Modelado de precisión finita

Javier Valls

Modelado de precisión finita

1. Introducción ¿necesidad y utilidad del modelado de precisión finita?
2. Codificación binaria
3. Formatos de punto fijo
4. Efectos de la precisión finita
5. Operaciones con precisión finita
6. Ejemplo de aplicación
7. Modelado de algoritmos con precisión finita y flujo de verificación

¿Modelado de precisión finita?

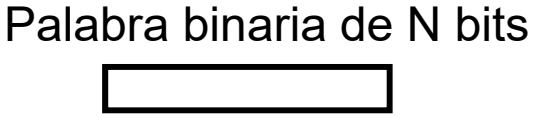
- Computar algoritmos en un dispositivo FPGA
  - implementación de operadores con tamaños de palabra fijos
- Reducir el tamaño de los operadores afecta a la implementación HW:
  - reducción área
  - mejora la frecuencia de operación
  - reducción de consumo de potencia
- Es necesario evaluar el comportamiento con precisión finita
  - el algoritmo debe mantener las prestaciones deseadas
- Disponer de un modelo de precisión finita facilita la verificación del HW:
  - proporciona vectores de test con los que comparar los modelos HDL del HW

Codificación binaria

- En los sistemas digitales los números se almacenan en formatos binarios
- El tipo de codificación de los datos determina:
  - el **rango** de valores representables
  - y la **resolución** con la que se puede operar
- Rango: conjunto de valores representables con un tipo de codificación
  - Los números fuera del rango dinámico se representan dentro del rango al cuantificarlo
- La resolución es la mínima distancia entre dos números cuantificados
- Sistemas de codificación:
  - Punto flotante
  - Punto fijo

Codificación binaria

- **Punto fijo**
  - Rango dinámico limitado
  - Resolución fija
  - Operaciones básicas sencillas



- **Punto flotante**
  - Rango dinámico grande
  - Resolución variable
  - Operaciones básicas complejas

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

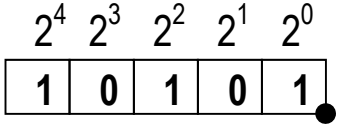
Codificación en punto fijo de enteros

Si el número de bits de la codificación es **N**

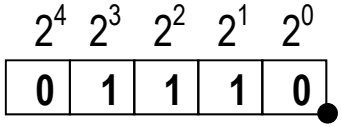
- Codificación sin signo:
  - Rango : $[0, (2^N-1)]$
  - Resolución : 1

$$X = \sum_{i=0}^{N-1} x_i 2^i$$

Ejemplo: N=5, rango $\subset [0,31]$



$X=2^4+2^2+2^0=21$



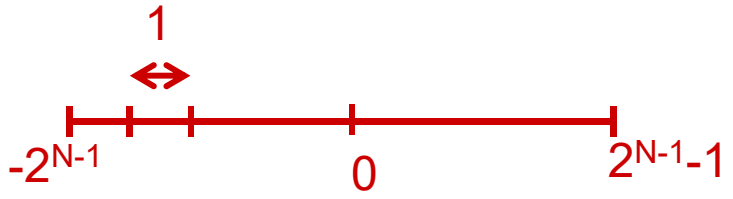
$X=2^3+2^2+2^1=14$

Codificación en punto fijo de enteros

Si el número de bits de la codificación es **N**

- Codificación con signo en complemento a dos:
  - Rango : $[-2^{N-1}, (2^{N-1}-1)]$
  - Resolución : 1

$$X = -x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i$$



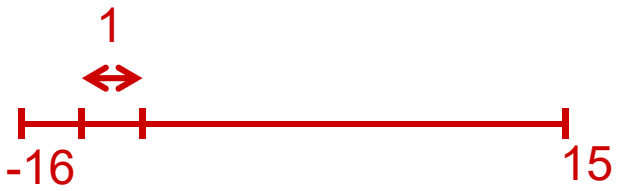
Ejemplo: N=5, rango $\subset$ [-16,15]

-2^4	2^3	2^2	2^1	2^0
1	0	1	0	1

$X=-2^4+2^2+2^0= -11$

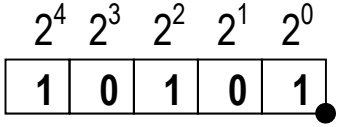
-2^4	2^3	2^2	2^1	2^0
0	1	1	1	0

$X=2^3+2^2+2^1=14$



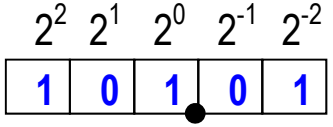
Codificación en punto fijo de reales

Número de bits: N=5



$X_E=2^4+2^2+2^0=21$

Rango (0,31) Resolución: 1



$X_Q=2^2+2^0+2^{-2}=5.25$

Rango (0,7.75) Resolución: 0.25

$5.4 = (101.01100110011\ldots)_2$

Codificación:

- 6 bits con 3 fraccionales: $5.4 \approx 5.375 = (101.011)_2$
- 5 bits con 2 fraccionales: $5.4 \approx 5.25 = (101.01)_2$
- 4 bits con 1 fraccionales: $5.4 \approx 5 = (101.0)_2$

Codificación en punto fijo de reales

$V_R \cong V_Q = V_E \cdot Q$

- V_R: Valor real
- V_Q: Valor codificado, aproximado de V_R
- V_E: Número entero que codifica V_Q
- Q : Escalado

Escalado de "b" bits $Q = 2^{-b}$:

- peso del bit menos significativo
- "b" indica la posición del punto fraccionario

$$5.4 = (\,101.01100110011\ldots)_2$$

Codificación:

- 6 bits con 3 fraccionales: $5.4 \approx 5.375 = (101.011)_2 = 43 \cdot 2^{-3}$
- 5 bits con 2 fraccionales: $5.4 \approx 5.25 = (101.01)_2 = 21 \cdot 2^{-2}$
- 4 bits con 1 fraccionales: $5.4 \approx 5 = (101.0)_2 = 10 \cdot 2^{-1}$

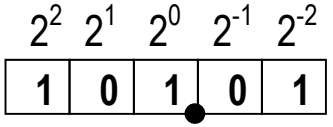
Codificación en punto fijo de reales

Si el número de bits de la codificación es **N** y **b** son fraccionales: **[N,b]**

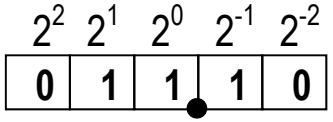
- Codificación sin signo:
  - Rango : $[0, (2^N-1)\cdot Q]$
  - Resolución : $Q=2^{-b}$

$$X = \left(\sum_{i=0}^{N-1} x_i 2^i\right) \cdot Q$$

Ejemplo: N=5, b=2, formato [5,2], rango $\subset (0,7.75)$, resolución=$2^{-2}=0.25$



$X=2^2+2^0+2^{-2}=5.25$



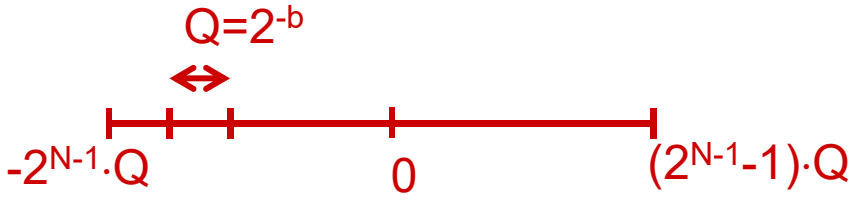
$X=2^1+2^0+2^{-1}=3.5$

Codificación en punto fijo de reales

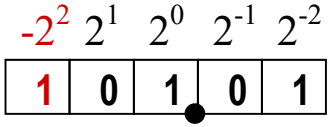
Si el número de bits de la codificación es **N** y **b** son fraccionales: **[N,b]**

- Codificación con signo en complemento a dos:
  - Rango : $[-2^{N-1}\cdot Q, (2^{N-1}-1)\cdot Q]$
  - Resolución : $Q=2^{-b}$

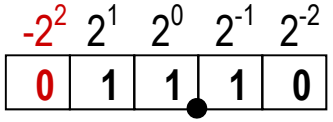
$$X = (-x_{N-1}2^{N-1} + \sum_{i=0}^{N-2} x_i 2^i)\cdot 2^{-b}$$



Ejemplo: N=5, b=2, formato [5,2], rango $\subset (-4,3.75)$, resolución=$2^{-2}=0.25$



$X=-2^2+2^0+2^{-2}= -2.75$

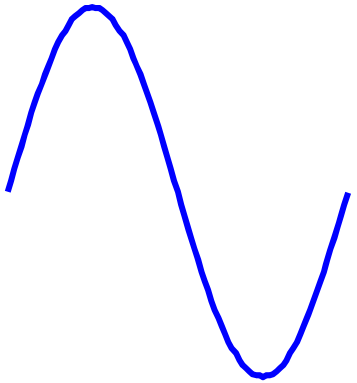


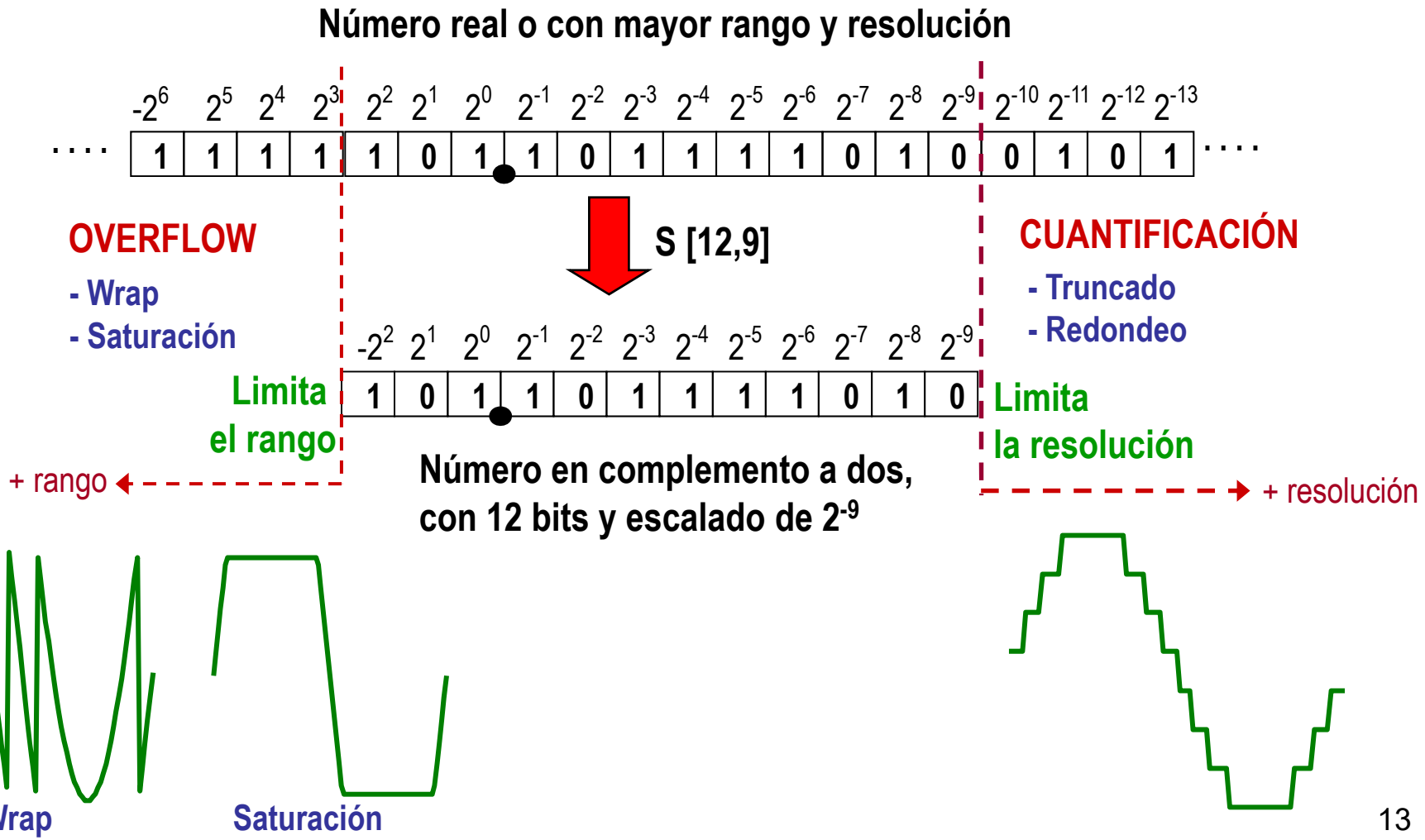
$X=2^1+2^0+2^{-1}=3.5$

Efectos de la precisión finita

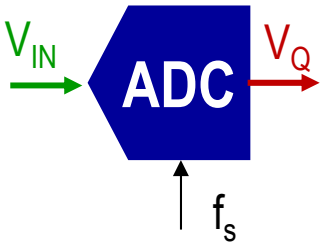
- Un ADC sólo digitaliza la señal analógica dentro de un rango y con una resolución determinada por su número de bits
- Al realizar operaciones aritméticas los datos crecen o aumenta su resolución
  - $\Rightarrow$ no suele ser conveniente dejar crecer siempre los operadores
    - $\Rightarrow$ crece el área, aumenta el consumo de potencia y disminuye la frecuencia de operación
- Limitar el número de bits tiene consecuencias:
  - Disminuir el número de bits enteros reduce el rango
    - $\Rightarrow$ ¡Puede ser catastrófico!
  - Disminuir el número de bits fraccionales reduce la resolución
    - $\Rightarrow$ Siempre introduce ruido
    - $\Rightarrow$ Se busca que no deteriore las prestaciones

