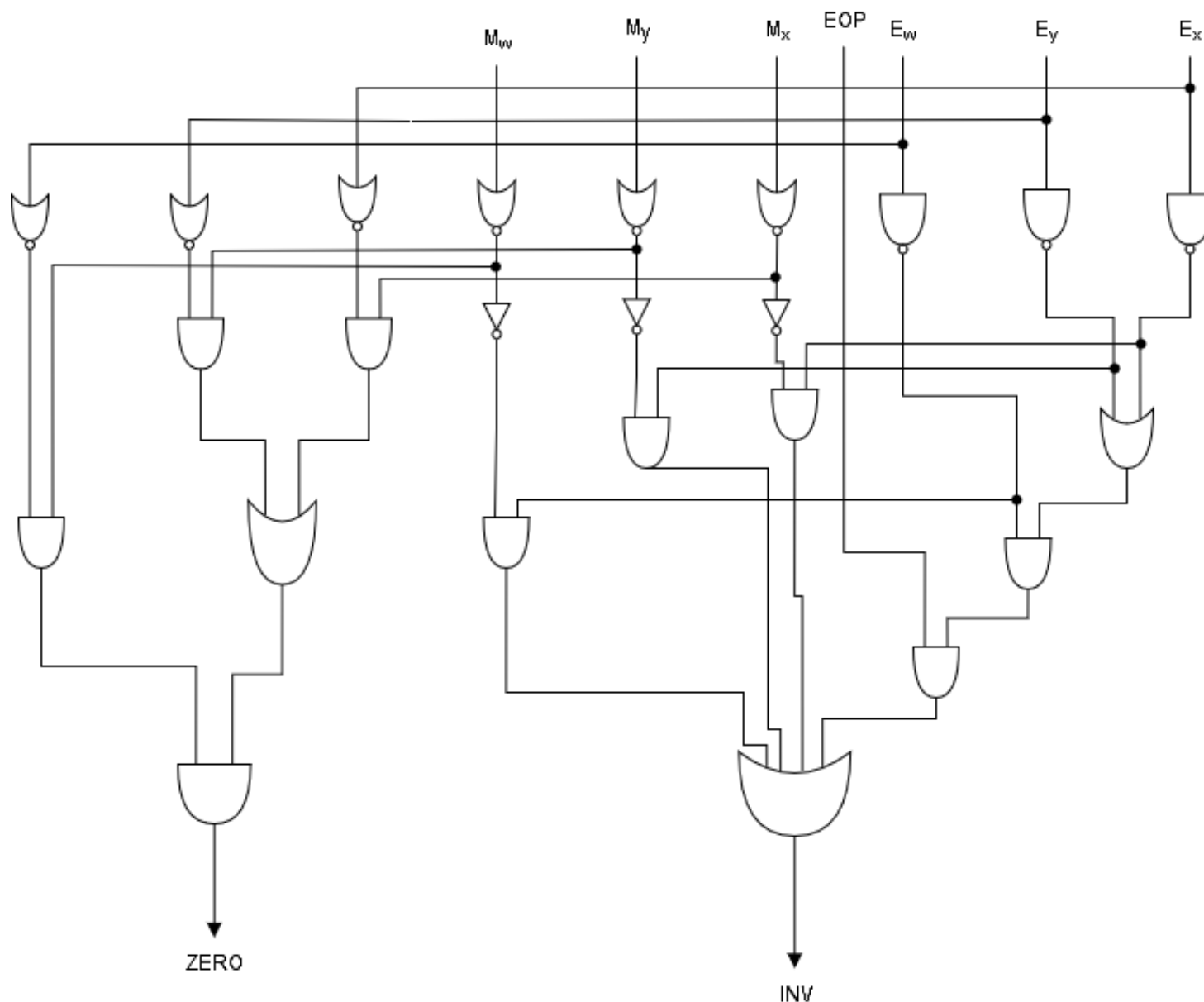
## EOP & Sign



$$EOP = (S_X \oplus S_Y) \oplus S_W$$



# Special Values



INV



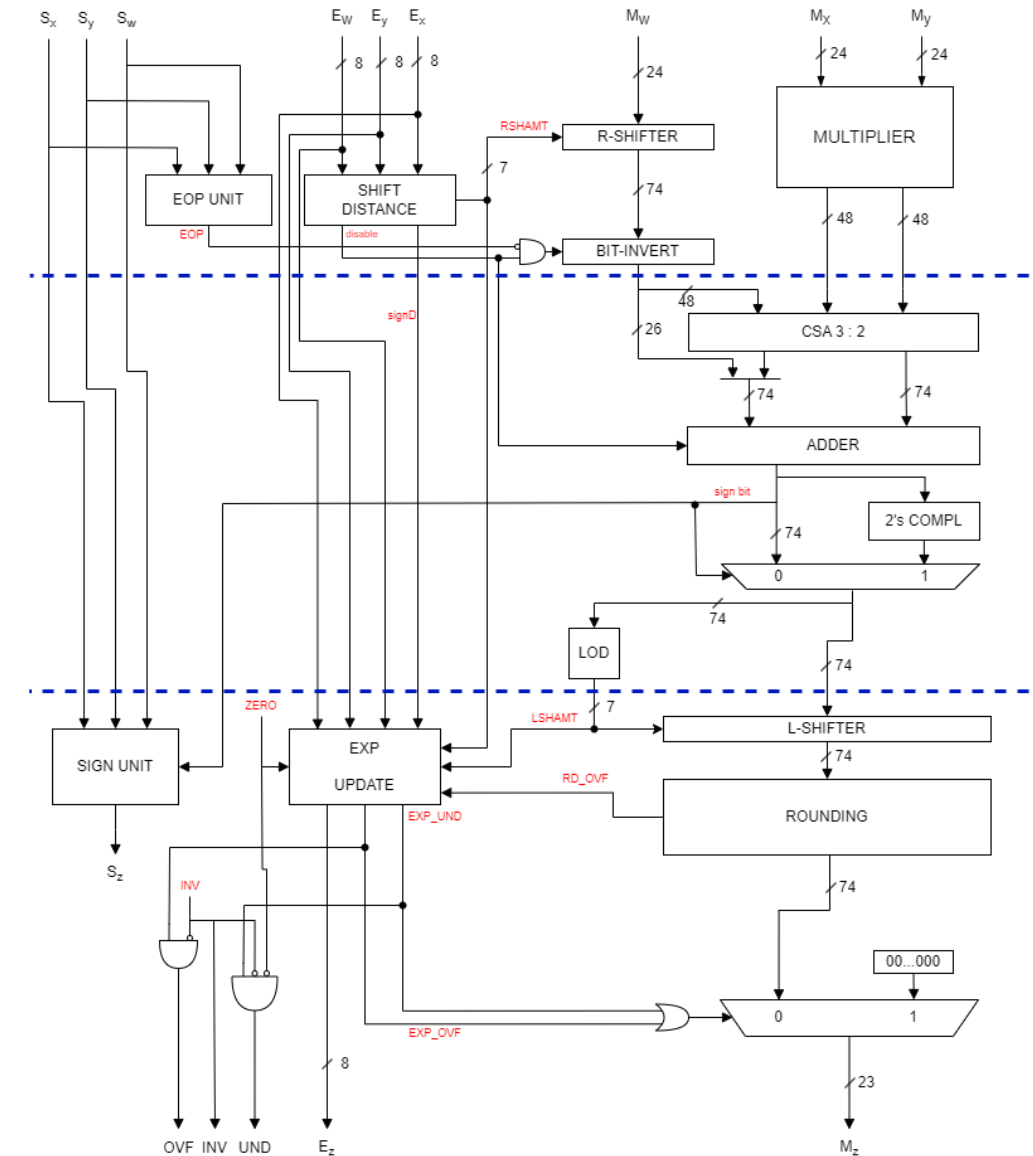
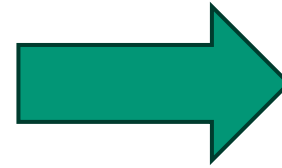
