Síntesis de un sumador paralelo de 5 bits empleando las herramientas de *Alliance*

Luis Arturo Reyes Llanos
Síntesis de circuitos digitales
Dr. Marco Antonio Gurrola Navarro
Centro Universitario de Ciencias Exactas e Ingeniería
Universidad de Guadalajara

***Resumen*— El siguiente reporte describe el desarrollo de un circuito sumador binario de dos números de 5 bits cada uno, desde su descripción en *VHDL* hasta la síntesis y preparación para la etapa pre-silicio.**

Palabras clave— *VHDL*, sumador, síntesis, *Alliance*, patrón de pruebas.

I. Introducción

ESTE documento tiene la finalidad describir el desarrollo de un sumador binario de dos números, *A* y *B*, cada uno de 5 bits, ofreciendo un resultado *C* de 5 bits, desde el diseño planteado a nivel *VHDL*, junto con las respectivas pruebas para la evaluación del funcionamiento del mismo a nivel de plano de circuito integrado.

II. Descripción del módulo

Como se mencionó en el apartado anterior, este circuito es un sumador binario de dos números de 5 bits cada uno, con un resultado de 5 bits. En lo sucesivo, a estos números se les llamará *A* y *B*, mientras que el resultado se representará con *C*, como se muestra en la Fig. 1.

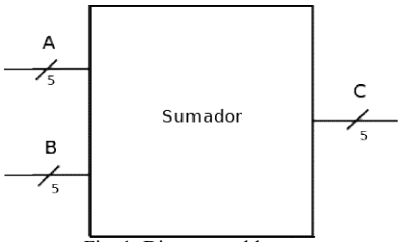

Fig. 1. Diagrama a bloques.

Ese módulo se encarga de sumar *A* y *B* de manera simultánea, mostrando el resultado en la salida *C*. Es importante mencionar ciertas limitaciones que existen en nuestro módulo, como la pérdida del *carry* o acareo de la suma.

Por este motivo, la longitud de *C* es de 5 bits. De la misma manera, este sumador carece de *carry* de entrada, por lo que para posibles aplicaciones de resta, tendría que incorporarse cierta modificación al módulo.

III. Descripción en *VHDL*

Una vez determinada la función del modulo es necesario plantear un código en *VHDL*, un lenguaje utilizado para la descripción de *hardware*.

Describir el módulo en *VHDL* nos permite plantear cómo se relacionarán las entradas para obtener cierto resultado a la salida.

Es importante mencionar que este es uno de los archivos iniciales que son requeridos por las herramientas de *Alliance*.

A. Entidad

Es necesario plantear una entidad, lo que equivale a describir las entradas y salidas con las que cuenta el sistema, además del ancho del bus de datos de cada una de ellas (Fig. 2).

B. Arquitectura

Después de haber descrito las entradas y salidas del módulo, es importante describir una arquitectura; es decir, cómo se relacionan las entradas y salidas previamente descritas para obtener el resultado esperado (Fig. 2).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;


entity Adder5 is

    port ( A   : in Std_Logic_Vector(4 downto 0) ;
           B   : in Std_Logic_Vector(4 downto 0) ;
           RESULT : out Std_Logic_Vector(4 downto 0) );

end Adder5;


architecture DataFlow OF Adder5 is
begin

   RESULT <= std_logic_vector( unsigned(A) + unsigned(B) );

