Universidad Nacional de Córdoba
Facultad de Ciencias Exáctas, Físicas y Naturales

Proyecto Integrador de la Carrera Ingeniería Electrónica

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

Noviembre 2014

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1
INTRODUCCIÓN

En el presente capítulo se describe en rasgos generales el flujo para el diseño de Circuitos Integrados de Aplicación Específica (*ASIC* por su sigla en inglés), y la metodología utilizada para llevar adelante el diseño, implementación y tape out del mismo.

1.1. Estructura del Proyecto Integrador

Este proyecto cuenta con 4 partes: Las primeras tres partes:
??: Selección de la Arquitectura
??: Implementación Física
III: Comparación de resultados.
Estas tres partes forman un flujo de trabajo circular e iterativo.
Y finaliza con un resumen de las conclusiones, además de conjunto de anexos técnicos ubicados al final para su consulta.

1.2. Planteamiento del problema y motivación

En la actualidad los microprocesadores, los DSP, los microcontroladores, y otro hardware específico para cálculo computacional son desarrollados en tecnología CMOS submicrónica. El problema planteado es, ¿Cómo hacer para diseñar circuitos integrados en esta tecnología, con herramientas flexibles, libres[1] y accesibles para todo tipo de uso: academico y comercial?.

1.3. Objetivo

El objetivo del trabajo es diseñar un sumador de n-bits, por ser este el elemento central de cualquier tipo de circuito digital de cálculo: Los multiplicadores, los MAC (*multiply-accumulate*), los filtros FIR, etc, que pueda ser enviado a fabricar utilizando procesos de fabricación CMOS para circuitos integrados. Integrar y documentar un flujo de diseño de este sistema digital utilizando herramientas de Software Libre, será un subproducto de este diseño, para lograr la base de conocimiento necesaria en el diseño de circuitos integrados con tecnología CMOS. Este trabajo

[1] En el sentido que no impongan restricciones de uso, estudio, mejora y distribución.

además de integrar todos los procesos de diseño de un Circuito Integrado, pretende facilitar el acceso a las herramientas de diseño de circuitos integrados a los estudiantes de grado.

1.4. Plan de Trabajo

Parte I

Diseño Digital

Capítulo 2

ESPECIFICACIONES DE DISEÑO

2.1. Introducción

Los sumadores binarios son utilizados en la adición, la resta, la multiplicación y la división. La velocidad de un sistema de procesamiento de señales, o un sistema de comunicación depende fuertemente de **estas unidades funcionales(10)**. Para cada una de esas operaciones, son necesarios sumadores de distinta cantidad de bits en el mismo diseño. Por lo cual, no se trata solamente de encontrar la arquitectura que para una determinada cantidad de bits logre el mejor compromiso de área, potencia y velocidad. Sino que también esta relacion se mantenga óptima para diferentes tamaños del sumador.

Por estas razones, precisamos diseñar un sumador de N-dígitos que sea lo más rápido posible, manteniendo una relación de compromiso óptima entre la velocidad, consumo de energía y área del circuito. Estas características las resumimos en el cuadro 2.1.

Parámetro	**Especificación**
Sumandos	Dos[1]
Cantidad de Dígitos	Parametrizable
Proceso de fabricación	Disponible por medio de MOSIS[2]
Retardo de propagación	Lo mas bajo posible
Potencia total disipada	Tan bajo como sea posible
Área del Circuito	La menor posible

Cuadro 2.1: Especificaciones de diseño para el sumador binario

2.2. Métricas de calidad

Definiremos las métricas que nos permitan dar cuenta de la calidad del diseño.

2.2.1. Performance

El término performance puede representar distintas métricas, según desde qué perpectiva se esté realizando el análisis. Pero si nos enfocamos puramente en el diseño, la performance se define usualmente(10) como la duración del período del clock (o su frecuencia). El valor mínimo

de período de clock que pueda ser usado para una tecnología y un diseño dado, está definido por múltiples factores, como el tiempo que le toma a las señales propagarse a través de la lógica (retardo de propagación), el tiempo que lleva entrar y salir los datos de los registros, la incertidumbre de llegada del reloj (*clock uncertainty*). Pero el núcleo de todo análisis de performance reside en la performance de una sola compuerta.

Retardo de propagación

El retardo de propagación t_p de una compuerta define cuán rápido responde un circuito a un cambio en su(s) entrada(s). Expresa el retardo experimentado por una señal cuando pasa a través de una compuerta. Medido entre el 50 % del punto de transición de entrada y salida, como mostramos en la figura 2.1, correspondiente al tiempo de propagación de una compuerta inversora. Ya que el tiempo de propagación es distinto según el flanco de entrada, se definen 2 tiempos de propagación. El t_{pLH} es el tiempo de respuesta de una compuerta para una transición de la salida desde bajo a alto, mientras que t_{pHL} se refiere a el tiempo para una transición de la salida desde alto a bajo. El retardo de propagación t_p se define como el promedio de estos dos.