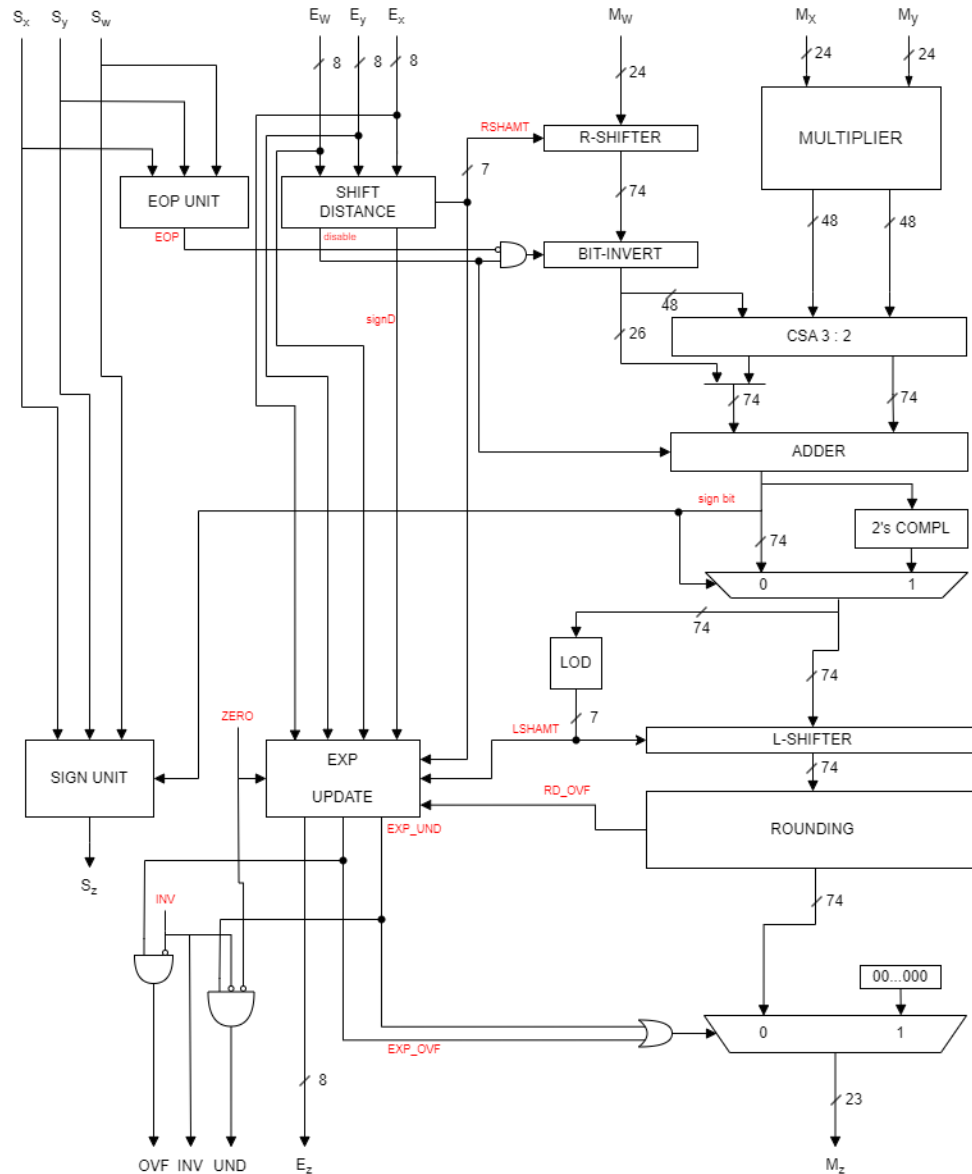
Rivela il caso  $+\infty - \infty$  oppure se uno degli input è NaN

ZERO

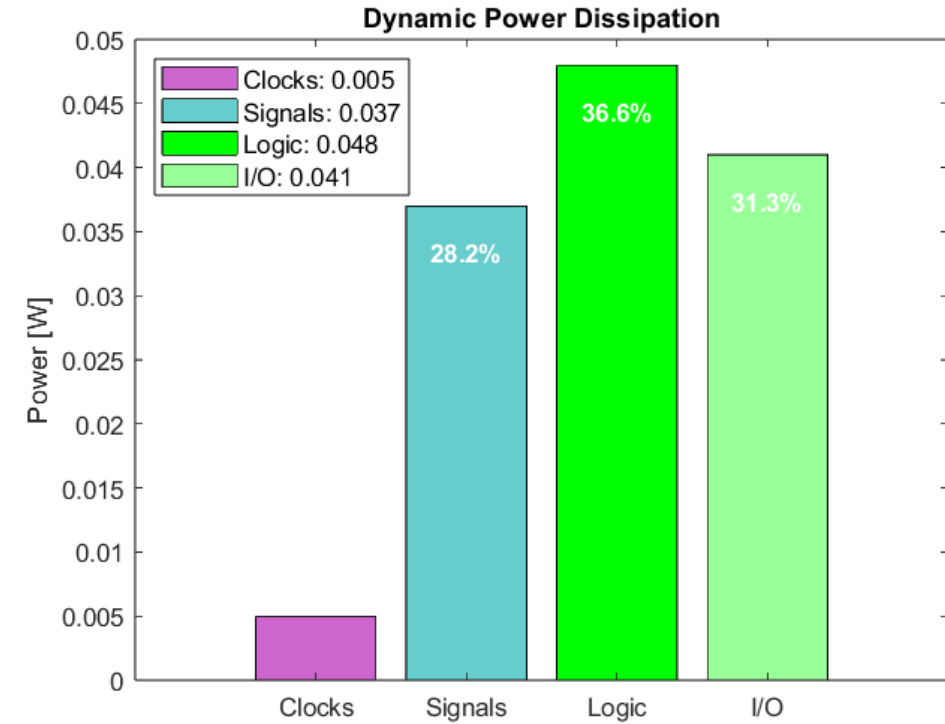
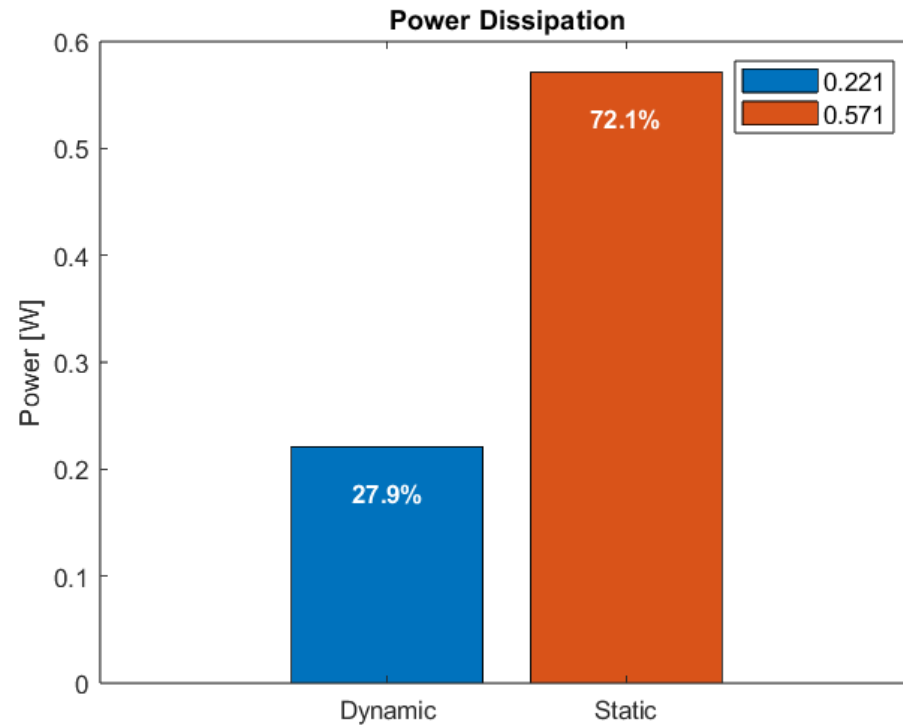


