Multipliers and Adders

April 23, 2024

1 Notación

Asumiremos que un vector binario x tiene la siguiente forma:

x_{N-1}	x_{N-2}	$\cdots$	x_3	x_2	x_1	x_0

2 Basic building blocks

2.1 Half adder

a_i	b_i	c_{i+1}	s_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\begin{aligned} s_i &= a_i \oplus b_i \\ c_{i+1} &= a_i.b_i \end{aligned} \tag{1}$$

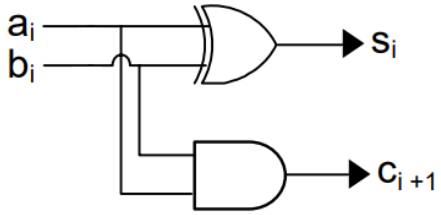

Figure 1: Half adder.

2.2 Full adder

a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$\begin{aligned} s_i &= a_i \oplus b_i \oplus c_i \\ c_{i+1} &= (a_i \oplus b_i).c_i + a_i.b_i \end{aligned} \tag{2}$$

3 Suma en codificación sin signo (US)

$$\begin{aligned} s &= x + y \\ &= \sum_{i=0}^{N-1} x_i.2^i + \sum_{i=0}^{N-1} y_i.2^i \\ &= \sum_{i=0}^{N-1} (x_i + y_i).2^i \end{aligned} \tag{3}$$

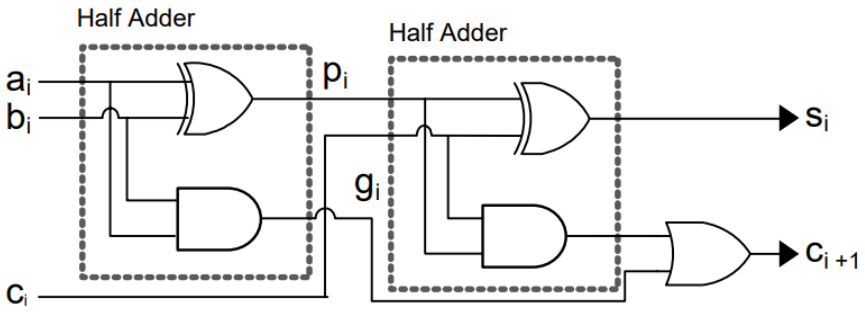

Figure 2: Full adder.

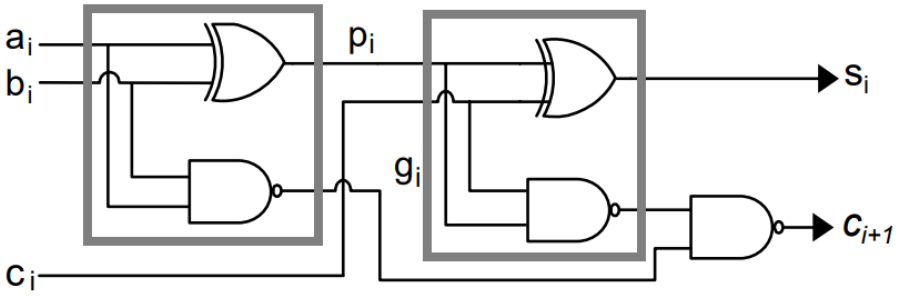

Figure 3: Full adder implementado con NANDs en el cómputo del carry.

3.1 Ripple carry adder - RCA

La implementación más sencilla es la llamada ripple carry adder

$$\begin{array}{cccccccc} & & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\ + & & & & & & & \\ \hline & & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 \\ \hline & s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0 \end{array}$$

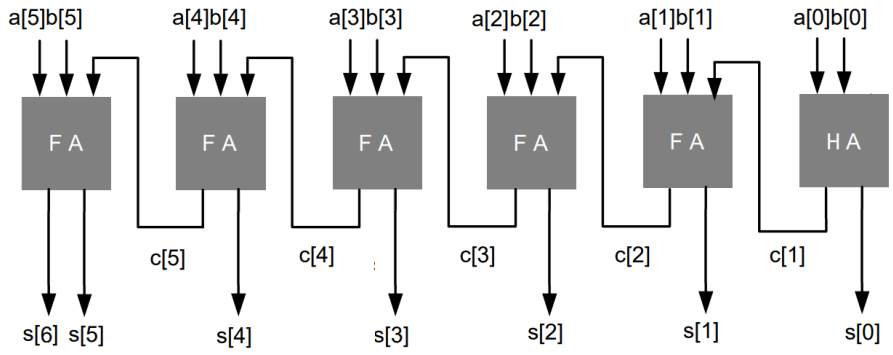

Figure 4: Riple carry adder.

3.2 Carry lookahead adder - CLA

Llamando

$$\begin{array}{rcll} g_i & = & a_i.b_i & \text{(Generate)} \\ p_i & = & a_i \oplus b_i & \text{(Propagate)} \end{array} \tag{4}$$

Entonces las ecuaciones del full-adder se convierten en:

$$\begin{array}{rcccl} s_i & = & a_i \oplus b_i \oplus c_i & = & p_i \oplus c_i \\ c_{i+1} & = & (a_i \oplus b_i).c_i + a_i.b_i & = & p_i.c_i + g_i \end{array} \tag{5}$$

Si por ejemplo, $N = 4$ bits, entonces se puede escribir

$$\begin{array}{rcl} c_1 & = & p_0.c_0 + g_0 \\ c_2 & = & p_1.c_1 + g_1 \\ c_3 & = & p_2.c_2 + g_2 \\ c_4 & = & p_3.c_3 + g_3 \end{array} \tag{6}$$

Sustituyendo c_1 en c_2, c_2 en c_3 y así sucesivamente:

$$\begin{array}{rcl} c_1 & = & p_0.c_0 + g_0 \\ c_2 & = & p_1.(p_0.c_0 + g_0) + g_1 = p_1.p_0.c_0 + p_1.g_0 + g_1 \\ c_3 & = & p_2.(p_1.p_0.c_0 + p_1.g_0 + g_1) + g_2 = p_2.p_1.p_0.c_0 + p_2.p_1.g_0 + p_2.g_1 + g_2 \\ c_4 & = & p_3.(p_2.p_1.p_0.c_0 + p_2.p_1.g_0 + p_2.g_1 + g_2) + g_3 = p_3.p_2.p_1.p_0.c_0 + p_3.p_2.p_1.g_0 + p_3.p_2.g_1 + p_3.g_2 + g_3 \end{array} \tag{7}$$

En general,

$$c_i = \prod_{k=0}^{i-1} p_j c_0 + \sum_{j=0}^{i-2} \left(\prod_{k=j+1}^{i-1} p_j \right) g_j + g_{i-1} \tag{8}$$

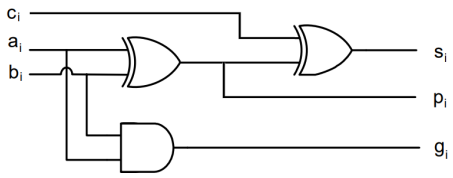

Figure 5: Full adder para un CLA.

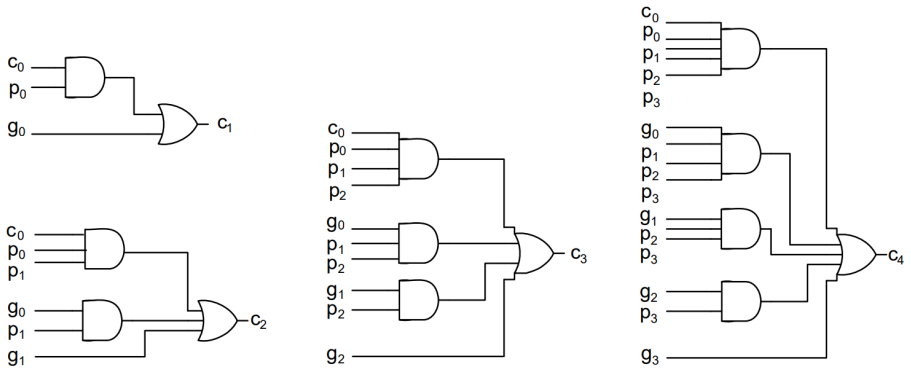

Figure 6: Lógica de carry para un CLA-4

Observar que la lógica de cálculo de carry se vuelve muy grande más allá e $N > 4$ bits. En dicho caso, limitando $N = 4$ se toma el cálculo de c_4:

$$c_4 = p_3.p_2.p_1.p_0.c_0 + p_3.p_2.p_1.g_0 + p_3.p_2.g_1 + p_3.g_2 + g_3 \tag{9}$$

y se escribe:

$$c_4 = P_0.c_0 + G_0 \tag{10}$$

donde G_0 y P_0:

$$\begin{array}{rcl} G_0 & = & p_3.p_2.p_1.g_0 + p_3.p_2.g_1 + p_3.g_2 + g_3 \\ P_0 & = & p_3.p_2.p_1.p_0 \end{array} \tag{11}$$

Si se tiene $N > 4$, por ejemplo $N = 8$ se agrupan los bits 0 a 3 para generar G_0 y $= P_0$ y los bits 4 a 7 para generar G_1, P_1 y así sucesivamente.
Vamos a llamar:

- CLA-FA: Carry lookahead adder - Full adder
- CLA-L0: Carry lookahead adder - Logic 0
- CLA-L1: Carry lookahead adder - Logic 1

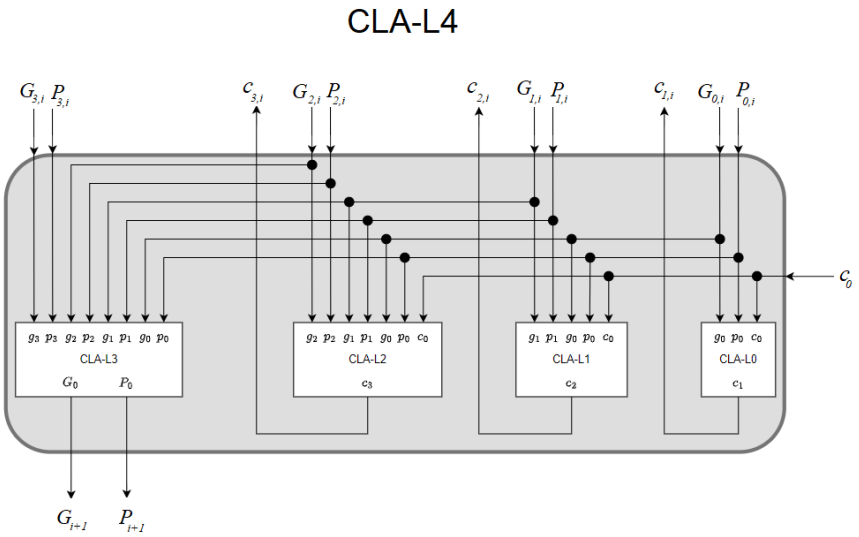

Figure 8: Carry lookahead logic, $N = 4$.