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

Camino Crítico

En un circuito digital con varias entradas y salidas, pueden existir mas de un camino desde la entrada hasta la salida. Se suele denominar camíno crítico a aquel camino que tenga el mayor retardo de propagación, ya sea por cantidad de lógica que atraviesa o por las capacidades parásitas de las conexiones. El retardo de propagación de este circuito será el retardo de propagación del camino crítico.

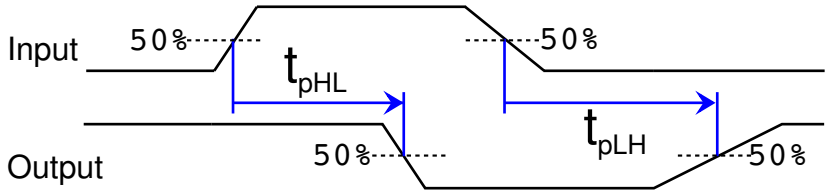

Figura 2.1: Retardo de propagación de un inversor

Mínimo retardo de propagación

Para poder comparar la performance de distintas tecnologías, se busca un circuito que no incluya parámetros como el fan-in o fan-out, que influyen en los tiempos t_f, t_r y t_f. Por ello, el circuito que es un estandard de facto para medir el tiempo de propagación, es el oscilador anillo (*ring oscillator*), que es un número impar de inversores conectados en serie, con la salida conectada a la entrada. Este circuito oscilla espontaneamente, a una frecuencia de $T = 2 \times t_p \times N$, con N el número de inversores en la cadena.

Contar con esta métrica nos permitirá tener una referencia del límite inferior impuesto por la tecnología que se esté utilizando. Por ejemplo, tomemos la tecnología TSMC de 180 nm: La frecuencia de un oscilador anillo de 31 etapas es de 377,13 MHz. Es decir que el tiempo de propagación de una celda inversora en esta tecnología es $t_p = 47,8\ ps$

2.2.2. Potencia promedio disipada

Realizamos el análisis de potencia a lo largo de un período de tiempo T. 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 2.1.

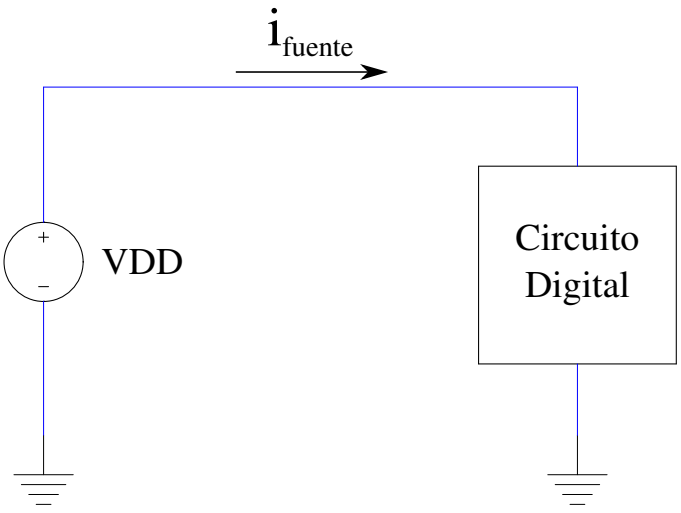

Figura 2.2: Estimación de Potencia Promedio Disipada

$$P_{av} = \frac{1}{\mathrm{T}} \int_0^T p(t)dt = \frac{\mathrm{V_{DD}}}{\mathrm{T}} \int_0^T i_{\mathrm{fuente}}(t)\mathrm{d}t \tag{2.1}$$

El período de tiempo que tomaremos para la integral es el retardo de propagación del camino crítico.

2.2.3. Área

La importancia de minimizar el área de los circuitos radica principalmente en que esta impacta fuertemente en el costo de cada *die*, ya que el costo es una función que depende de la cuarta potencia del área del circuito(10). Adeás, los circuitos de menor área tienden a consumir menor energía.

Capítulo 3

VERIFICACIÓN FORMAL

Como aclaramos en el capítulo **??**, el correcto funcionamiento del circuito se garantiza por medio de la verificación formal de las propiedades de la suma.

Nuestro flujo para esta etapa tiene que ver con la verificación de propiedades que se denominan *safety properties*. Estas son propiedades que se mantienen como verdaderas siempre (o lo que es equivalente, nunca son falsas). En Lava escribimos estas propiedades de la misma forma en que escribimos los circuitos, inclusive utilizando otros circuitos que nos sirvan para expresar una condición. Esto se verá con mas claridad cuando avancemos con la verificación. Entonces, la pregunta que estamos haciendo para verificar cualquier propiedad descrita de esta forma es: ¿Este circuito de verificación siempre tiene como salida el estado `True` sin importar cuales son las entradas? Para responder esta pregunta, en Lava usamos la operación `verify`.

Este proceso funciona asi: Tal como podemos generar un netlist VHDL (o la simulación) a partir de la descripción del circuito, también podemos generar una fórmula lógica que representa al circuito. Esta fórmula lógica se la damos a un probador de teorema externo que nos probará (o desaprobará) la validez de la fórmula. El probador externo que usaremos es miniSAT[1].