rileva se uno o entrambi gli input x e y sono zero e l'input w è zero

# Pipelining



# Risultati Implementativi

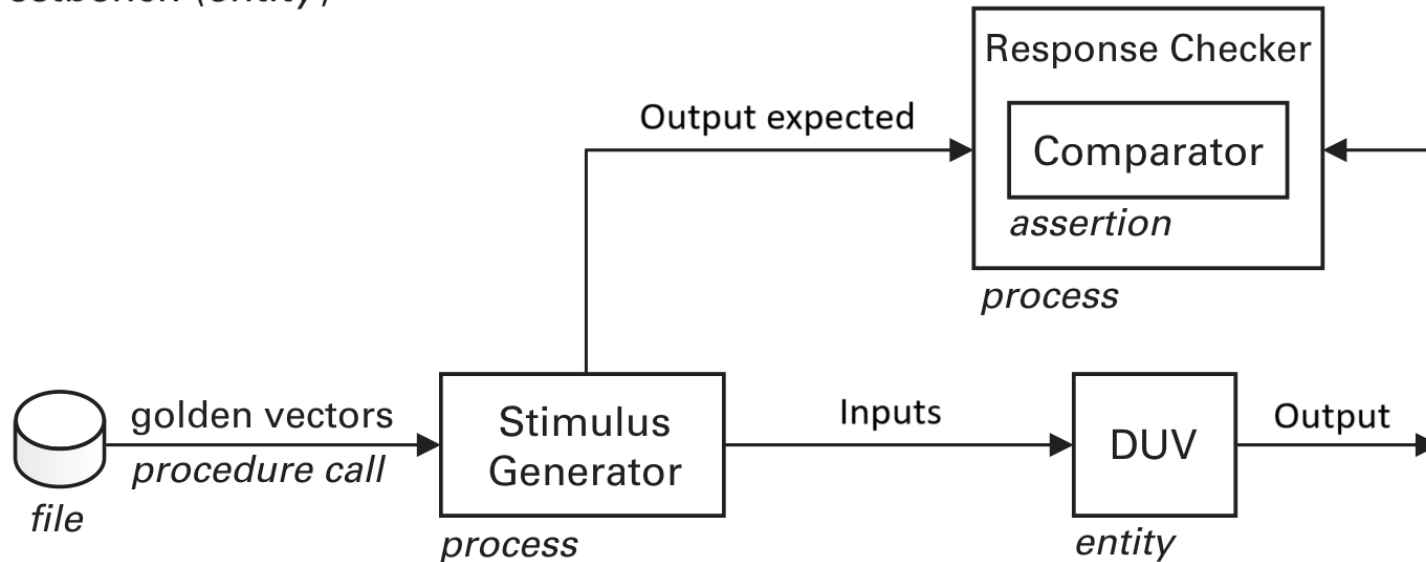


Risorse	Utilizzate	Disponibili	Utilizzo (%)
LUT	1823	230400	0.79
FF	301	460800	0.07
IO	133	204	65.20

- ❑ FPGA → Zynq UltraScale+ MPSoCs
- ❑ Frequenza massima di clock → 333 MHz

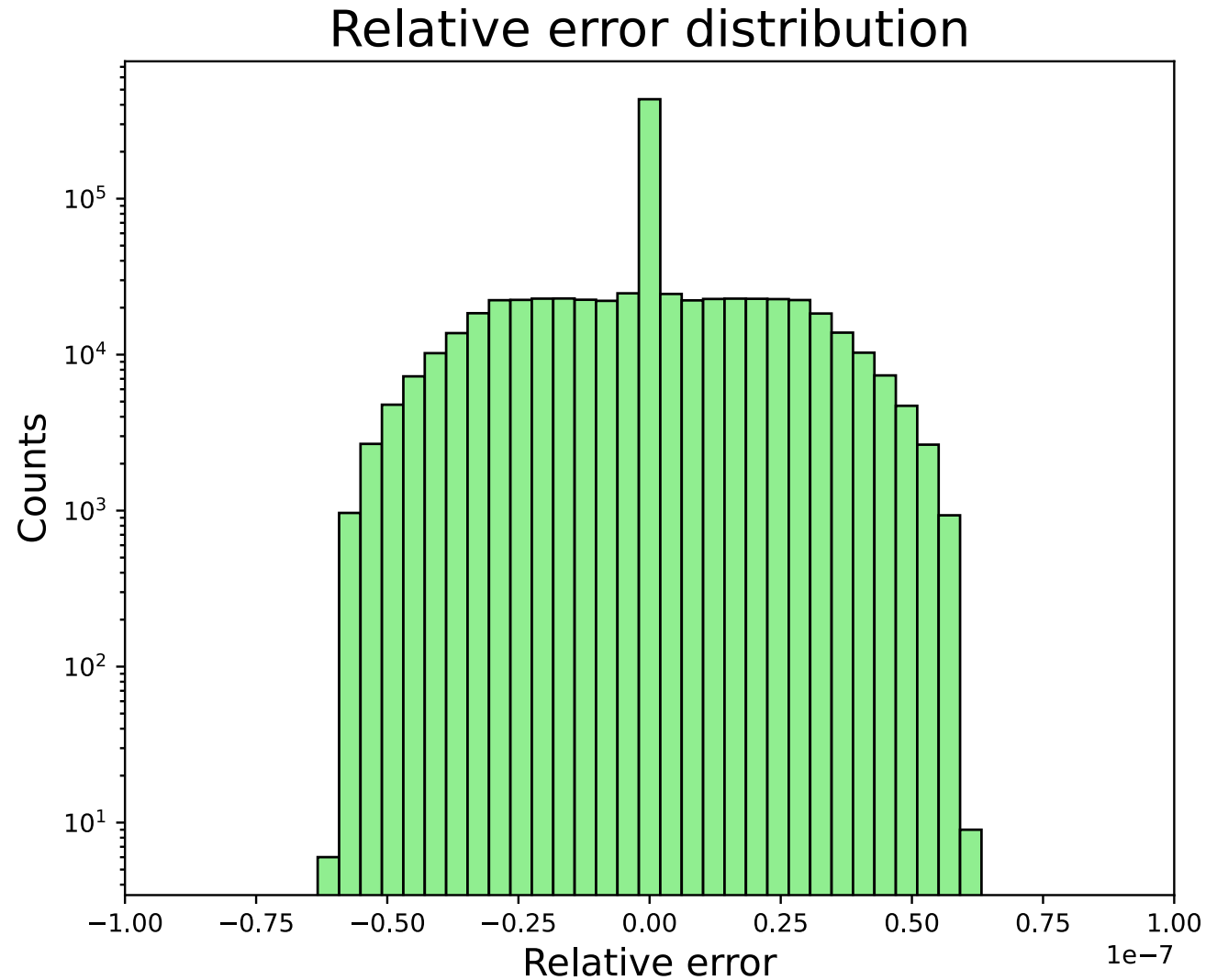
# Risultati Simulativi

Testbench (*entity*)



