

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

Leandro Marsó¹

¹ Universidad Nacional de Córdoba, Argentina

Facultad de Ciencias Exactas, Físicas y Naturales, 2015

Temario

1 Introducción

- Definiciones generales
- Planteamiento del problema y motivación

2 Implementación

- Diseño digital
- Diseño físico

3 Conclusiones

- Definiciones generales

- Planteamiento del problema y motivación

- Diseño digital

- Diseño físico

3 Conclusiones

Un sumador binario, de n bits que calcula de forma rápida el acarreo.

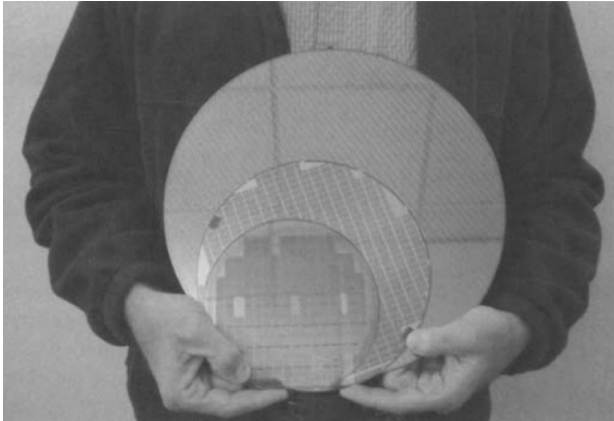
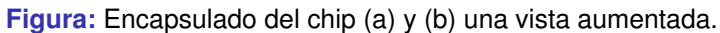
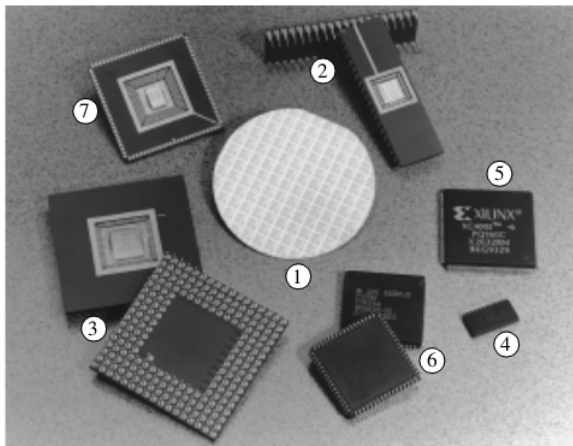


Figura: Obleas de silicio de 150, 200 y 300 mm de diámetro, de un proceso CMOS.





- 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.

¿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 μ m - 0.7 μ m
Austria Micro Systems	180 nm - 0.35 μ m

Tabla: Procesos disponibles por medio de MOSIS

¿Cómo accedemos a fabricar circuitos integrados?

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

Tabla: Procesos disponibles por medio de CMP

¿Qué es un circuito integrado?

¿Cuánto podemos integrar?

CMOS 350 nm de AMS

- 18kGates/mm²
- 650 €/mm²
- Área mínima 3 mm²
- 25 chips

CMOS 180 nm de AMS

- 118 kGates/mm²
- 1200 €/mm²
- Área mínima 5 mm²
- 25 chips

¿Qué es el Software Libre?

Definición

«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
- La libertad de estudiarlo
- La libertad de redistribuir copias
- La libertad de distribuir copias de sus versiones modificadas

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
- La libertad de estudiarlo
- La libertad de redistribuir copias
- La libertad de distribuir copias de sus versiones modificadas

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
- La libertad de estudiarlo
- La libertad de redistribuir copias
- La libertad de distribuir copias de sus versiones modificadas

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
- La libertad de estudiarlo
- La libertad de redistribuir copias
- La libertad de distribuir copias de sus versiones modificadas