3.1. Modelo de referencia

A los fines de la verificación, usaremos un sumador de referencia `adder` bien simple, en el cual podamos probar todas las propiedades de la suma, para luego hacer un chequeo de equivalencia lógica (LEC por sus siglas en inglés) entre el sumador de referencia y el sumador que queremos implementar. Esto es conveniente porque es una gran ventaja (desde el punto de vista de tiempo de cálculo) hacer todas las pruebas sobre circuitos mas simples (pequeños), para luego realizar una sola comprobación de equivalencia lógica entre este circuito simple y el circuito diseñado, garantizando así que si todas las propiedades se cumplen en uno, también se cumplen en el otro.

[1] Minisat es un programa que resuelve problemas conocidos como *Boolean satisfiability problem (SAT)*, o directamente *SAT solver*

3.2. Verificación de las Propiedades

3.2.1. Propiedades de la suma

La suma tiene las siguientes propiedades:

- Asociativa
- Conmutativa
- Existencia del elemento neutro cero.

3.2.2. Descripción de las propiedades en el RCA

Debido a que nuestro sumador de Brent-Kung asume que el acarreo de entrada es cero, debemos modificar nuestro sumador de referencia (un Ripple Carry Adder) para que desprecie el acarreo de entrada. Por lo tanto describimos nuevamente una versión del RCA de la siguiente forma:

```
adder2 ([],[]) = []
adder2 (a:as, b:bs) = sum:sums
  where
    (sum, carry)     = halfAdd (a, b)
    (sums, carryOut) = adder (carry, (as, bs))
```

Propiedad Conmutativa

Ahora declaramos la propiedad conmutativa de la suma de la siguiente forma:

```
prop_AdderCommutative (as, bs) = ok
   where
      out1 = adder2 (as, bs)
      out2 = adder2 (bs, as)
      ok   = out1 <==> out2
```

Notar que el operador `<==>` es la versión infija de una función que mapea dos listas a la compuerta `xnor2`, la cual es la operación de equivalencia lógica. Como es muy difícil verificar automáticamente para cualquier tamaño, definimos una nueva propiedad que incluye el tamaño del circuito a ser verificado:

```
prop_AdderCommutative_ForSize n =
   forAll (list n) $ \as ->
     forAll (list n) $ \bs ->
       prop_AdderCommutative (as, bs)
```

Luego hacemos la verificación corriendo Minisat desde Lava, dando el tamaño del sumador:

```
minisat (prop_AdderCommutative_ForSize 32)
```

Si nuestro circuito está correctamente diseñado, tenemos:

```
Minisat: ... (t=0.00system) Valid.
```

De otro modo, podemos tener uno de estos resultados:

```
Minisat: ... (t=0.00system) Falsifiable.
Minisat: ... (t=0.00system) Inderterminate.
```

Propiedad Asociativa

La propiedad Asociativa la declaramos como:

```
prop_AdderAssociative (as, bs, cs) = ok
   where
      out1 = adder2 (adder2 (as, bs), cs)
      out2 = adder2 (as, adder2 (bs, cs))
      ok   = out1 <==> out2
```

Existencia del Elemento Neutro

Para verificar que el cero es el elemento neutro de la adición, necesitamos escribir un poco mas de lógica al circuito para transformar uno de los operandos a cero:

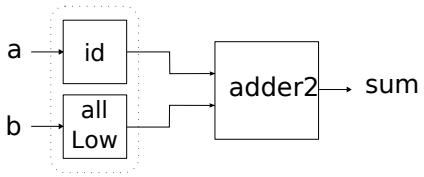

Figura 3.1: circuito addZero

```
alwaysLow :: [Signal Bool] -> [Signal Bool]
alwaysLow (as) = [low | n <- [1..n]]
   where
      n = length as

addZero = (id -|- alwaysLow) ->- adder2
-- id es la funcion identidad
```

Y la verificación de esta propiedad en el circuito:

```
prop_AdderZero (as,bs) = ok
   where
      out = addZero (as, bs)
      ok   = out <==> as
```

Equivalencia lógica entre el RCA y el sumador de Brent-Kung

Finalmente, hacemos la equivalencia lógica entre los dos circuitos: `fastAdd` (el BKA) y `adder2` (el RCA). Eso se declara en Lava de la siguiente forma:

```
prop_Equivalent adder2 fastadd a = ok
   where
      out1 = adder2   a
      out2 = fastadd a
      ok   = out1 <==> out2
```