- ❑ **N° di vettori:** 1.000.000
- ❑ **Distribuzione inputs:** Uniforme
- ❑ Inputs esatti in float32 (senza arrotondamenti)
- ❑ **Outputs desiderati calcolati in float128**
- ❑ **0 errori**, esclusi i casi con output subnormal

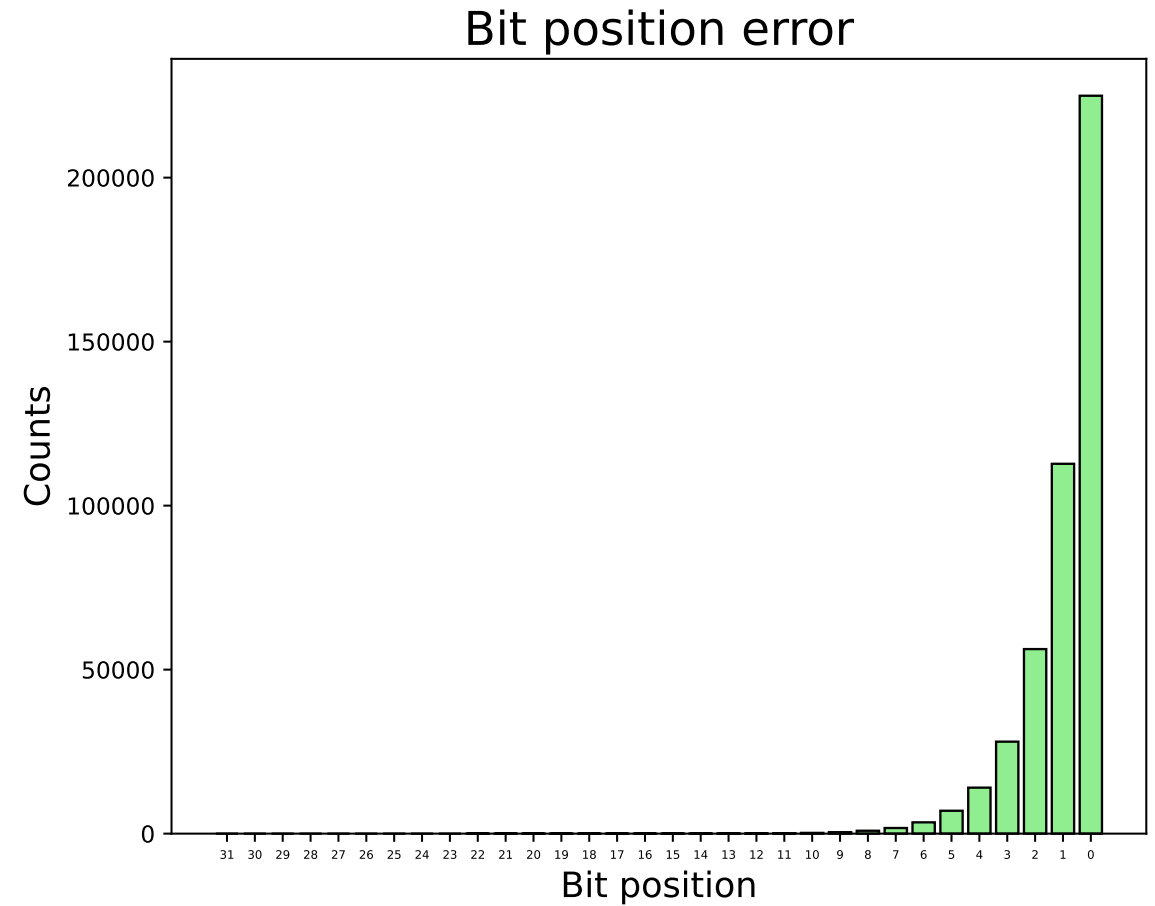
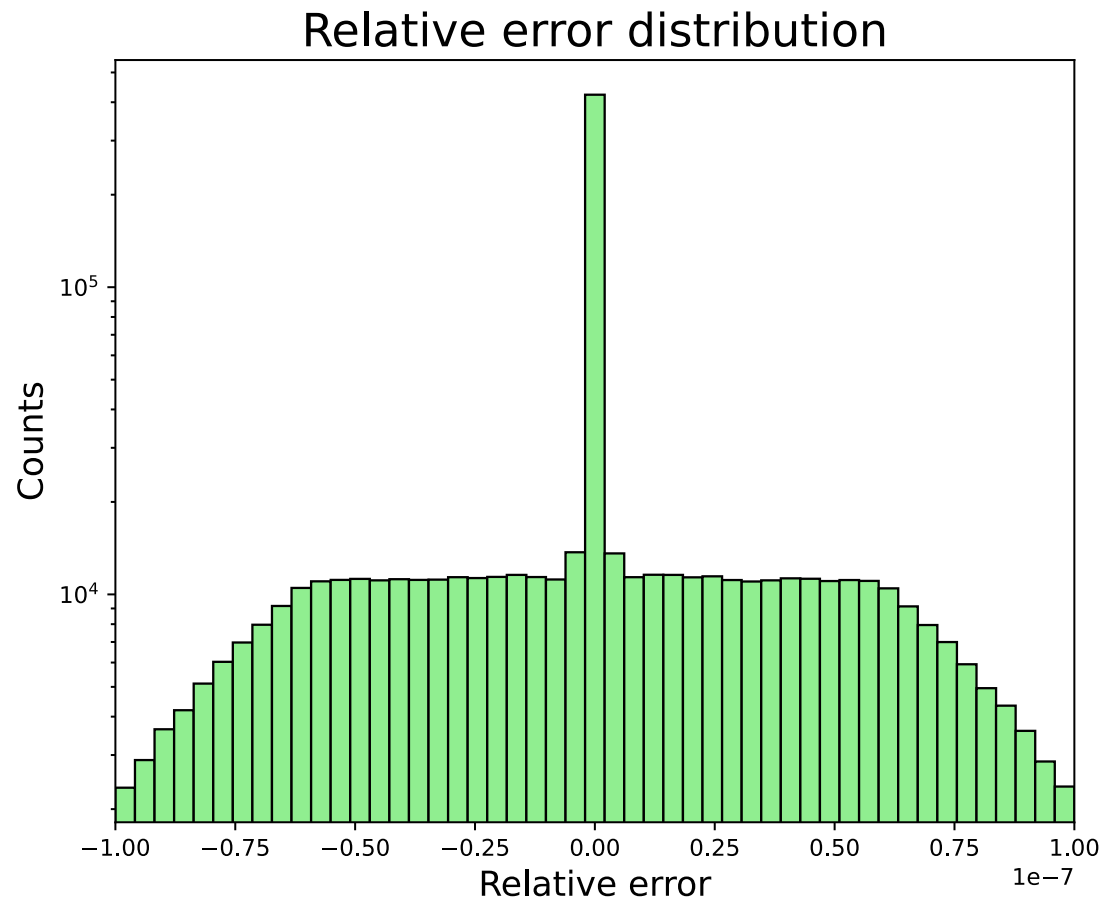
## Error PDF - RTNE



