Sumador de cinco bits sintetizado con las herramientas de Alliance

Diana Elsa Tenorio Cortés
Circuitos digitales
Dr. Marco Antonio Gurrola Navarro
Ingeniería electrónica y comunicaciones
Universidad de Guadalajara

Resumen— **Este documento presenta el desarrollo de un sumador de cinco bits, para el cual se utilizan las herramientas de síntesis y verificación en *Alliance*.**

Palabras clave— Alliance, Linux, síntesis, verificación, flujo de síntesis, sumador paralelo.

I. Introducción

ESTE documento presenta el desarrollo de un sumador de cinco bits, se utilizan las herramientas de síntesis y verificación de Alliance. En la sección II se puede encontrar la explicación detallada del módulo, en la sección III el código VHDL, que contiene la arquitectura del sumador, seguido del diagrama de disposición de conectores (.ioc), después en la sección V se presenta el plan de pruebas, así como el archivo de patrones (.pat), el script con el que se realizó el flujo de síntesis se encuentra descrito en la sección VII de este documento.

Finalmente se demuestra su correcto funcionamiento con la captura de pantalla de la herramienta LVX donde muestra que el netlist de LOON es igual al que extrae COUGAR y también una captura de pantalla del plano final en GRAAL.

II. EXPICACIÓN DEL MÓDULO.

Se desarrolló un sumador paralelo de dos datos A y B de cinco bits cada uno, con una salida del mismo ancho, del cual se espera que, al introducir los datos en las entradas, a la salida se obtenga la suma booleana de estos. En la tabla I se muestra un ejemplo del comportamiento esperado de este módulo.

Tabla I
Tabla ejemplo de sumas

ENTRADAS										SALIDA
A					B					A+B
a4	a3	a2	a1	a0	b4	b3	b2	b1	b0	
0	0	0	0	0	0	0	0	0	0	00000
0	0	0	0	0	0	0	0	0	1	00001
0	0	0	0	0	1	1	1	1	1	11111
1	1	1	1	1	0	0	0	0	0	11111
1	1	1	1	1	1	1	1	1	1	11110

III. CÓGIGO INICIAL VHDL

En la figura 1 se muestra el código VHDL del sumador paralelo de cinco bits, en él se describe la entidad, es decir las entradas y salidas del sistema, que para este caso A y B son entradas, definidas como vectores de 5 bits, y una salida RESULT del mismo ancho, ya que decidimos descartar el bit más signinicativo. También se describe la arquitectura, que refiere a como se relacionan las entradas para generar una salida.

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


entity Adder4 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 Adder4;


architecture DataFlow OF Adder4 is
begin

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

end DataFlow;
```

Fig. 1. Código VHDL inicial.

IV. DIAGRAMA DE DISPOSICION DE CONECTORES

El diagrama de disposición de conectores se presenta en la figura 2 y de este partimos para crear el archivo IOC, (inputs and outputs conectors) que es uno de los archivos iniciales, tiene la extensión. ioc y resulta indispensable ya que contiene la información de cómo se conectarán las entradas y salidas alrededor del Core, estas pueden estar en cuatro grupos (top, botton, right, left). Este archivo en el flujo de síntesis es utilizado por la herramienta OCP, ya que está ubica las celdas estándar con los conectores externos, tal como se muestra en la figura 3.

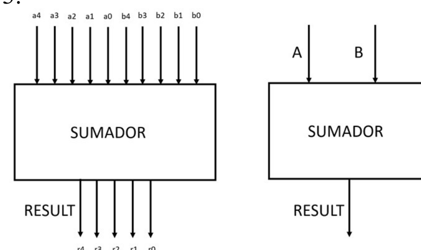

Fig. 2. Diagrama de bloques de disposición de conectores.

```
#
######################################################################
# In each of TOP()/BOTTOM()/LEFT()/RIGHT() section, there are       #
# placed IOs. In the IGNORE() section, the IOs are ignored           #
# by the IOPlacer. In every section, the IO syntax could be:         #
#        for pin:       (IOPIN iopinName.0 );
#        for pad:       iopadName orientation ;                      #
#        for space:     SPACE  value;                                #
# The capital words are keywords. orientation is not required.      #
# The value is the space between the IO above and the IO below it.#
######################################################################