Parte II

Diseño Físico

Capítulo 4
Flujo de Diseño Físico

4.1. Introducción

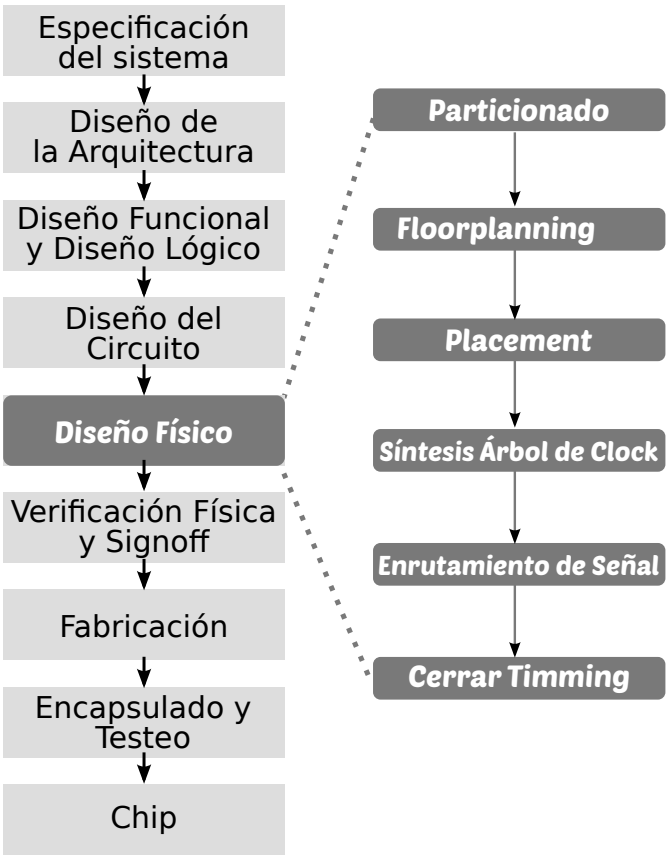

Figura 4.1: Flujo de diseño Físico

4.2. Relevamiento, comparación y selección de las herramientas disponibles

Criterios para la selección

- Desarrollo activo y que exista una comunidad de usuarios que brinden soporte
- Mayor cantidad de herramientas integradas
- Flexibilidad para importar y exportar datos.
- Design kits disponibles para la tecnología seleccionada.
- Diseño físico a partir de la descripción estructural en VHDL[1].

La tecnología que se usará para fabricar es TSMC 180 nm[2]. Luego de una inspección de esos criterios, las herramientas candidatas que cumplen con esos criterios definidos son:

Alliance VLSI CAD System[3] Alliance es un conjunto de herramientas libres, y librerías portables para el diseño de VLSI. Incluye un compilador vhdl y un simulador, herramientas de síntesis de lógica, y herramientas de *place & route* automáticas. Brinda un conjunto completo de librerías CMOS portables.

Electric VLSI Design System Es un sistema de Automatización de Diseño Electrónico. En un entorno integrado muy flexible que permite la descripción del circuito de varias formas (esquemáticos, VHDL, Layout, generadores de ROM y MOSIS PLA)

4.3. Selección de las Celdas estándar

El resultado de la síntesis que realizamos en el capítulo **??** es un netlist VHDL que contiene sólo compuertas lógicas. Estas son compuertas lógicas abstractas, es decir que nuestro circuito fué mapeado a un conjunto finito de funciones logicas como las `and`, `or`, `xor`, `xnor`, etc.

Ahora es necesario mapear estas funciones lógicas a compuertas lógicas reales, que serán tambien un conjunto finito de compuertas, pero con dimensiones físicas definidas, y con una caracterización de su funcionamiento real. Estas compuertas lógicas se denominan celdas estándards, que se diseñan específicamente para la tecnología de fabricación que hayamos definido usar. Por cada función lógica existen distintas versiones para poder manejar distintas capacidades.

Caraterización de las celdas: Se realizan simulaciones analógicas de un netlist de estas celdas, donde variando paramétricamente la carga (capacitiva) de salida y el tiempo de crecida y caída (t_r y t_f) de la entrada, se obtiene el retardo de propagación, tiempo de crecida y de bajada de la salida y la disipación de potencia.

Es común elegir las celdas estándars según el tipo de aplicación a desarrollar. Existen celdas estandars que fueron diseñadas para bajo consumo, o alta velocidad, o de mínima área. También existe la posibilidad de diseñar celdas que busquen la mejor relación velocidad-consumo-area que puedan ser utilizadas en muchas aplicaciones. En circuitos integrados para sistemas alimentados a batería se intentará utilizar las celdas de menor consumo y evitar siempre que sea posible las de mayor velocidad, en función del presupuesto de potencia disponible para el mismo.

En nuestro caso, seleccionamos una librería de celdas estándars diseñadas para tamaño mínimo de canal n, y p. Otra característica de la librería es la distancia entre el riel de VSS hasta el

[1] Condición necesaria ya que el resultado de nuestro diseño en el capítulo **??** es una descripción estructural en VHDL.