- CLA-L2: Carry lookahead adder - Logic 2
- CLA-L3: Carry lookahead adder - Logic 3
- CLA-L4: Carry lookahead adder - Logic 4

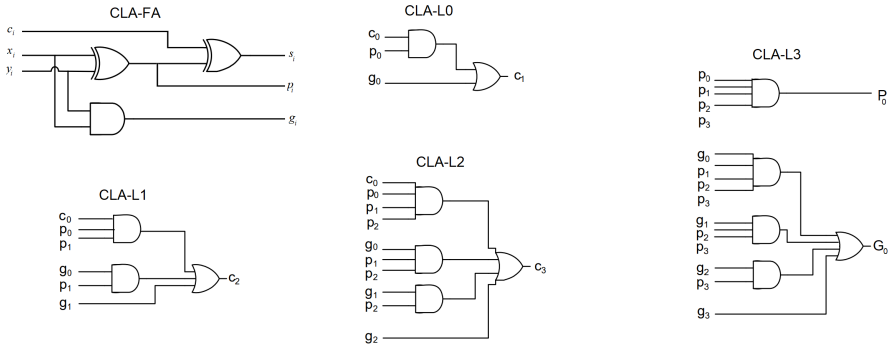

Figure 7: Carry looakahed logic functions.

4 Hybrid Ripple Carry and Carry Look-ahead Adder

Se conectan en ripple los CLA4.

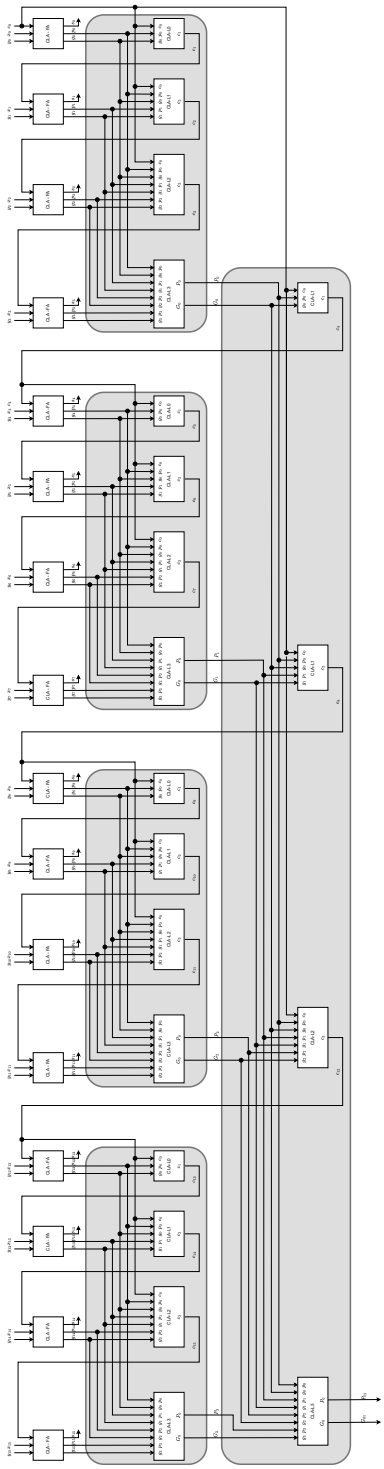

Figure 9: Carry lookahead adder, $N = 16$.

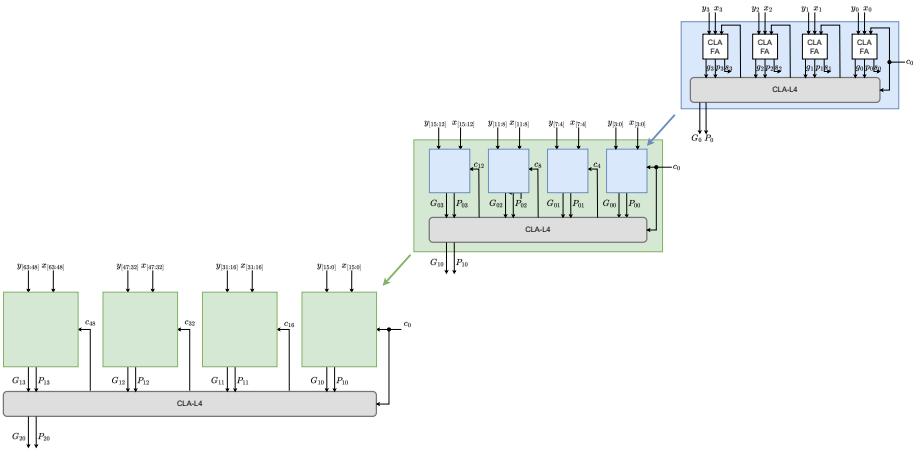

Figure 10: Carry lookahead adder, $N = 64$.

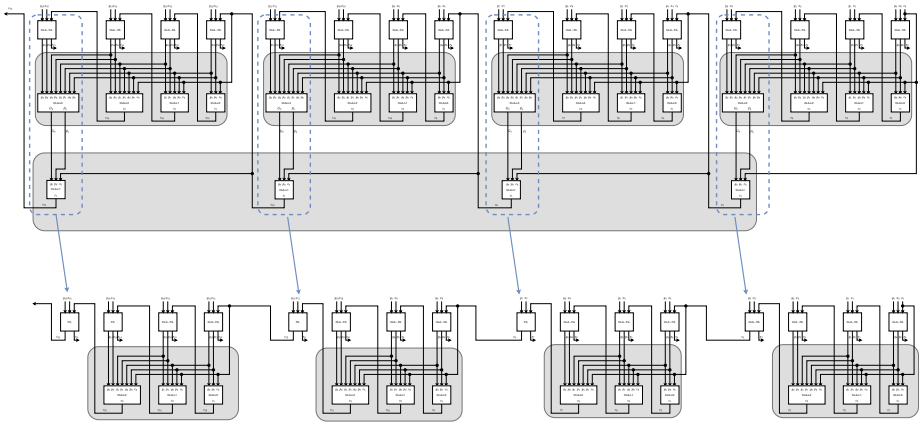

Figure 11: Hybrid Ripple Carry and Carry Look-ahead Adder, $N = 16$.

5 Carry skip adder

Se toma ell full adder de la Fig. 3.

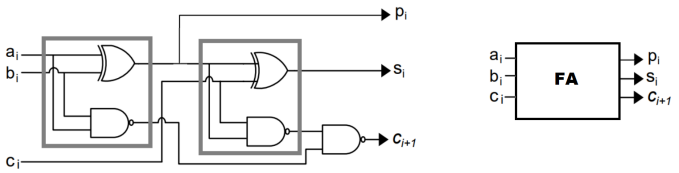

Figure 12: Full adder con salida p_i.

Se arma un ripple carry adder de por ejemplo 4 bits. Se arma una señal P_i de grupo:

$$P_i = p_i.p_{i+1}.p_{i+2}.p_{i+3} \tag{12}$$

Si un grupo genera un carry, se propaga al siguiente grupo. Sino lo genera, entonces pero propaga el carry de la etapa previa, entones lo transmite al grupo siguiente.

Se puede armar con distintos tamaño de grupo.

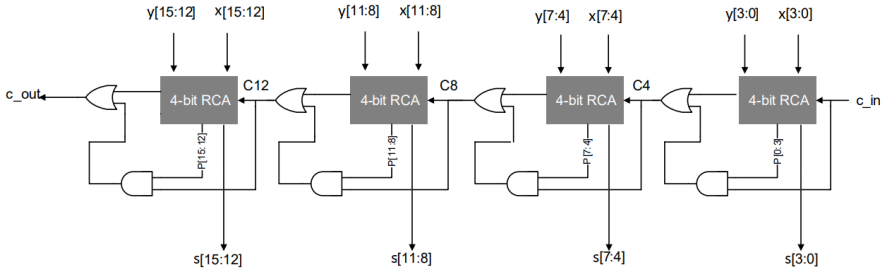

Figure 13: Carry skip adder para $N = 16$.

6 Carry select adder

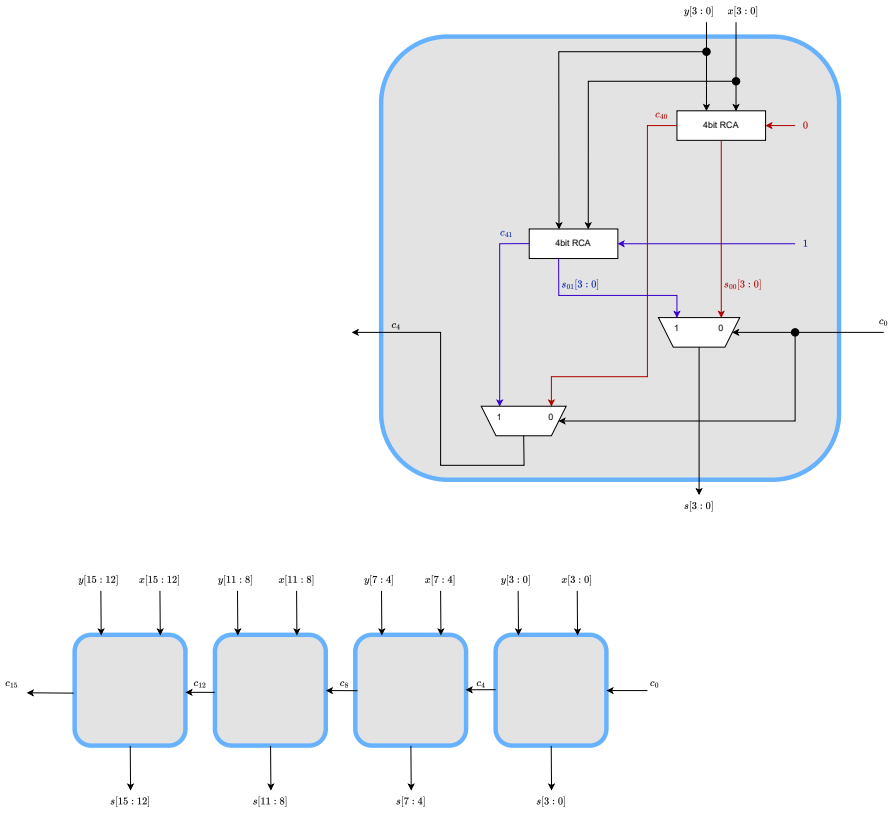

Figure 14: Carry select adder para $N = 16$.