Temario

1

Introducción

- Definiciones generales
- Planteamiento del problema y motivación

2

Implementación

- Diseño digital
- Diseño físico

3

Conclusiones

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

Económico

Factibilidad de proyectos según la escala

Académico

Oportunidad de abordaje multidisciplinario

Otras razones

Soberanía tecnológica

Temario

1 Introducción

- Definiciones generales
- Planteamiento del problema y motivación

2 Implementación

- Diseño digital
- Diseño físico

3 Conclusiones

Diseño digital

Selección de la arquitectura

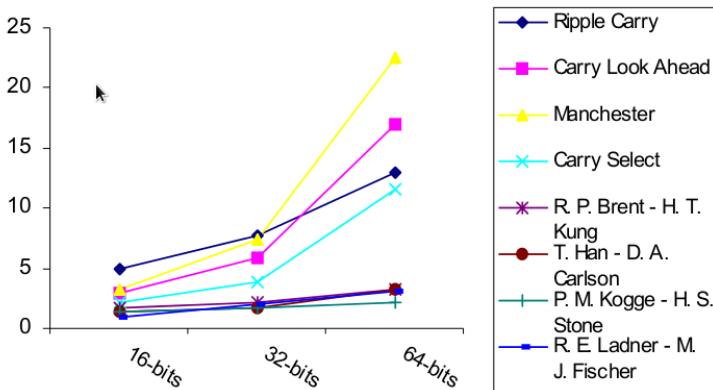


Figura: Retardo respecto al tamaño de los operandos

Diseño digital

Selección de la arquitectura

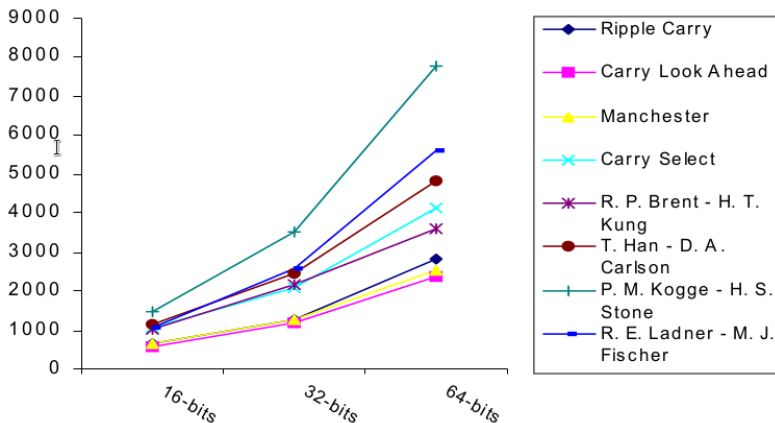


Figura: Área respecto al tamaño de los operandos

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	$O(\log_2(n))$	$O(n)$

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

Carry Look-ahead

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

$$s_i = a_i \oplus b_i \oplus c_i \quad (1)$$

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:

$$c_{i+1} = g_i + c_i p_i \quad (2)$$

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 si cada una de las posiciones propaga; y el bloque genera un acarreo si en la posición $i + 3$ se genera uno, o se produce en la posición $i + 2$ y es propagado por la posición $i + 3$, etc.

Problema de prefijos paralelos

Dado:

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

Un operador + asociativo

Computar : x_0

$$x_0 + x_1$$

$$x_0 + x_1 + x_2$$

$$\vdots$$

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

Cómputo del acarreo como un problema de prefijos paralelos

Pensemos la ecuación 3 de la siguiente forma, asumiendo que $c_0 = c_{in}$ 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: $(g_0, p_0), (g_1, p_1), \dots, (g_{k-1}, p_{k-1})$

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

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

Operador de Brent-Kung

El operador \circ se define como:

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

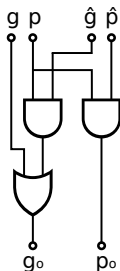


Figura: Operator Punto de Brent-Kung

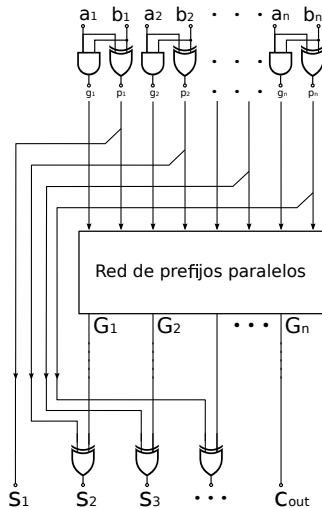


Figura: Sumador de prefijo paralelo

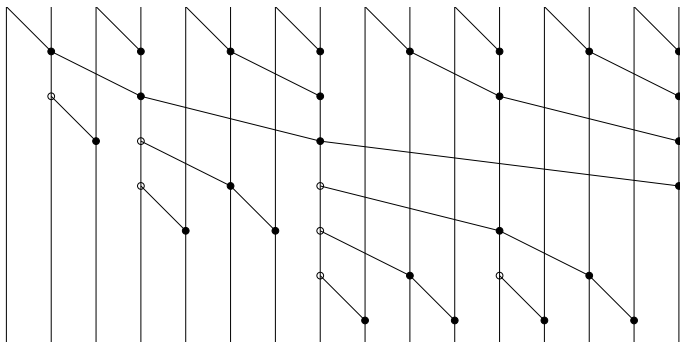


Figura: Red de prefijo paralelo para Brent-Kung (ejemplo de 16 bits)

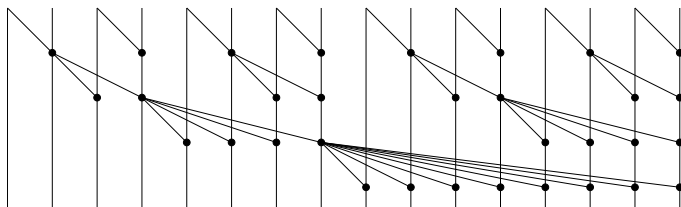


Figura: Red de prefijo paralelo para Sklansky (ejemplo de 16 bits)

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 descritos 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:

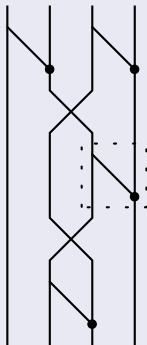
$$(g, p) \circ (\hat{g}, \hat{p}) = (g \vee (p \wedge \hat{g}), p \wedge \hat{p})$$

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

Primeras iteraciones que construyen la red de BK



```

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

Funciones auxiliares:

comb

```
comb []      = []  
comb [a]     = []  
comb (a:as) = dop [a, head as] ++ comb (tail as)
```

posComb

```
posComb (a:as) = a: (comb (init as)) ++ [last as]
```

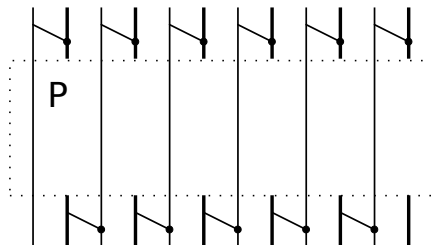
half

