Diseño de un Sumador Rápido en tecnología CMOS submicrónica utilizando Herramientas de Software Libre

Leandro Marsó[1] Pablo Cayuela (Director)[2]
Hugo Carrer (Codirector)[1]

[1]Universidad Nacional de Córdoba, Argentina
[2]Universidad Tecnológica Nacional, FRC, Argentina

Facultad de Ciencias Exáctas, Físicas y Naturales, 2015

Temario

Temario

¿Qué es un sumador rápido?

¿Qué es un circuito integrado?

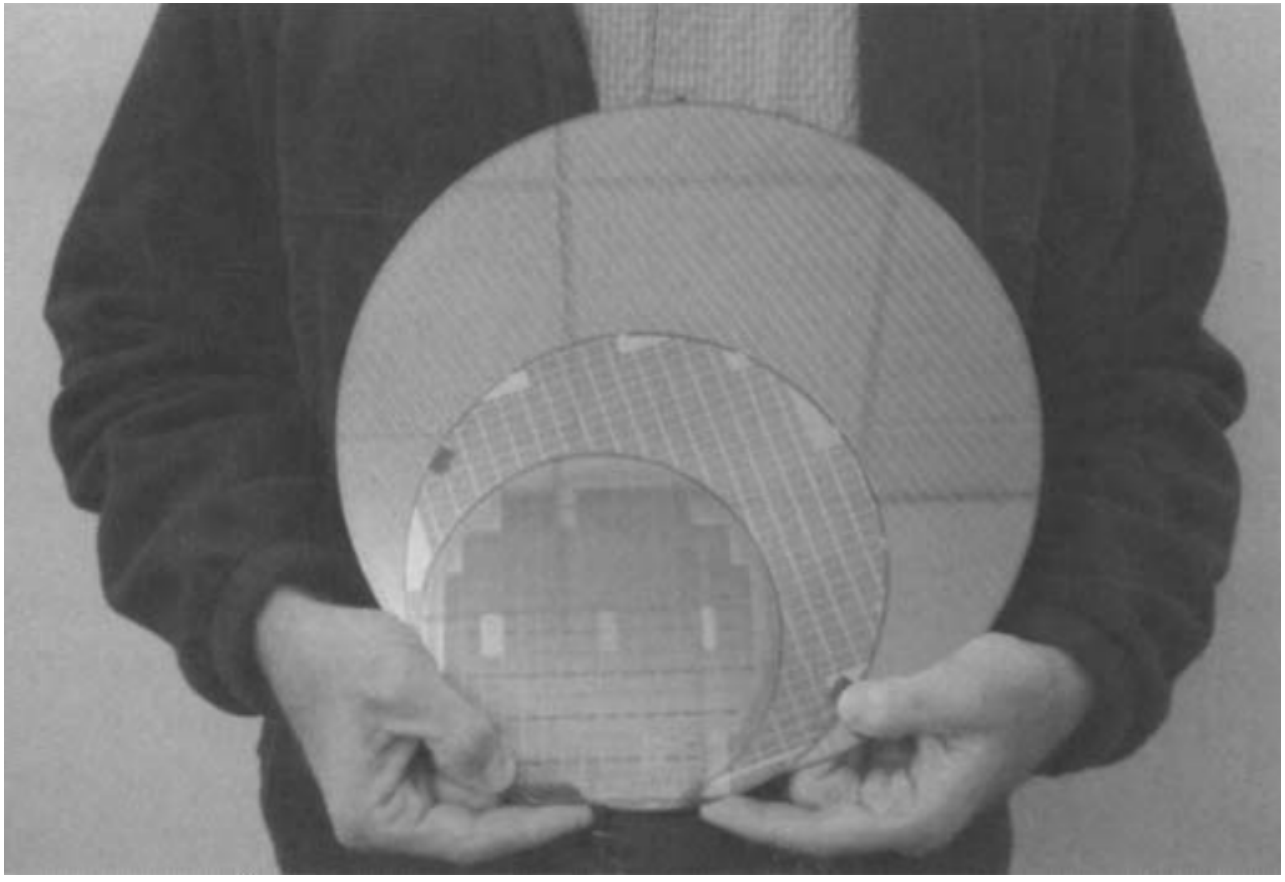


Figura: Obleas de silicio de 150, 200 y 300 mm de díametro, de un proceso CMOS.

¿Qué es un circuito integrado?

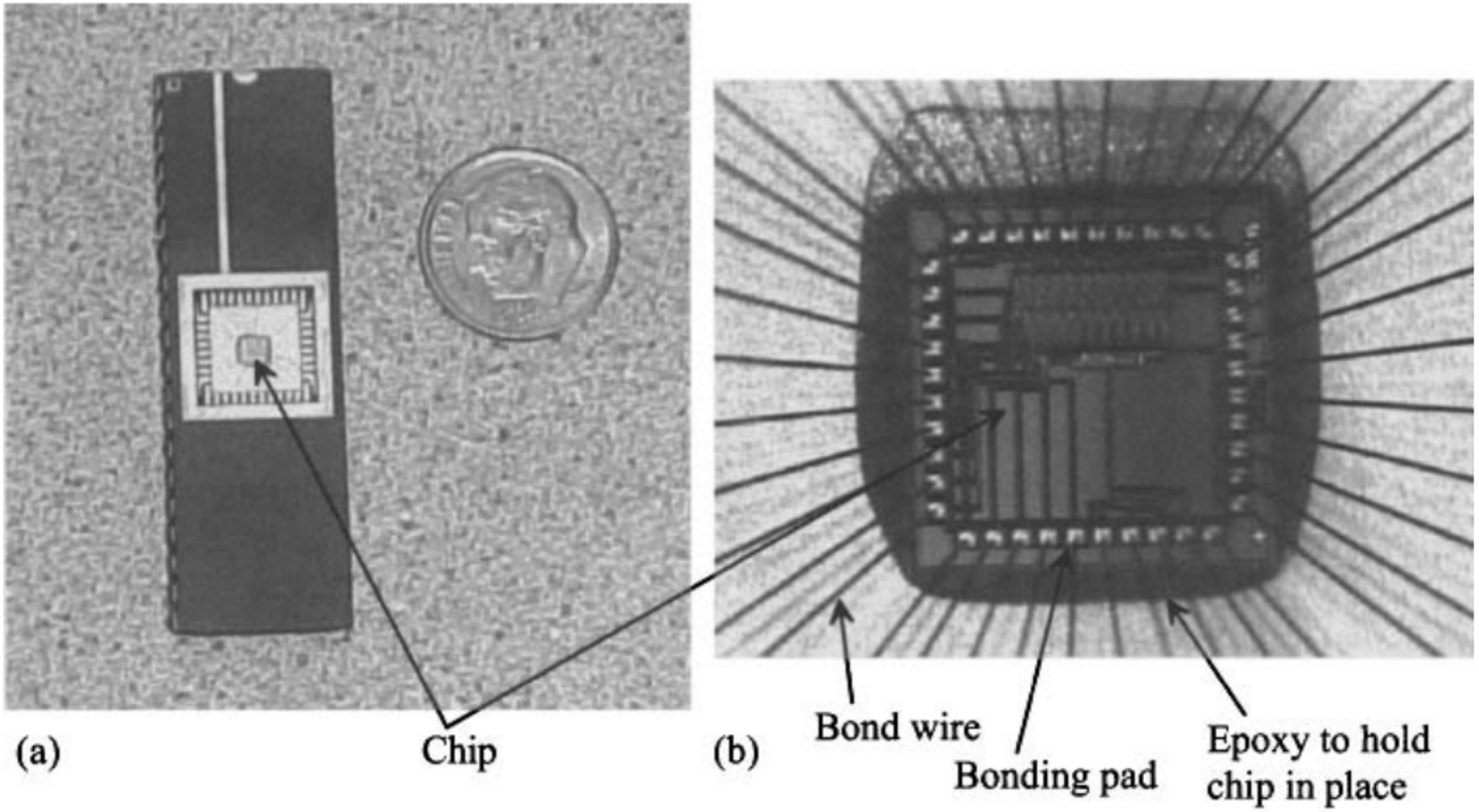
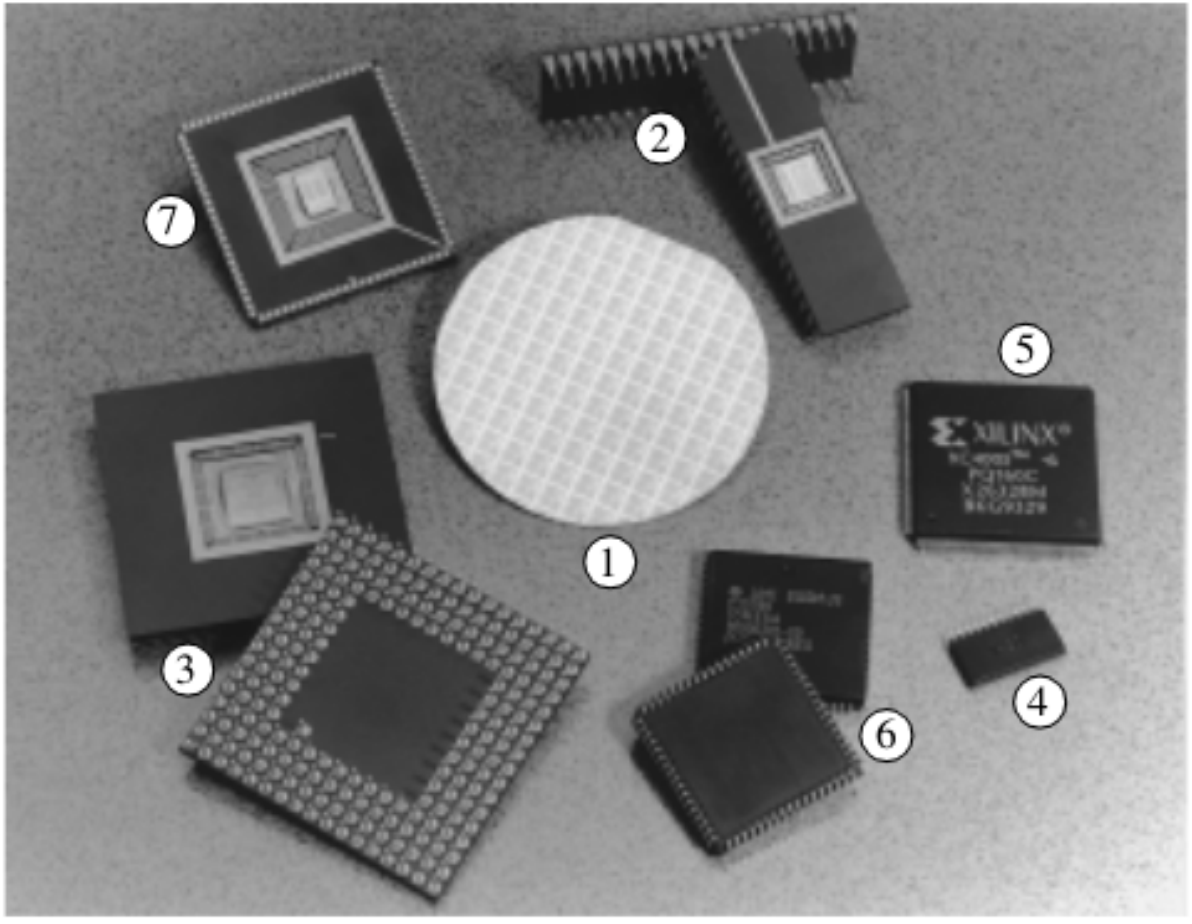


Figura: Encapsulado del chip (a) y (b) una vista aumentada.