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 );
   (IOPIN b(4).0 );
   (IOPIN b(3).0 );
   (IOPIN b(2).0 );
   (IOPIN b(1).0 );
   (IOPIN b(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 );
)
IGNORE ( # IOs are ignored(not placed) by IO Placer
)
```

Fig. 3. Archivo IOC

V. PLAN DE PRUEBAS

El plan de pruebas que se realizó fue tomar los posibles casos críticos, es decir los que involucren multiplicaciones por cero, o por $1F_H$ (donde H representa la nomenclatura hexadecimal), se seleccionaron l6 casos para análisis y de los que se conoce el resultado de su suma aritmética. Estos están descritos en la tabla II que se presenta a continuación.

Tabla II
TABLA PLAN DE PRUEBAS

ENTRADAS										SALIDA
A					B					A+B
a4	a3	a2	a1	a0	b4	b3	b2	b1	b0	
0	0	0	0	0	0	0	0	0	0	00000
					0	0	0	0	1	00001
					0	0	0	1	0	00010
					0	0	0	1	1	00011
0	0	0	0	1	0	0	0	0	0	00001
					0	0	0	0	1	00010
					1	0	0	0	0	10001
					1	1	1	1	1	00000
0	0	0	1	0	0	0	0	0	0	00010
					0	0	0	0	1	00010
					0	0	0	1	0	00100
					0	0	0	1	1	00101
1	1	1	1	1	0	0	0	0	0	11111
					0	0	0	0	1	00001
					1	0	0	0	0	01111
					1	1	1	1	1	11110

VI. ARCHIVO DE PATRONES

Para crear el archivo de patrones simplemente se vacía la información que se tiene en el plan de pruebas (tabla II) en un archivo que tendrá como extensión .pat, indicando entradas y salidas, además del tiempo que habrá entre una prueba y otra (90nS).

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

begin

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

<   0ns>: 00000  00000  ?*****  0 1;
< +90ns>: 00001  00001  ?*****  0 1;
< +90ns>: 00010  00010  ?*****  0 1;
< +90ns>: 00011  00011  ?*****  0 1;
< +90ns>: 00100  00100  ?*****  0 1;
< +90ns>: 01001  01001  ?*****  0 1;
< +90ns>: 10010  10010  ?*****  0 1;
< +90ns>: 11001  10001  ?*****  0 1;
< +90ns>: 11111  11111  ?*****  0 1;
< +90ns>: 00001  00001  ?*****  0 1;
< +90ns>: 00010  00010  ?*****  0 1;
< +90ns>: 00011  00011  ?*****  0 1;
< +90ns>: 00100  00100  ?*****  0 1;
< +90ns>: 01001  01001  ?*****  0 1;
< +90ns>: 10010  10010  ?*****  0 1;
< +90ns>: 11001  10001  ?*****  0 1;
< +90ns>: 11111  11111  ?*****  0 1;
< +90ns>: 00001  00001  ?*****  0 1;
< +90ns>: 00010  00010  ?*****  0 1;
< +90ns>: 00011  00011  ?*****  0 1;
< +90ns>: 00100  00100  ?*****  0 1;
< +90ns>: 01001  01001  ?*****  0 1;
< +90ns>: 10010  10010  ?*****  0 1;
< +90ns>: 11001  10001  ?*****  0 1;
< +90ns>: 11111  11111  ?*****  0 1;

end;
```

Fig. [4]. Archivo de patrones

VII. SCRIPT

El script es el conjunto de herramientas de síntesis y verificación guardadas en un archivo que tiene extensión .sh, es una manera simple de ejecutar todas las herramientas en una sola instrucción y verificar en que parte se detiene para detección de errores.

```
#!/bin/bash
export VH_MAXERR=1 &&
vasy -apo -I vhdl adder4 adder4 &&
asimut -b adder4 adder4 salida_vbe &&
boom -VA adder4 adder4_opt &&
boog adder4_opt adder4 &&
loon adder4 adder4_opt &&
asimut -b adder4 adder4 salida_vst &&
ocp -v -ioc adder4 adder4_opt adder4_p &&
nero -V -2 -p adder4_p adder4_opt adder4_f &&
export MBK_OUT_LO=al &&
cougar -v adder4_f adder4 &&
export MBK_OUT_LO=vst &&
lvx vst al adder4_opt adder4 -f &&
graal -l adder4_f
```

Fig. 5. Script

VIII. NETLIST IDENTICOS

La herramienta COUGAR hace una extracción del archivo estructural a partir del plano generado por OCP o NERO y lo compara con el archivo estructural generado por LOON, si estos dos son iguales quiere decir que no hubo errores en el proceso de *place and route* (posicionado y ruteado). En la figura [6] se muestra una captura de pantalla donde indica que los netlist son idénticos.

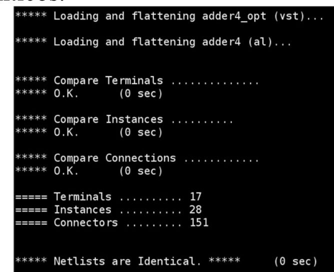

Fig. 6. Comparación de netlists

IX. GRAAL

Como último paso se muestra el plano final en la herramienta GRAAL, esto como resultado del flujo de síntesis. Las herramientas de verificación indican que es un plano funcional.

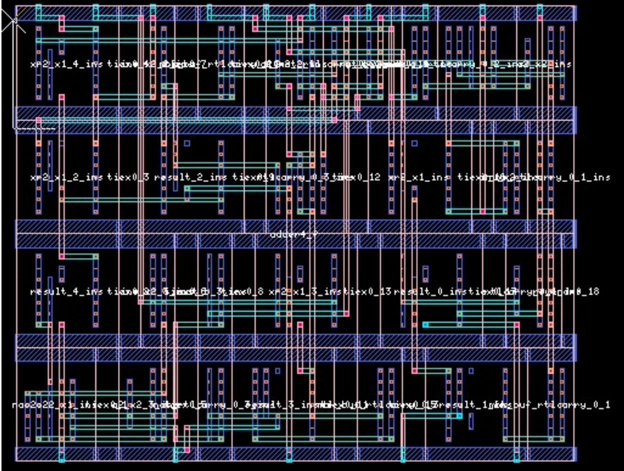

Fig. 7. Plano final en Graal

X. REFERENCIAS

- [1] Mano, M. Morris, Diseño Digital, pp. 119 - 124, Tercera Edición, Editorial Prentice Hall, 2003