7 Prefix addition

Otras formas de propagar el carry que ahora no vamos a estudiar:

- Brent–Kung adder
- Ladner–Fischer
- Kogge–Stone
- Han–Carlson

8 Carry save adder

Todos los sumadores vistos hasta acá se conocen como de tipo Carry propagate adder (CPA).

Para el caso de suma de tres operandos o más, se puede implementar el método Carry save adder (CSA).

8.1 Compression

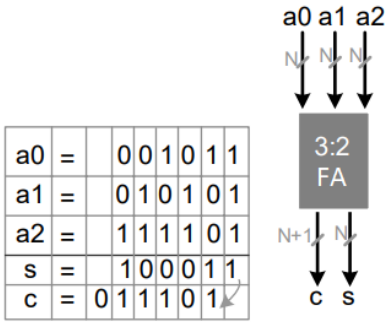

Figure 15: Compression 3:2.

Observar que la reducción 3:2 se hace "carry-free". Luego la reducción 2:1 se puede implementar con cualquier CPA de los vistos. También puede usar cualquier otra relación de compresión. Cuando se explique multiplicación se mostrarán ejemplos típicos.

9 Suma en codificación con signo (2C)

Aplicando el teorema de inversión de signo para 2C ($-x = \overline{x} + 1$), se aplica cualquier método de suma sin signo (US) invirtiendo uno de los operandos y asignando '1' al carry-in.

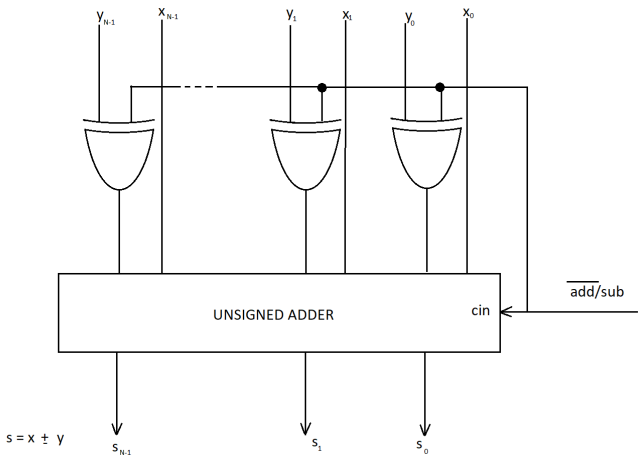

Figure 16: Add/sub.

10 Serial addition

Las entradas x,y son 1-bit serie, con codificación LSB first. Cada operando tiene N bits en forma serie.

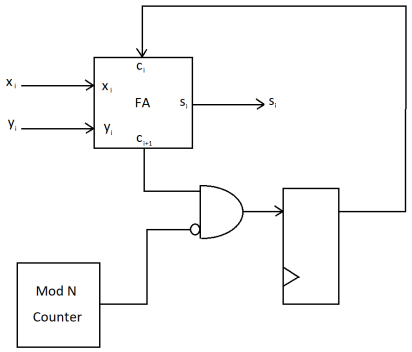

Figure 17: Serial unsigned adder

Multiplicación sin signo

				a_2	a_1	a_0
			$*$	b_2	b_1	b_0
		0	0	$a_2 \cdot b_0$	$a_1 \cdot b_0$	$a_0 \cdot b_0$
$+$		0	$a_2 \cdot b_1$	$a_1 \cdot b_1$	$a_0 \cdot b_1$	0
$+$		$a_2 \cdot b_2$	$a_1 \cdot b_2$	$a_0 \cdot b_2$	0	0
	p_5	p_4	p_3	p_2	p_1	p_0

donde p_5 es el carry out de la suma.

Multiplicación con signo

10.1 Introducción

Veamos primero la intución antes de demostrar formalmente las propiedades, para eso vamos a utilizar una multiplicación de números signados 3x3:

- Two's complement (2C) $X = -x_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} x_k \cdot 2^k$
- El bit de signo de los primeros Productos Parciales (PPs) se extiende.
- Se toma complemento a 2 del último PP
- La lógica de la extensión de signo es significativa. Es deseable encontrar una forma de removerla.

				a_2	a_1	a_0
			$*$	b_2	b_1	b_0
	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_1 \cdot b_0$	$a_0 \cdot b_0$
$+$	$a_2 \cdot b_1$	$a_2 \cdot b_1$	$a_2 \cdot b_1$	$a_1 \cdot b_1$	$a_0 \cdot b_1$	
$-$	$a_2 \cdot b_2$	$a_2 \cdot b_2$	$a_1 \cdot b_2$	$a_0 \cdot b_2$		
	p_5	p_4	p_3	p_2	p_1	p_0

Reemplazando el operador $-$ obtenemos:

				a_2	a_1	a_0
			$*$	b_2	b_1	b_0
	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_1 \cdot b_0$	$a_o \cdot b_0$
$+$	$a_2 \cdot b_1$	$a_2 \cdot b_1$	$a_2 \cdot b_1$	$a_1 \cdot b_1$	$a_o \cdot b_1$	
$+$	$\overline{a_2 \cdot b_2}$	$\overline{a_2 \cdot b_2}$	$\overline{a_1 \cdot b_2}$	$\overline{a_o \cdot b_2}$		
$+$				1		
	p_5	p_4	p_3	p_2	p_1	p_0

Luego, sumamos y restamos 111:

				a_2	a_1	a_0
			$*$	b_2	b_1	b_0
	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_2 \cdot b_0$	$a_1 \cdot b_0$	$a_o \cdot b_0$
$+$				1		
$+$	$a_2 \cdot b_1$	$a_2 \cdot b_1$	$a_2 \cdot b_1$	$a_1 \cdot b_1$	$a_o \cdot b_1$	
$+$			1			
$+$	$\overline{a_2 \cdot b_2}$	$\overline{a_2 \cdot b_2}$	$\overline{a_1 \cdot b_2}$	$\overline{a_o \cdot b_2}$		
$+$		1		1		
$-$	0	1	1	1		
	p_5	p_4	p_3	p_2	p_1	p_0

Se observa que:

- Si $a_2 \cdot b_0 == 0$:
  - La extensión de signo en la fila 1 desaparece.
  - Al sumarle 1, tendremos 1, por lo tanto estaremos invirtiendo el valor original.
- Si $a_2 \cdot b_0 == 1$:
  - Al sumarle 1, tendremos carry y 0 por lo tanto estaremos invirtiendo el valor original.
  - El carry se encarga de eliminar la extensión de signo.

Realizando el mismo procedimiento en los 3 PPs, y haciendo 2C en la última

resta, llegamos a:

				a_2	a_1	a_0
			$*$	b_2	b_1	b_0
				$\overline{a_2 \cdot b_0}$	$a_1 \cdot b_0$	$a_o \cdot b_0$
$+$			$\overline{a_2 \cdot b_1}$	$a_1 \cdot b_1$	$a_o \cdot b_1$	
$+$		$a_2 \cdot b_2$	$\overline{a_1 \cdot b_2}$	$\overline{a_o \cdot b_2}$		
$+$				1		
$+$	1	0	0	1		
	p_5	p_4	p_3	p_2	p_1	p_0

Realizando la operación de las últimas 2 filas, se obtiene:

				a_2	a_1	a_0
			$*$	b_2	b_1	b_0
				$\overline{a_2 \cdot b_0}$	$a_1 \cdot b_0$	$a_o \cdot b_0$
$+$			$\overline{a_2 \cdot b_1}$	$a_1 \cdot b_1$	$a_o \cdot b_1$	
$+$		$a_2 \cdot b_2$	$\overline{a_1 \cdot b_2}$	$\overline{a_o \cdot b_2}$		
$+$	1		1			
	p_5	p_4	p_3	p_2	p_1	p_0

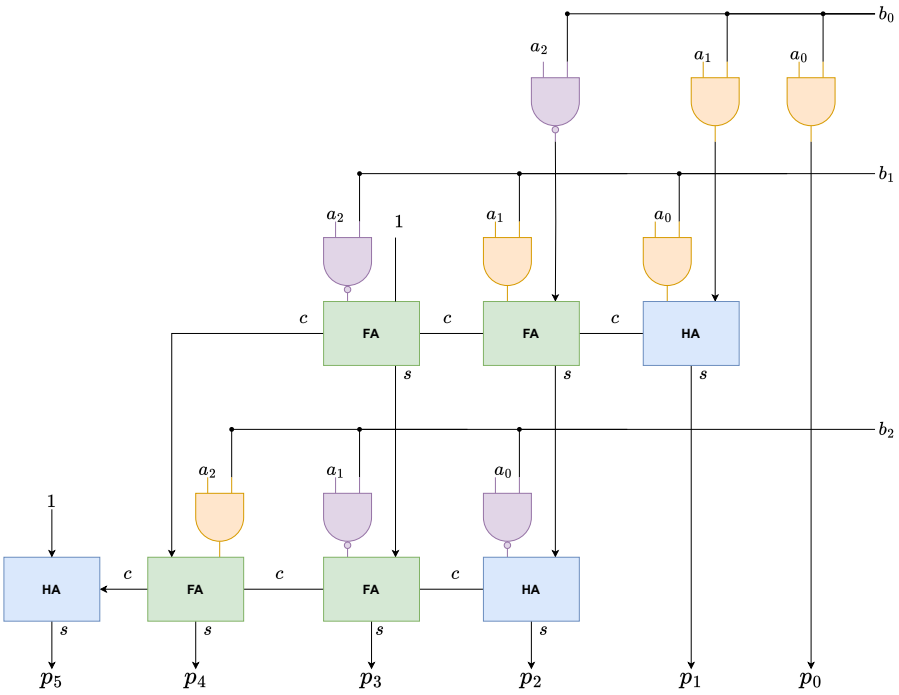

Figure 18: 3-bit Circuito Multiplicador Signado

10.2 Extensión de Signo - Eliminación

- Asumiremos que un vector binario x tiene la siguiente forma:

x_{N-1}	x_{N-2}	$\cdots$	x_3	x_2	x_1	x_0

  Two's complement (2C) $x = -x_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} x_k \cdot 2^k$