¿Qué es un circuito integrado?



1 Bare die
2 DIP
3 PGA
4 Small-outline IC
5 Quad flat pack
6 PLCC
7 Leadless carrier

Figura: Algunos tipos de encapsulados comunes.

¿Qué es un circuito integrado?

¿Cómo accedemos a fabricar circuitos integrados?

Fabrica	Proceso CMOS
TSMC	28 nm - 180 nm
Globalfoundries	14 nm - 180 nm
IBM	32 nm - 250nm
ON Semi	0.35 um - 0.7 um
Austria Micro Systems	180 nm - 0.35 um

Cuadro: Procesos disponibles por medio de MOSIS

¿Qué es un circuito integrado?

¿Cómo accedemos a fabricar circuitos integrados?

Fabrica	Proceso CMOS
STMicroelectronics	28 nm - 130 nm
Austria Micro Systems	180 nm - 0.35 um

Cuadro: Procesos disponibles por medio de CMP

¿Qué es un circuito integrado?

¿Cuánto podemos integrar?

CMOS 350 nm de AMS

- 18kGates/mm^2
- 650 €/mm^2
- Área mínima 3 mm^2
- 25 chips

CMOS 180 nm de AMS

- 118 kGates/mm^2
- 1200 €/mm^2
- Área mínima 5 mm^2
- 25 chips

¿Qué es el Software Libre?

Definition

«Software libre» es el software que respeta la libertad de los usuarios y la comunidad. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Es decir, el «software libre» es una cuestión de libertad, no de precio.

Las cuatro libertades del Software Libre

Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Las cuatro libertades del Software Libre

Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Las cuatro libertades del Software Libre

Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Las cuatro libertades del Software Libre

Un programa es software libre si los usuarios tienen las cuatro libertades esenciales:

- La libertad de ejecutar el programa como se desea, con cualquier propósito (libertad 0).
- La libertad de estudiar cómo funciona el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para ayudar a su prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (libertad 3). Esto le permite ofrecer a toda la comunidad la oportunidad de beneficiarse de las modificaciones. El acceso al código fuente es una condición necesaria para ello.

Temario

¿Cómo diseñar circuitos integrados con herramientas flexibles y accesibles para todo tipo de uso: académico e industrial?

Económico

G:

Académico

G:

Otras razones

G:

Temario

Diseño digital

Selección de la arquitectura.

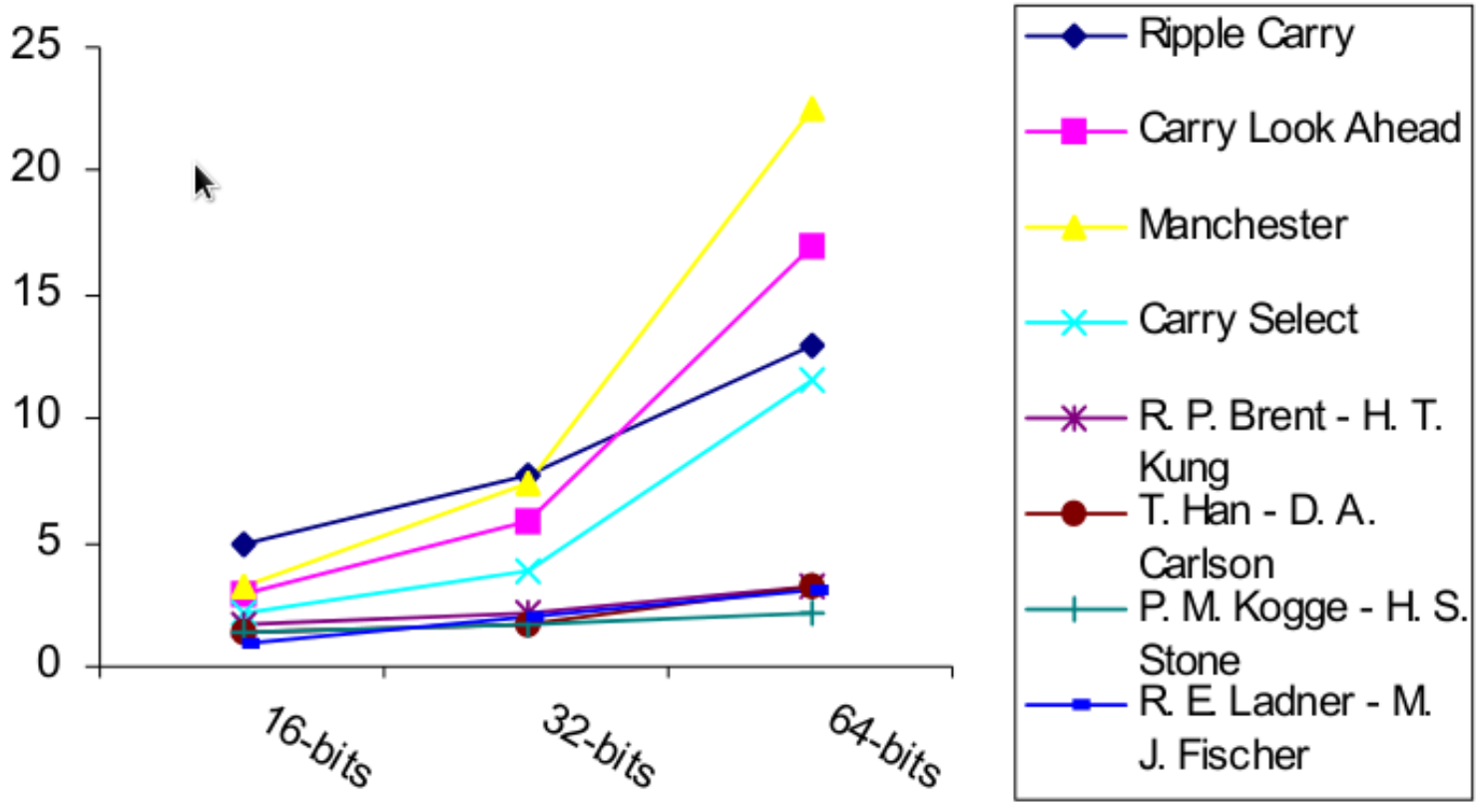


Figura: Retardo respecto al tamaño de los operandos

Diseño digital

Selección de la arquitectura.

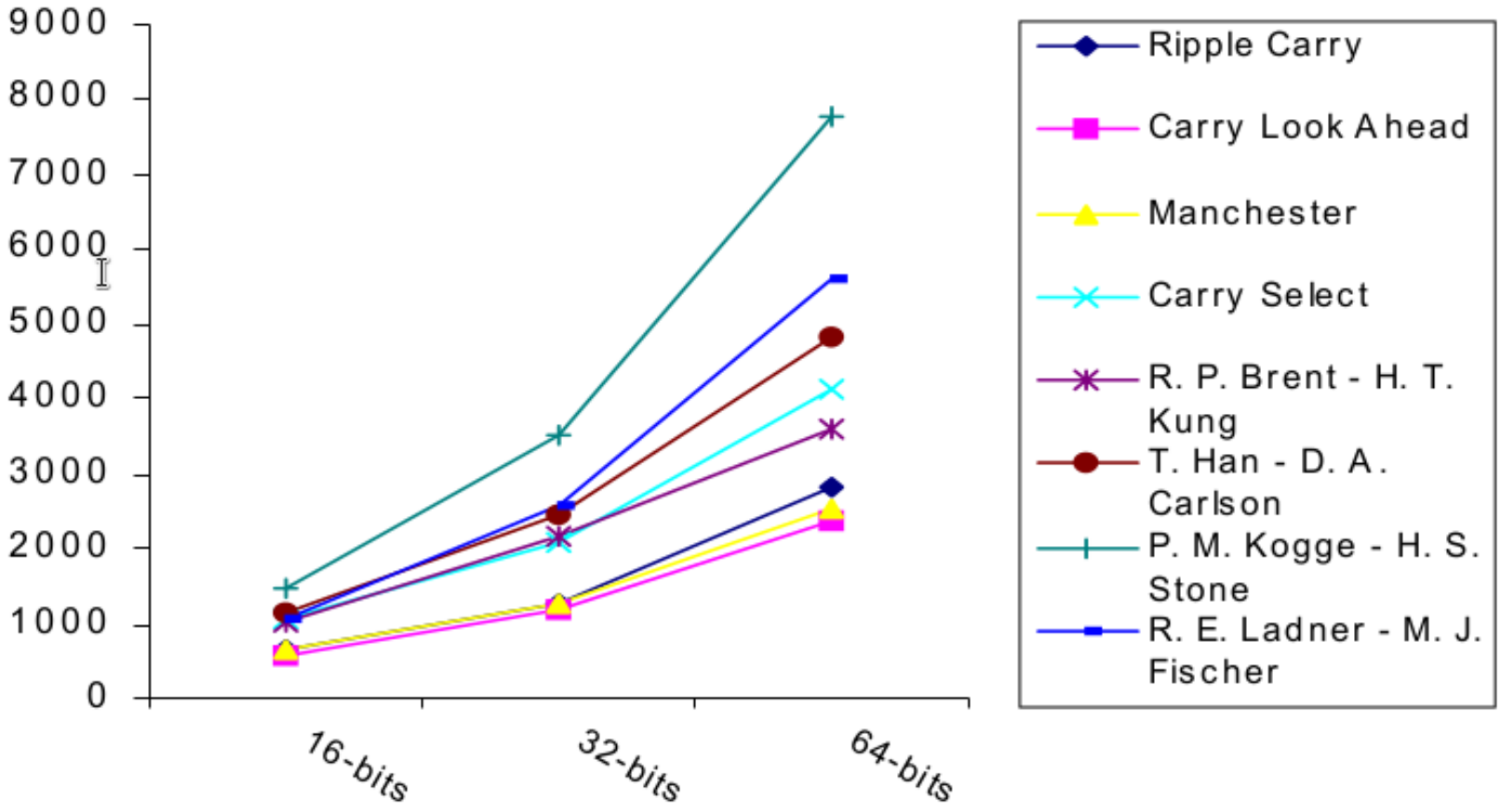


Figura: Área respecto al tamaño de los operandos

Arquitectura	Retardo Máx.	Área
Ripple Carry	$O(n)$	$O(n)$
Carry Look-Ahead	$O(\log_2(n))$	$O(n\log_2(n))$
Ladner-Fisher	$O(\log_2(n))$	$O(n\log_2(n))$
Sklansky	$O(\log_2(n))$	$O(n\log_2(n))$
Kogge-Stone	$O(\log_2(n))$	$O(n\log_2 n)$
Brent-Kung	$O(\log_2(n))$	$O(n)$