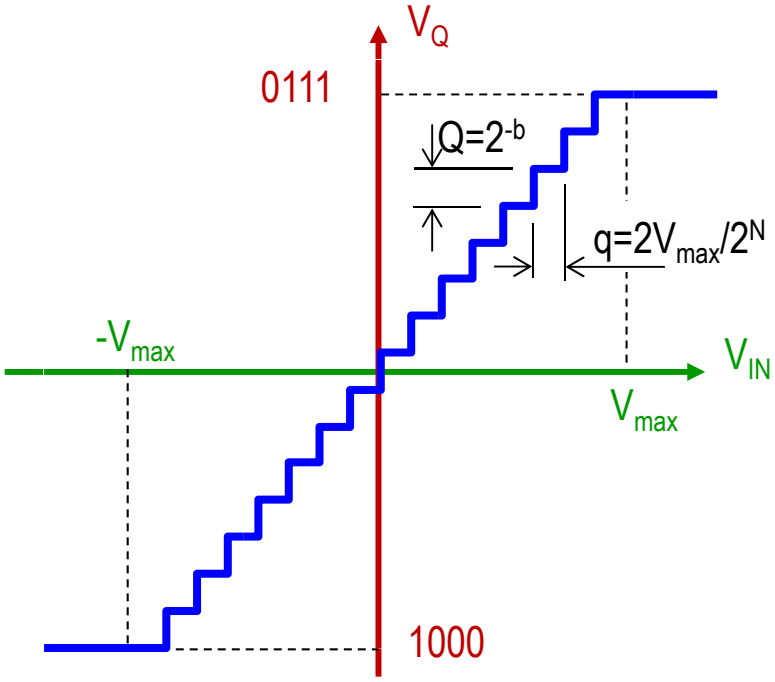
Efectos de la precisión finita



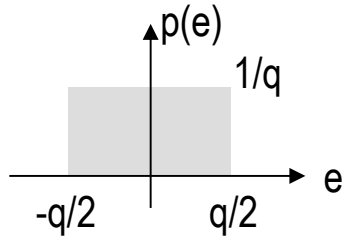


Cuantificación





Función densidad de probabilidad de error

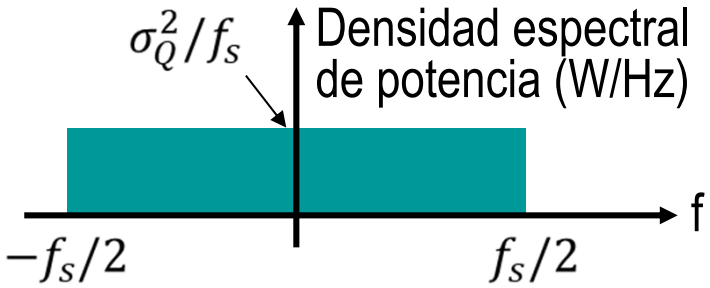


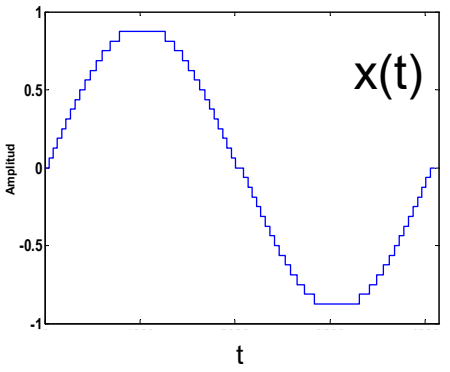
$$\sigma_Q^2 = \frac{q^2}{12}$$

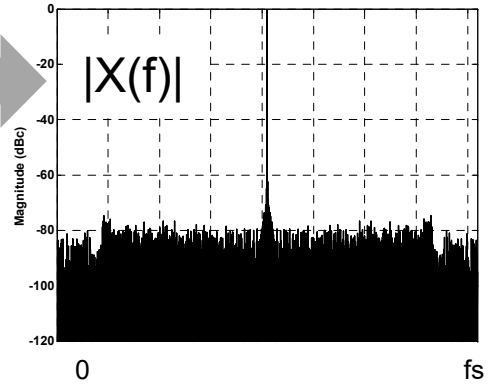
Varianza de ruido

$$\sigma_Q^2 = \int_{-\infty}^{+\infty} e^2 p(e)de = \frac{e^3}{3q}\Big|_{-\frac{q}{2}}^{+\frac{q}{2}} = \frac{q^2}{12}$$

<u>Ruido de cuantificación</u>: ruido blanco que se extiende por toda la banda







Efectos de la cuantificación

Resolución ($Q=2^{-b}$):
peso del bit menos significativo
de la representación (b # bits frac)

- **Truncar:**
  Redondear hacia abajo

$$\text{Error} \le -Q = -2^{-b}$$

$$\text{Error medio} = -\frac{Q}{2} = -2^{-b-1}$$

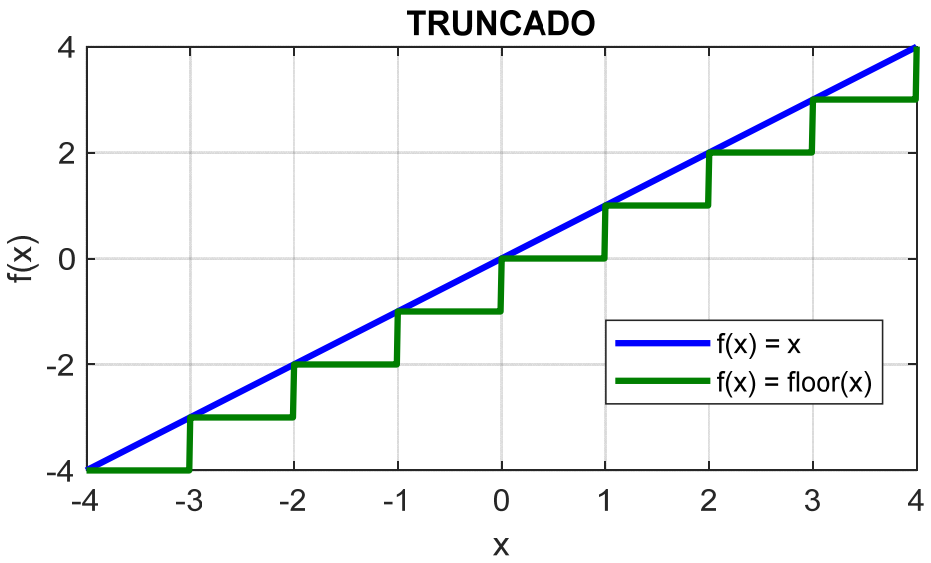
$$\sigma_Q^2 = \frac{Q^2}{12} = \frac{2^{-2b}}{12}$$

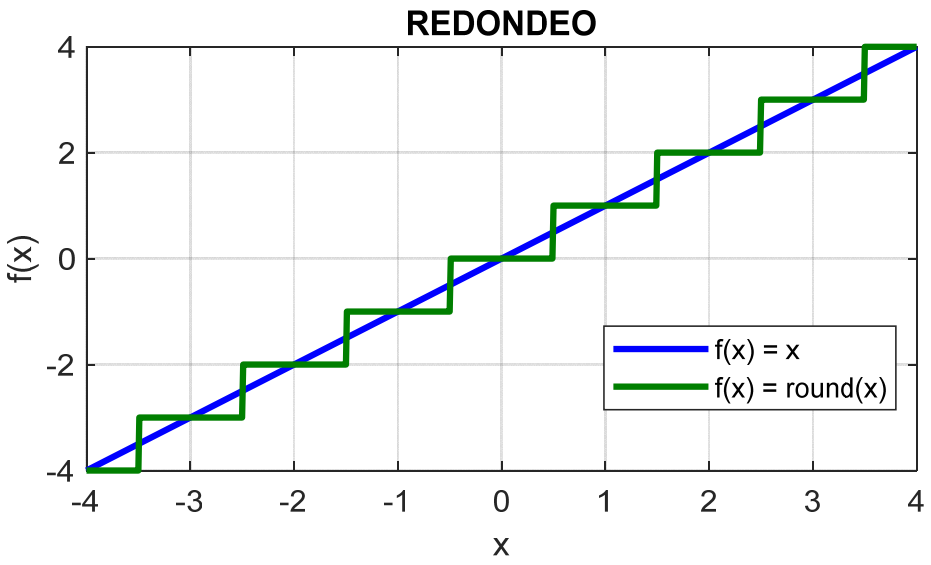
- **Redondear:**
  Redondear hacia el más cercano

$$|\text{Error}| \le \left|\frac{Q}{2}\right| = 2^{-b-1}$$

$$\text{Error medio} = 0$$

$$\sigma_Q^2 = \frac{Q^2}{12} = \frac{2^{-2b}}{12}$$

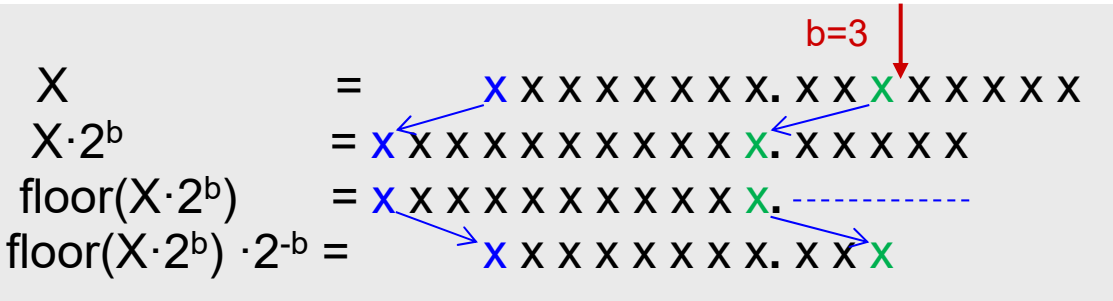




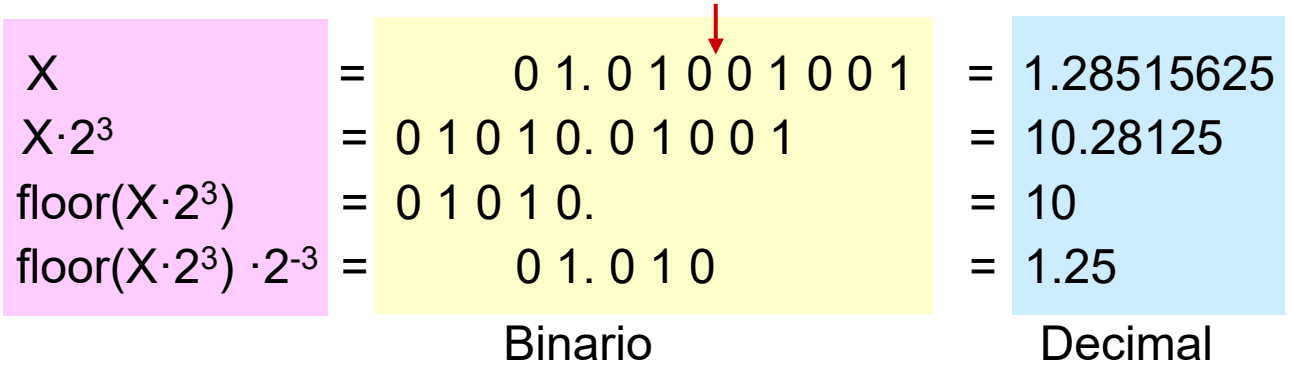
Cuantificación en matlab

- floor(x)/round(x) trunca/redondea la parte entera de x
- Para truncar/redondear en un bit arbitrario (fraccional) hay que hacer la siguiente operación:

```
Xt = floor(X*2^b)*2^-b;
Xr = round(X*2^b)*2^-b;
```