- Sea $\tilde{x}$ expresado en 2C:

0	$\bar{x}_{N-1}$	x_{N-2}	$\cdots$	x_2	x_1	x_0

  Entonces:

$$\begin{aligned} \tilde{x} &= -0 \cdot 2^N + \bar{x}_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} x_k \cdot 2^k \\ &= \bar{x}_{N-1} \cdot 2^{N-1} + (x_{N-1} \cdot 2^{N-1} - x_{N-1} \cdot 2^{N-1}) + \sum_{k=0}^{N-2} x_k \cdot 2^k \\ &= (\bar{x}_{N-1} + x_{N-1}) \cdot 2^{N-1} - x_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} x_k \cdot 2^k \\ &= 2^{N-1} + x \end{aligned} \tag{13}$$

  Obtenemos: $\tilde{x} = 2^{N-1} + x$

- Sea x' expresado en 2C,

0	x_{N-1}	$\bar{x}_{N-2}$	$\cdots$	$\bar{x}_2$	$\bar{x}_1$	$\bar{x}_0$

Entonces:

$$\begin{aligned} x' &= -0 \cdot 2^N + x_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} \bar{x}_k \cdot 2^k \\ &= x_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} \bar{x}_k \cdot 2^k \\ &= x_{N-1} \cdot 2^{N-1} + \bar{x}_{N-1} \cdot 2^{N-1} - \bar{x}_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} \bar{x}_k \cdot 2^k \\ &= (\bar{x}_{N-1} + x_{N-1}) \cdot 2^{N-1} - \bar{x}_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} \bar{x}_k \cdot 2^k \\ &= 2^{N-1} + x \end{aligned} \tag{14}$$

Finalmente: $x' = 2^{N-1} - x - 1$

- Sea $\bar{x}$ expresado en 2C

$\bar{x}_{N-1}$	$\bar{x}_{N-2}$	$\cdots$	$\bar{x}_2$	$\bar{x}_1$	$\bar{x}_0$

Entonces:

$$-\sum_{k=0}^{M} 2^k = 2^0 + 2^1 + 2^2 + \cdots + 2^M = 2^{M+1} - 1$$

$$\begin{aligned} \bar{x} &= -\bar{x}_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} \bar{x}_k \cdot 2^k \\ &= -\bar{x}_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} \bar{x}_k \cdot 2^k + x - x \\ &= -\bar{x}_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} \bar{x}_k \cdot 2^k + (-x_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} x_k \cdot 2^k) - x \\ &= -(\bar{x}_{N-1} + x_{N-1}) \cdot 2^{N-1} + \sum_{k=0}^{N-2} (\bar{x}_k + x_k) \cdot 2^k - x \\ &= -2^{N-1} + \sum_{k=0}^{N-2} 2^k - x \\ &= -2^{N-1} + (2^{N-1} - 1) - x \\ &= -x - 1 \end{aligned} \tag{15}$$

Finalmente: $\bar{x} = -x - 1$

10.3 Demostración Multiplicación Signada

- Sea $a = -a_{N-1} \cdot 2^{N-1} + \sum_{j=0}^{N-2} a_j \cdot 2^j$ (2C)
- Sea $b = -b_{N-1} \cdot 2^{N-1} + \sum_{k=0}^{N-2} b_k \cdot 2^k$ (2C)
- Sea $p = a \cdot b$

Entonces

$$
\begin{aligned}
p &= \left(-a_{N-1}\cdot 2^{N-1}+\textstyle\sum_{j=0}^{N-2} a_j\cdot 2^j\right)\cdot\left(-b_{N-1}\cdot 2^{N-1}+\textstyle\sum_{k=0}^{N-2} b_k\cdot 2^k\right)\\
&= \left(-a_{N-1}\cdot 2^{N-1}+a_{N-2}\cdot 2^{N-2}+a_{N-3}+\ldots+a_1\cdot 2^1+a_0\cdot 2^0\right)\cdot\left(-b_{N-1}\cdot 2^{N-1}+\textstyle\sum_{k=0}^{N-2} b_k\cdot 2^k\right)\\
&= \begin{array}{l}
+\left(-a_0\cdot b_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2} a_0\cdot b_k\cdot 2^k\right)\cdot 2^0\\
+\left(-a_1\cdot b_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2} a_1\cdot b_k\cdot 2^k\right)\cdot 2^1\\
+\left(-a_2\cdot b_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2} a_2\cdot b_k\cdot 2^k\right)\cdot 2^2\\
\vdots\\
+\left(-a_{N-2}\cdot b_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2} a_{N-2}\cdot b_k\cdot 2^k\right)\cdot 2^{N-2}\\
-\left(-a_{N-1}\cdot b_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2} a_{N-1}\cdot b_k\cdot 2^k\right)\cdot 2^{N-1}
\end{array}
\end{aligned}
\tag{16}
$$

Luego $p=-p_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2} p_k\cdot 2^k$ donde $p_k=a_k\cdot b$

Sea ahora $p^*=\tilde{p}_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2}\tilde{p}_k\cdot 2^k$, entonces

$$
\begin{aligned}
p^* &= (-p_{N-1}-1)\tilde{\cdot}\, 2^{N-1}+\textstyle\sum_{k=0}^{N-2}(2^{N-1}+p_k)\cdot 2^k\\
&= \left(2^{N-1}-p_{N-1}-1\right)\cdot 2^{N-1}+2^{N-1}\cdot\textstyle\sum_{k=0}^{N-2} 2^k+\sum_{k=0}^{N-2} p_k\cdot 2^k\\
&= \left(2^{N-1}-1\right)\cdot 2^{N-1}+2^{N-1}\cdot\textstyle\sum_{k=0}^{N-2} 2^k-p_{N-1}\cdot 2^{N-1}+\sum_{k=0}^{N-2} p_k\cdot 2^k\\
&= \left(2^{N-1}-1\right)\cdot 2^{N-1}+2^{N-1}\cdot\textstyle\sum_{k=0}^{N-2} 2^k+p\\
&= \left(2^{N-1}-1\right)\cdot 2^{N-1}+2^{N-1}\cdot\left(2^{N-1}-1\right)+p\\
&= \left(2^{N-1}-1\right)+2^{N-1}\left(2^{N-1}-1\right)+p\\
&= \left(2^{N-1}-1\right)\cdot 2^N+p
\end{aligned}
\tag{17}
$$

Luego $p=p^*+2^N\left(-2^{N-1}+1\right)$

10.4 Ejemplos

$2N-1$	$2N-2$	$2N-3$	$2N-4$	$\ldots$	N	N-1	N-2	N-3	$\ldots$	1	0	Exp
						$\overline{p_0}$	p_0	p_0	$\ldots$	p_0	p_0	$\bar{p}_0 \cdot 2^0$
					$\overline{p_1}$	p_1	p_1	p_1	$\ldots$	p_1		$\bar{p}_1 \cdot 2^1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
			$\overline{p_{N-3}}$	$\ldots$	p_{N-3}	p_{N-3}	p_{N-3}	p_{N-3}				$\bar{p}_{N-3} \cdot 2^{N-}$
		$\overline{p_{N-2}}$	p_{N-2}	$\ldots$	p_{N-2}	p_{N-2}	p_{N-2}					$\ldots$
	p_{N-1}	$\overline{p_{N-1}}$	$\overline{p_{N-1}}$	$\ldots$	$\overline{p_{N-1}}$	$\overline{p_{N-1}}$						$\ldots$
1	0	0	0	$\ldots$	1							$(-2^{N-1}+1)$

N N

Entonces:

$2N-1$	$2N-2$	$2N-3$	$2N-4$	$\ldots$	N	N-1	N-2	N-3	$\ldots$	1	0
					1	$\overline{p_0}$	p_0	p_0	$\ldots$	p_0	p_0
					$\overline{p_1}$	p_1	p_1	p_1	$\ldots$	p_1	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
			$\overline{p_{N-3}}$	$\ldots$	p_{N-3}	p_{N-3}	p_{N-3}	p_{N-3}			
		$\overline{p_{N-2}}$	p_{N-2}	$\ldots$	p_{N-2}	p_{N-2}	p_{N-2}				
	$\overline{p_{N-1}}$	p_{N-1}	p_{N-1}	$\ldots$	p_{N-1}	p_{N-1}					
1	p_{N-1}	$\overline{p_{N-1}}$	$\overline{p_{N-1}}$	$\ldots$	$\overline{p_{N-1}}$						

Ejemplos:

1.

$$
\begin{array}{cccccccccc}
 & 3 & & & & & 0 & 1 & 1 \\
* & 2 & & & & * & 0 & 1 & 0 \\
\hline
 & & & & & \boxed{1} & \boxed{1} & 0 & 0 \\
 & & & & & \boxed{1} & 1 & 1 & \\
 & & & \boxed{1} & 0 & \boxed{1} & \boxed{1} & & \\
\hline
 & 6 & \boxed{1} & 0 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

2.

$$
\begin{array}{cccccccccc}
 & -3 & & & & & 1 & 0 & 1 \\
* & -2 & & & & * & 1 & 1 & 0 \\
\hline
 & & & & & \boxed{1} & \boxed{1} & 0 & 0 \\
 & & & & & \boxed{0} & 0 & 1 & \\
 & & & \boxed{1} & 1 & \boxed{1} & \boxed{0} & & \\
\hline
 & 6 & \boxed{1} & 0 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

3.

	-3				1	0	1
$*$	$+2$			$*$	0	1	0
				$\boxed{1}$	$\boxed{1}$	0	0
				$\boxed{0}$	0	1	
		$\boxed{1}$	0	$\boxed{1}$	$\boxed{1}$		
	-6	1	1	1	0	1	0

4.

	$+3$				0	1	1
$*$	-2			$*$	1	1	0	
				$\boxed{1}$	$\boxed{1}$	0	0
				$\boxed{1}$	1	1	
		$\boxed{1}$	0	$\boxed{0}$	$\boxed{0}$		
	-6	1	1	1	0	1	0

10.5 Multiplicación en Paralelo

10.5.1 Introducción

Dado entradas de N bits, el multiplicador tiene 3 componentes básicos:

- Generación de Producto Parcial = N números binarios desplazados.
- Reducción Producto Parcial = Reducción a 2 números binarios
- Calculo Suma final = $2N$ bits

Para cada una de estas operaciones, existen diversas técnicas para optimizar el HW necesario para implementar el multiplicador.