```
half p = unzipl ->- (id -|- p) ->- ziplt
```

wrap

```
wrap p = comb ->- half p ->- posComb
```

Luego finalmente, podemos describir ppNet:



ppNet

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

Circuito completo

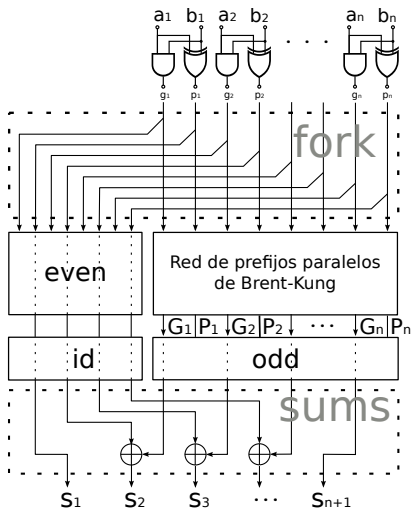


Figura: Sumador de Brent-Kung

De Lava a VHDL

A partir de nuestra implementación del circuito podemos:

- Simular
- Verificar formalmente
- Generar un netlist VHDL estructural

Temario

1 Introducción

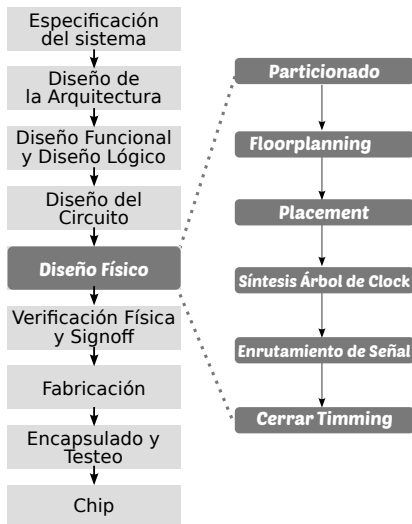
- Definiciones generales
- Planteamiento del problema y motivación

2 Implementación

- Diseño digital
- Diseño físico

3 Conclusiones

Flujo de diseño físico



Selección del proceso de fabricación

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 fabricación

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

Tabla: Procesadores fabricados en CMOS 180nm

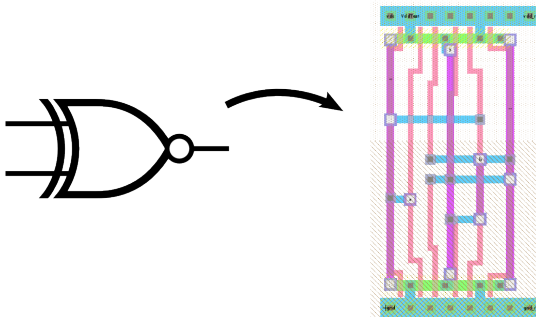
Reglas de diseño para TSMC 180nm

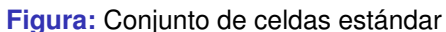
MOSIS denomina a las reglas de diseño **SCN6M_DEEP**, que significa:

- 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* (lambda 90nm).

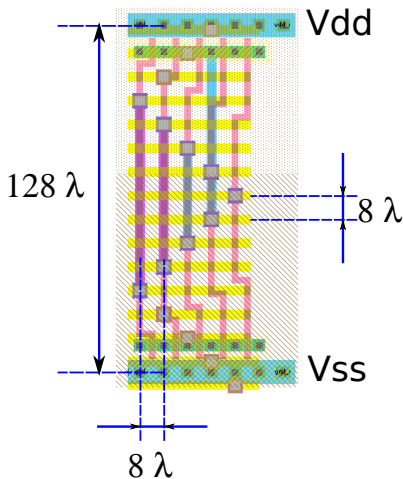
Mapeo de lógica a compuertas

Mapeo de una función lógica a una celda estándar





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