[2] http://www.mosis.com/vendors/view/tsmc/018

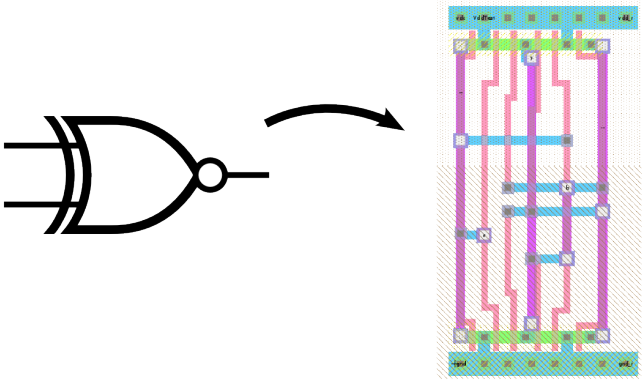

Figura 4.2: Mapeo de funciones lógicas a celdas estándars

riel de VDD. Esta distancia es de 117 λ (En nuestra tecnología de 180 nm, el valor de lambda es: $\lambda = 100\ \mathrm{nm}$), lo cual permite el ruteo horizontal de 16 pistas de metal por encima de las celdas, con metal 3 hasta capas superiories.

4.4. Ubicación y Cableado (*Place & Route*)

Partimos desde la descripción estructural[1] del vhdl que producimos en el capítulo **??**. De *Electric* usaremos la herramienta llamada *Silicon Compiler*, que se encarga de ubicar y conectar las celdas según el archivo vhdl. En el apéndice B mostramos el resultado de sintetizar a compuertas nuestro circuito. Las compuertas que se usan son compuertas lógicas genéricas, aún podemos modificar este netlist primero para hacer optimizaciones lógicas, que en nuestro caso esto no es conveniente ya que hemos elegido la arquitectura para lograr la mejor relación de compromiso entre velocidad y menor interconectividad. Otra optimización a realizar es eligiendo la fuerza de la celda.

place & route

Arquero para que este archivo pueda ser usado con Electric fué necesario hacer algunas modificaciones:

1. Reemplazar los nombres de las instancias de las celdas estandards que seleccionamos en la sección 4.3 y *Electric* utilizará.

2. En la parte del vhdl que comienza la descripción estructural es necesario agregar las celdas estandards que serán utilizadas en el circuito.

3. Quitar todas las instancias de `wire port map (...)` volviendo a conectar lo que queda desconectado al quitar estas instancias.

Estas tareas las realizamos modificando el código del programa de lava que crea el netlist vhdl llamado `VhdlNew.hs`, y con un script[2] en perl que es lanzado desde el mismo.

[1] El resultado de la síntesis hecha con lava es un netlist a nivel de compuerta, listo para ser usado por una herramienta de *place & route*Gate-level netlist vhdl.

[2] Adjuntamos el script en el apéndice **??**

4.4.1. Métricas de Calidad de un circuito digital

Para cuantificar la calidad de un circuito, podemos analizar un circuito desde diferentes perspectivas: Costo, funcionalidad, robustez, performance y consumo de energía. Tomaremos estas dos últimas métricas, mas la del área (propiedad que define el costo) del circuito, para realizar la comparación entre las distintas arquitecturas analizadas.

4.4.2. Performance

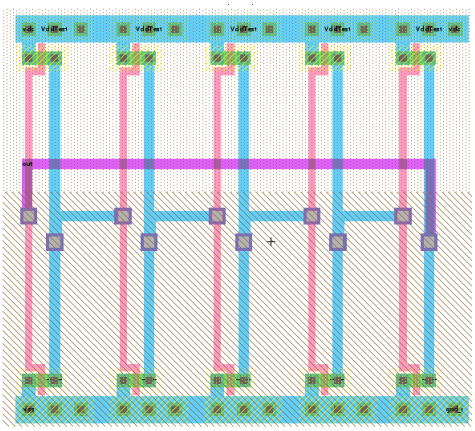

Figura 4.3: Layout del Oscilador anillo (N=5)

Realizamos un oscilador anillo de 5 etapas, utilizando el inversor mas chico de nuestra librería de celdas estándard elegida. En la figura 4.3 podemos ver el *layout* que realizamos para esta prueba. Hacemos la extracción de parásitos y simulamos el circuito para obtener la forma de onda de la figura 4.4.

Como podemos deducir de la figura 4.4, el tiempo de propagación es de 32 pS, lo cual no significa que nuestros circuitos puedan correr a 31,25 GHz. Cuando medimos el tiempo de propagación usando este circuito, deberemos considerar que este es sólo una forma de comparar las tecnologías de fabricación y las topologías de las compuertas, pero de ninguna forma será la frecuencia máxima a la que pueda funcionar nuestro circuito, ya que la complejidad y cantidad de compuertas es enormemente mayor que la de un inversor. Según Rabaey(10) se puede esperar velocidades entre 50 a 100 veces mas bajas, aunque optimizando el diseño se puede lograr acercarnos un poco mas a esta frecuencia ideal.