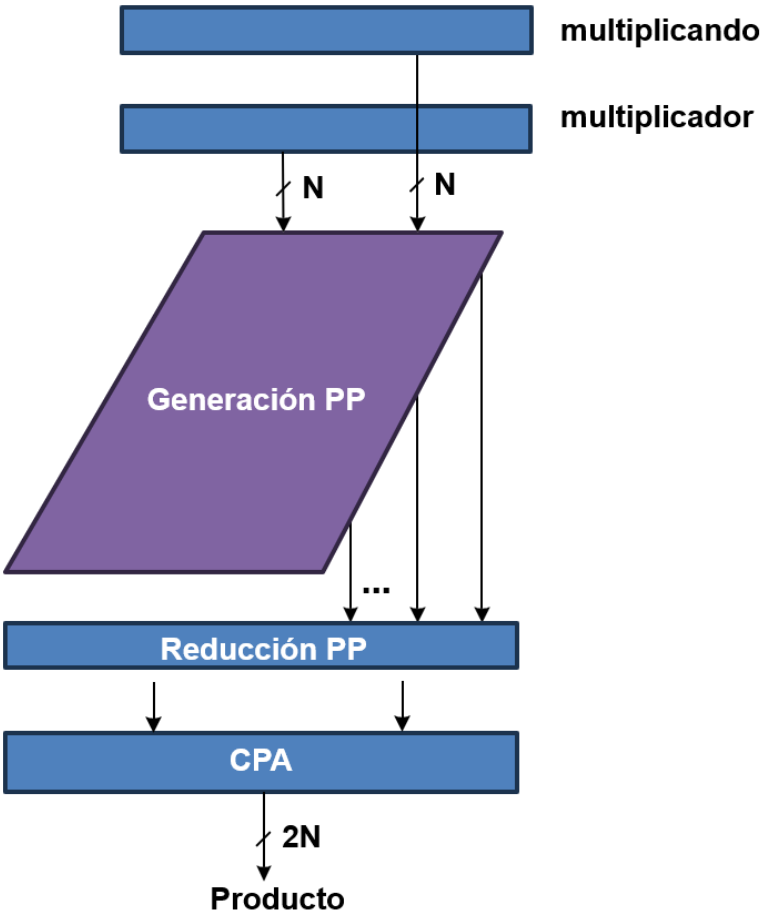

Figure 19: Componentes Multiplicador

10.5.2 Generación de Producto Parcial

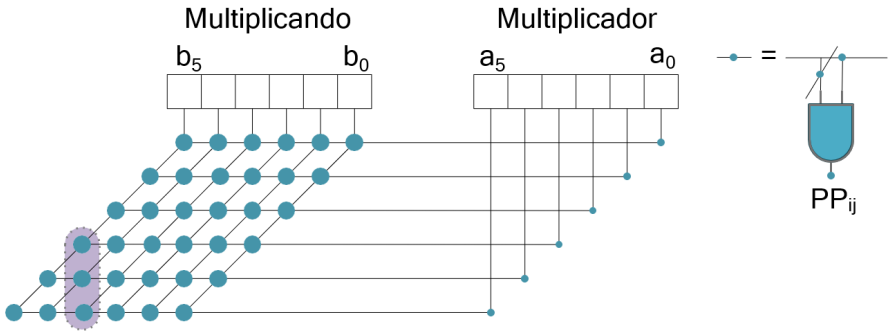

Figure 20: Generación de Productos Parciales para un Multiplicador de 6x6

```
module multiplier(
    input  [ 5:0] a,b,
    output [11:0] prod
);
    // 6 productos parciales
    reg [5:0] pp [0:5];
    integer i;

    always @* begin
        for (i=0; i<6; i=i+1) begin
            pp[i] = b & {6{a[i]}};
        end
    end

    assign prod = pp[0] +
                  {pp[1], 1'd0} +
                  {pp[2], 2'd0} +
                  {pp[3], 3'd0} +
                  {pp[4], 4'd0} +
                  {pp[5], 5'd0};
endmodule
```

Listing 1: Ejemplo: Generación de Productos Parciales

10.5.3 Esquemas de reducción de productos parciales

- Carry Save
- Dual Carry Save
- Wallace Tree
- Dadda Tree

10.5.4 Carry Save

- Toma las 3 primeras filas y utiliza CSA para reducirlas a 2
- Iterativamente, toma 2 filas del resultado previo mas una nueva del PP y las reduce a solo 2 utilizando CSA
- Finalmente produce solo 2 filas que son sumadas utilizadas cualquier CPA

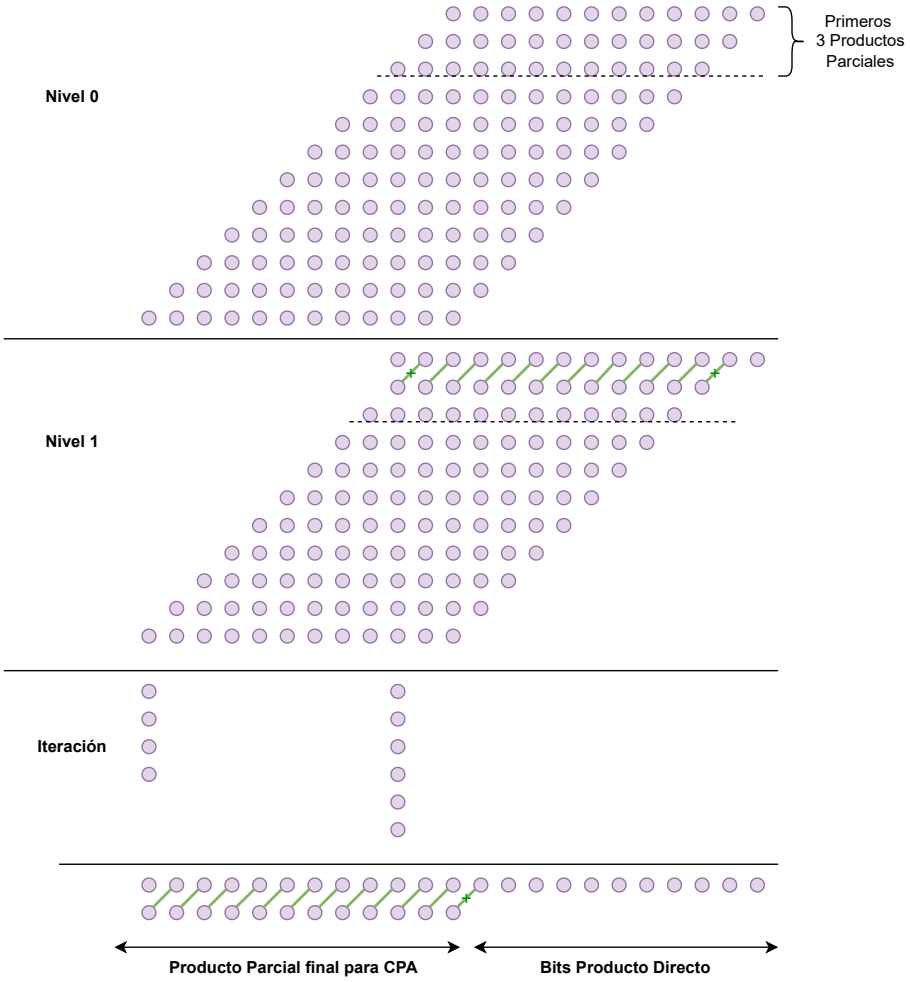

Figure 21: Reducción PP para un multiplicador de 12x12 utilizando Carry Save Reduction

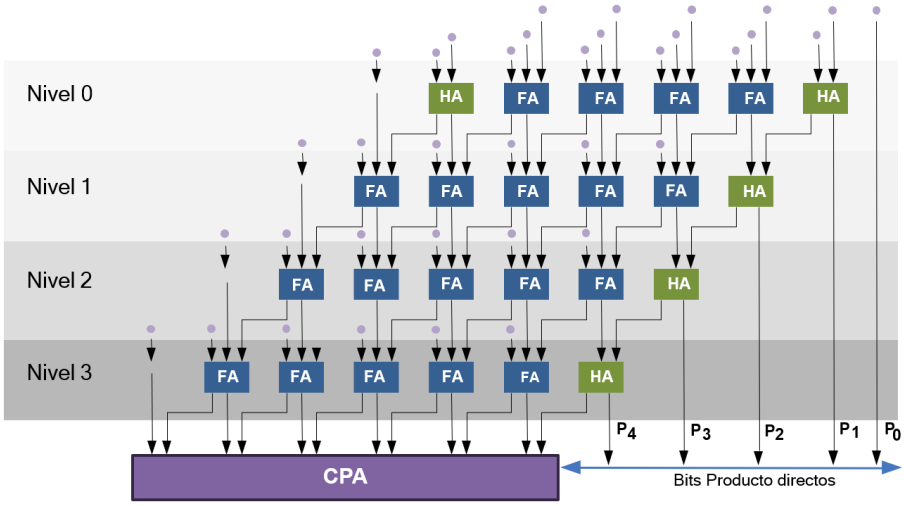

Figure 22: Reducción Carry Save

10.5.5 Dual Carry Save

- Los productos parciales se dividen en 2 grupos de igual tamaño
- El esquema de Carry Save Reduction se aplica en ambos grupos simultáneamente
- Esto resulta en 2 filas de productos parciales en cada grupo
- Las últimas 4 filas se reducen usando Carry Save Reduction
- Finalmente, las útimas 2 filas se suman utilizando cualquier CPA

10.5.6 Reducción Wallace

- Uno de las arquitecturas más comunes y utilizadas para multiplicadores
- Complejidad de: $O\left(log(n)\right)$
  - El número de niveles de sumadores se incrementa logarítmicamente a medida de el producto parcial de filas aumenta.

–

Número de Productos Parciales	Número de niveles de Full Adders
3	1
4	2
$5 \leq n \leq 6$	3
$7 \leq n \leq 9$	4
$10 \leq n \leq 13$	5
$14 \leq n \leq 19$	6
$20 \leq n \leq 28$	7
$29 \leq n \leq 42$	8
$43 \leq n \leq 63$	9

- El procedimiento consiste en:
  - Se arman grupos de a 3 filas y se aplica reducción CSA en paralelo
  - Cada grupo CSA produce 2 filas
  - Iterativamente se aplica la reducción CSA en las nuevas matrices.
  - Se continua el procedimiento hasta que unicamente queden 2 filas que se suman para obtener el producto final.

Hay un total de **102 FullAdders** y **34 Half Adders** en el diagrama de Wallace de 12x12, totalizando 1054 gates. El multiplicador Wallace 12x12 tiene un total de 5 etapas de reducción. Cada etapa de reducción tiene **6 gate delays** lo que resulta en un total de **30 gate delays**.