- Ejemplo: trucar dejando hasta el bit fraccional b=3



Efectos de la cuantificación

Modelado con matlab

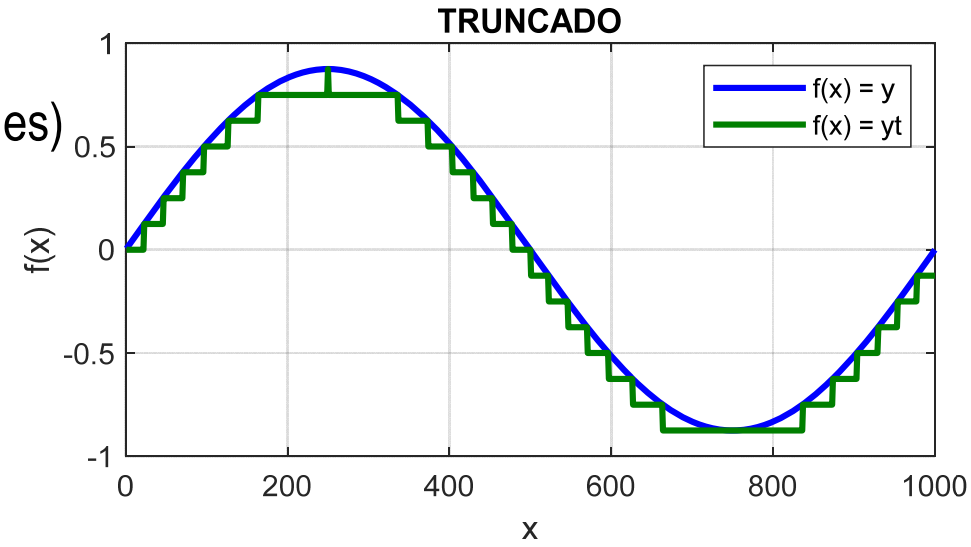
```
Tsim=1000;
b=3;  % Bits de escalado (bits fraccionales)
Resolución =2^-b
t=[1:1:Tsim];
y=(1-2^-b)*sin(2*pi*t/Tsim);
```

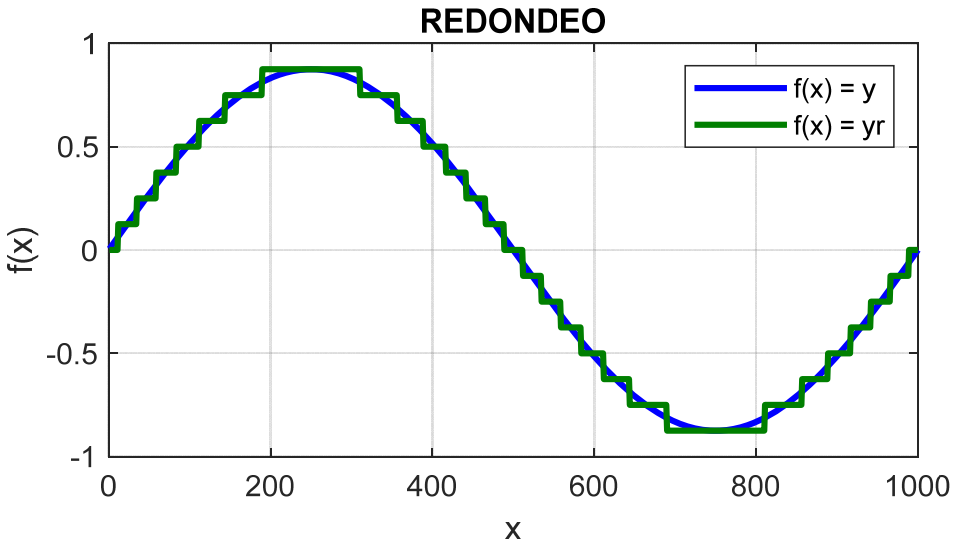
- **Truncar:**

```
yt = floor(y*2^b)*2^-b;
```

- **Redondear:**

```
yr = round(y*2^b)*2^-b;
```





Redondeo: round vs. nearest

Ejemplo: redondear la parte entera sumando el MSB que se trunca

0.25 → 0

000.01 →000
+ 0
000

round(0.25)=0
nearest(0.25)=0

0.75 → 1

000.11 →001
+ 1
001

round(0.75)=1
nearest(0.75)=1

-0.25 → 0

111.11 →000
+ 1
000

round(-0.25)=0
nearest(-0.25)=0

-0.75 → -1

111.01 →111
+ 0
111

round(-0.75)=-1
nearest(-0.75)=-1

2.5 → 3

010.10 →011
+ 1
011

round(2.5)=3
nearest(2.5)=3

0.5 → 1

000.10 →001
+ 1
001

round(0.5)=1
nearest(0.5)=1

-0.5 → 0

111.10 →000
+ 1
000

round(-0.5)= -1
nearest(-0.5)= 0

-2.5 → -2

101.10 →110
+ 1
110

round(-2.5)= -3
nearest(-2.5)= -2

¿Qué ocurre con -x.5?

round: redondeo simétrico con error medio nulo
Al sumar solo el MSB que se trunca, se calcula la operación "**nearest**".
Para hacer el "**round**" es necesario detectar el caso especial:

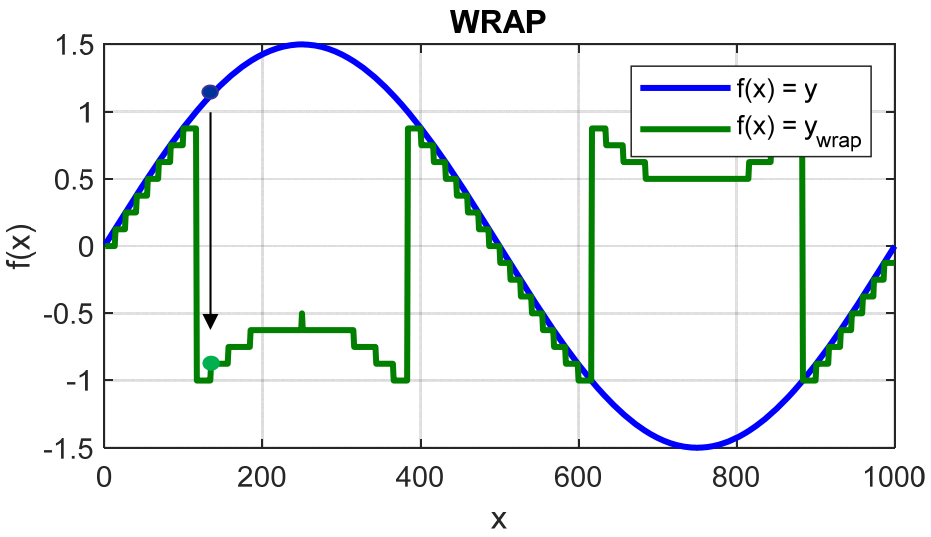
- bit de signo 1 (números negativos) y cadena "1000…0" en bits truncados.

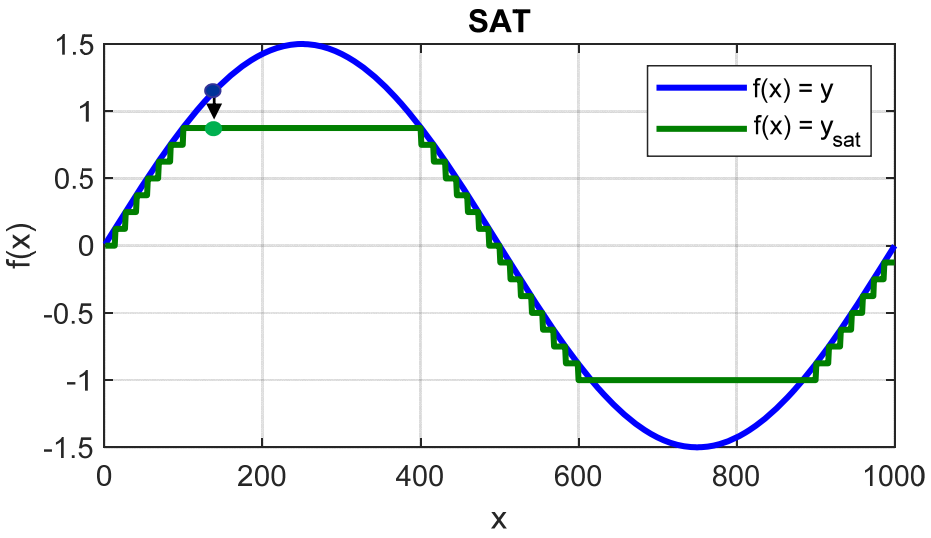
Efectos del desbordamiento (*overflow*)

y=1.5*sin(2*pi*t/Tsim);
N=4; %Numero de bits
b=3; % Bits de escalado
Formato [4,3] ⇒rango: [-1, 1-2^{-3}]

- **Wrap:**
  Elimina bits superiores
  1.125 $(01.001)_{2C}$
  → -0.875 $(1.001)_{2C}$

- **Satura:**
  Si la señal es mayor que el valor máximo representable, se le asigna ese valor
  1.125 $(01.001)_{2C}$
  → 0.875 $(0.111)_{2C}$





Matlab: objeto “quantizer”

q=quantizer(Format, Mode, Roundmode, Overflowmode)

Values sorted by property name:

'Format'

[wordlength fractionlength] - The format for fixed and ufixed mode.

'Mode'

'fixed' - Signed fixed-point mode.

'ufixed' - Unsigned fixed-point mode.

'Roundmode'

'floor' - Round towards negative infinity.

'round' - Round towards nearest integer with greater absolute value

'nearest' - Round towards positive infinity.

'Overflowmode'

'saturate' - Saturate at max value on overflow.

'wrap' - Wrap on overflow.

Matlab: “quantizer” object

q=quantizer(Format, Mode, Roundmode, Overflowmode)

Ejemplo:

```
% Se declara el objeto quantizer
q=quantizer([8 7],'fixed','saturate','round');

% Se aplica con la función quantize
a=0.87

aq=quantize(q,a);
```

Resultado:

```
aq =   0.867187500000000
```

Funciones asociadas: *range, num2bin, bin2num*

Simulink:

Function Block Parameters: Gain

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Main | Signal Attributes | Parameter Attributes

Output minimum: []

Output maximum: []

Output data type: fixdt(1,16,15) <<

Data Type Assistant

Mode: Fixed point Signedness: Signed Word length: 16

Scaling: Binary point Fraction length: 15

Data type override: Inherit Calculate Best-Precision Scaling

Fixed-point details

Lock output data type setting against changes by the fixed-point tools

Integer rounding mode: Floor

Saturate on integer overflow

OK | Cancel | Help | Apply

Operaciones con precisión finita

- Modificación de la resolución/escalado
- Modificación del rango
- Suma/resta
- Sumatorio
- Multiplicación
- Multiplica y suma
- Suma de productos
- Recursividad y precisión finita

Modificación de la resolución/escalado

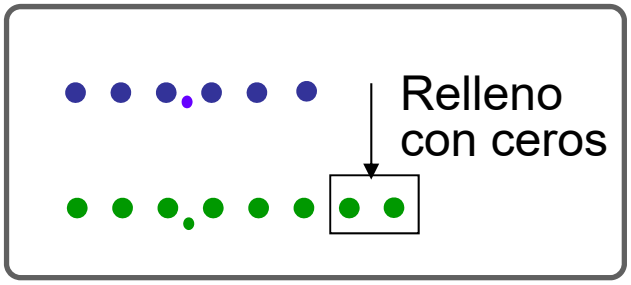
Ampliación de la resolución / reducción de escalado

Ejemplo: $A=2.625 = (010.101)_2$

$A\ [6,3]\quad A=Ae\cdot 2^{-3} \Rightarrow Ae=21 = (010101)_2$

$A'\ [8,5]\quad A'=Ae'\cdot 2^{-5} \Rightarrow Ae'=84 = (01010100)_2$

$\Rightarrow A = A' = 2.625$



Reducción de la resolución / aumento de escalado

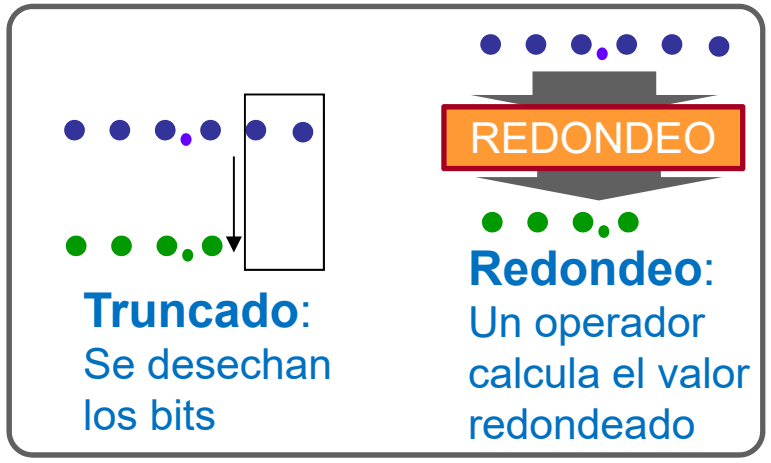
Ejemplo: $A=2.625 = (010.101)_2$

$A\ [6,3]\quad A=Ae\cdot 2^{-3} \Rightarrow Ae=21 = (010101)_2$

$A'\ [4,1]\quad A'=Ae'\cdot 2^{-1} \Rightarrow Ae'=5 = (0101)_2$

$Ae'=\text{floor}(Ae\cdot 2^{-2})=5 \Rightarrow A'=5\cdot 2^{-1} = 2.5$

↑ se multiplica por 2^{-2} para pasar de esc. 2^3 a 2^1



→ Modelado directo: $A'=\text{floor}(A\cdot 2^1)\cdot 2^{-1} = 2.5 \Rightarrow A \neq A' = 2.5$