Cuadro: Resumen de las funciones de retardo y área de algunos sumadores

▸ Implementación

Carry Look-ahead

Ya que una vez que el acarreo en la posición *i* es conocido, se puede calcular la suma como:

$$s_i = a_i \oplus b_i \oplus c_i \tag{1}$$

El acarreo se *genera* ó se *propaga*, según las siguientes ecuaciones:

$$g_i = a_i b_i$$

$$p_i = a_i \oplus b_i$$

Cálculo recursivo del acarreo:

$$c_{i+1} = g_i + c_i p_i \tag{2}$$

Desenrollando la recurrencia del acarreo

Uno puede desenrollar esta fórmula recursiva del acarreo hasta lograr una función que dependa directamente de los operandos (a y b) y del acarreo de entrada c_{in}:

$$
\begin{aligned}
c_i &= g_{i-1} + p_{i-1} c_{i-1} \\
&= g_{i-1} + p_{i-1}(g_{i-2} + p_{i-2} c_{i-2}) = g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} c_{i-2} \\
&= g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3} + p_{i-1} p_{i-2} p_{i-3} c_{i-3} \\
&= g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3} + p_{i-1} p_{i-2} p_{i-3} g_{i-4} + p_{i-1} p_{i-2} p_{i-3} p_{i-4} c_{i-4}
\end{aligned}
$$

Podemos interpretar estas ecuaciones de la siguiente forma: las cuatro posiciones de bits propagan colectivamente un acarreo c_{in} si y solo sí cada una de las posiciones propaga; y el bloque gener a un acarreo si en la posición $i + 3$ se genera uno, o se podrouce en la posición $i + 2$ y es propagado por la posición $i + 3$, etc.

Problema de prefijos paralelos

Dado:

Entradas: $x_0, x_1, \ldots, x_{k-1}$

Un operador + asociativo

Computar : x_0

$x_0 + x_1$

$x_0 + x_1 + x_2$

$\vdots$

$x_0 + x_1 + x_2 + \cdots + x_{k-1}$

Cómputo del acarreo como un problema de prefijos paralelos

Pensemos la ecuación 3 de la siguiente forma, asumiendo que $c_0 = c_{\text{in}}$ viene desde otro bloque:

$$g_{[i,i+3]} = g_{i+3} + g_{i+2}p_{i+3} + g_{i+1}p_{i+2}p_{i+3} + g_i p_{i+1} p_{i+2} p_{i+3}$$

$$p_{[i,i+3]} = p_i p_{i+1} p_{i+2} p_{i+3}$$

Cómputo del acarreo como un problema de prefijos paralelos

Dados:

Entradas:$(g_0, p_0), (g_1, p_1), \ldots, (g_{k-1}, p_{k-1})$

Un operador $\circ$ asociativo

Computar :

$$(G_0, P_0) = (g_{[0,0]}, p_{[0,0]})$$

$$(G_1, P_1) = (g_{[0,0]}, p_{[0,0]}) \circ (g_{[0,1]}, p_{[0,1]})$$

$$\vdots$$

$$(G_{k-1}, P_{k-1}) = (g_{[0,0]}, p_{[0,0]}) \circ (g_{[0,1]}, p_{[0,1]}) \ldots \circ (g_{[0,k-2]}, p_{[0,k-2]}) \circ (g_{[0,k-1]}, p_{[0,k-1]})$$

Operador de Brent-Kung

El operador $\circ$ se define como:

$$(g, p) \circ (\hat{g}, \hat{p}) = (g \vee (p \wedge \hat{g}), p \wedge \hat{g}) \tag{3}$$

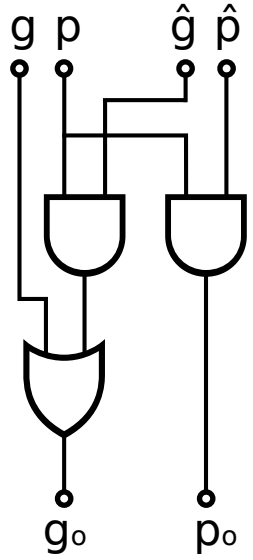


Figura: Operator Punto de Brent-Kung

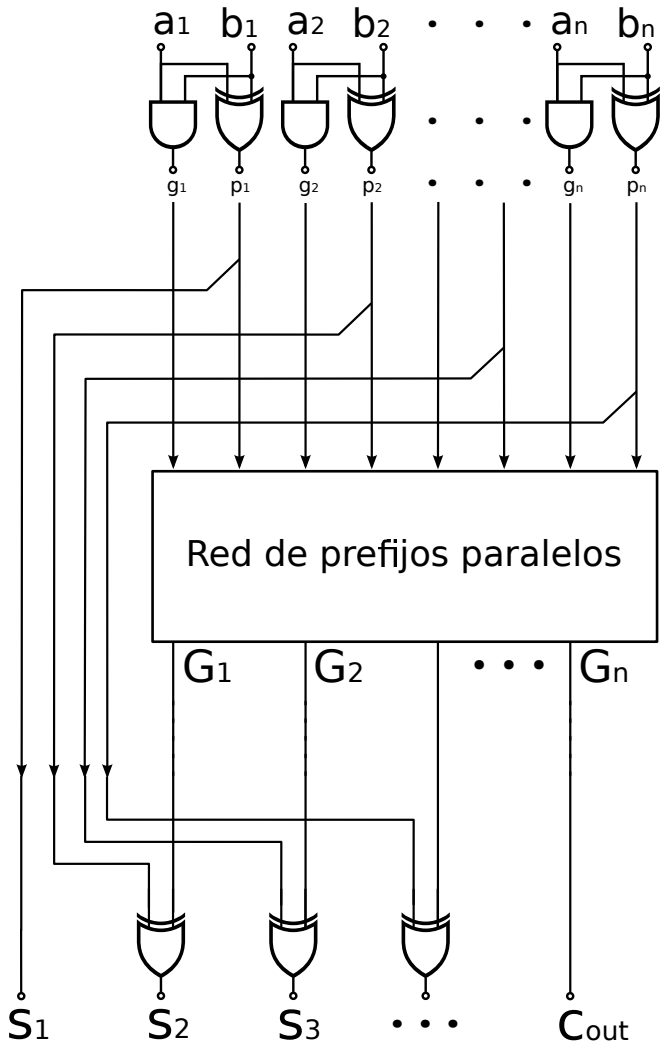


Figura: Sumador de prefijo paralelo

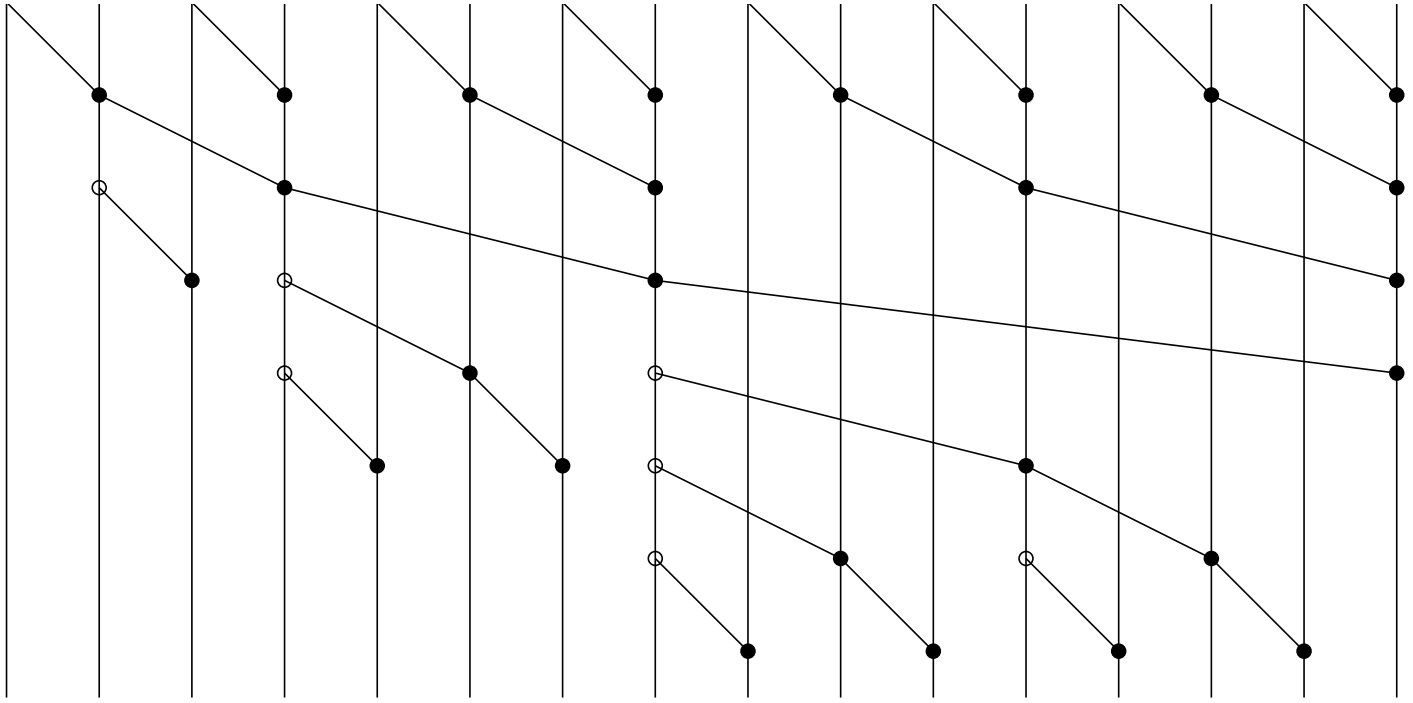


Figura: Red de prefijo paralelo para Brent-Kung (ejemplo de 16 bits)