Etapa	1	2	3	4	5	Total
Full Adders	40	20	20	11	11	102
Half Adders	8	6	8	5	7	34

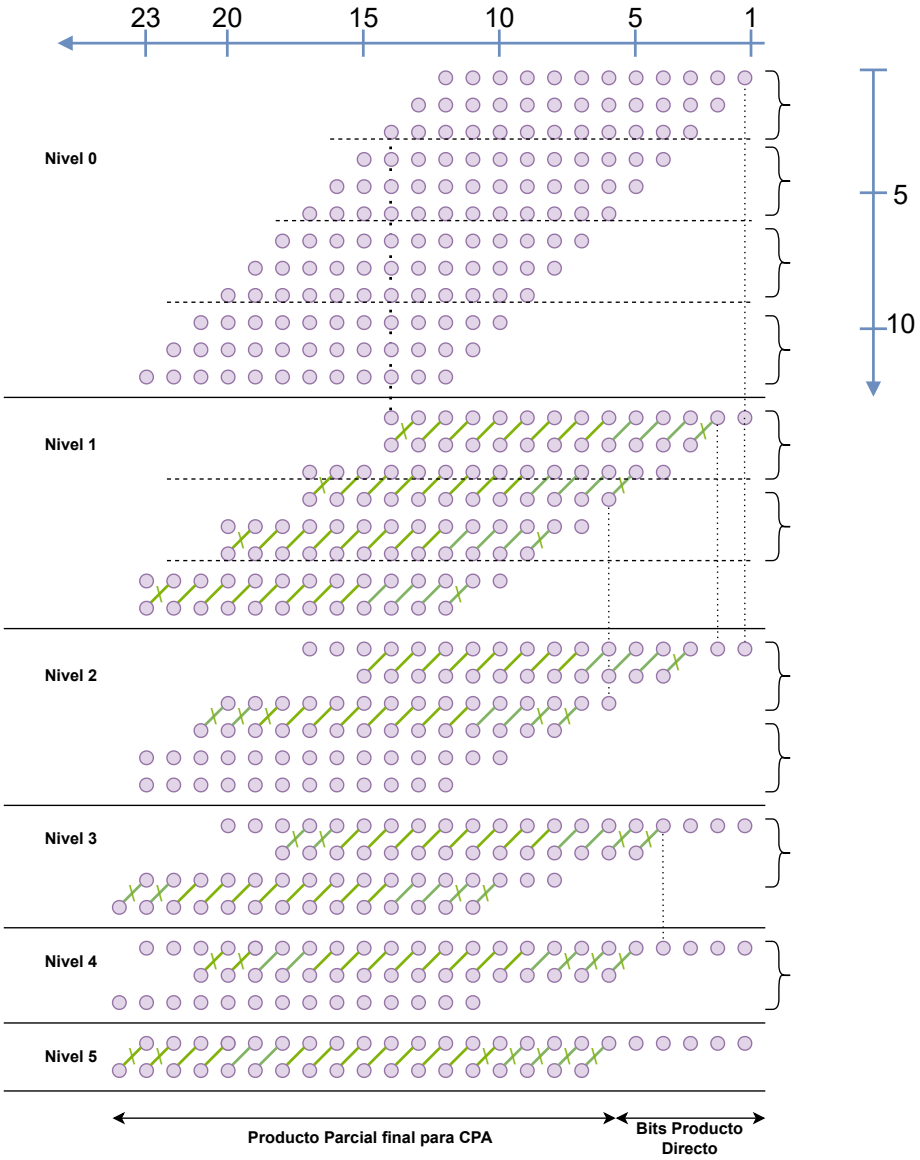

Figure 23: Reducción de Wallace aplicada en PP_{12}

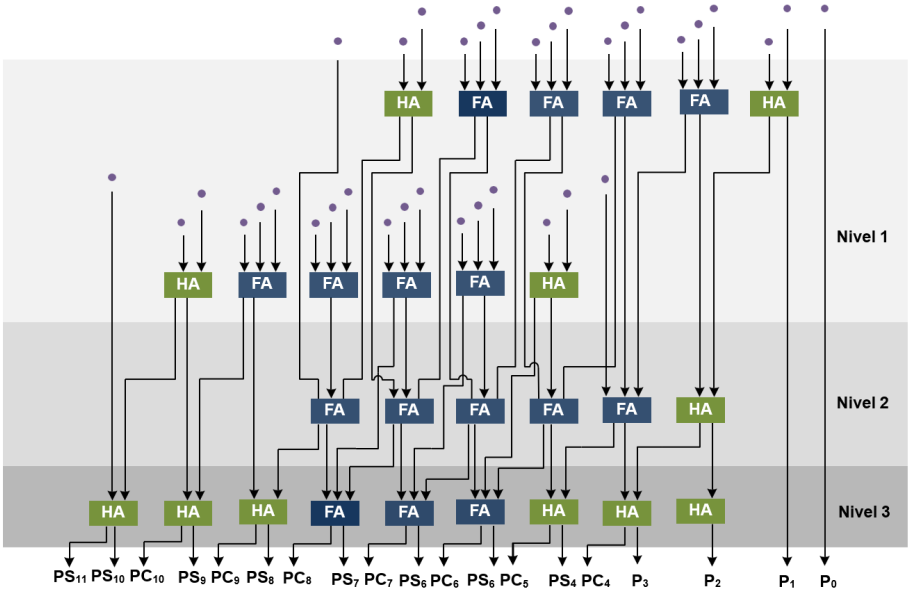

Figure 24: Reducción de Wallace para un arreglo de 6x6 PPs

10.5.7 Reducción Dadda

- Minimiza el número de **HA** y de **FAs**.
- La reducción considera cada columna de forma separada.
- Reduce el máximo número de puntos en cada columna siguiendo la secuencia númerica de la reducción de Wallace.

El enfoque de Dadda se diferencia del enfoque de Wallace en el uso del número mínimo de FAs y HAs necesarios para cumplir con las alturas predeterminadas para cada etapa. Los límites de altura de cada etapa son: #2, #3, #4, #6, #9, #13, #19, #28, #42, etc.. Cada etapa es 1.5 veces mayor que la etapa subsiguiente. La relación de reducción de 1.5 resulta del uso de sumadores completos que toman 3 productos parciales y los reducen a 2 productos parciales. Por lo tanto, la altura máxima de la etapa siguiente es $\frac{2}{3}$ de la de la etapa anterior. Al hacer esto, Dadda busca optimizar el área del multiplicador utilizando el menor número de sumadores para alcanzar la etapa de CPA.

Debido a que el número de Dadda más cercano a 12 es 9, la primera etapa de reducción tiene una altura de 9 puntos. Las siguientes etapas proceden siguiendo los números de Dadda, de 9 a 6, de 6 a 4, etc. El multiplicador Dadda tiene el mismo número de etapas de reducción que un multiplicador Wallace; en este caso, el multiplicador Dadda de 12 bits por 12 bits tiene 5 etapas de reducción. Por lo tanto, el retardo es de 30 gate delay, que es el mismo retardo que usando la redución de Wallace.

Etapa	1	2	3	4	5	Total
Full Adders	8	27	28	17	19	99
Half Adders	4	3	2	1	1	11

Se puede observar que el número de componentes aritméticos es mayor en las etapas intermedias de su reducción. Hay un total de 99 FAs y 11 HAs, lo que da una complejidad total de 945 gates. Debido a que la reducción de Dadda sigue un conjunto de alturas de etapa, el número de sumadores completos en un multiplicador Dadda de N bits por N bits se puede encontrar con: $N^2 - 4N + 3$, mientras que, el número de medio sumadores se puede encontrar con: $N - 1$

La complejidad de la etapa de reducción en un multiplicador Wallace es aproximadamente un 10% mayor que la de Dadda para un tamaño de entrada dado. A pesar de esta reducción en hardware, los multiplicadores Dadda contienen el mismo número de etapas de reducción en comparación con los multiplicadores Wallace. Como resultado, se espera que tengan el mismo retraso en la etapa de reducción. Sin embargo, el retraso total de un multiplicador Wallace podría diferir del multiplicador Dadda ya que el tamaño del CPA es menor en el multiplicador Wallace. El CPA observa incrementos en los retrasos cada vez que su tamaño de palabra de entrada aumenta en un factor de 4[**?**, **?**, **?**, **?**, **?**, **?**]