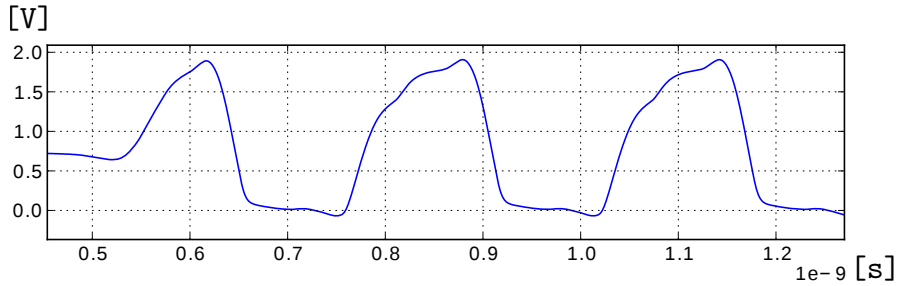

Figura 4.4: Simulación con parásitos extraidos del layout de la figura 4.3

4.4.3. Tiempo mínimo de propagación

$t_p = 32p$

Capítulo 5
Sign Out y Tape Out

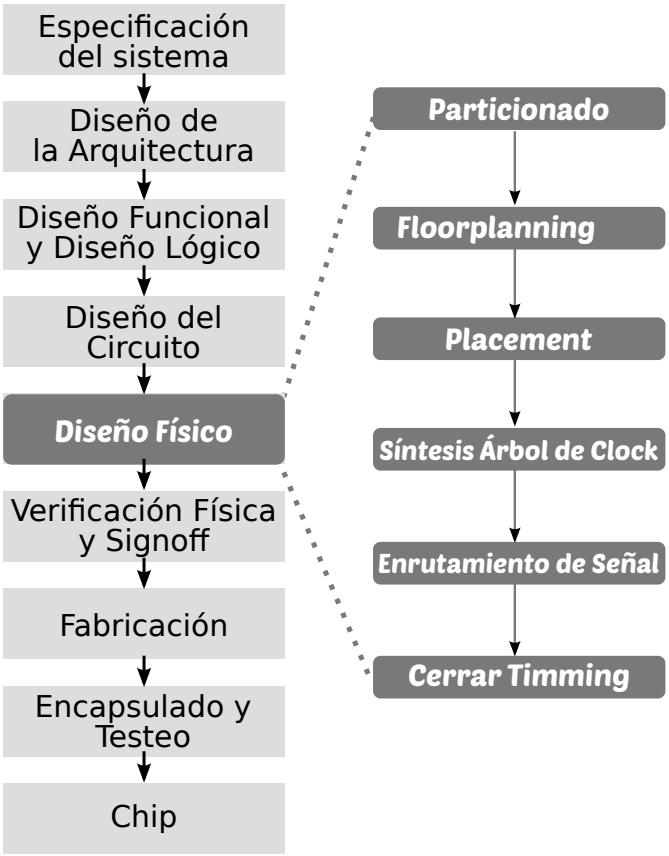

Figura 5.1: Flujo de diseño Físico

...

Parte III

Comparación de resultados

Capítulo 6
Comparación de resultados

6.1. Tablas comparativas

Parte IV

Conclusiones

Capítulo 7
Conclusiones

...