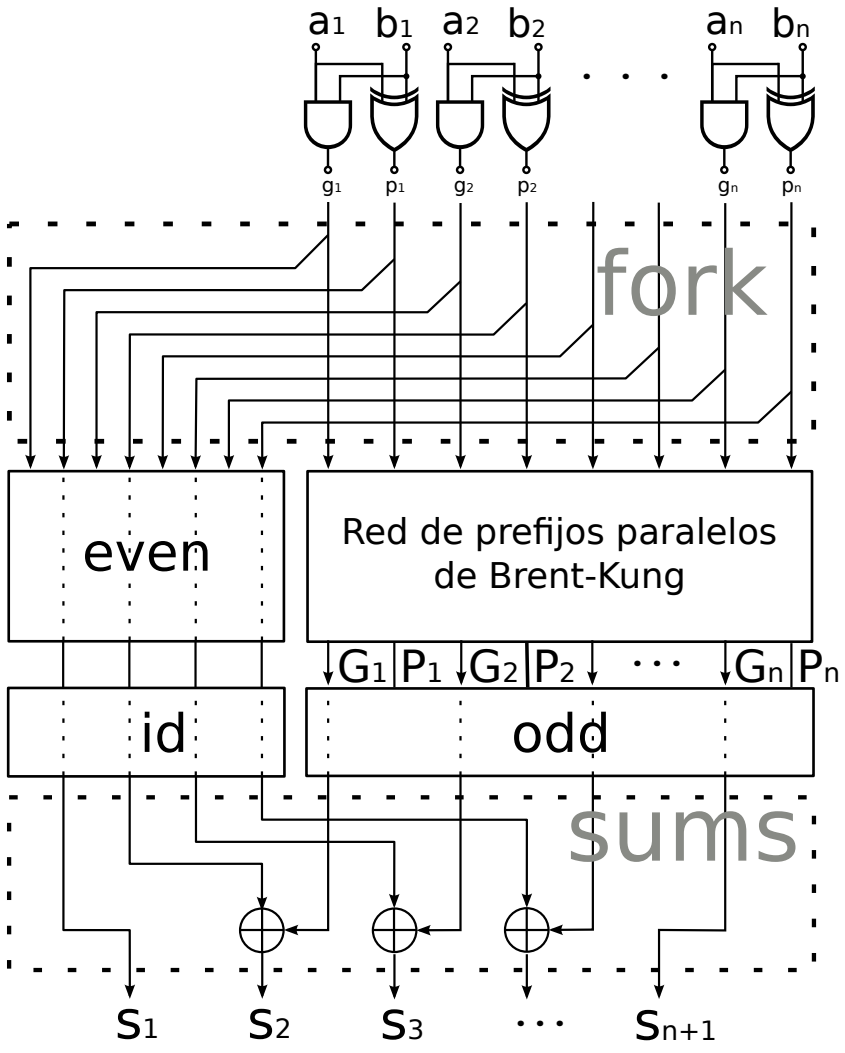


Figura: Sumador de Brent-Kung

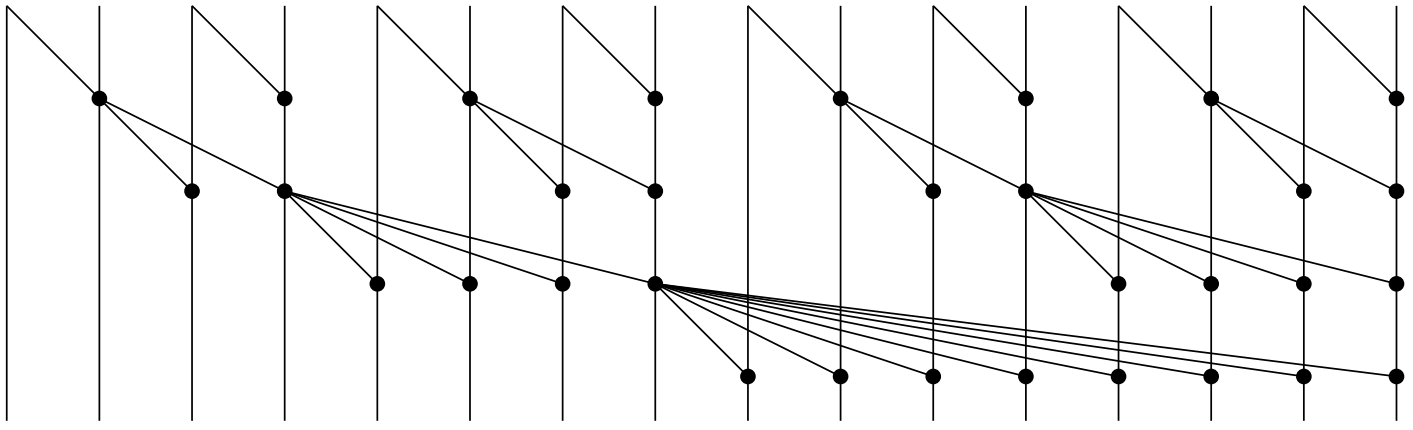


Figura: Red de prefijo paralelo para Sklansky (ejemplo de 16 bits)

Implementación de los circuitos en lenguaje de descripción de hardware

Nuevos lenguajes de descripción de hardware

- Usar un único lenguaje para describir, simular, verificar e implementar el circuito
- Los circuitos se describen en Haskell, Scala o Python, el HDL es simplemente un conjunto de módulos o librerías
- Generar automáticamente una descripción en VHDL o Verilog
- Describir circuitos que construimos a partir de subcircuitos, además de la posibilidad de reutilizar fácilmente patrones de conexión

¿Por qué Lava?

- Conocimiento previo del lenguaje
- Genera un netlist VHDL (Fácil integración con Electric)
- Los circuitos son descriptos como funciones que operan sobre listas, tuplas o sobre circuitos
- Fácil integración con un SAT solver para verificación formal

Operador de Brent-Kung en Lava

A partir de la definición del operador:

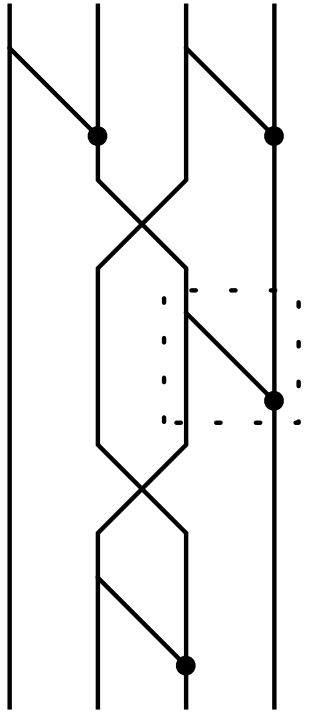
$$(g, p) \circ (\hat{g}, \hat{p}) = (g \vee (p \wedge \hat{g}), p \wedge \hat{g})$$

En Lava la escribimos:

```
dotOp ((g1, p1) ,(g, p)) = (go, po)
   where
      go = or2 (g, and2 (p, g1))
      po = and2 (p, p1)
```

Red de prefijo paralelo de Brent-Kung en Lava

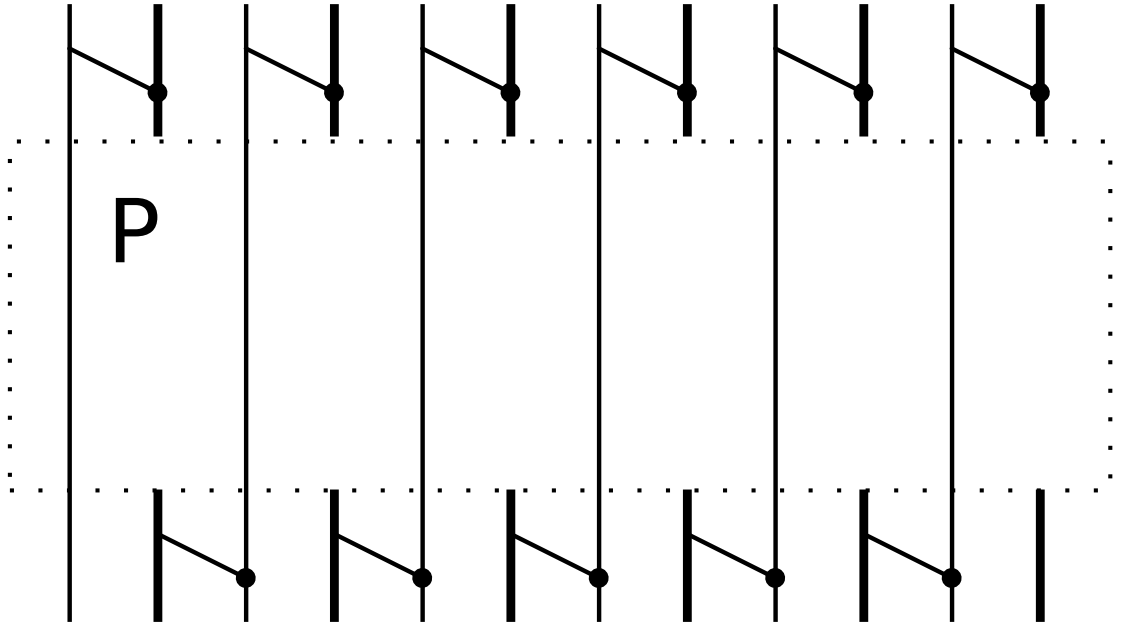
Funciones auxiliares:



```
comb
 ->-
unzippl
 ->-
id -|- p
 ->-
zippl
 ->-
posComb
```

```
comb []     = []
comb [a]    = []
comb (a:as) = dop [a, head as] ++ comb (tail as)
--
posComb (a:as)  = a: (comb (init as))++ [last as]
--
half p = unzipl ->- (id -|- p) ->- zipl
--
wrap p = comb ->- half p ->- posComb
```