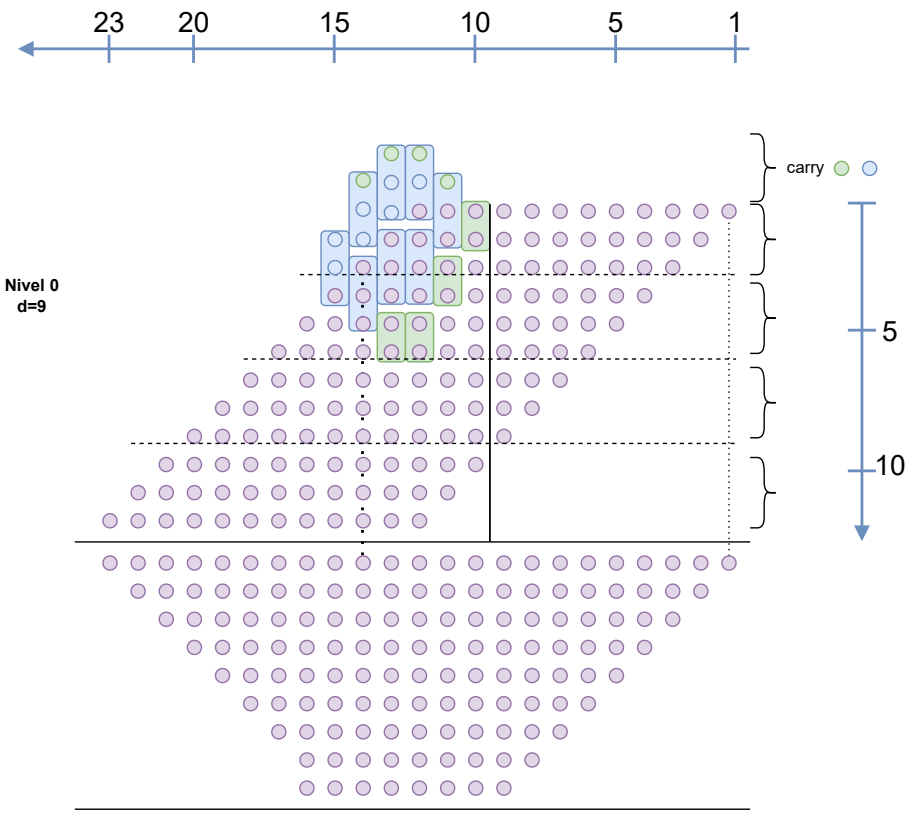

Figure 25: Reducción Dadda - 12x12 - Nivel 1

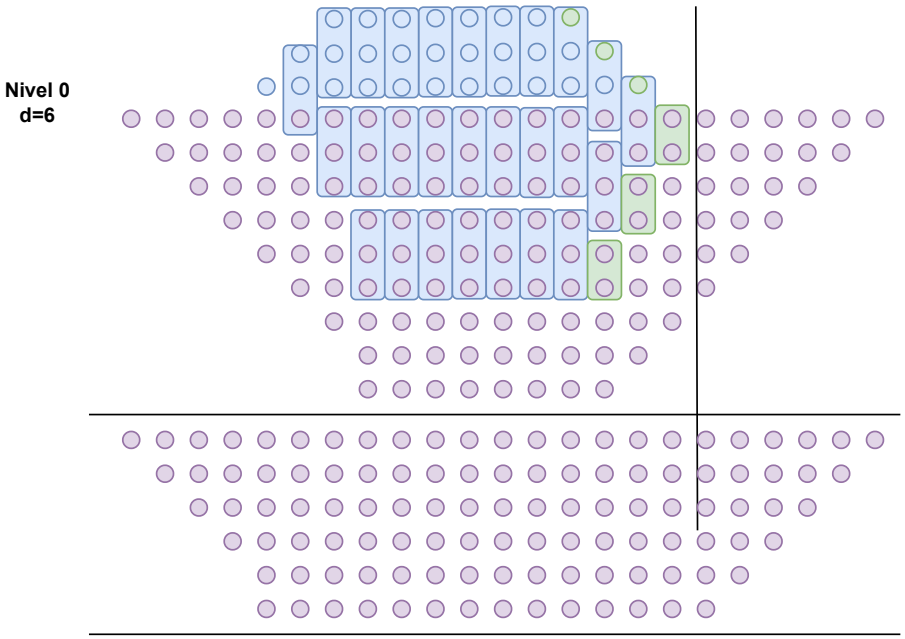

Figure 26: Reducción Dadda - 12x12 - Nivel 2

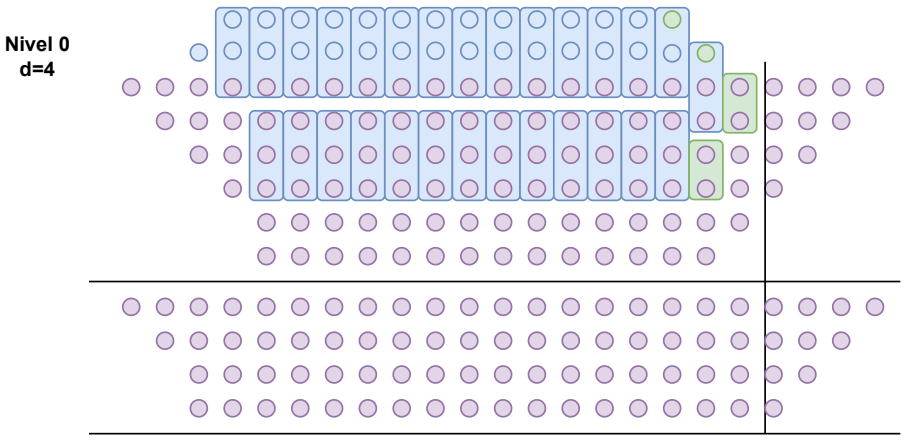

Figure 27: Reducción Dadda - 12x12 - Nivel 3

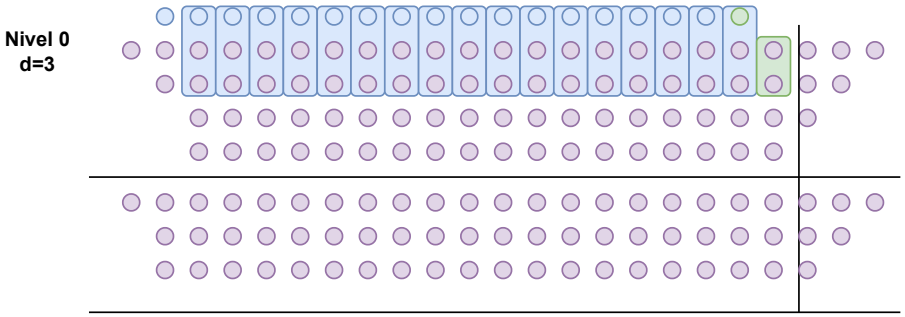

Figure 28: Reducción Dadda - 12x12 - Nivel 4

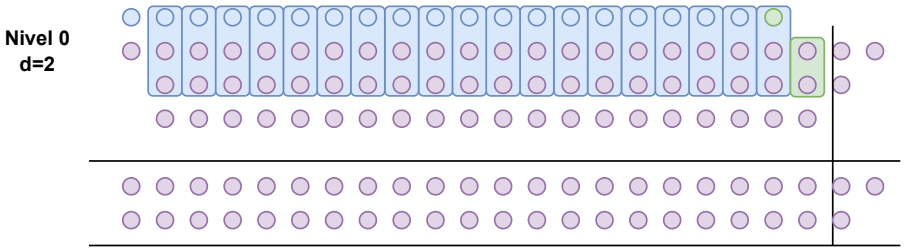

Figure 29: Reducción Dadda - 12x12 - Nivel 5

10.6 Decomposición de Multiplicador

Ejemplo: Un multiplicador de 16x16 se puede realizar considerando los 2 operandos de 16-bits como 4 de 8-bits.

$$\begin{array}{lcccccccc} a_L & = & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\ a_H & = & a_{15} & a_{14} & a_{13} & a_{12} & a_{11} & a_{10} & a_9 & a_8 \\ b_L & = & b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ b_H & = & b_{15} & b_{14} & b_{13} & b_{12} & b_{11} & b_{10} & b_9 & b_8 \end{array}$$

$$\left(a_L + 2^8 \cdot a_H\right) \times \left(b_L + 2^8 \cdot b_H\right) = a_L \times b_L + a_L \times b_H 2^8 + a_H \times b_L 2^8 + a_H \times b_H 2^{16}$$

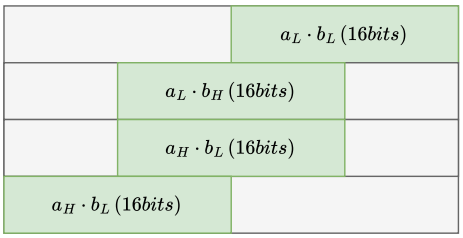

Figure 30: Multiplicador de 16x16 bits compuesto por 4 multiplicadores de 8x8 bits

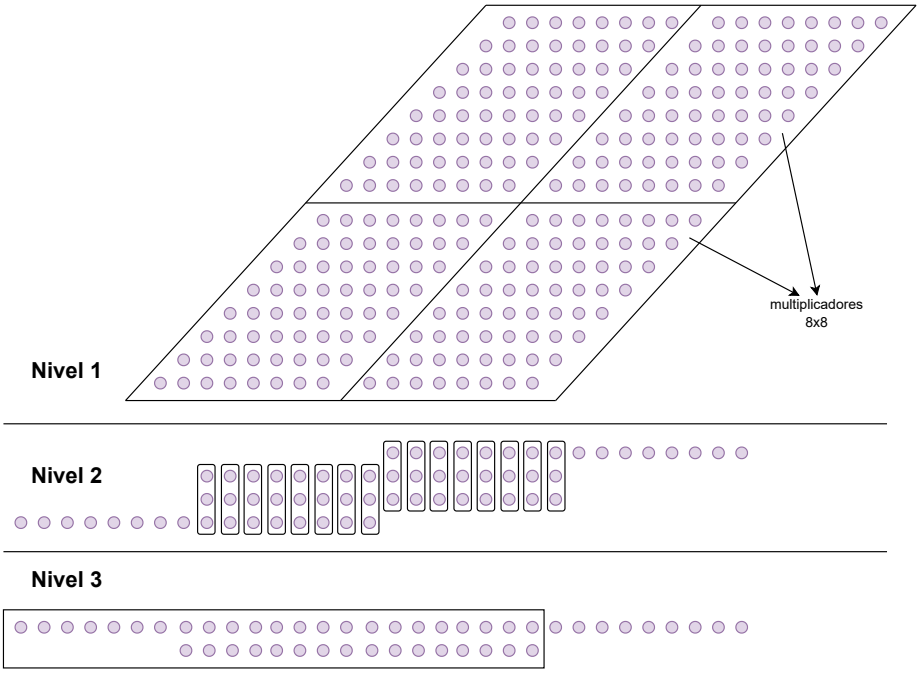

Figure 31: Ejemplo 16x16 utilizando 4 8x8

11 Multiplicadores secuenciales

11.1 Multiplicador basado en contar y acumular

Sean x, y enteros positivos y sea $m = y \times x$, entonces $m = \sum_{i=0}^{x-1} y$, es decir

$$m = \underbrace{y + y + y + \ldots + y}_{x \text{ veces}} \tag{18}$$

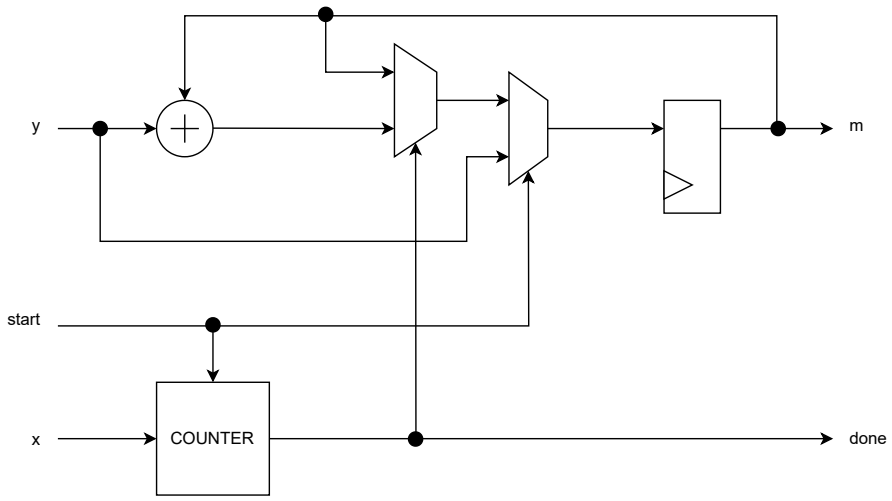

Figure 32: Sequential unsigned counter-and-add multiplier.