Apéndice A
NETLIST VHDL

```
library ieee;

use ieee.std_logic_1164.all;

entity
  BrentKungFastAdder
is
port
  (


    a_0 : in std_logic
  ; a_1 : in std_logic
  ; a_2 : in std_logic
  ; a_3 : in std_logic
  ; a_4 : in std_logic
  ; a_5 : in std_logic
  ; a_6 : in std_logic
  ; a_7 : in std_logic
  ; b_0 : in std_logic
  ; b_1 : in std_logic
  ; b_2 : in std_logic
  ; b_3 : in std_logic
  ; b_4 : in std_logic
  ; b_5 : in std_logic
  ; b_6 : in std_logic
  ; b_7 : in std_logic


  ; sum_0 : out std_logic
  ; sum_1 : out std_logic
  ; sum_2 : out std_logic
  ; sum_3 : out std_logic
  ; sum_4 : out std_logic
  ; sum_5 : out std_logic
  ; sum_6 : out std_logic
  ; sum_7 : out std_logic
  ; cout : out std_logic
  );
```

```
end BrentKungFastAdder;

architecture
  structural
of
  BrentKungFastAdder
is
  signal w1 : std_logic;
  signal w2 : std_logic;
  signal w3 : std_logic;
  signal w4 : std_logic;
  signal w5 : std_logic;
  signal w6 : std_logic;
  signal w7 : std_logic;
  signal w8 : std_logic;
  signal w9 : std_logic;
  signal w10 : std_logic;
  signal w11 : std_logic;
  signal w12 : std_logic;
  signal w13 : std_logic;
  signal w14 : std_logic;
  signal w15 : std_logic;
  signal w16 : std_logic;
  signal w17 : std_logic;
  signal w18 : std_logic;
  signal w19 : std_logic;
  signal w20 : std_logic;
  signal w21 : std_logic;
  signal w22 : std_logic;
  signal w23 : std_logic;
  signal w24 : std_logic;
  signal w25 : std_logic;
  signal w26 : std_logic;
  signal w27 : std_logic;
  signal w28 : std_logic;
  signal w29 : std_logic;
  signal w30 : std_logic;
  signal w31 : std_logic;
  signal w32 : std_logic;
  signal w33 : std_logic;
  signal w34 : std_logic;
  signal w35 : std_logic;
  signal w36 : std_logic;
  signal w37 : std_logic;
  signal w38 : std_logic;
  signal w39 : std_logic;
  signal w40 : std_logic;
  signal w41 : std_logic;
  signal w42 : std_logic;
  signal w43 : std_logic;
  signal w44 : std_logic;
  signal w45 : std_logic;
  signal w46 : std_logic;
  signal w47 : std_logic;
  signal w48 : std_logic;
  signal w49 : std_logic;
```

```
  signal w50 : std_logic;
  signal w51 : std_logic;
  signal w52 : std_logic;
  signal w53 : std_logic;
  signal w54 : std_logic;
  signal w55 : std_logic;
  signal w56 : std_logic;
  signal w57 : std_logic;
  signal w58 : std_logic;
  signal w59 : std_logic;
  signal w60 : std_logic;
  signal w61 : std_logic;
  signal w62 : std_logic;
  signal w63 : std_logic;
  signal w64 : std_logic;
  signal w65 : std_logic;
begin
  c_w2      :  wire  port map (a_0, w2);
  c_w3      :  wire  port map (b_0, w3);
  c_w1      :  xor2  port map (w2, w3, w1);
  c_w6      :  wire  port map (a_1, w6);
  c_w7      :  wire  port map (b_1, w7);
  c_w5      :  xor2  port map (w6, w7, w5);
  c_w8      :  and2  port map (w2, w3, w8);
  c_w4      :  xor2  port map (w5, w8, w4);
  c_w11     :  wire  port map (a_2, w11);
  c_w12     :  wire  port map (b_2, w12);
  c_w10     :  xor2  port map (w11, w12, w10);
  c_w14     :  and2  port map (w6, w7, w14);
  c_w15     :  and2  port map (w5, w8, w15);
  c_w13     :  or2   port map (w14, w15, w13);
  c_w9      :  xor2  port map (w10, w13, w9);
  c_w18     :  wire  port map (a_3, w18);
  c_w19     :  wire  port map (b_3, w19);
  c_w17     :  xor2  port map (w18, w19, w17);
  c_w21     :  and2  port map (w11, w12, w21);
  c_w22     :  and2  port map (w10, w13, w22);
  c_w20     :  or2   port map (w21, w22, w20);
  c_w16     :  xor2  port map (w17, w20, w16);
  c_w25     :  wire  port map (a_4, w25);
  c_w26     :  wire  port map (b_4, w26);
  c_w24     :  xor2  port map (w25, w26, w24);
  c_w29     :  and2  port map (w18, w19, w29);
  c_w30     :  and2  port map (w17, w21, w30);
  c_w28     :  or2   port map (w29, w30, w28);
  c_w32     :  and2  port map (w17, w10, w32);
  c_w31     :  and2  port map (w32, w13, w31);
  c_w27     :  or2   port map (w28, w31, w27);
  c_w23     :  xor2  port map (w24, w27, w23);
  c_w35     :  wire  port map (a_5, w35);
  c_w36     :  wire  port map (b_5, w36);
  c_w34     :  xor2  port map (w35, w36, w34);
  c_w38     :  and2  port map (w25, w26, w38);
  c_w39     :  and2  port map (w24, w27, w39);
  c_w37     :  or2   port map (w38, w39, w37);
  c_w33     :  xor2  port map (w34, w37, w33);
```

```
  c_w42     :  wire  port map (a_6, w42);
  c_w43     :  wire  port map (b_6, w43);
  c_w41     :  xor2  port map (w42, w43, w41);
  c_w46     :  and2  port map (w35, w36, w46);
  c_w47     :  and2  port map (w34, w38, w47);
  c_w45     :  or2   port map (w46, w47, w45);
  c_w49     :  and2  port map (w34, w24, w49);
  c_w48     :  and2  port map (w49, w27, w48);
  c_w44     :  or2   port map (w45, w48, w44);
  c_w40     :  xor2  port map (w41, w44, w40);
  c_w52     :  wire  port map (a_7, w52);
  c_w53     :  wire  port map (b_7, w53);
  c_w51     :  xor2  port map (w52, w53, w51);
  c_w55     :  and2  port map (w42, w43, w55);
  c_w56     :  and2  port map (w41, w44, w56);
  c_w54     :  or2   port map (w55, w56, w54);
  c_w50     :  xor2  port map (w51, w54, w50);
  c_w60     :  and2  port map (w52, w53, w60);
  c_w61     :  and2  port map (w51, w55, w61);
  c_w59     :  or2   port map (w60, w61, w59);
  c_w63     :  and2  port map (w51, w41, w63);
  c_w62     :  and2  port map (w63, w45, w62);
  c_w58     :  or2   port map (w59, w62, w58);
  c_w65     :  and2  port map (w63, w49, w65);
  c_w64     :  and2  port map (w65, w27, w64);
  c_w57     :  or2   port map (w58, w64, w57);


  c_sum_0   :  wire  port map (w1, sum_0);
  c_sum_1   :  wire  port map (w4, sum_1);
  c_sum_2   :  wire  port map (w9, sum_2);
  c_sum_3   :  wire  port map (w16, sum_3);
  c_sum_4   :  wire  port map (w23, sum_4);
  c_sum_5   :  wire  port map (w33, sum_5);
  c_sum_6   :  wire  port map (w40, sum_6);
  c_sum_7   :  wire  port map (w50, sum_7);
  c_cout    :  wire  port map (w57, cout);
end structural;
```