end DataFlow;
```

Fig. 2. Descripción en *VHDL*.

IV. Diagrama de disposición de conectores

Una vez descrito el módulo en *VHDL*, lo siguiente es determinar la disposición de los conectores en el *core* o núcleo del circuito integrado.

En este punto, es importante tener ciertas consideraciones presentes, tales como la disposición de los demás elementos del PCB donde el *chip* será colocado (si es que ya se cuenta con esta información).

Esto nos ayudará a saber qué tan conveniente, por cuestiones de ubicación y simetría, es colocar los puertos de entrada y salida en alguna posición en particular de nuestro circuito integrado.

Debido a la naturaleza de este ejercicio, puramente didáctico, el *chip* no será montado en ningún circuito PCB, por lo que la disposición de los conectores se determina a gusto del diseñador (Fig. 3).

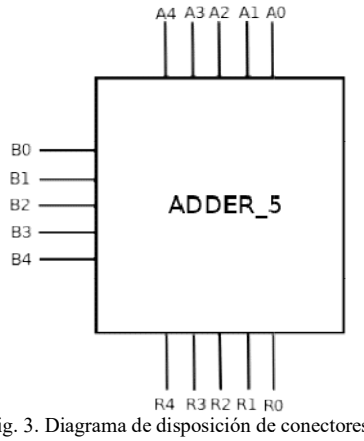

Fig. 3. Diagrama de disposición de conectores.

Conociendo la disposición de los conectores alrededor del *core*, procedemos a crear el archivo que dice a las herramientas de síntesis dónde se colocarán los puertos para efectos del plano final del circuito integrado, como se puede observar en Fig. 4.

```
TOP ( # IOs are ordered from left to right
    (IOPIN a(4).0 );
    (IOPIN a(3).0 );
    (IOPIN a(2).0 );
    (IOPIN a(1).0 );
    (IOPIN a(0).0 );
)
BOTTOM ( # IOs are ordered from left to right
    (IOPIN result(4).0 );
    (IOPIN result(3).0 );
    (IOPIN result(2).0 );
    (IOPIN result(1).0 );
    (IOPIN result(0).0 );
)
LEFT ( # IOs are ordered from bottom to top
    (IOPIN b(4).0 );
    (IOPIN b(3).0 );
    (IOPIN b(2).0 );
    (IOPIN b(1).0 );
    (IOPIN b(0).0 );
)
IGNORE ( # IOs are ignored(not placed) by IO Placer
)
```

Fig. 4. Archivo de disposición de conectores (.ioc).

V. Plan de pruebas

Junto con las herramientas de diseño y síntesis se utilizan un par de herramientas de pruebas y simulación, que nos ayudan a evaluar la funcionalidad del módulo.

Estas herramientas requieren la creación de ciertos planes de prueba; combinaciones de entradas proponiendo un resultado en la salida. Es decir, si es la suma de dos números *A* y *B*, con un resultado *C*, es importante plantear combinaciones que nos ayuden a comprobar el funcionamiento del módulo en puntos críticos. Con base en ello, el plan de pruebas propuesto es:

- Sumar 0 en *A* y 0 en *B* (00_H+00_H=00_H).
- Sumar el mayor número posible de representar con 5 bits ($1F_H$), tanto en *A* como en *B* y observar lo que sucede ($1F_H$+$1F_H$=$3E_H$).
- Sumar cierto número distinto de 0 en *A* con un 0 en *B*, lo que nos daría como resultado *A*, o viceversa. (05_H+00_H=05_H).
- Sumar una unidad en *A* con cierto número en *B*, obteniendo como resultado *B*+1, o viceversa (01_H+$0A_H$=$0B_H$).

A. Archivo de patrones

A continuación se captura en un archivo las pruebas del plan de pruebas descrito en la sección anterior, el cual se puede apreciar en Fig. 5. Nótese que si el dato de algún puerto se escribirá en notación binaria o hexadecimal, esto será indicado con una "B" o con una "X" respectivamente al declarar el puerto.

```
in       a (4 downto 0) X;;;
in       b (4 downto 0) X;;;
out      result (4 downto 0) X;;;
in       vss B;;
in       vdd B;;

begin

-- Pattern description :
--          A  B   Res     V V

<   0ns>: 00  00  ?**  0 1;
< +10ns>: 00  00  ?**  0 1;
< +10ns>: 00  00  ?**  0 1;
< +10ns>: 1f  1f  ?**  0 1;
< +10ns>: 1f  1f  ?**  0 1;
< +10ns>: 05  00  ?**  0 1;
< +10ns>: 05  00  ?**  0 1;
< +10ns>: 01  0a  ?**  0 1;
< +10ns>: 01  0a  ?**  0 1;
```

Fig. 5. Archivo de patrones.

VI. Proceso de síntesis

Una vez creados los archivos con los que el proceso de síntesis se ejecutará con la ayuda de las herramientas de *Alliance*, se procede al uso de las mismas, partiendo de un *script* o un archivo ejecutable que contiene en él el conjunto de instrucciones del proceso de síntesis (Fig. 6).