11.2 Shift and add multiplier

Sean x, y enteros positivos y sea $m = y \times x$. Se cargan inicialmente x e y en dos registros X e Y respectivamente. El registro P se inicializa en 0.

- Si el LSB del registro X es '1', se suma el contenido del registro Y al registro P, sino, el registro P queda sin modificaciones.
- Los registros P y X se desplazan como conjunto a derecha una posición. El carry out de la suma se carga en el MSB de P. Luego de que se shiftearon todos los bits de X, en el conjunto de registros $\{P, X\}$ se encuentra el valor de m.

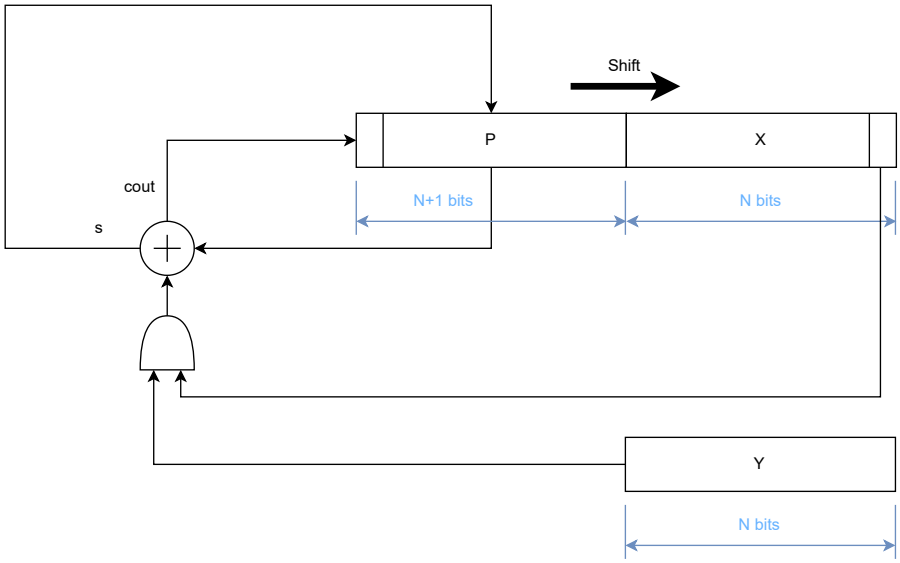

Figure 33: Shift-and-add multiplier.

11.3 Booth Multiplier

Recordamos Booth:

Es una representación SD por lo cual:

$$x = \sum_{i=0}^{N-1} x_i 2^i, \quad x_i \in \{-1, 0, 1\} \tag{19}$$

Se utiliza para números signados. Observar que si la representación 2C es de N bits, entonces la representación de Booth tamibién.

11.4 Conversión 2C a Booth

Algoritmo: Sea $x_{-1} = 0$, se aplica la siguiente Tabla.

Table 1: Conversión 2C a Booth.

x_i^{2C}	x_{i-1}^{2C}	x_i^{Booth}
0	0	0
0	1	1
1	0	$\overline{1}$
1	1	0

Observamos desde la tabla que si las entradas son iguales, no se hace nada, si son distintas entonces se suma o resta. Podemos tomar el bit x_i para decidir si se suma o resta.

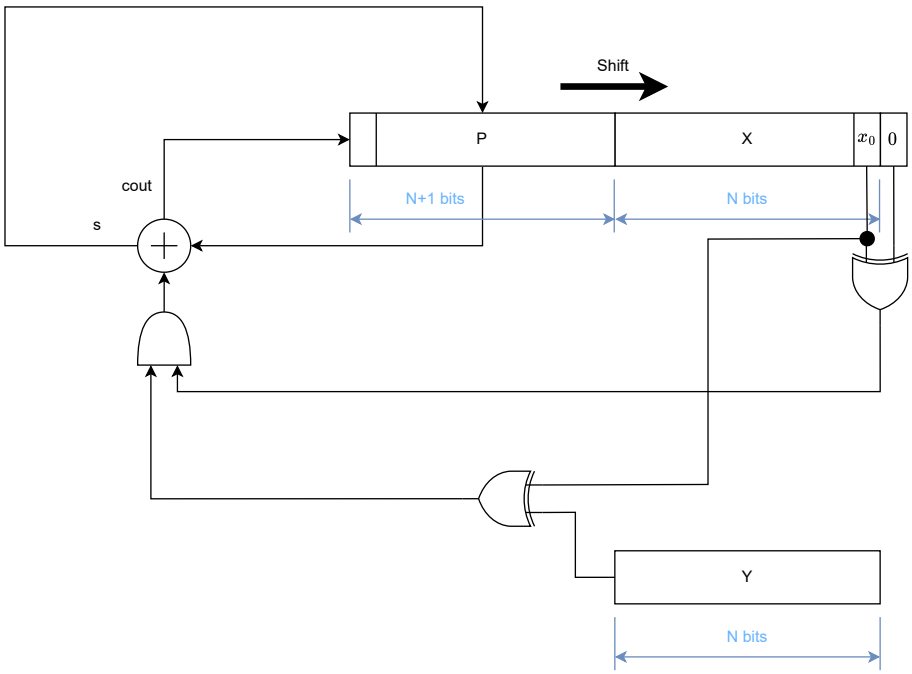

Figure 34: Booth multiplier multiplier.

11.5 CSD Multiplier

Ejercicio para entregar: Sea x un entero expresado en 2C con N_x bits y sea y un entero expresado en 2C con N_y bits. Describir en Verilog un circuito que obtenga $m = x \times y$ expresado en 2C con $N_m = N_x + N_y$ bits tal que opere secuencialmente con codificación CSD. Verifique que opere correctamente para todos los casos posibles de entrada.

TIP: Se puede utilizar la siguiente codificación:

x_i^{2C}	x_{i1}^{CSD}	x_{i0}^{CSD}
0	0	0
1	1	0
$\overline{1}$	1	1

De esta forma bit x_{i1}^{CSD} significa que o bien no hay que sumar nada o bien hay que restar o sumar. Bit x_{i0}^{CSD} significa que si es 0 se suma, si es 1 se resta (se puede usar para invertir el operando y y como entrada del carry in del sumador.

11.6 Parallel-serial multiplier

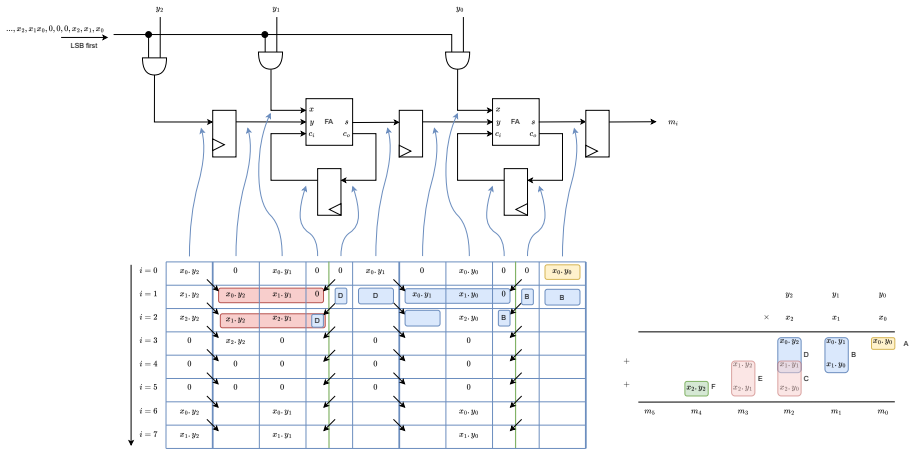

Figure 35: Parallel-serial multiplier.

Ejercicio para entregar: Extenderlo al caso de x e y ambos signados 2C.

12 Inferencia en Síntesis

Las herramientas de síntesis tienen bibliotecas de diseños reutilizables. En el caso de Cadence Genus se llama ChipWare (CW) que incluye:

- Componentes básicos combinacionales y sequenciales
- Componentes aritméticos (sumadores, restadores, multiplicadores)
- Componentes de memoria (FFs, FIFOs)
- La síntesis asigna automáticamente operadores a componentes CW disponibles.
- Muchos componentes CW tienen múltiples arquitecturas que permiten a la síntesis lógica hacer compensaciones entre área y tiempo.

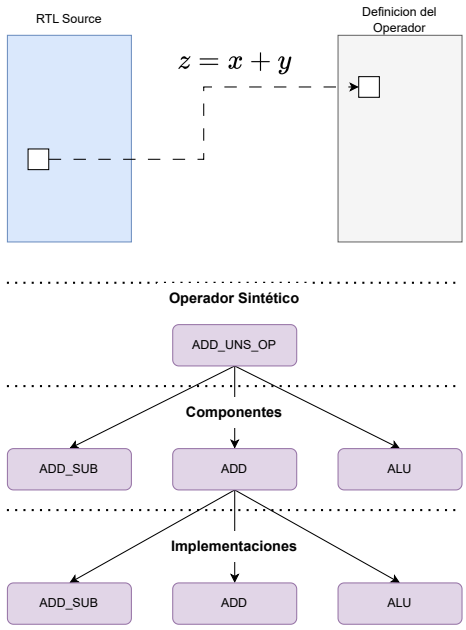

Figure 36: Selección de Arquitectura - ChipWare

Observar que los siguientes dos casos corresponden al mismo hardware y arrojan en binario los mismos resultados:

```
// Caso 1
wire signed [N-1:0] x;
wire signed [N-1:0] y;
wire signed [N-1:0] s;

assign s = x + y;

// Caso 2
wire [N-1:0] x;
wire [N-1:0] y;
wire [N-1:0] s;

assign s = x + y;
```

Sin embargo, esto no ocurre para la multiplicación, es decir no es el mismo hardware y no arrojan el mismo resultado binario los casos 1 y 2 siguientes:

```
// Caso 1
wire signed [N-1:0] x;
wire signed [N-1:0] y;
wire signed [2*N-1:0] m;

assign m = x * y;

// Caso 2
wire [N-1:0] x;
```

```
10 wire [N-1:0] y;
11 wire [2*N-1:0] m;
12
13 assign m = x * y;
```