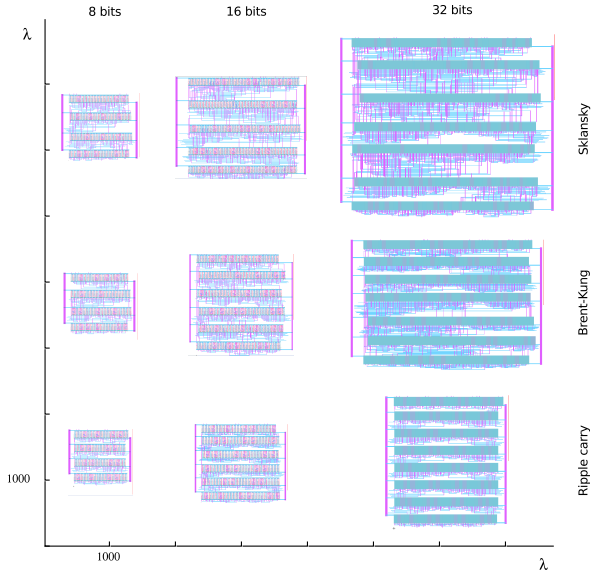


Ubicación y conexionado del sumador de *Brent-Kung*

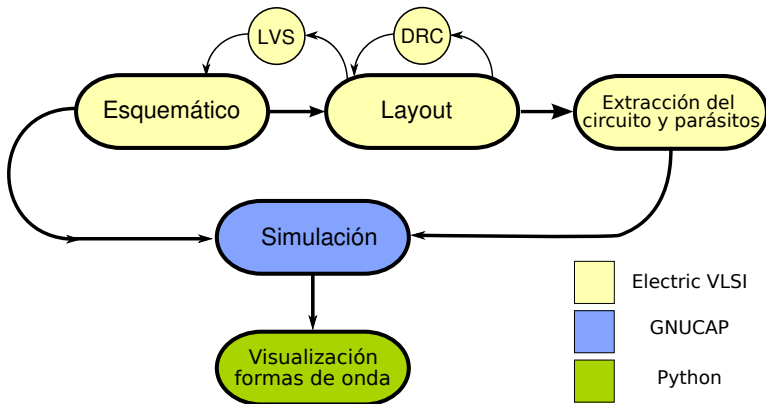
Brent-Kung	8 bits			16 bits				32 bits			
filas	3	4	5	4	5	6	7	6	7	8	9
ancho	1386	1090	945	2268	1757	1545	1429	3196	1983	2569	2424
alto	746	910	1199	1255	1436	1540	1959	2024	2871	2927	2882
área	1033956	991900	1133055	2846340	2523052	2379300	2799411	6468704	5693193	7519463	6985968
ancho/alto	0,54	0,83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

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

Layout de todas las arquitecturas y tamaños

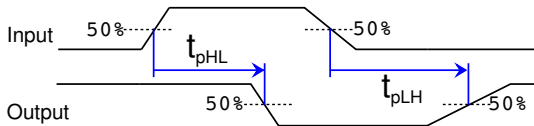


Realizamos extracción de parásitos del *layout* y utilizamos un motor de simulación analógico tipo SPICE, llamado gnucap.



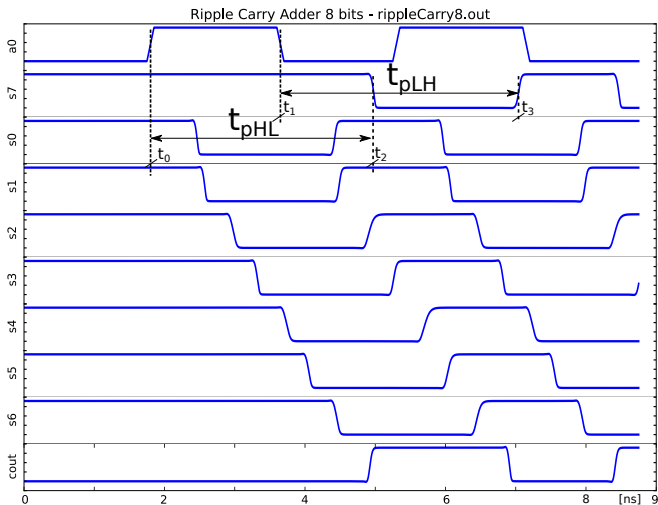
Realizamos las simulaciones de todas las arquitecturas y de los tres tamaños.

Retardo de propagación de un inversor:



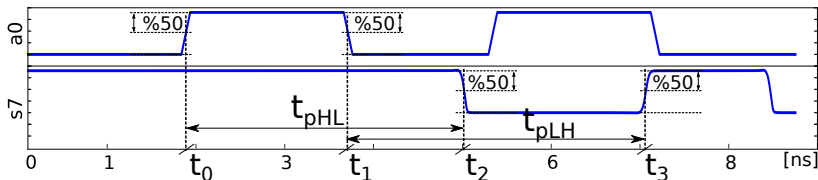
Se define usualmente el retardo de propagación como:

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



Simulación post *layout* para calcular performance

Sumador de 8 bits de *ripple carry*

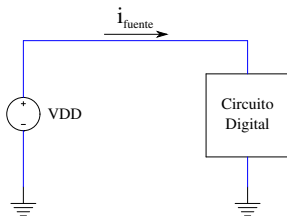


$$t_{pHL} = t_2 - t_0 = 5,01 \text{ ns} - 1,8 \text{ ns} = 3,21 \text{ ns}$$

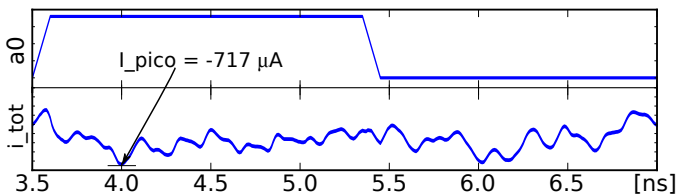