*\*Uscite overflow e subnormal escluse*

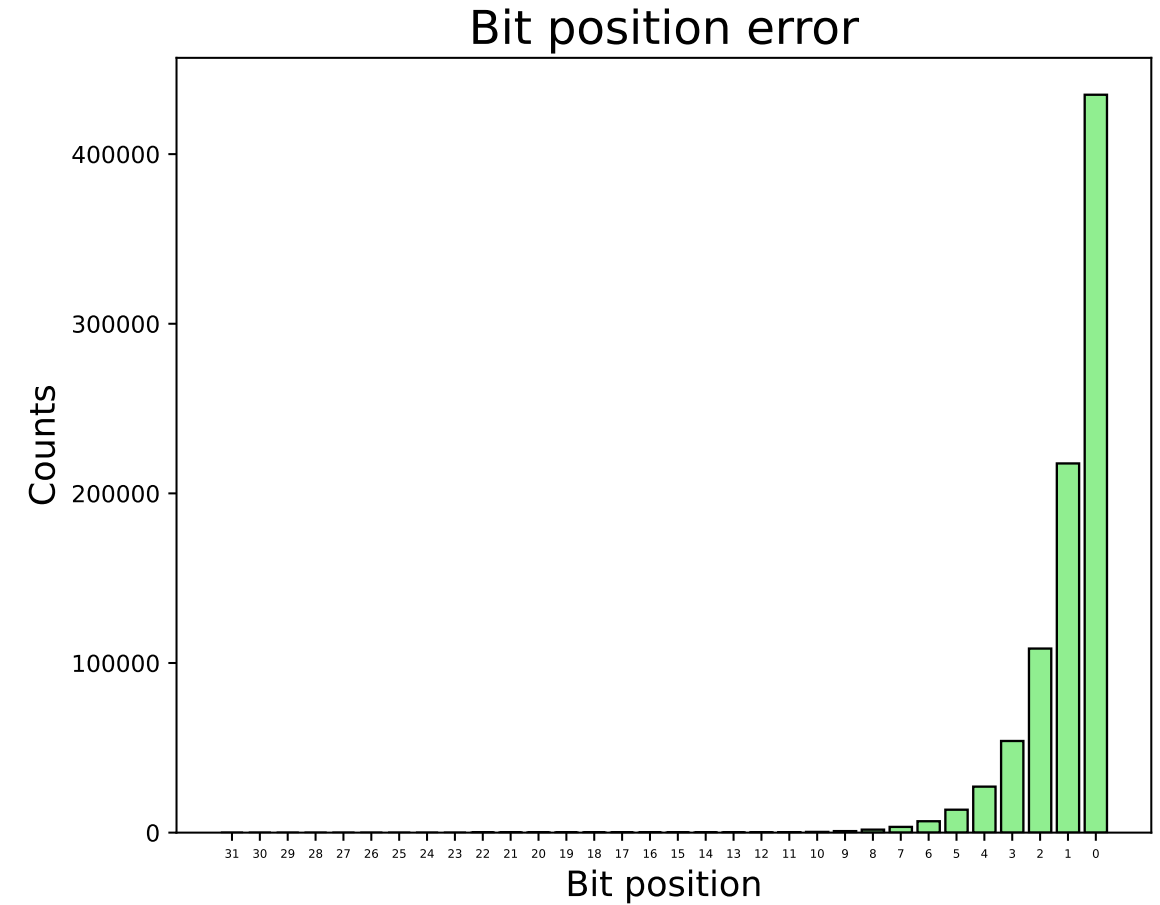
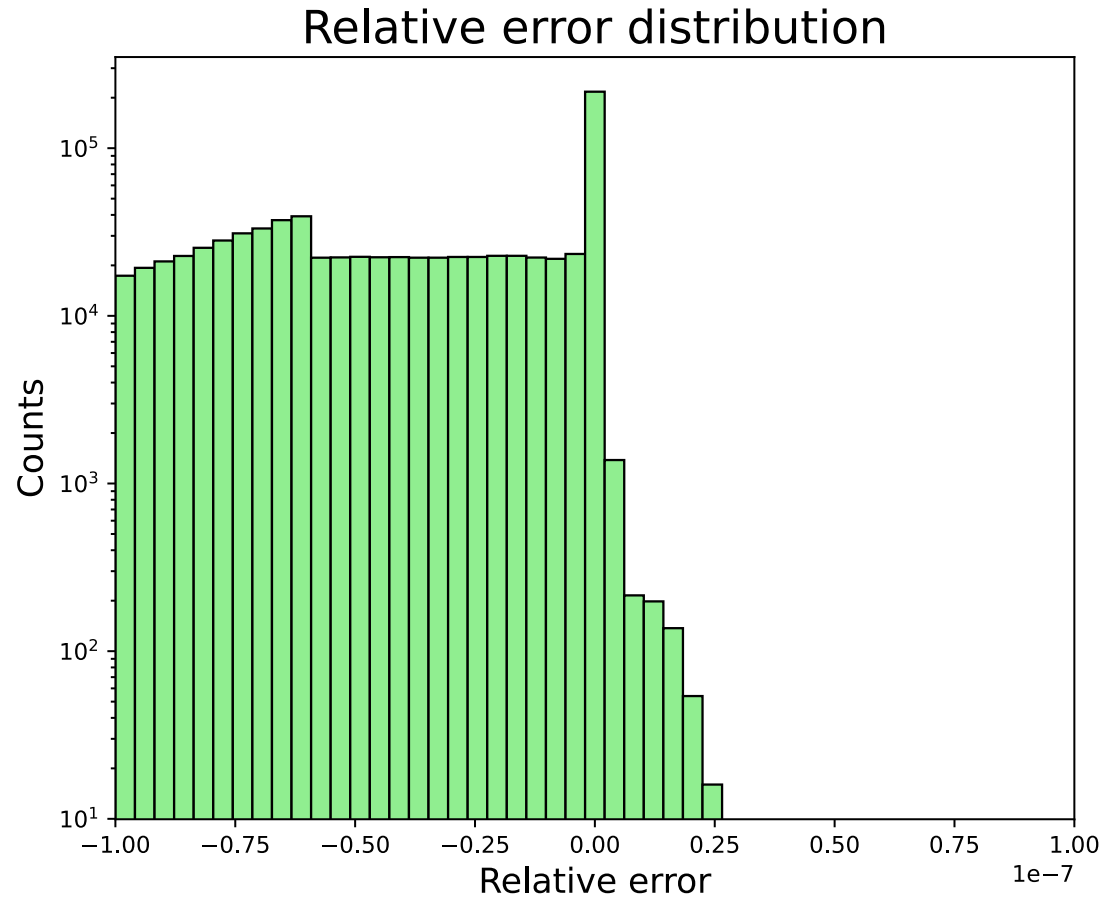
$$Error_{rel} = \frac{OUT_{exp} - OUT}{|OUT_{exp}|}$$