Modificación del rango

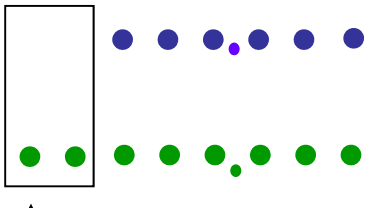
Ampliación del rango

Ejemplo: $A=2.625 = (010.101)_2$

$A\ [6,3]\quad A=Ae\cdot 2^{-3} \Rightarrow Ae=21 = (010101)_2$

$A'\ [8,3]\quad A'=Ae'\cdot 2^{-3} \Rightarrow Ae'=21 = (00010101)_2$

$\Rightarrow A = A' = 2.625$



Relleno con:

- ceros en cod. sin signo
- copias del bit de signo en cod. en complemento a dos

Ejemplo: $A= -1.375 = (11.0101)_2$

$A\ [6,3]\quad A=Ae\cdot 2^{-3} \Rightarrow Ae=-11 = (110101)_2$

$A'\ [8,3]\quad A'=Ae'\cdot 2^{-3} \Rightarrow Ae'=-11 = (11110101)_2$

$\Rightarrow A = A' = -1.375$

Modificación del rango

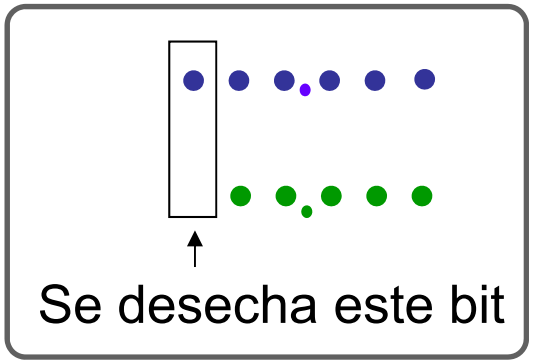
Reducción del rango

Ejemplo con wrap: $A=2.625 = (010.101)_2$

A [6,3] $A=Ae\cdot 2^{-3} \Rightarrow Ae=21 = (010101)_2$

A' [5,3] $A'=Ae'\cdot 2^{-3} \Rightarrow Ae'=-11 = (10101)_2$

$\Rightarrow A \neq A' = -1.375$

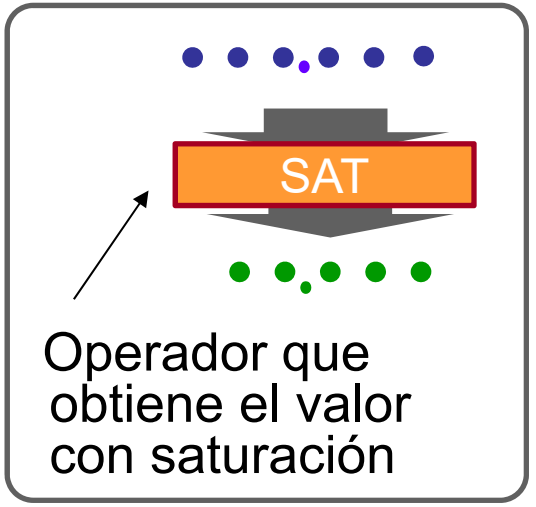


Ejemplo con saturación: $A=2.625 = (010.101)_2$

A [6,3] $A=Ae\cdot 2^{-3} \Rightarrow Ae=21 = (010101)_2$

A' [5,3] $A'=Ae'\cdot 2^{-3} \Rightarrow Ae'=15 = (01111)_2$

$\Rightarrow A \neq A' = 1.875$



$\Rightarrow$ Usar **WRAP** si se puede dimensionar el rango para que **NUNCA** se produzca **OVERFLOW;**

$\Rightarrow$ si no es posible, usar **SATURACIÓN**

Suma/resta: S=A±B

A [Na, a]	$A=Ae\cdot 2^{-a}$
B [Nb, b]	$B=Be\cdot 2^{-b}$
S [Ns, s]	$S=Se\cdot 2^{-s}$

Sólo se puede hacer la suma si ambos operandos tienen el mismo tamaño y escalado (a=b=s)

$$S=A+B=(Ae+Be)\ 2^{-a}=Se\cdot 2^{-a}$$

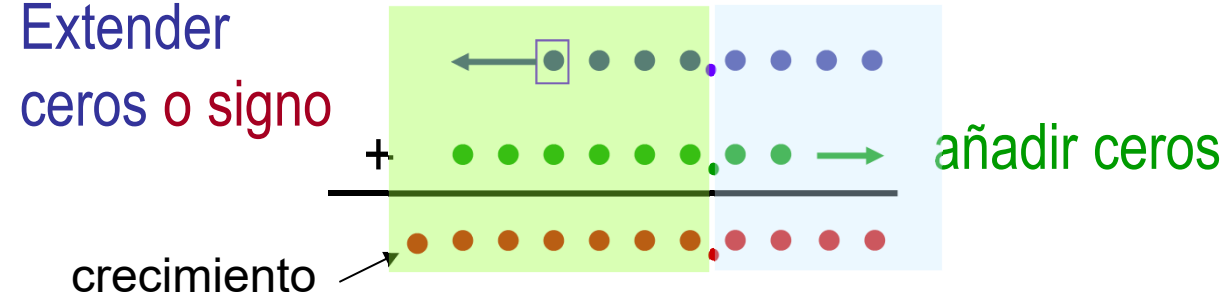
$$Ns = \max\{(Na-a),(Nb-b)\}+1 + \max\{a,b\}$$

Formato de salida de la suma:

⇒ El mismo escalado que el del operando con menor escalado

⇒ Amplia el rango del operador con mayor rango en 1 bit

Ejemplo: B[8,4]; A[8,2] ; S=A+B [11,4]



Suma/resta: S=A±B

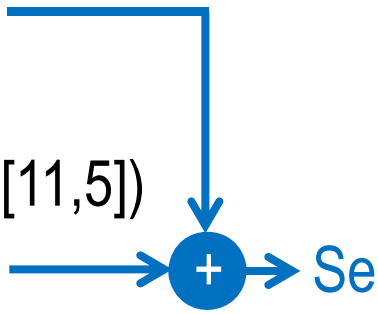
Ejemplo: A[6,5]; B[8,2] ;S=A+B [12,5]

$A=0.53125=Ae\cdot 2^{-5} \Rightarrow Ae=17$ $\quad (010001)_2$

$B=3.25=Be\cdot 2^{-2} \Rightarrow Be=13$ $\quad (00001101)_2$

Hay que cambiar la escala de B a 2^{-5} (nuevo formato de B [11,5])

$B=3.25=Be' \cdot 2^{-5} \Rightarrow Be'=104$ $\quad (00001101000)_2$



Ahora se puede calcular la suma a través de sus enteros equivalentes

$Se=Ae+Be'=17+104=121 \Rightarrow S=Se\cdot 2^{-5}=3.78125$ (=A+B)

En Matlab lo modelaríamos directamente: S = A + B = 3.78125

⇒ Si en la operación se respetan los crecimientos naturales de los operadores, no hay pérdida de precisión:

- ✓no hay diferencia entre operar con el número real o con el formato de precisión finita
- ✓no hay que incluir nada en el modelo (de Matlab) para forzar el formato numérico de precisión finita

Suma/resta: S=A±B

Si limitamos el tamaño del operador manteniendo el escalado del operador que mayor escalado tiene, se pierde precisión en el cálculo

Ejemplo: A[6,5]; B[8,2] ; S=A+B [9,2]

$A=0.53125=Ae\cdot 2^{-5} \Rightarrow Ae=17$

$B=3.25=Be\cdot 2^{-2} \Rightarrow Be=13$

Hay que cambiar la escala de A a 2^{-2} (nuevo formato de A [3,2])

$Ae'=floor(Ae\cdot 2^{-3})=2$ (se multiplica por 2^{-3} para pasar de esc. 2^5 a 2^2)

Ahora se puede calcular la suma a través de sus enteros equivalentes

$Se=Ae'+Be=2+13=15 \Rightarrow S=Se\cdot 2^{-2}=3.75 (\approx A+B)$

En Matlab lo modelaríamos directamente:

S = floor(A*2^2)*2^-2 + B = 3.75

Sumatorio: S=ΣP

Suma de M palabras con el mismo formato

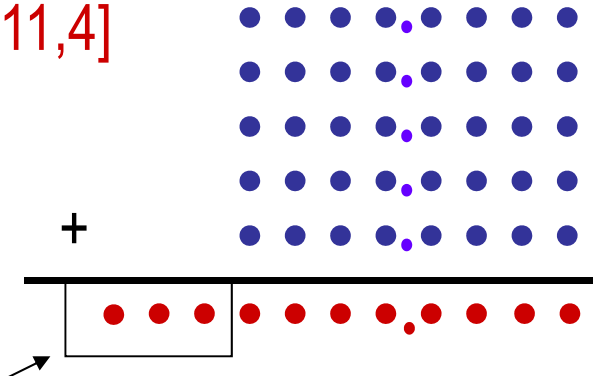
$$S = \sum_{i=0}^{M-1} P_i$$

P [Np,p] $P=Pe\cdot 2^{-p}$
S [Ns,s] $S=Se\cdot 2^{-s}$

s=p
$Ns = Np + ceil(\log_2(M))$

Formato de salida de la suma:
⇒ Se mantiene el escalado
⇒ Hay que ampliar el rango

Ejemplo: M=5; P[8,4] ; S [11,4]



Multiplicación: P=A·B

A [Na, a]
B [Nb, b]
P [Np, p]

$A=Ae\cdot 2^{-a}$
$B=Be\cdot 2^{-b}$
$P=Pe\cdot 2^{-p}$

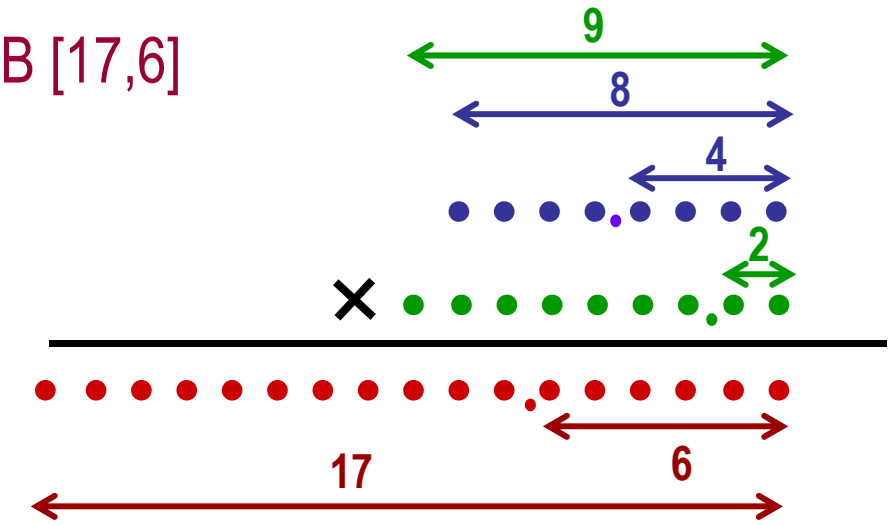
$Np=Na+Nb$
$p=a+b$

El tamaño del multiplicador sólo depende del tamaño de sus operandos

$$\mathbf{P=A\cdot B=(Ae\cdot Be)\cdot 2^{(-a-b)}}$$

Formato de salida de la multiplicación:
⇒ Se amplía el rango y la resolución

Ejemplo: B[8,4]; A[9,2] ; P=A·B [17,6]

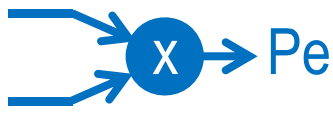


Multiplicación: P=A·B

Ejemplo: A[6,5]; B[8,2] ; P=A·B [14,7]

$A=0.53125=Ae\cdot 2^{-5} \Rightarrow Ae=17$ $\quad (010001)_2$

$B=3.25=Be\cdot 2^{-2} \Rightarrow Be=13$ $\quad (00001101)_2$



Se puede calcular la multiplicación a través de sus enteros equivalentes

$Pe=Ae\cdot Be=17\cdot 13=221 \Rightarrow P=Pe\cdot 2^{-7}=1.7265625$ (P=A·B)

$\Rightarrow$ **Frecuentemente se recorta la precisión de la salida**

Ejemplo: Pr=A·B [12,5]

A través de sus enteros equivalentes:

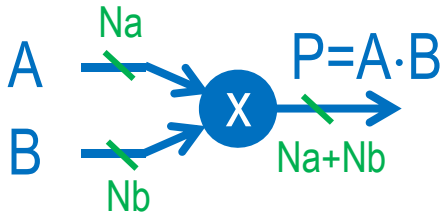
$Pre=floor(Pe\cdot 2^{-2})= 55 \Rightarrow Pr=Pre\cdot 2^{-5}=1.71875$

(se multiplica por 2^{-2} para pasar de escala 2^7 a 2^5)

Directamente:

$Pr=floor(A\cdot B\cdot 2^5)\cdot 2^{-5} = 1.71875$

Multiplicación: P=A·B, caso extremo



El rango en 2'C es asimétrico:

- [N,b] → rango $[-2^{(N-1)}, 2^{(N-1)}-1]$

Caso extremo: de todas las combinaciones de números en A (de Na bits) y B (de Nb bits) solo hay una cuyo producto P=A·B requiere Na+Nb bits:

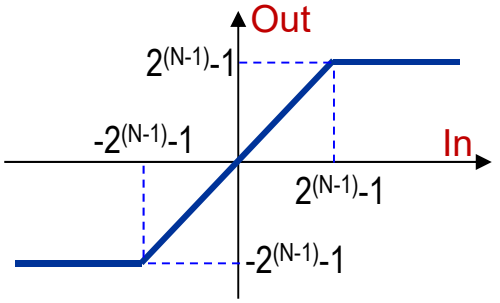
- $A = B = -2^{(N-1)}$

Ej. Números con formato [4,3] tienen un rango [-1,0.875]

- Si multiplicamos (-1)x(-1) = 1⇒ se necesitan 2 bits enteros para representar la salida [8,6]
- Ningún otro caso de multiplicación requiere 2 bits enteros a la salida.