$$t_{pLH} = t_3 - t_1 = 7,05 \text{ ns} - 3,61 \text{ ns} = 3,44 \text{ ns}$$

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

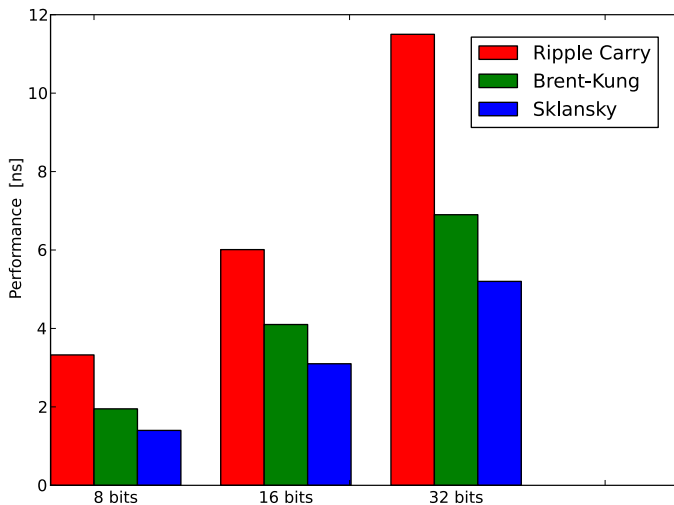
La potencia promedio disipada total la podemos calcular si conocemos la corriente instantánea que brinda la fuente de tensión V_{DD} , como podemos ver en la ecuación 4.

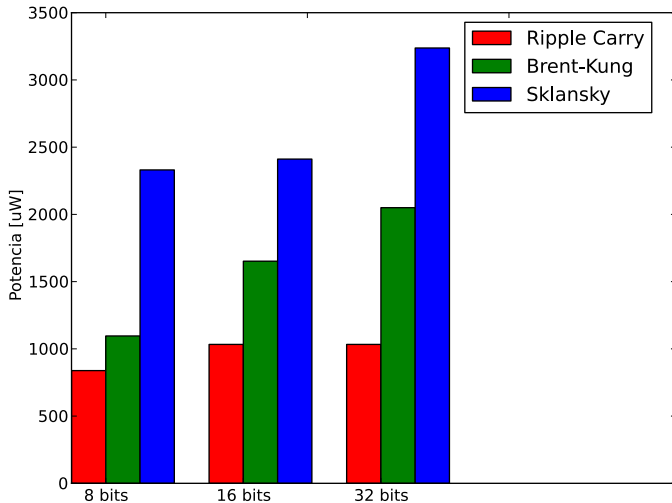


$$P_{av} = \frac{1}{T} \int_0^T p(t) dt = \frac{V_{DD}}{T} \int_0^T i_{fuente}(t) dt \quad (4)$$



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.





Resumen

Hemos logrado un conjunto de sumadores que según los requerimientos de área, potencia y performance, podremos elegir la arquitectura más adecuada.

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 óptima de estos tres, ya que ahorra área y energía.

Conclusiones

- Metodología flexible
 - Selección de la herramienta según la necesidad
 - Integración del flujo con modificaciones simples
 - Todos los procesos son automáticos
- Resultados del mismo orden de magnitud que otros estudios
 - Sumadores de cualquier tamaño
 - Sumadores rápidos, eficientes o de bajo consumo
- Metodología para circuitos combinacionales
 - Unidades aritméticas, decodificadores, codificadores, funciones lógicas
 - También diseños analógicos

¡Gracias!