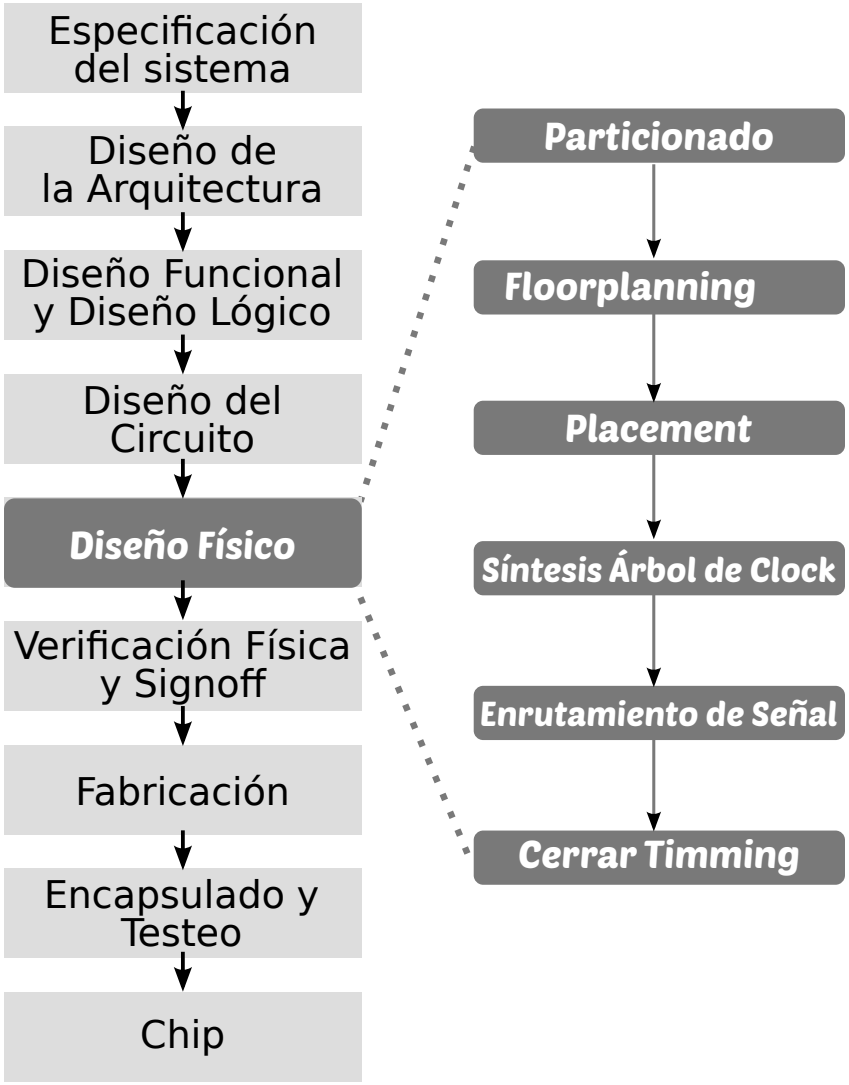
Luego finalemente, podemos describir `ppNet`:




```
ppNet [a]    = []
ppNet [a, b] = dop [a, b]
ppNet as     = wrap ppNet as
```

Temario

Flujo de diseño físico



Selección del proceso de fabicación

Seleccionamos TSMC 180nm por las siguientes razones:

- Existen herramientas de software libre para esta tecnología.
- Bajo costo de fabricación
- Posibilidad de integrar sistemas de gran complejidad y alta performance

Selección del proceso de fabicación

Ejemplos de microprocesadores que fueron fabricados en esta tecnología:

Procesador	Año de lanzamiento
Intel Coppermine E	1999
AMD Athlon Thunderbird	2000
Intel Celeron (Willamette)	2002
Motorola PowerPC 7445 y 7455 (Apollo 6)	2002

Cuadro: Procesadores fabricados en CMOS 180nm

Diseño físico

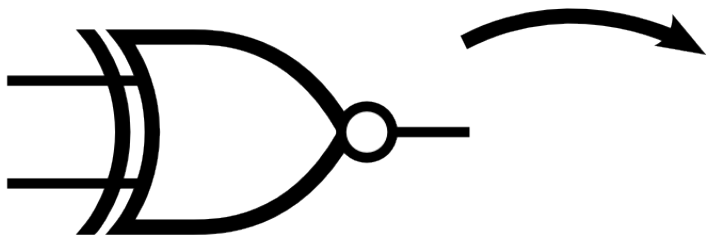
Reglas de diseño para TSMC 180nm

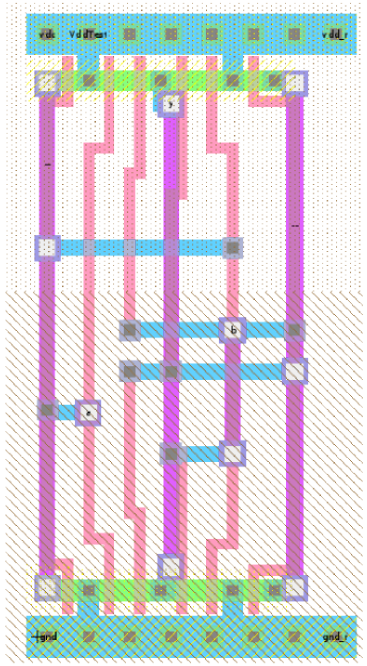
MOSIS denomina a las reglas de diseño **SCN6M_DEEP**, que significa:

- S: Escalable
- C: Tecnología de fabricación CMOS
- N: Pozo N.
- 6M: 6 metales y un conductor policristalino (*poly*) para crear las compuertas.
- DEEP: Reglas *deep submicron* (lamda 90nm).