# Error PDF – RTZ



*\*Uscite overflow e subnormal escluse*

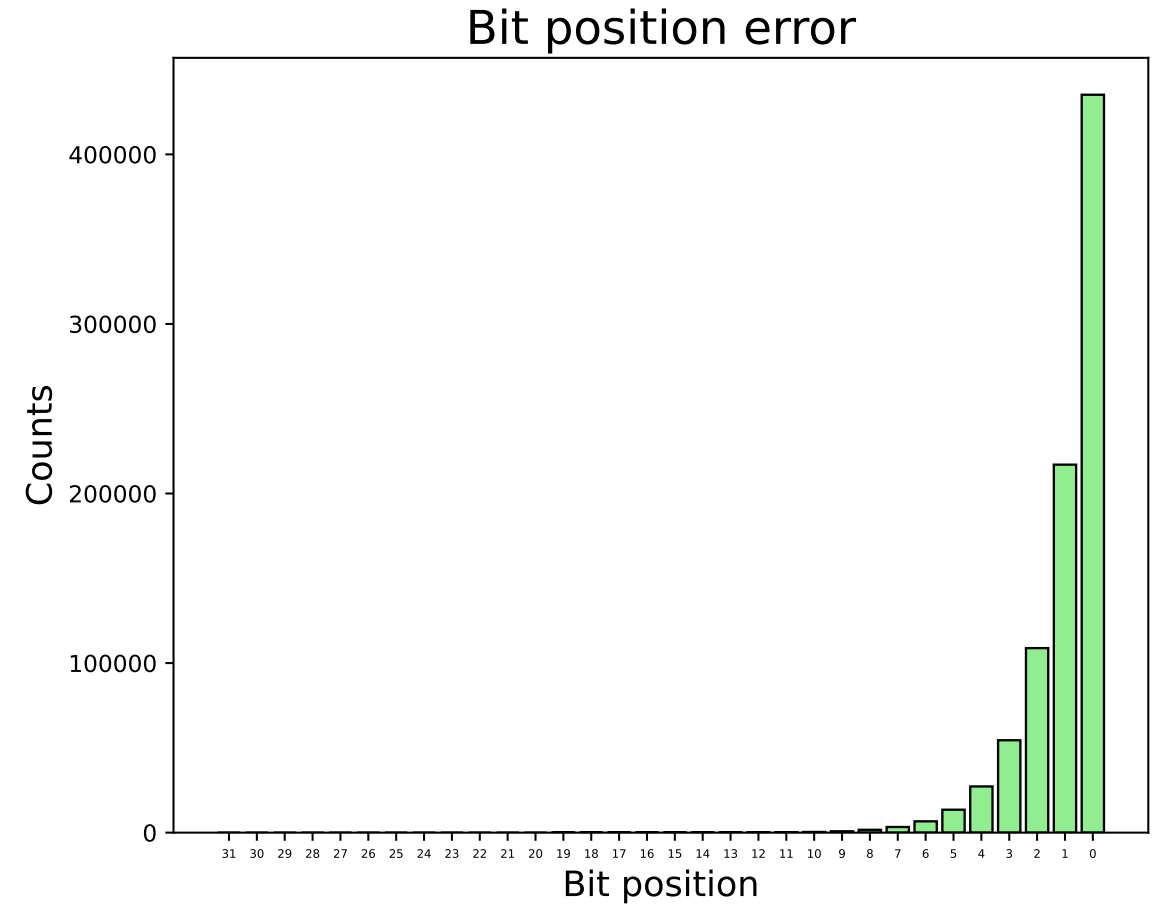
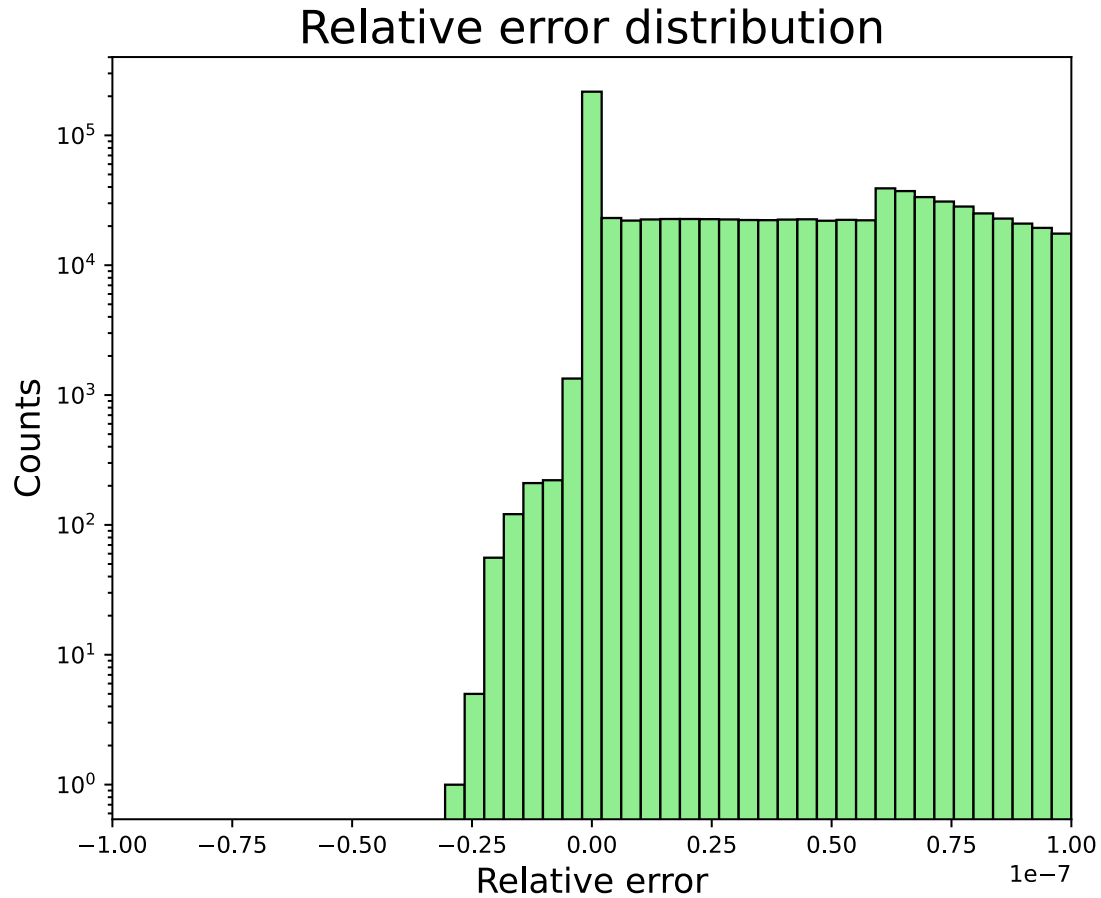
# Error PDF – Round To Plus Infinity