⇒ Si se sabe que no se puede dar ese caso se puede dimensionar la salida con 1 bit entero

⇒ Se puede evitar aplicando una saturación simétrica a la entrada de la cadena de procesado



Multiplicación por cte: P=A·K

A [Na, a]
K [Nk, k]
P [Np, p]

$A=Ae\cdot 2^{-a}$
$K=Ke\cdot 2^{-k}$
$P=Pe\cdot 2^{-p}$

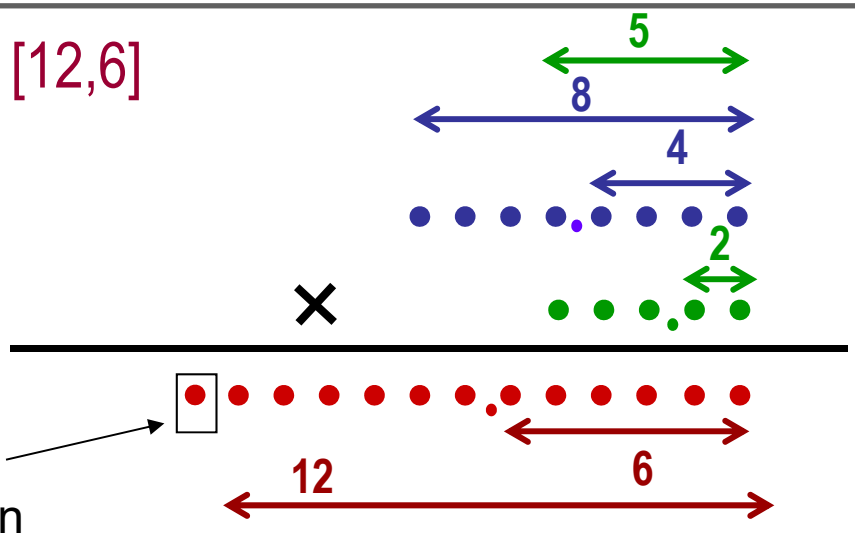
El crecimiento de los datos de salida depende del valor de la constante K

$Np=Na+ceil(\log_2(Ke))$
$p=a+b$

Ejemplo: A[8,4]; K[5,2] ; P=A·K [12,6]

K=2.25
$Ke=2.25\cdot 2^2=9$
$Crecimiento=ceil(\log_2(Ke))=4$

Sobra ese bit
Nunca llevará información



Multiplica y suma: S=A·B+C

S=P+C=A·B+C

A [Na,a] $A=Ae\cdot 2^{-a}$

B [Nb,b] $B=Be\cdot 2^{-b}$

P [Np,p] $P=Pe\cdot 2^{-p}$

C [Nc,c] $C=Ce\cdot 2^{-c}$

S [Ns,s] $B=Be\cdot 2^{-s}$

Como $P=A\cdot B=(Ae\cdot Be)\cdot 2^{(-a-b)}$, si c<(a+b), C debe tener escalado $2^{(-a-b)}$ para realizar correctamente la suma:

$C=Ce\cdot 2^{(-a-b)}$

$S=A\cdot B+C=(Ae\cdot Be+Ce)\cdot 2^{(-a-b)}$

s=max(p,c)

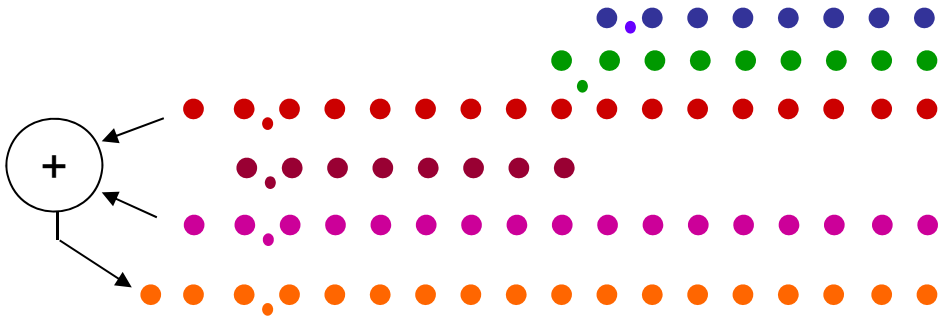
Ns=max{(Np-p),(Nc-c)}+max{p,c}+1

Ejemplo: A [8,7]; B [9,8]; C [8,7]

$P=A\cdot B=(Ae\cdot Be)2^{-(a+b)} \Rightarrow$[17,15]

$C \Rightarrow C'$ [17,15]

$S \Rightarrow$ [18,15]



Multiplica y suma: S=A·B+C

Ejemplo: A[6,5]; B[8,2] ; C [5,4]; S=A·B+C [15,7]

$A=0.53125=Ae\cdot 2^{-5} \Rightarrow Ae=17$

$B=3.25=Be\cdot 2^{-2} \Rightarrow Be=13$

$C=0.3125=Ce\cdot 2^{-4} \Rightarrow Ce=5$

A través de sus enteros equivalentes

$Pe=Ae\cdot Be=17\cdot 13=221$

Ce hay que ponerlo en la misma escala que P

$Ce'=Ce\cdot 2^{3}=40$

$\Rightarrow Se=Pe+Ce'=221+40=261 \Rightarrow S=Se\cdot 2^{-7}=2.0390625$ (S=A·B+C)

Suma de productos: $S= \Sigma(A_i \cdot B_i)$

A [Na,a] $A=Ae \cdot 2^{-a}$
B [Nb,b] $B=Be \cdot 2^{-b}$
S [Ns,s] $S=Se \cdot 2^{-s}$

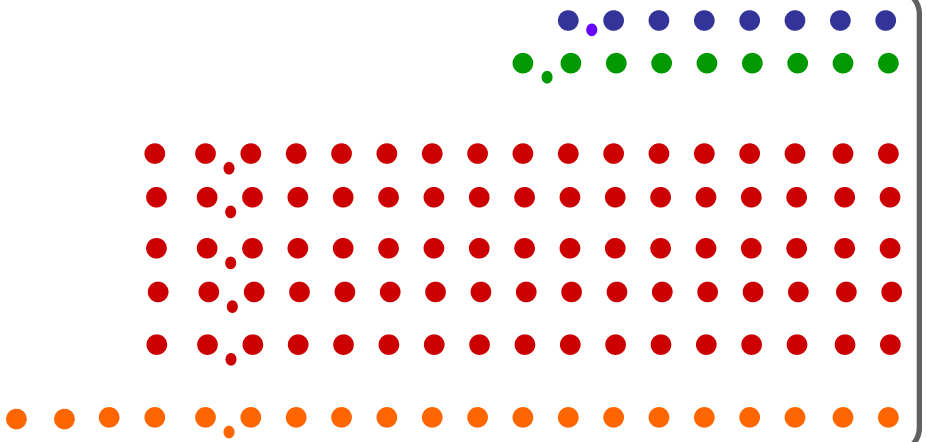
$$S = \sum_{i=0}^{M-1} (A_i \cdot B_i)$$

$s=a+b$
$Ns=Na + Nb + ceil(\log_2(M))$

1 bit menos si no tenemos el caso extremo de la multiplicación

Si uno de los operadores son coeficientes conocidos en la fase de diseño (ej. Filtro FIR), se puede conocer el crecimiento de los datos y reducir los bits de guarda

Ejemplo: A [8,7]; B [9,8]; M=5
$P=A \cdot B=(Ae \cdot Be)2^{-(a+b)} \Rightarrow [17,15]$
$S \Rightarrow [20,15]$

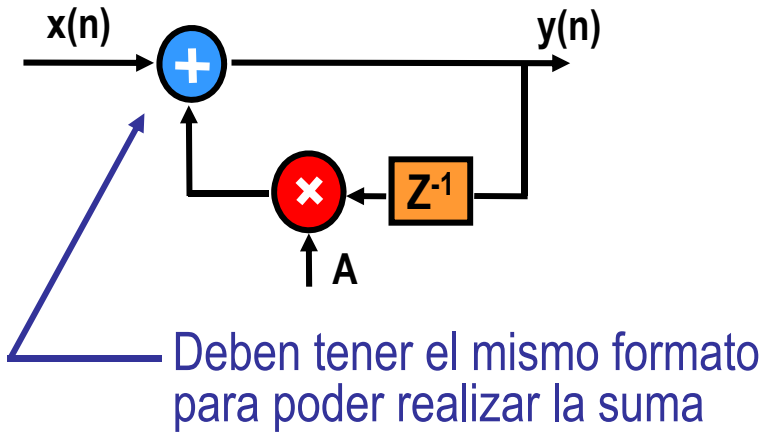


Recursividad y precisión finita

Ejemplo: Integrador

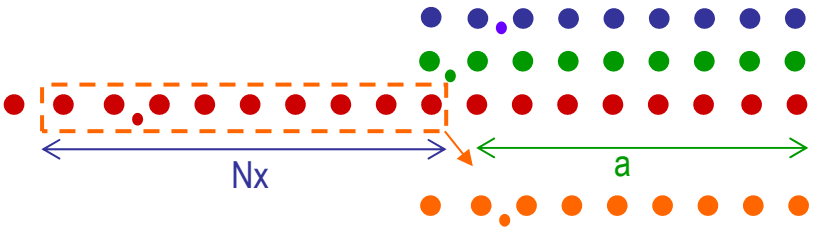
$$y(n) = x(n) + Ay(n-1)$$

X [Nx,x] $X=Xe\cdot 2^{-x}$
Y [Ny,y] $Y=Ye\cdot 2^{-y}$; y=x
A [Na,a] $A=Ae\cdot 2^{-a}$
P [Ny+Na,y+a] $P=Y\cdot A$



Ejemplo:

X, Y $\Rightarrow$[9,7]
A $\Rightarrow$[9,8]
$P=Y\cdot A=(Ye\cdot Ae)2^{-(x+a)} \Rightarrow$[18,15]
$P_Q = Q(P\cdot 2^{-a}) = Pe_Q 2^{-x} \Rightarrow$[9,7]

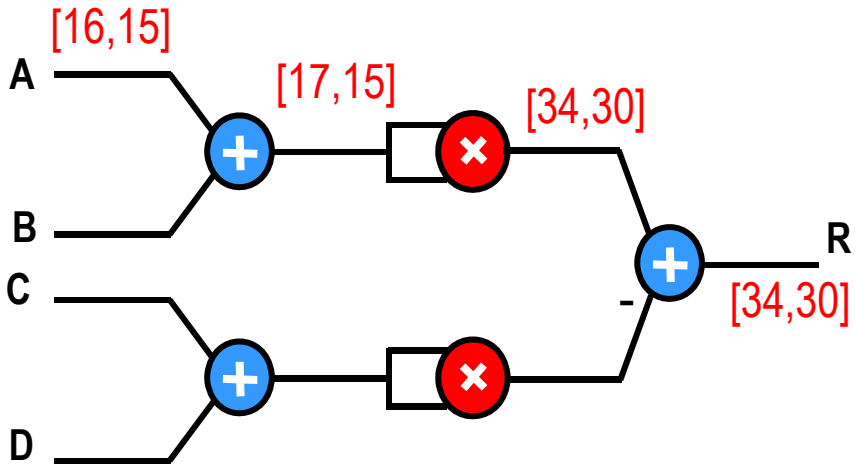


$$Ye(n) = Xe(n) + \text{floor}(Ye(n) * Ae * 2^{-a})$$

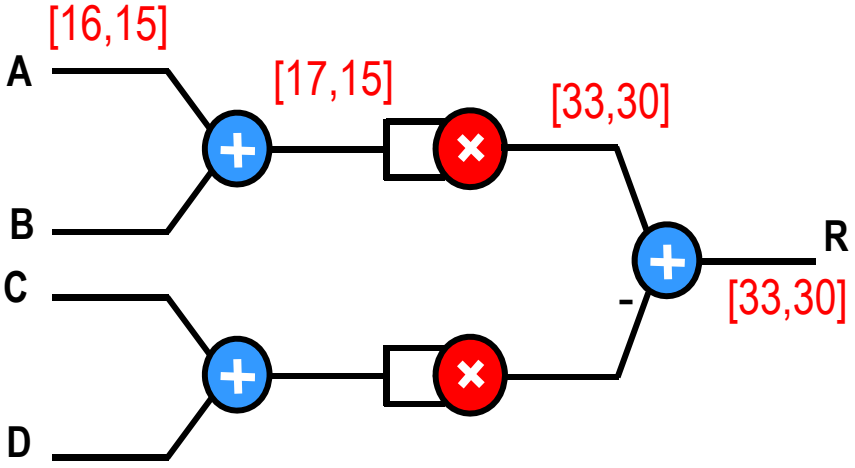
Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V1: Cómputo sin pérdida de precisión