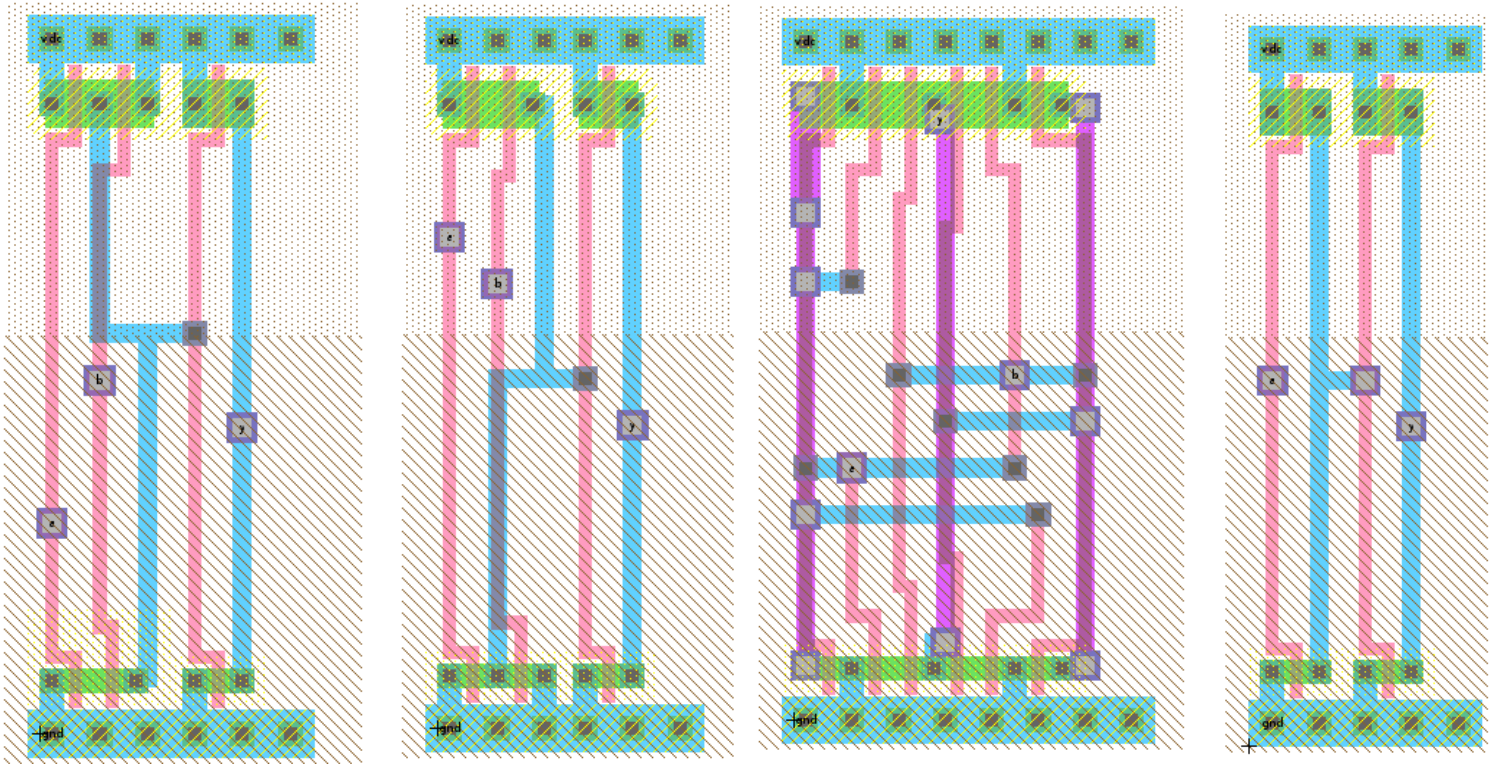
Mapeo de lógica a compuertas

Mapeo de una función lógica a una celda estándar





Librería de celdas estándar

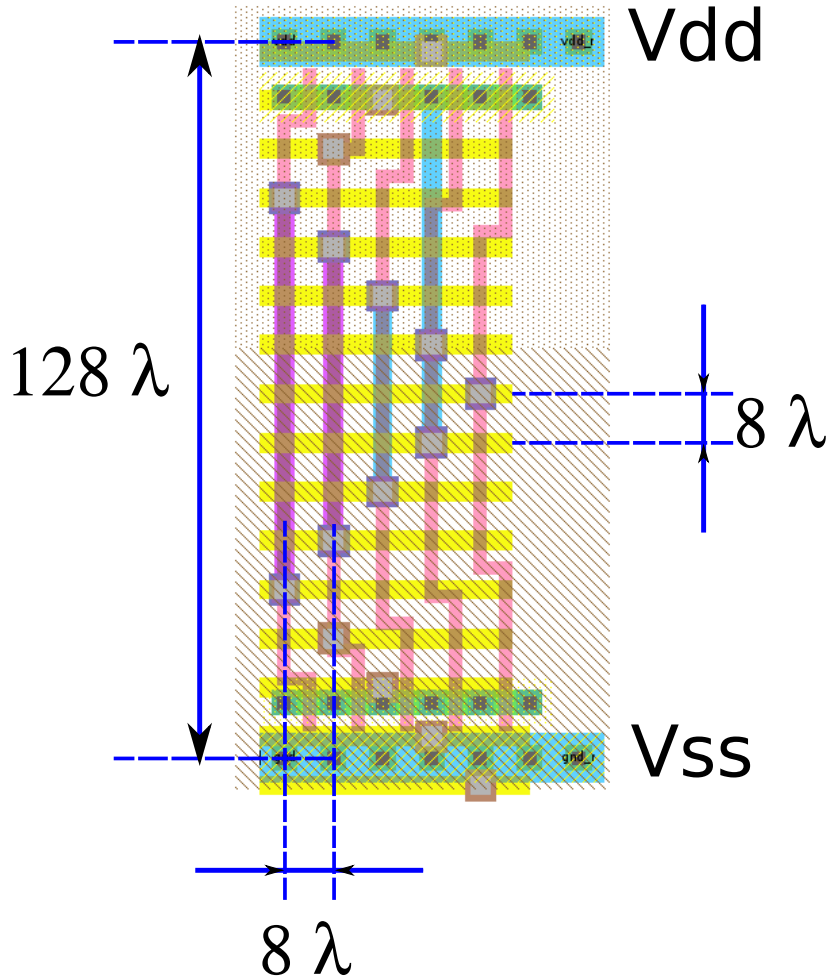


(a) and2 **(b)** or2 **(c)** xor **(d)** buffer

Figura: Conjunto de celdas estándar

Celdas estándar

Grilla de interconexionado y riel de alimentación de las celdas estándar de 128λ



Ubicación y conexionado del *ripple carry adder*

Ripple Carry	8 bits			16 bits			32 bits		
filas	3	4	5	5	6	7	8	7	6
ancho	1297	966	843	1562	1350	1142	1881	2169	2581
alto	665	839	958	1227	1196	1600	2000	1850	1360
área	862505	810474	807594	1916574	1614600	1827200	3762000	4012650	3510160
ancho/alto	0,51	0,87	1,14	0,79	0,89	1,40	1,06	0,85	0,53

Cuadro: Las dimensiones de los lados y el área están en λ y λ^2 respectivamente.

Ubicación y conexionado del sumador de *Brent-Kung*

Brent-Kung	8 bits			16 bits				32 bits			
filas	3	4	5	4	5	6	7	6	7	8	9
ancho	1386	1090	945	2268	1757	1545	1429	3196	1983	2569	2424
alto	746	910	1199	1255	1436	1540	1959	2024	2871	2927	2882
área	1033956	991900	1133055	2846340	2523052	2379300	2799411	6468704	5693193	7519463	6985968
ancho/alto	0,54	0,83	1,27	0,55	0,82	1,00	1,37	0,63	1,45	1,14	1,19

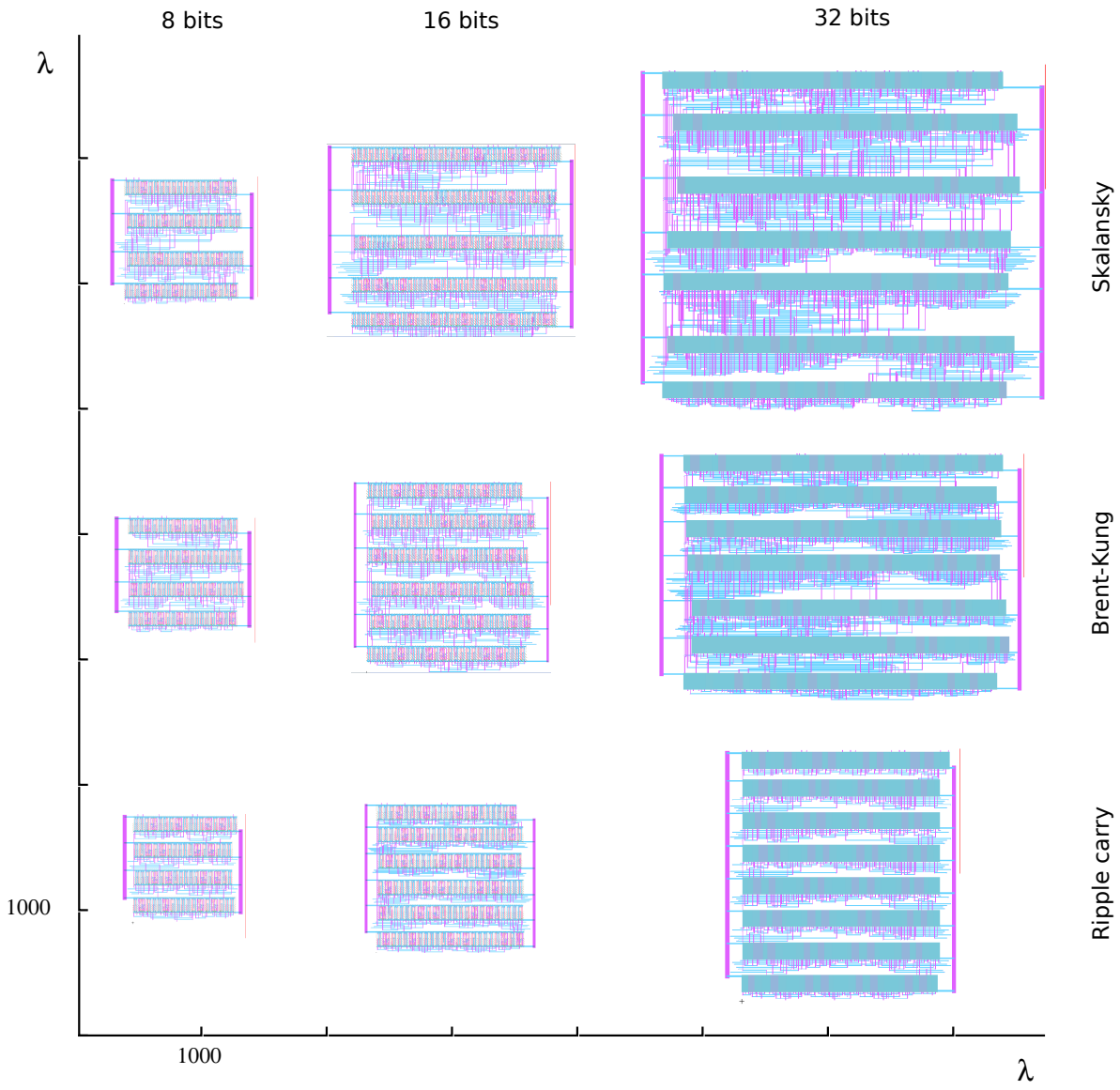
Cuadro: Las dimensiones de los lados y el área están en λ y λ^2 respectivamente.

Ubicación y conexionado del sumador de *Sklansky*

Sklansky	8 bits			16 bits				32 bits		
filas	3	4	5	4	5	6	7	6	7	8
ancho	1516	1167	954	3538	2042	1825	1536	3678	3229	2860
alto	810	973	1252	1345	1581	1878	2063	2639	2695	3072
área	1227960	1135491	1194408	4758610	3228402	3427350	3168768	9706242	8702155	8785920
ancho/alto	0,53	0,83	1,31	0,38	0,77	1,03	1,34	0,72	0,83	1,07

Cuadro: Las dimensiones de los lados y el área están en λ y λ^2 respectivamente.

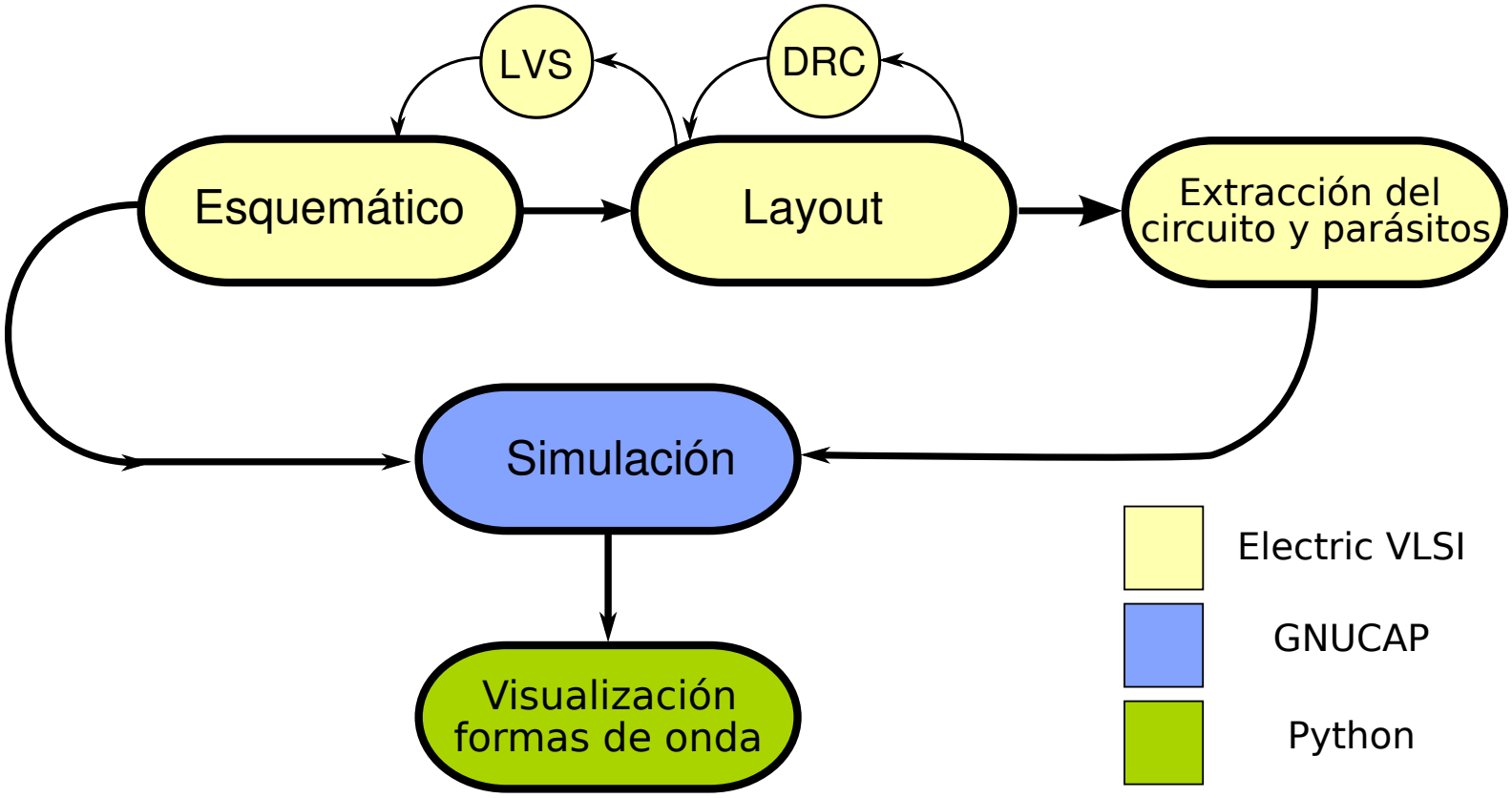
Layout de todas las arquitecturas y tamaños



Simulación post *layout* para calcular performance y potencia

Realizamos extracción de parásitos del *layout* y utililizamos un motor de simulación analógico tipo SPICE, llamado gnucap.