```
#! /bin/bash
export VH_MAXERR=1 &&
vasy -apo -I vhdl adder5 adder5 &&
asimut -b adder5 adder5 salida_vbe &&
boom -VA adder5 adder5_opt &&
boog adder5_opt adder5 &&
loon adder5 adder5_opt &&
asimut -b adder5 adder5 salida_vst &&
ocp -v -ioc adder5 adder5_opt adder5_p &&
nero -v -2 -p adder5_p adder5_opt adder5_f &&
export MBK_OUT_LO=al &&
cougar -v adder5_f adder5 &&
export MBK_OUT_LO=vst &&
lvx vst al adder5_opt adder5 -f &&
graal -l adder5_f
```

Fig. 6. Archivo script ejecutable de herramientas de síntesis.

Es importante destacar que la funcionalidad del módulo se verificará a través de los archivos de salida obtenidos con la herramienta *asimut*, los cuales no deberán arrojar ningún error o advertencia.

Al finalizar el proceso de síntesis, definido por los comandos del script, nos encontramos con el plano final generado a partir de las herramientas, mostrado en Fig. 7.

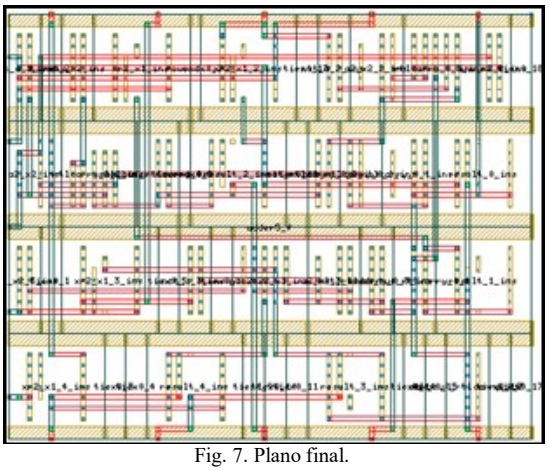

Fig. 7. Plano final.

De igual manera cabe destacar que la herramienta *lvx*, después de un paso previo realizado con *cougar*, nos demuestra que la síntesis tuvo éxito, pues este analiza los estructurales creados a nivel *layout* y a nivel esquemático, mostrando el mensaje de Fig. 8.

```
***** Loading and flattening adder5_opt (vst)...

***** Loading and flattening adder5 (al)...


***** Compare Terminals ..............
***** O.K.       (0 sec)

***** Compare Instances ..........
***** O.K.       (0 sec)

***** Compare Connections ............
***** O.K.       (0 sec)

===== Terminals .......... 17
===== Instances .......... 28
===== Connectors ......... 151


***** Netlists are Identical. *****     (0 sec)
```

Fig. 8. Mensaje "*Netlists are Identical*".

Al observar este mensaje sabemos que nuestro proceso de síntesis fue realizado de manera satisfactoria y la funcionalidad del módulo está garantizada.

VII. Conclusiones

El proceso de diseño de un módulo desde el lenguaje *VHDL* en conjunto con el uso de las herramientas de síntesis de *Alliance* permite desarrollar circuitos digitales complejos de manera muy sencilla, incluyendo ciertas pruebas sobre el funcionamiento del mismo.

De esta manera, el diseño se simplifica, de manera que se puede invertir más tiempo en desarrollar una arquitectura robusta y un patrón de pruebas útil para garantizar la funcionalidad del módulo.