- Formato A, B, C, D:
  - S[16,15], rango [-1,1-2^{-15}]
- Formato R
  - rango de posibles R [-4,4]
    - casos extremos A=B=-1, C=D=0 o A=B=0, C=D=-1
  - S[34,30], rango [-8,8-2^{-30}]



- Formato A, B, C, D:
  - S[16,15] con **saturación simétrica**, rango [-1+2^{-15},1-2^{-15}]
- Formato R
  - rango de posibles R]-4,4[
    - casos extremos A=B=1-2^{-15}, C=D=0 o A=B=0, C=D=1-2^{-15}
  - S[33,30], rango [-4,4-2^{-30}]

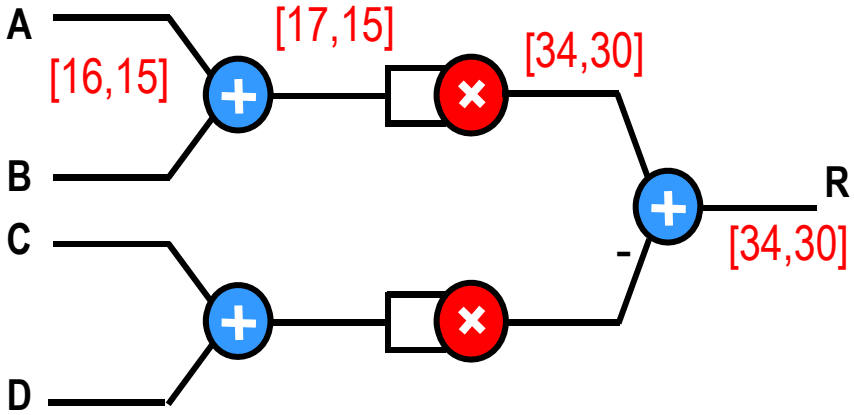


Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V1: Cómputo sin pérdida de precisión

Matlab

```
r=(a+b)^2-(c+d)^2
```



a	0.003265380859375	0.000366210937500	0.017181396484375	-1
b	0.000213623046875	0.015258789062500	0.121887207031250	-1
c	0.000518798828125	0.006103515625000	0.047698974609375	0
d	0.094818115234375	0.000121421180665	0.017181396484375	0
r	-0.009077023714781	0.000000477768481	0.015130613930523	4

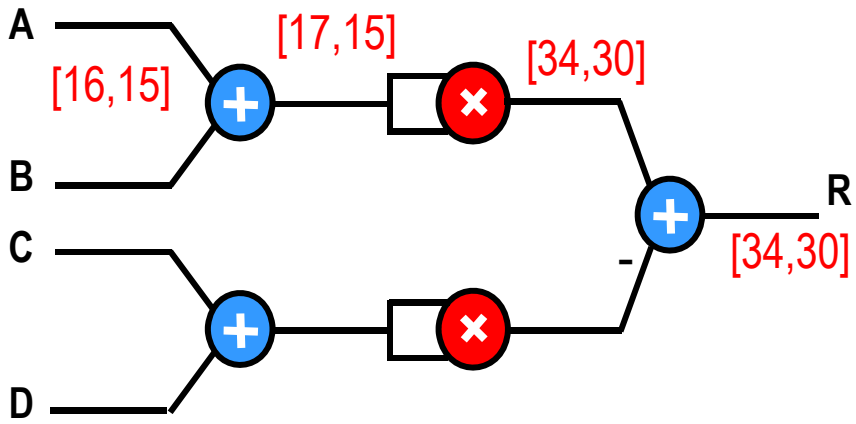
$x2^{15}$	A	107	12	563	-32768
	B	7	500	3994	-32768
	C	17	200	1563	0
	D	3107	163	563	0
$x2^{30}$	R	-9746380	130375	16246373	4294967296

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V1: Cómputo sin pérdida de precisión

Verilog

```
module RESTA_SUM_CUAD1 (A,B,C,D,R);
input signed [15:0] A,B,C,D;
output wire signed [33:0] R;

wire signed [16:0] S1,S2;
wire signed [33:0] M1,M2;

assign S1 = A+B;
assign S2 = C+D;
assign M1 = S1*S1;
assign M2 = S2*S2;
assign R = M1-M2;

endmodule
```



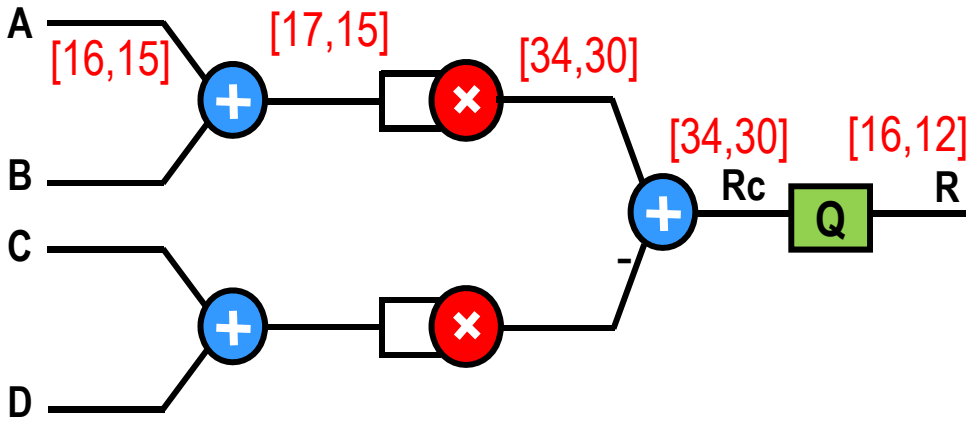
A	107	12	563	-32768
B	7	500	3994	-32768
C	17	200	1563	0
D	3107	163	563	0
R	-9746380	130375	16246373	4294967296

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V2: Sin pérdida de precisión interna y salida truncada a [16,12]

Matlab

```
rc=(a+b)^2-(c+d)^2;
r=floor(rc*2^12)*2^-12
- - - - - - - - - - - - - - - - - - -
qf=quantizer([16 12],'fixed','wrap','floor')
r=quantize(qf,rc)
```



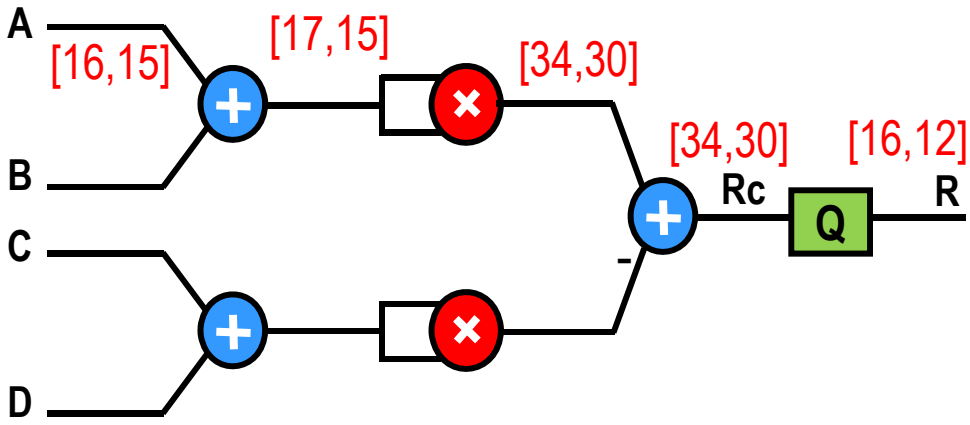
	a	0.003265380859375	0.000366210937500	0.017181396484375	-1
	b	0.000213623046875	0.015258789062500	0.121887207031250	-1
	c	0.000518798828125	0.006103515625000	0.047698974609375	0
	d	0.094818115234375	0.000121421180665	0.017181396484375	0
	r	-0.009277343750000	0	0.014892578125000	4
$x2^{15}$	A	107	12	563	-32768
	B	7	500	3994	-32768
	C	17	200	1563	0
	D	3107	163	563	0
$x2^{12}$	R	-38	0	61	16384

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V2: Sin pérdida de precisión interna y salida truncada a [16,12]

Verilog

```
module RESTA_SUM_CUAD2 (A,B,C,D,R);
input signed [15:0] A,B,C,D;
output  wire signed[15:0] R;

wire signed [16:0] S1,S2;
wire signed [33:0] M1,M2,Rc;

assign S1 = A+B;
assign S2 = C+D;
assign M1 = S1*S1;
assign M2 = S2*S2;
assign Rc = M1-M2;
assign R=Rc[33:18];

endmodule
```



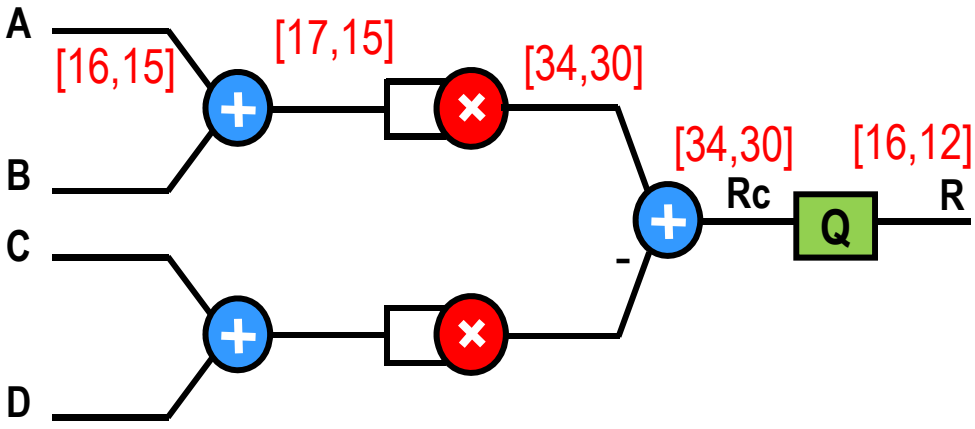
A	107	12	563	-32768
B	7	500	3994	-32768
C	17	200	1563	0
D	3107	163	563	0
R	-38	0	61	16384

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V3: Sin pérdida de precisión interna y salida redond. con *nearest* a [16,12]

Matlab

```
rc=(a+b)^2-(c+d)^2;
r=nearest(rc*2^12)*2^-12
- - - - - - - - - - - - - - - - - - -
qn=quantizer([16 12],'fixed','wrap','nearest')
r=quantize(qf,rc)
```



a	0.003265380859375	0.000366210937500	0.017181396484375	-1
b	0.000213623046875	0.015258789062500	0.121887207031250	-1
c	0.000518798828125	0.006103515625000	0.047698974609375	0
d	0.094818115234375	0.000121421180665	0.017181396484375	0
r	-0.009033203125000	0	0.015136718750000	4

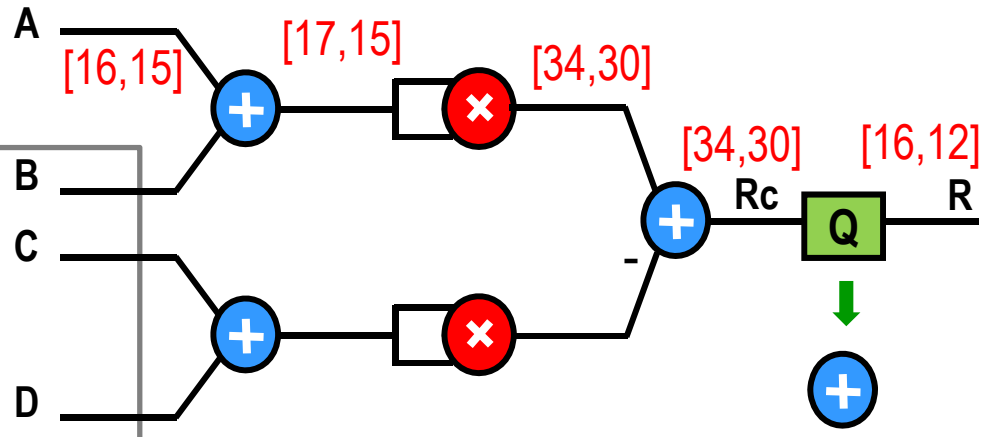
$x2^{15}$	A	107	12	563	-32768
	B	7	500	3994	-32768
	C	17	200	1563	0
	D	3107	163	563	0
$x2^{12}$	R	-37	0	62	16384

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V3: Sin pérdida de precisión interna y salida redond. con *nearest* a [16,12]

Verilog

```
module RESTA_SUM_CUAD3(A,B,C,D,R);
input signed [15:0] A,B,C,D;
output wire signed [15:0] R;

wire signed [16:0] S1,S2;
wire signed [33:0] M1,M2,Rc;

assign S1 = A+B;
assign S2 = C+D;
assign M1 = S1*S1;
assign M2 = S2*S2;
assign Rc = M1-M2;
assign R=Rc[33:18] + Rc[17];

endmodule
```



A	107	12	563	-32768
B	7	500	3994	-32768
C	17	200	1563	0
D	3107	163	563	0
R	-37	0	62	16384

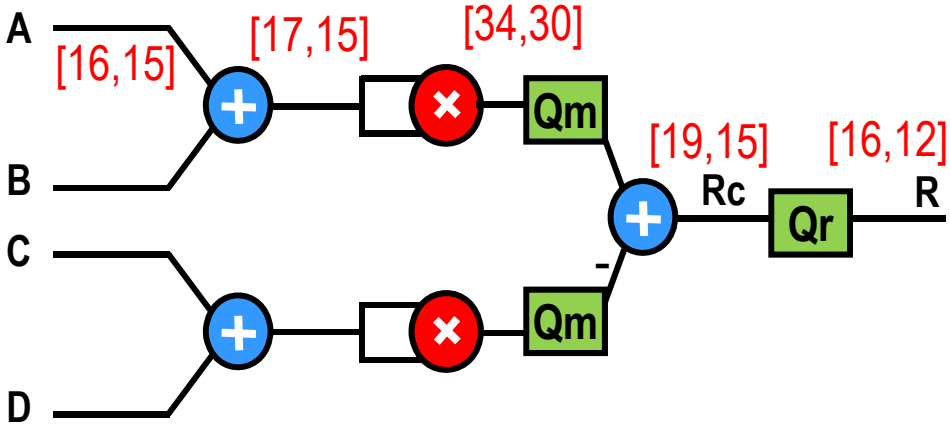
Al sumar solo el MSB que se trunca, se calcula la operación "**nearest**".