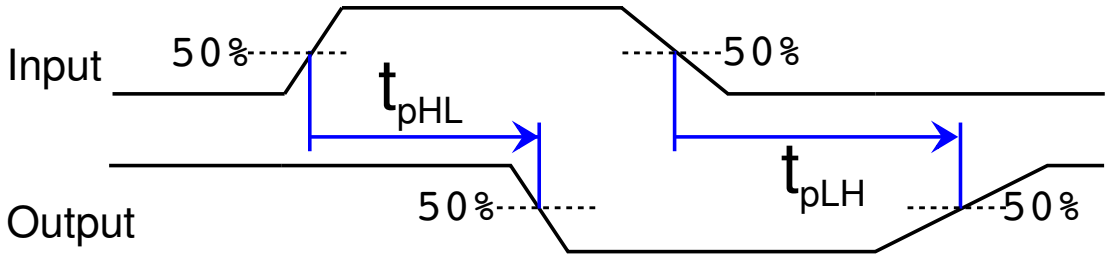
Flujo para simulaciones analógicas



Retardo de propagación

Definición:

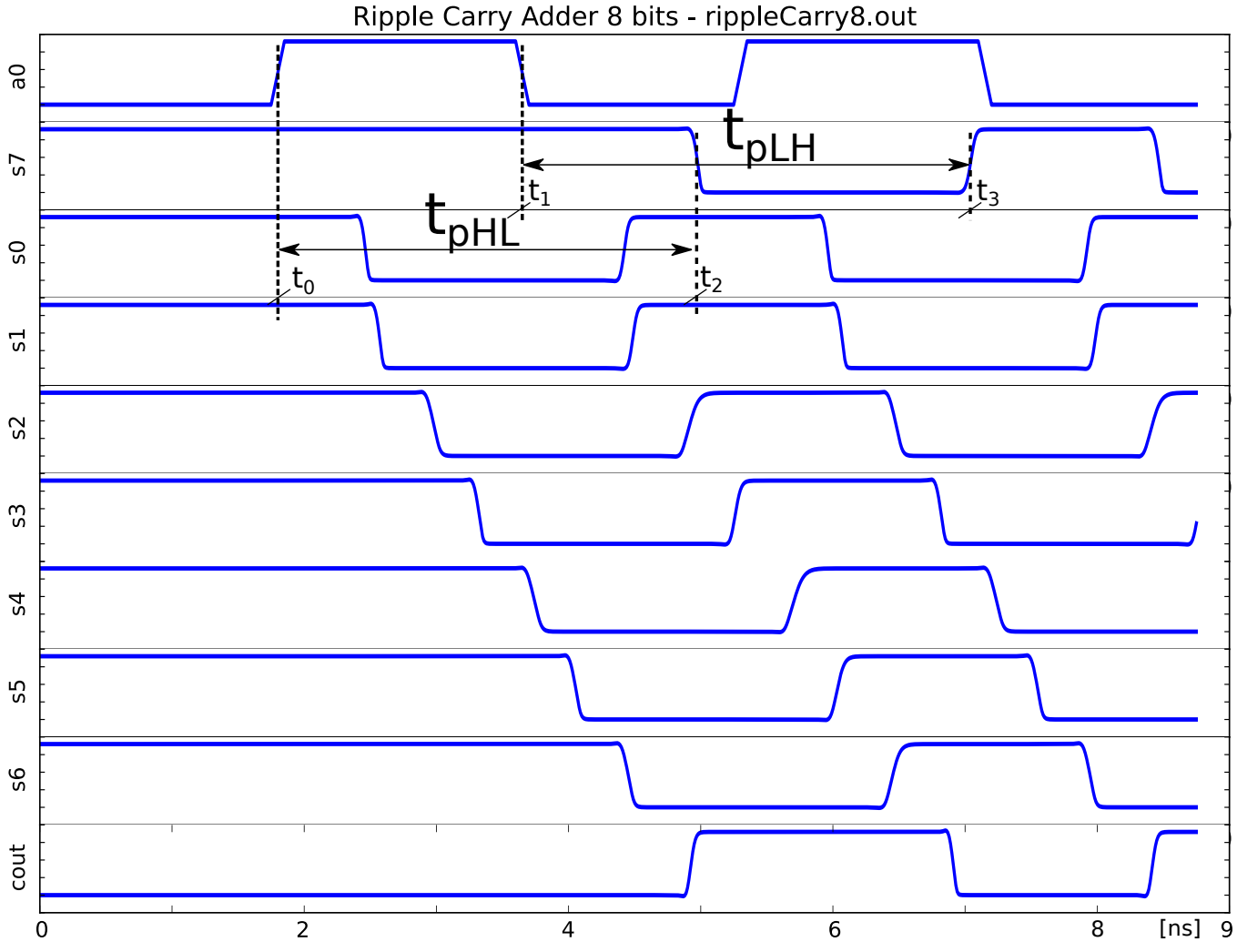
Retardo de propagación de un inversor:



Se define usualmente el retardo de propagación como:

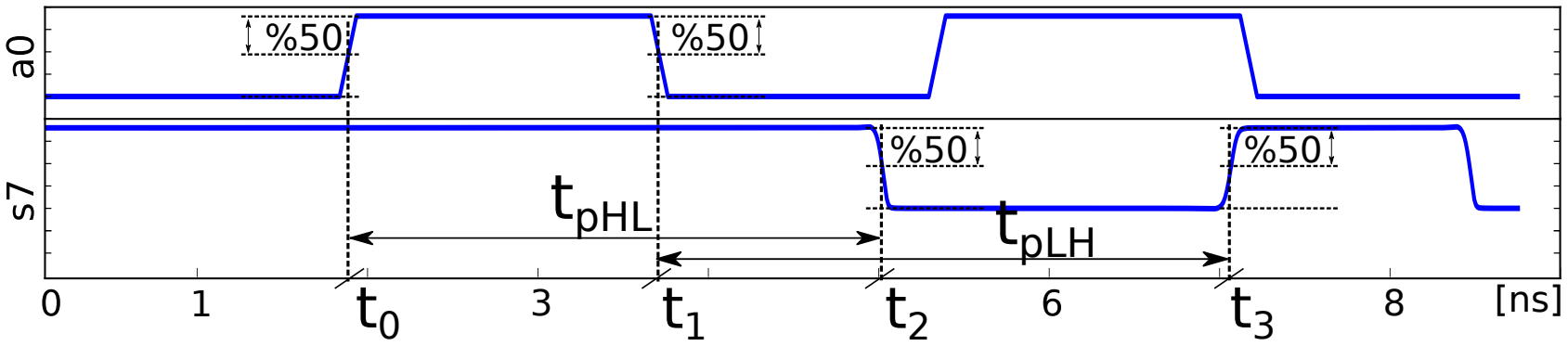
$$t_p = \frac{t_{pLH} + t_{pHL}}{2}$$

Simulación post *layout* para calcular performance



Simulación post *layout* para calcular performance

Sumador de 8 bits de *ripple carry*



$$t_{pHL} = t_2 - t_0 = 5{,}01 \text{ ns} - 1{,}8 \text{ ns} = 3{,}21 \text{ ns}$$

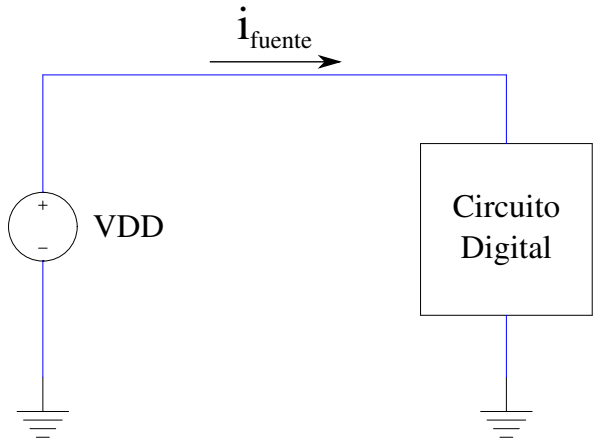
$$t_{pLH} = t_3 - t_1 = 7{,}05 \text{ ns} - 3{,}61 \text{ ns} = 3{,}44 \text{ ns}$$

$$t_p = \frac{(t_{pHL} + t_{pLH})}{2} = 3{,}325 \text{ ns}$$

Potencia promedio disipada

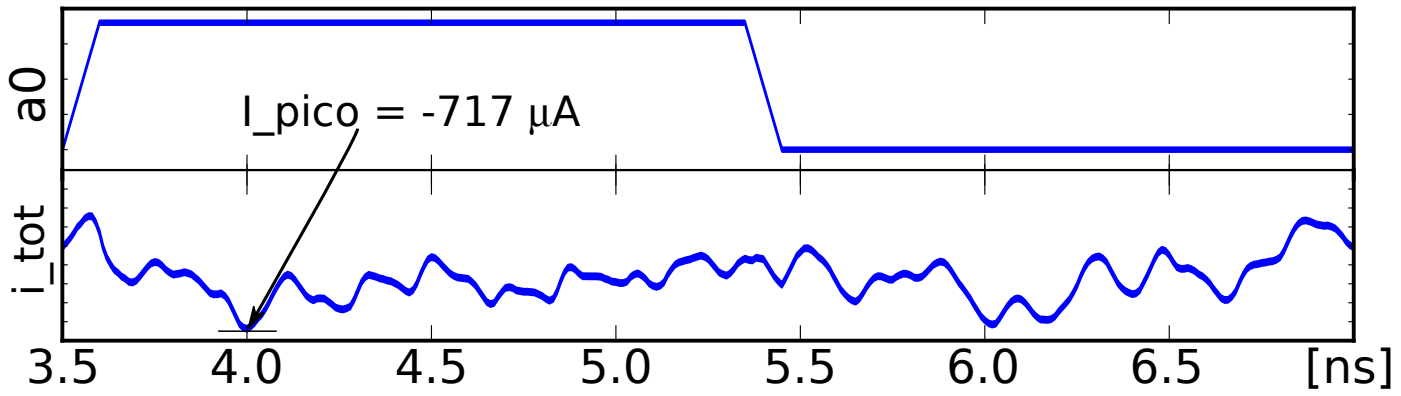
Definición:

La potencia promedio disipada total la podemos calcular si conocemos la corriente instantánea que brinda la fuente de tensión V_{DD}, como podemos ver en la ecuación 4.



$$P_{av} = \frac{1}{\mathrm{T}} \int_0^T p(t)dt = \frac{\mathrm{V_{DD}}}{\mathrm{T}} \int_0^T i_{\mathrm{fuente}}(t)\mathrm{d}t \qquad (4)$$

Simulación de régimen transitorio del circuito Ripple Carry 8 bits



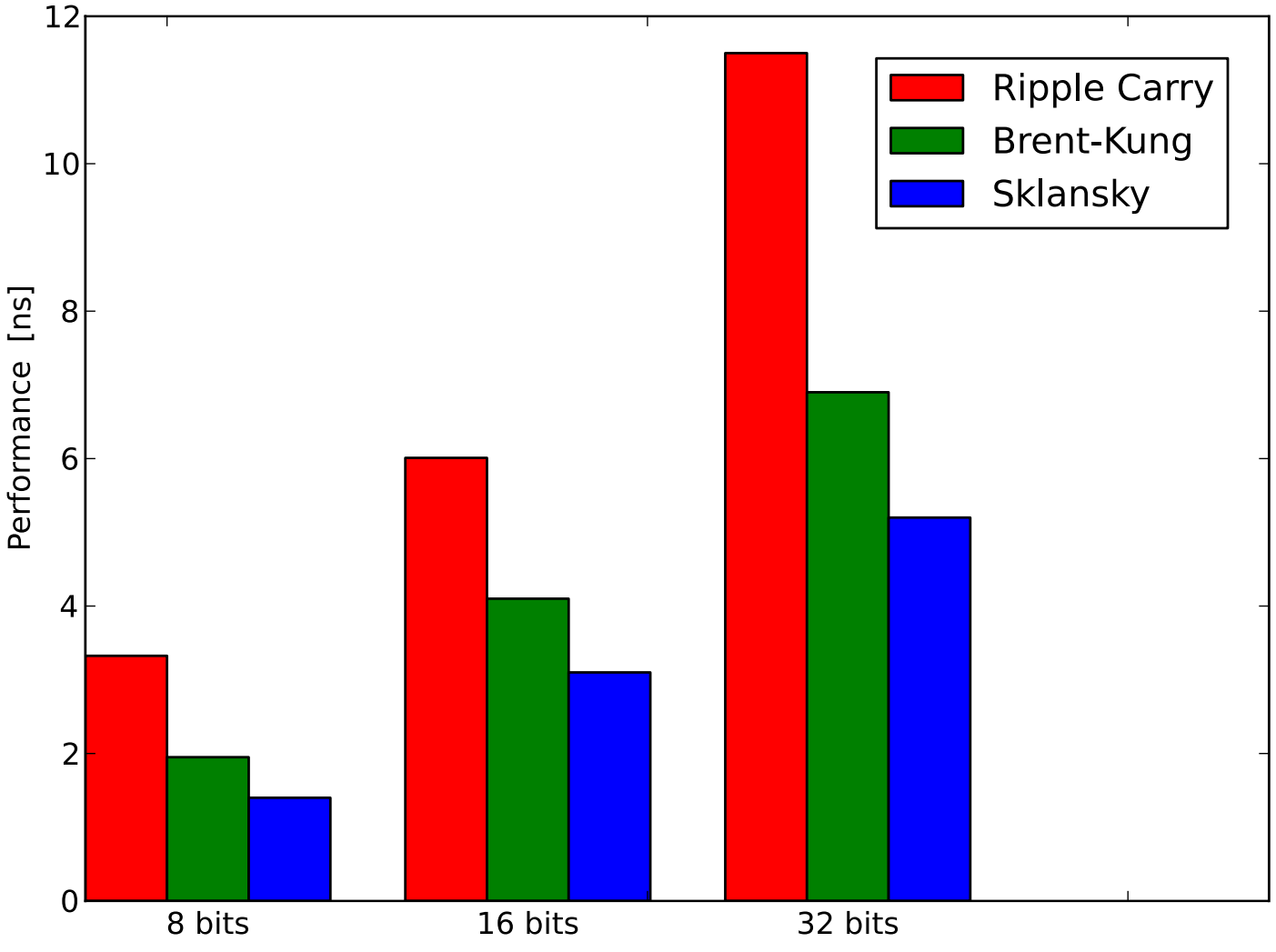
El período de integración que elegimos está determinado por el t_p del circuito, lo que físicamente quiere decir: Medimos la potencia del circuito cuando está funcionando a la mayor velocidad posible.

Simulación post *layout* para calcular performance y potencia

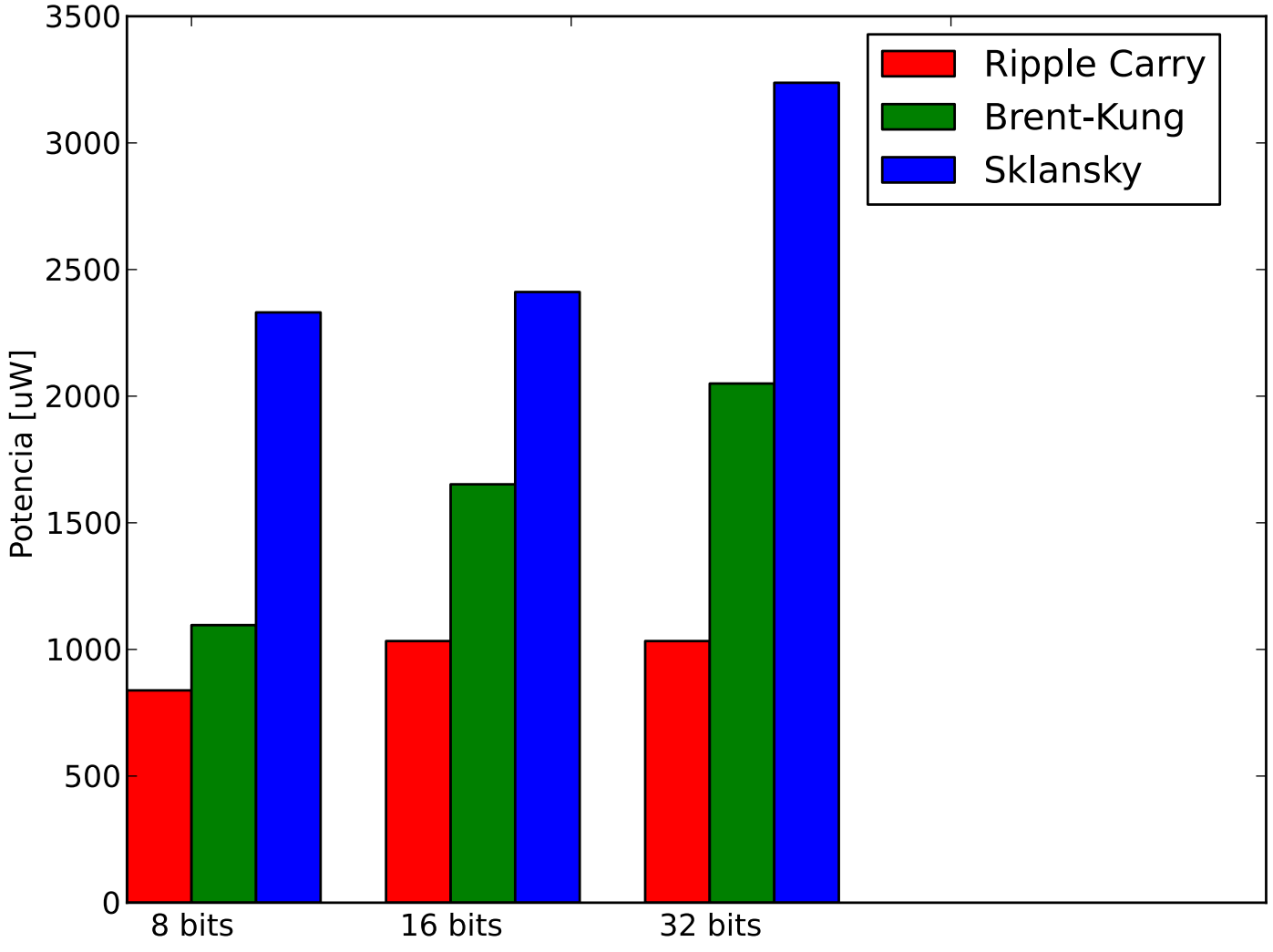
Simulación post *layout*

Realizamos las simulaciones de todas las arquitecturas y de los tres tamaños.

Resultados: Performance



Resultados: Potencia



Resumen

Por lo tanto, hemos logrado un conjunto de sumadores que según los requerimientos de área, potencia y performance, podremos elegir la arquitectura más adecuada.
Para todos los tamaños de sumadores,

Para sumadores de 32 bits

La mayor velocidad se logra con Sklansky y el mejor compromiso entre velocidad, potencia y área con Brent-Kung.

Para todos los tamaños

Si la performance no es un problema, un ripple carry es la solución optima de estos tres, ya que ahorra área y energía.

Conclusiones

- Sumadores rápidos, eficientes o de bajo consumo.

Resumen