*\*Uscite overflow e subnormal escluse*



# Error PDF – Round To Minus Infinity

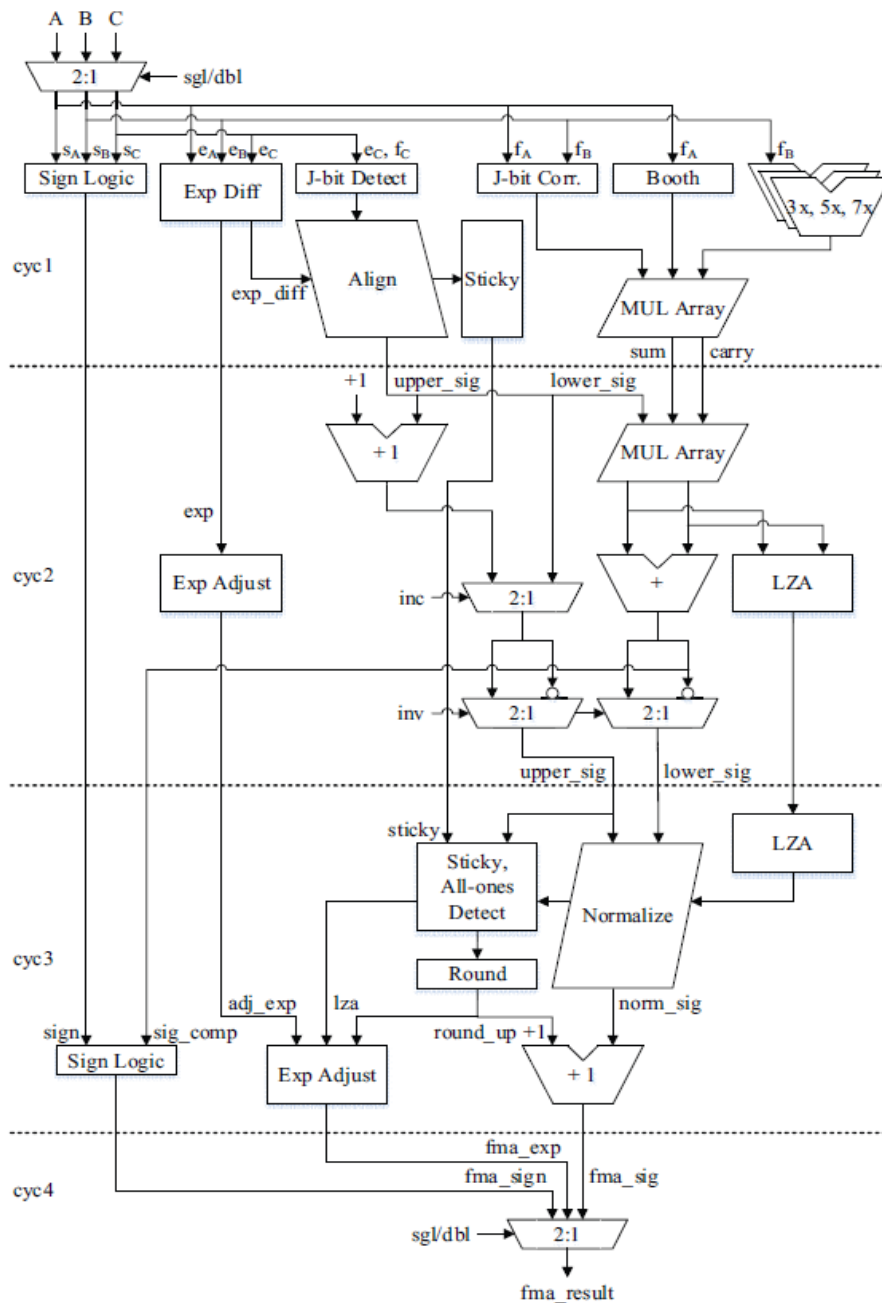


*\*Uscite overflow e subnormal escluse*

## Simulazione Corner Case

$X$	$Y$	$W$	$OUT$	$OVF$	$UND$	$INV$
0	0	0	0	0	0	0
-0	0	-0	-0	0	0	0
2.5	0	0	0	0	0	0
3.403e38	3.403e38	3.403e38	$\infty$	1	0	0
1.175e-38	1.175e-38	0	0	0	1	0
$\infty$	2.5	2.5	$\infty$	1	0	0
$-\infty$	2.5	2.5	$-\infty$	1	0	0
$\infty$	$\infty$	$\infty$	$\infty$	1	0	0
$-\infty$	$-\infty$	$\infty$	$\infty$	1	0	0
$\infty$	2.5	$-\infty$	NaN	0	0	1
$\infty$	$-\infty$	$\infty$	NaN	0	0	1
NaN	2.5	2.5	NaN	0	0	1
NaN	NaN	NaN	NaN	0	0	1

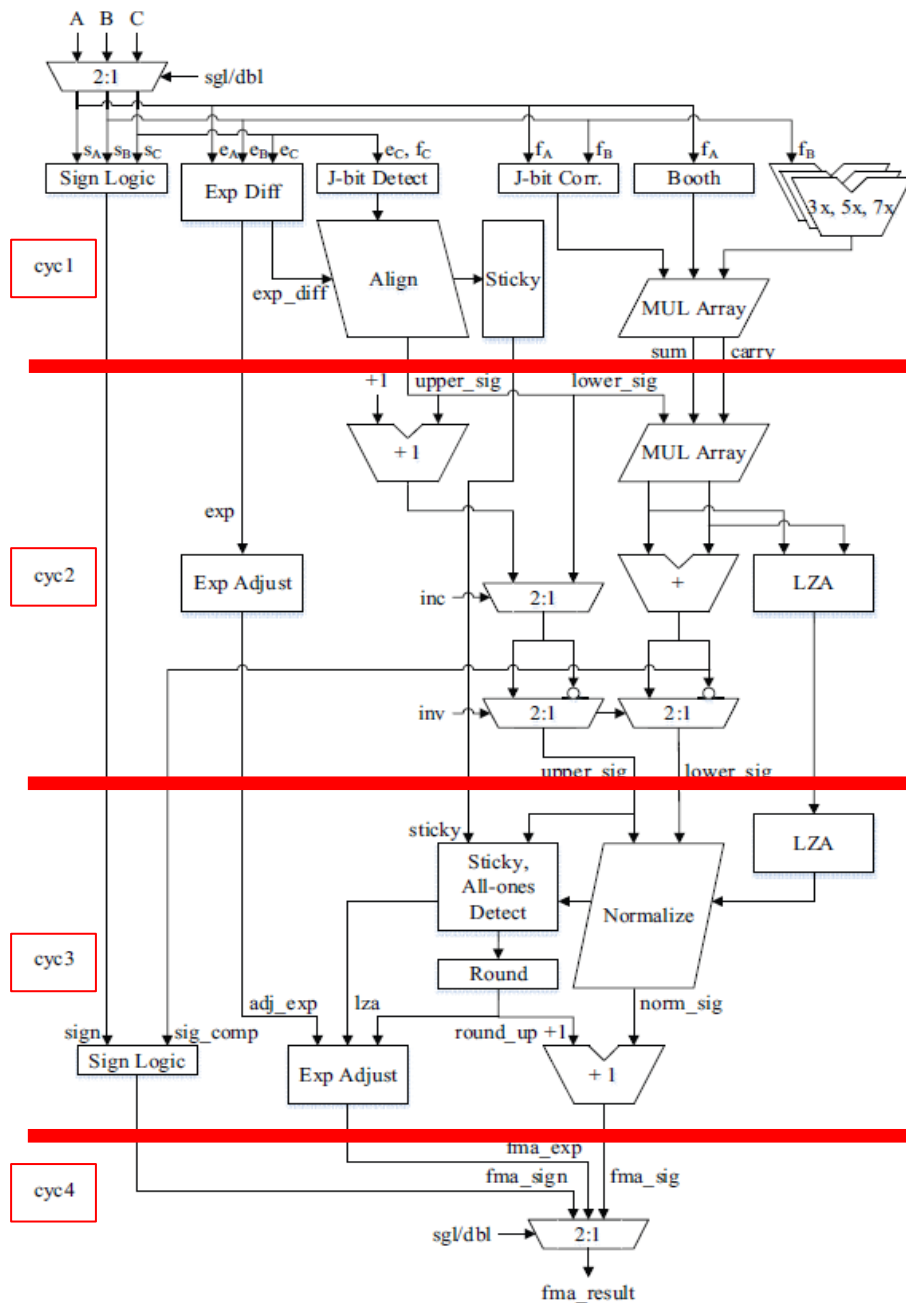
# Subnormal (1)



- ❑ Design ottimizzato di un FMA per processori Intel E-Core
- ❑ Supporto completo per input *subnormal* e output *underflow*
- ❑ Supporto per tutti le 4 modalità di rounding (per IEEE 754)
- ❑ Rispetto alle FMA tradizionali:
  - ❖ Riduzione area del 10/30 %
  - ❖ Riduzione latenza 10/20 %

[5] J. Sohn, D. K. Dean, E. Quintana, W. S. Wong, Enhanced Floating-Point Multiply-Add with Full Denormal Support