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V4: Con multiplicador truncado a [19,15] y salida redondeada a [16,12]

Matlab

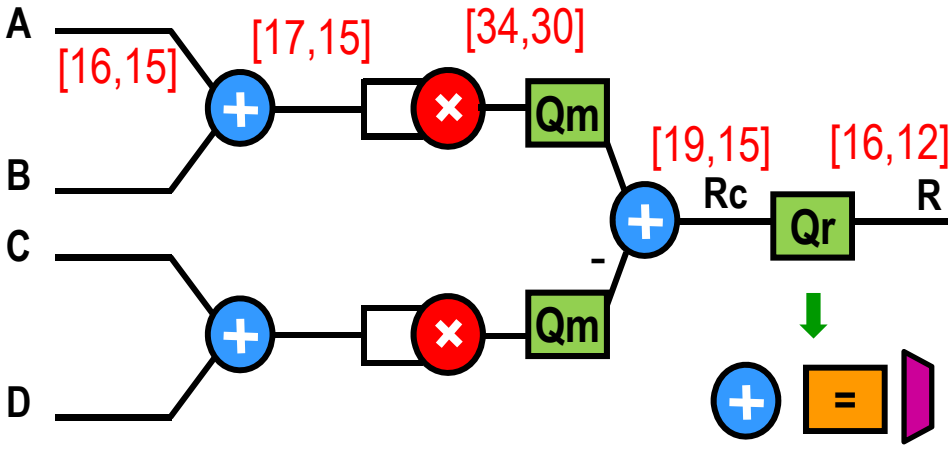
```
m1=floor((a+b)^2*2^15)*2^-15;
m2=floor((c+d)^2*2^15)*2^-15;
r=round((m1-m2)*2^12)*2^-12;

- - - - - - - - - - - - - - - - - - - -

qm=quantizer([19 15],'fixed','wrap','floor')
qr=quantizer([16 12],'fixed','wrap','round')
m1=quantize(qm,(a+b)^2);
m2=quantize(qm,(c+d)^2);
r=quantize(qr,(m1-m2));
```



	d	0.094818115234375	0.000121421180665	0.017181396484375	0
	r	-0.009033203125000	0.000244140625000	0.015136718750000	4
$x2^{15}$	A	107	12	563	-32768
	B	7	500	3994	-32768
	C	17	200	1563	0
	D	3107	163	563	0
$x2^{12}$	R	-37	1	62	16384

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V4: Con multiplicador truncado a [19,15] y salida redondeada a [16,12]



Verilog

```
module RESTA_SUM_CUAD4(A,B,C,D,R);
input signed [15:0] A,B,C,D;
output wire signed [15:0] R;

wire signed [16:0] S1,S2;
wire signed [33:0] M1,M2;
wire signed [18:0] Rc;

assign S1 = A+B;
assign S2 = C+D;
assign M1 = S1*S1;
assign M2 = S2*S2;

assign Rc = M1[33:15]-M2[33:15];
assign R = ((Rc[18] & (Rc[2:0]))==3'b100) ? Rc[18:3] : (Rc[18:3] + Rc[2]);

endmodule
```

A	107	12	563	-32768
B	7	500	3994	-32768
C	17	200	1563	0
D	3107	163	563	0
R	-37	1	62	16384

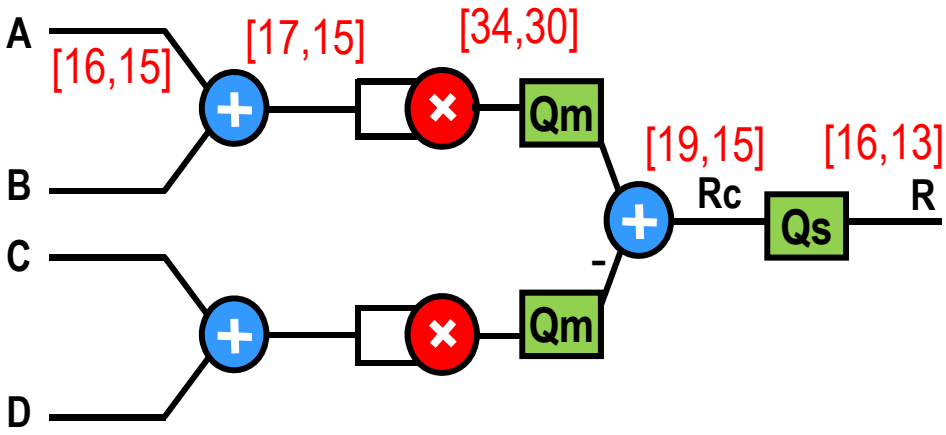
round: redondeo simétrico con error medio nulo
Al sumar solo el MSB que se trunca, se calcula la operación "**nearest**".
Para hacer el "**round**" es necesario detectar el caso especial:
- bit de signo 1 (números negativos) y cadena "1000…0" en bits truncados.

Ej. Modelado de PF: $R=(A+B)^2-(C+D)^2$

V5: Con multiplicador truncado a [19,15] y salida redond. y sat. a [16,13]

Matlab

```
m1=floor((a+b)^2*2^15)*2^-15;
m2=floor((c+d)^2*2^15)*2^-15;
rr=round((m1-m2)*2^13)*2^-13;
if rr >= 4-2^-13
   r = 4-2^-13;
elseif rr < -4
   r = -4;
else r = rr;
end
   - - - - - - - - - - - - - - - - - -
qm=quantizer([19 15],'fixed','wrap','floor')
qs=quantizer([16 13],'fixed', 'saturate','round')
m1=quantize(qm,(a+b)^2);
m2=quantize(qm,(c+d)^2);
r=quantize(qs,(m1-m2));
```



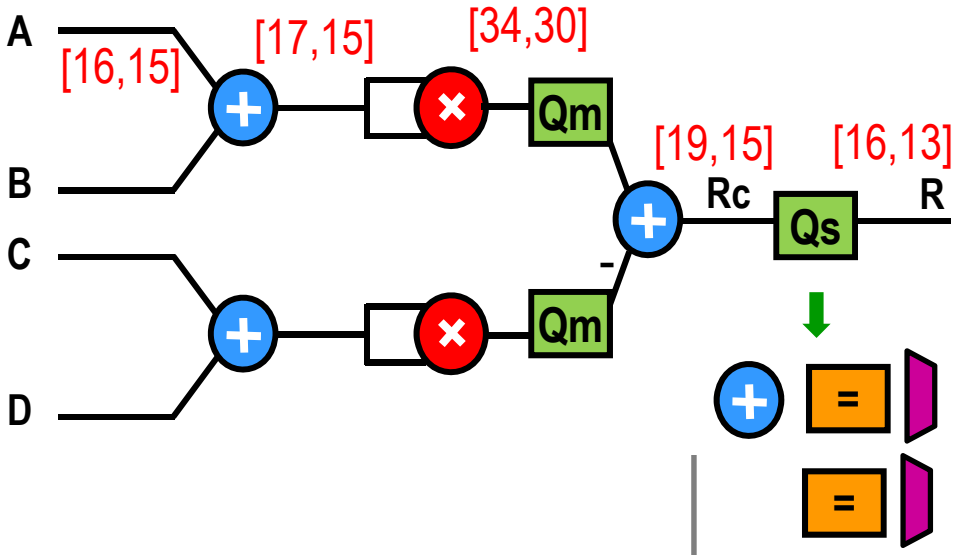
r	R
-0.009033203125000	-74
0.000122070312500	1
0.015136718750000	124
3.999877929687500	32767

$x2^{13}$

V5: $R=(A+B)^2-(C+D)^2$

Verilog

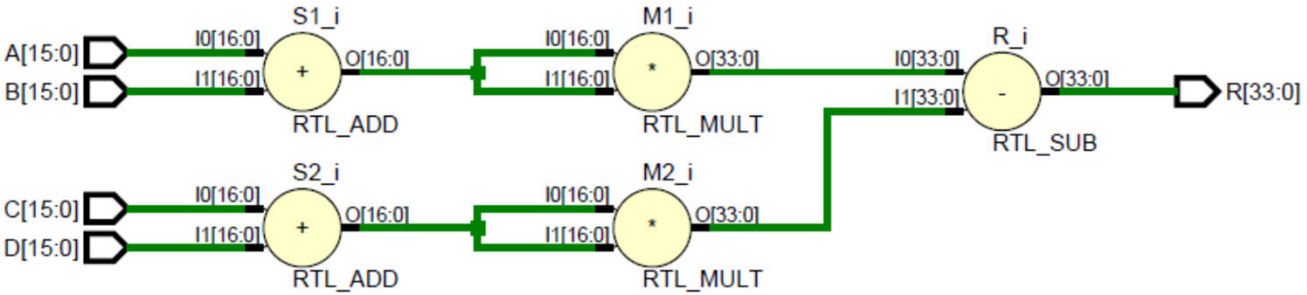
```
module RESTA_SUM_CUAD5(A,B,C,D,R);
input signed [15:0] A,B,C,D;
output  wire signed [15:0] R;

wire signed [16:0] S1,S2;
wire signed [33:0] M1,M2;
wire signed [18:0] Rc;
wire signed [16:0] Rr;

wire signed [16:0] SATpos = 32767;
wire signed [16:0] SATneg = -32768;

assign S1 = A+B;
assign S2 = C+D;
assign M1 = S1*S1;
assign M2 = S2*S2;
assign Rc = M1[33:15]-M2[33:15];

assign Rr = (Rc[18] & (Rc[1:0]==2'b10)) ? Rc[18:2] : (Rc[18:2] + Rc[1]);
assign R = (Rr > SATpos) ? SATpos[15:0] : (Rr<SATneg) ? SATneg[15:0] : Rr[15:0];

endmodule
```



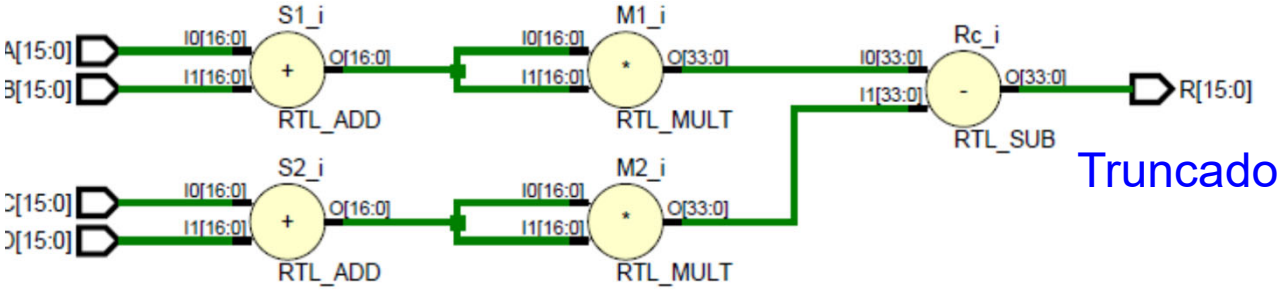
A	107	12	563	-32768
B	7	500	3994	-32768
C	17	200	1563	0
D	3107	163	563	0
R	-74	1	124	32767