Apéndice B

SCRIPT PERL

```
#!/usr/bin/perl

#### Import Classes
use File::Copy;
use FileHandle;
####
#### Define constants
my $idPort = "id";

# Input File
my $file =$ARGV[0];

#
my %ports; # to store ports (ins & outs)
# and wire's name given
# by the VhdlNew.hs

### OPEN INPUT FILE
print "$file\n";
open(my $fhi, '<',$file) or die "archivo no encontrado";
open(my $fho, '+>',"temp-$file");
while(<$fhi>){
#Primero obtengo las entradas
if(m/.+$idPort.+port.map.+\(([A-Za-z0-9_]+).+(w[0-9]+)/)
{
push @wires,$2;
push @wires,$1;
$ports {$2} = $1;
# print $fho "--$_"; # comento la linea
}

#Ahora obtengo las salidas, por ejemplo:
# c_sum_0   :  std_wire  port map (w1, sum_0);
# o como estas:
# c_cout    :  std_wire  port map (w131, cout);
if(m/.+$idPort.+(w[0-9]+)\,.?([A-Za-z0-9_]+)\).*/)
```

```
{
$ports {$1} = $2;
print $fho "--$_"; # comento lo que quiero eliminar
} else {
print $fho "$_";}
# and the outputs
#if(m/([a-z]+_[a-z]*_*[0-9]+).+out/) {push @outs,$1;}
# I could use ins and outs to make the %ports hash table
}#while
close($fhi);
close($fho);


#copy("temp-$file", "temp2-$file");
# Replace all signals connected to wire with the inputs
#   c_w131    :  std_or2   port map (w132, w141, w131);
# w131 should be replaced by the output


while(my($key,$value) = each(%ports)) {
open(my $fhi, '<',"temp-$file") or die "archivo no encontrado";
open(my $fho, '+>', "stripped-$file");
while(<$fhi>){
if(s/(.+map.+)$key(\,|\))/$1$value$2/g) {

#need to delete entries with the next pattern:  c_w18     :  std_wire  port map ...
if(m/.+$idPort.+/) {print $fho "-- deleted $_";}
else {print $fho "$_";}}
else { print $fho "$_";}
      }# while file

close($fho);
close($fhi); #atenti no hacer close($fho, $fhi) porque no es lo mismo que en 2 reng
copy("stripped-$file","temp-$file");
}#while hash table
```

Bibliografía

[1] MANUEL VALENCIA ADRIÁN ESTRADA, CARLOS J. JIMÉNEZ. Características de Sumadores en Tecnologías Fuertemente Submicrónicas. *IBERCHIP*, **DOC (TEC 2004-01509/MIC)**, 1982.

[2] A. BALIGA AND D. YAGAIN. Design of high speed adders using cmos and transmission gates in submicron technology: A comparative study. In *Emerging Trends in Engineering and Technology (ICETET), 2011 4th International Conference on*, pages 284–289, 2011.

[3] R. P. BRENT AND H. T. KUNG. . *IEEE Transaction on Computers*, **C-31, Issue: 3**:260–264, 2006.

[4] K. CLAESSEN AND M. SHEERAN. A tutorial on Lava: A hardware descritption and verification system. Website, 2014. http://projects.haskell.org/chalmers-lava2000/Doc/.

[5] S. KNOWLES. A family of adders. In *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, pages 277–281, 2001.

[6] PETER M. KOGGE AND HAROLD S. STONE. A parallel algorithm for the efficient solution of a general class of recurrence equations. *Computers, IEEE Transactions on*, **C-22**(8):786–793, Aug 1973.

[7] RICHAR E. LADNER AND MICHAEL J. FISCHER. Parallel prefix computation. *JACM, Journal of the ACM*, **C-27**(4):831,838, Oct 1980.

[8] L. MARSO. Brent-kung fast adder dscription, simulation and formal verification using lava. In *Micro-Nanoelectronics, Technology and Applications, 2008. EAMTA 2008. Argentine School of*, pages 111–114, Sept 2008.

[9] B. PARHAMI. *Computer Arithmetic: Algorithms and Hardware Designs*. Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press, 2000.

[10] J.M. RABAEY, A.P. CHANDRAKASAN, AND B. NIKOLIC. *Digital integrated circuits: a design perspective*. Prentice Hall electronics and VLSI series. Pearson Education, 2ed edition, 2003. 3, 5, 16

[11] M. SHEERAN. Parallel prefix network generation: an application of functional programming In Hardware Design and Functional Languages. In *Hardware design and Functional Languages (HFL07), Braga, Portugal*, March 2007.