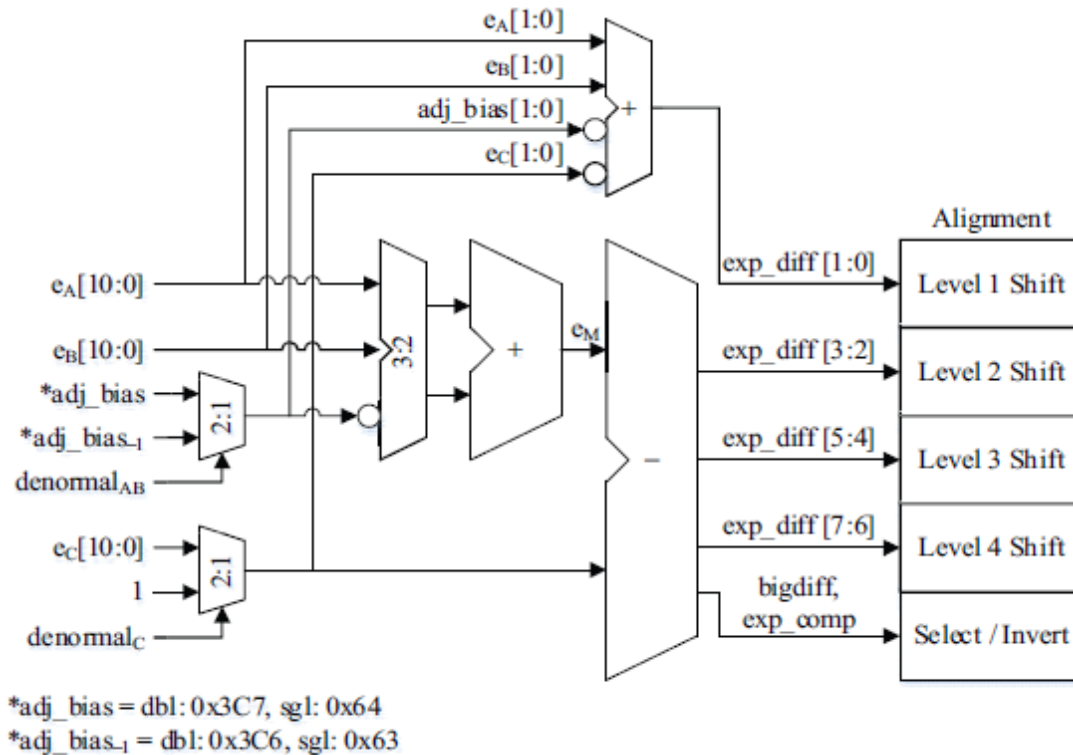
## Subnormal (2)



- ❑ Input a 64-bit vengono formattati o in un valore *double precision* o in due *single precision*
- ❑ **Cyc1** → exponent difference, allineamento, Booth encoding, e la prima parte del multiply array
- ❑ **Cyc2** → resto del multiply array, main adder, incrementor e LZA
- ❑ **Cyc3** → normalizzazione e rounding
- ❑ **Cyc4** → ultimo MUX e bypass/writeback logic

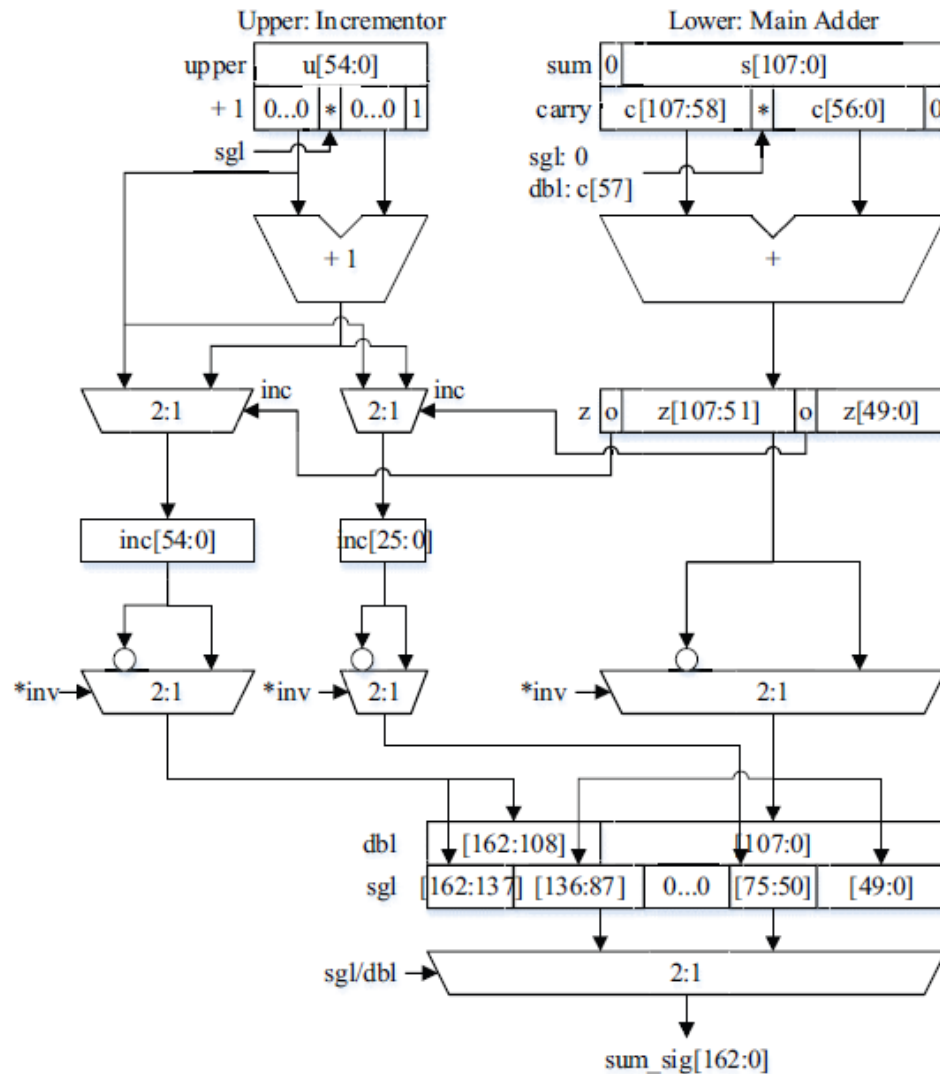
[5] J. Sohn, D. K. Dean, E. Quintana, W. S. Wong, Enhanced Floating-Point Multiply-Add with Full Denormal Support

## Subnormal (3)



- ❑ All'interno del blocco **Exponent Difference**, il bias viene sottratto di uno se uno dei due operandi del moltiplicatore è *subnormal*. Allo stesso modo,  $E_C$  viene regolato ad uno se  $C$  è *subnormal*.
- ❑ Il J-bit è implicito per i numeri normalizzati, ma, per gestire i numeri *subnormal*, il bit J deve essere trattato come zero:  $Jbit = \begin{cases} 1 & \text{if } exp \neq 0 \\ 0 & \text{otherwise} \end{cases}$
- ❑ Il J-bit della mantissa dell'addendo viene rilevato in parallelo con il primo livello della differenza dell'esponente in modo che non vi sia alcun *delay* aggiuntivo
- ❑ L'allineamento della mantissa consiste in quattro livelli di shifter e un MUX

## Subnormal (4)



\*inv = upper allones & inc & truesub

- ❑ Si presuppone che entrambi i J-bit siano uno, quindi si sottrae una linea di correzione J-bit nell'array di moltiplicazione, che richiede un'altra linea di prodotto parziale e alcuni bit per il complemento a due. Se A è *subnormal*, viene sottratto  $M_B$  e se B è *subnormal*, viene sottratto  $M_A$
- ❑ Il risultato del **Main Adder** deve essere normalizzato. Per accelerare la normalizzazione, la LZA viene eseguita in parallelo con il **Main Adder**
- ❑ L'*underflow* viene rilevato se il J-bit dopo la normalizzazione è zero, il che significa avere una mantissa risultante *subnormal*, e l'esponente è impostato su zero

## Conclusioni

- ❑ Realizzazione di un architettura FMA floating-point
- ❑ Analisi risultati simulativi e implementativi
- ❑ Studio e analisi architettura FMA *subnormal*

## Sviluppi futuri

- ❑ Confronto con altri design FMA floating-point
- ❑ Implementazione architettura *subnormal*
- ❑ Progettazione di altri design per i *subnormal*