Esquemas RTL : $R=(A+B)^2-(C+D)^2$

V1

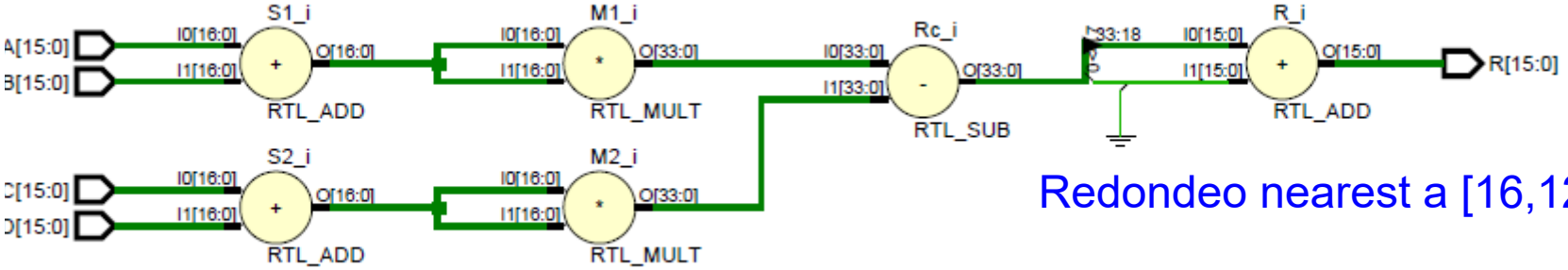


V2



Truncado a [16,12]

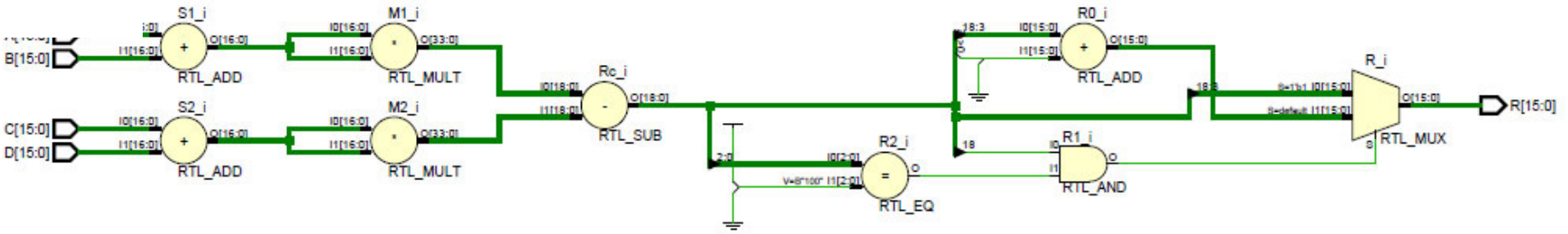
V3



Redondeo nearest a [16,12]

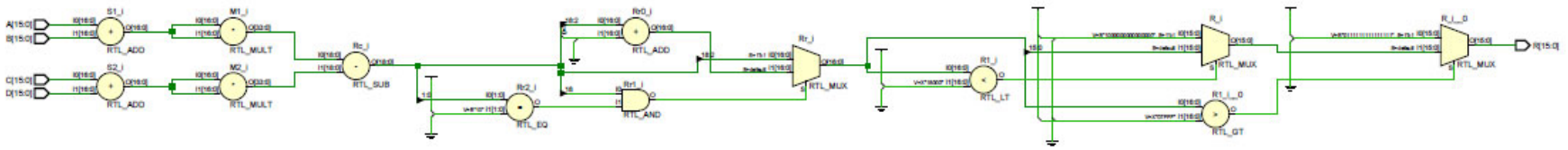
Esquemas RTL : $R=(A+B)^2-(C+D)^2$

V4



Redondeo round a [16,12]

V5



Redondeo saturación a [16,13]

La selección del formato numérico no sólo afecta a los resultados numéricos
También afecta al área y tiempo de propagación del circuito que se implementa

Modelado con precisión finita de algoritmos

- Objetivo:
  $\Rightarrow$ Limitar los tamaños de palabra sin que se vean deterioradas las prestaciones

- ¿Cómo sabemos que no se deterioran las prestaciones?
  $\Rightarrow$ Se necesita un modelo de referencia "*golden model*" para compararse
  $\Rightarrow$ Hay que establecer la medida de "calidad": exactitud, SNR, BER…

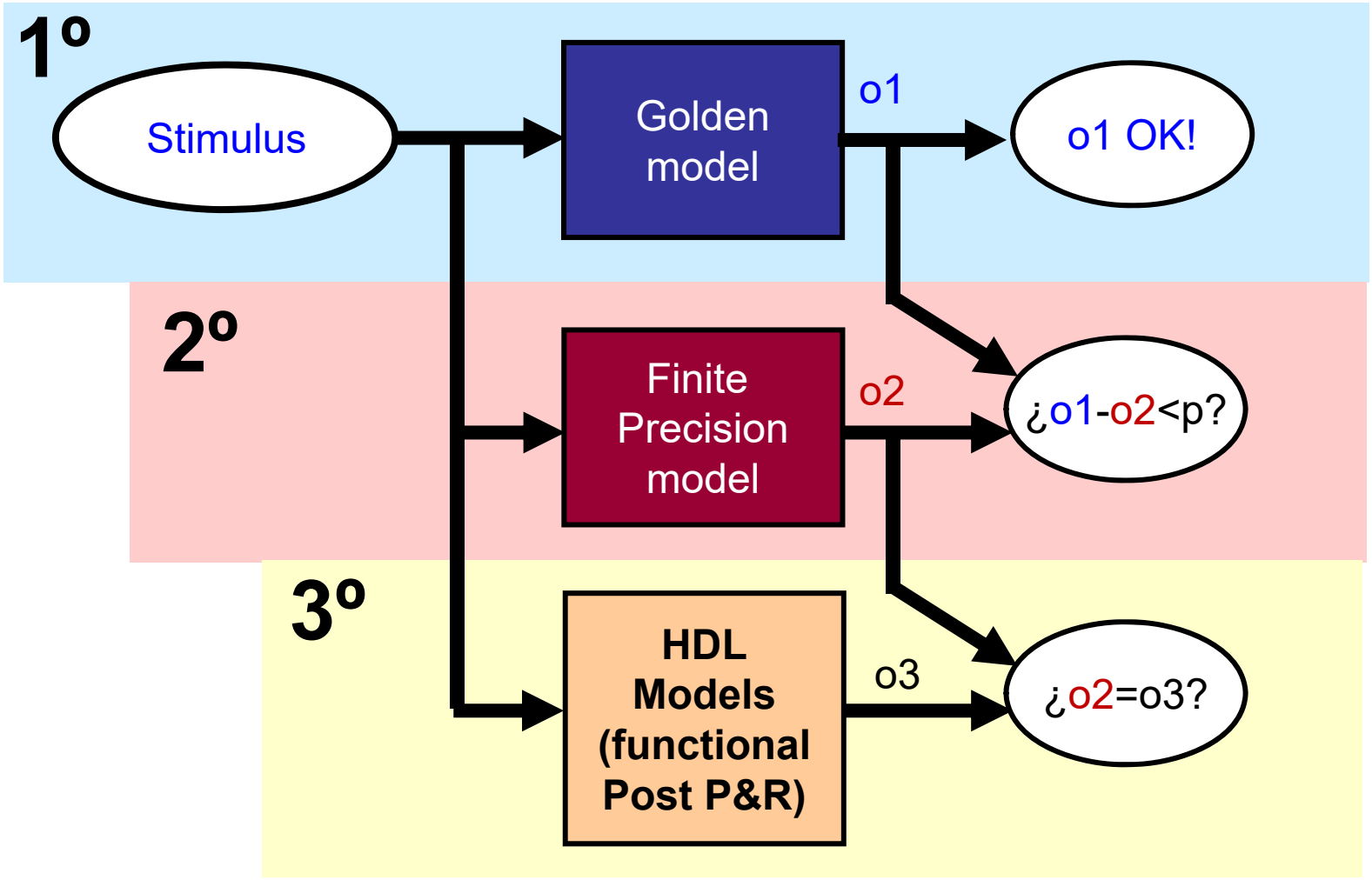
- Método de aplicación:
  1. Codificar el modelo de referencia
     - Excitarlo con las señales típicas de la aplicación
       - Utilizar amplitudes máximas y mínima admisibles
     - Visualizar variables intermedias para conocer su crecimiento
  2. Cuantificar coeficientes y comparar con el modelo de referencia
     - Una vez decidida la cuantificación de coeficientes, dejarlos cuantificados para evaluar la cuantificación de la ruta de datos
  3. Cuantificar datos y comparar con el modelo de referencia

Modelado con precisión finita de algoritmos

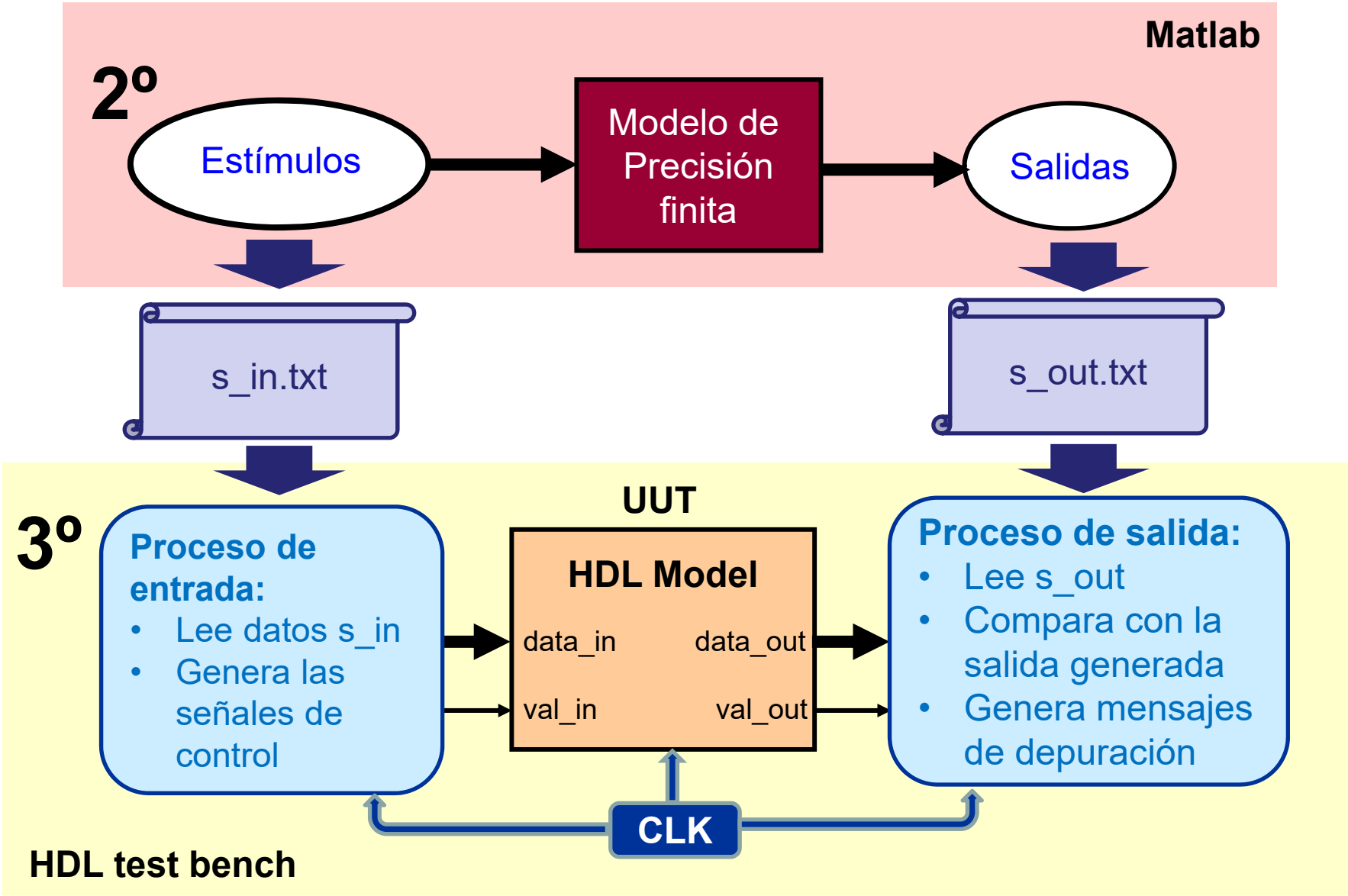
- Método de aplicación:
  - La aplicación de 2 y 3 es secuencial:
    - Se empieza aplicándolo desde la entrada y propagándolo hacia la salida
    - Cada vez que se cuantifica un coeficiente o variable se debe comprobar que el modelo sigue siendo correcto
  - La aplicación de 2 y 3 es iterativa:
    - Suele ser común el volver atrás y ajustar las cuantificaciones

$\Rightarrow$ Realizar modelos de precisión finita es un proceso laborioso que consume mucho tiempo de diseño y simulación

Flujo de verificación



Flujo de verificación



Conclusiones

- En los dispositivos FPGA los algoritmos se implementan con precisión finita (mayoritariamente con formato de punto fijo)
- El modelado de precisión finita de los algoritmos es necesario para:
  - determinar si el algoritmo alcanza las prestaciones deseadas al implementarse con precisión finita
  - para facilitar la depuración y verificación del sistema
- En este tema:
  - se han presentado las herramientas y metodología para modelar algoritmos con precisión finita
  - y se ha presentado el flujo de verificación hardware de un módulo que implemente un algoritmo, utilizando su modelo de precisión finita