

Universidad Autónoma de Baja California  
Facultad de Ciencias Químicas e Ingeniería



## **CIRCUITOS DIGITALES AVANZADOS**

### **Practica 6 Dispositivos Programables**

**Docente:** Lara Camacho Evangelina

**Alumnos:**  
Gómez Cárdenas Emmanuel Alberto **1261509**

## Contenido

OBJETIVO.....	3
EQUIPO .....	3
FUNDAMENTO TEORICO.....	3
Field Programmable Gate Array (FPGA) .....	3
Arquitectura de FPGAs Xilinx .....	4
Lut           6	
Flip-Flop   6	
Bloque DSP .....	6
Elementos de almacenamiento.....	7
FPGA y Procesadores .....	7
Field Programmable Analog Array (FPAA) .....	7
DESARROLLO.....	8
Simulación8	
Medio Sumador.....	8
Medio Sumador (Test Bench) .....	8
Medio Sumador (Simulación) .....	9
Sumador Completo .....	9
Sumador Completo (Test Bench) .....	9
Sumador Completo (Simulación) .....	10
Investigación.....	10
Síntesis: .....	10
Implementación:.....	10
CONCLUSIONES.....	12
REFERENCIAS.....	12

## OBJETIVO

Diseñar circuitos combinacionales en dispositivos programables FPGA.

## EQUIPO

Computadora con el compilador Xilinx Vivado u otro software para desarrollo de código para FPGAs.

## FUNDAMENTO TEORICO

### Field Programmable Gate Array (FPGA)

Los dispositivos programables FPGA ofrecen un número grande de bloques lógicos (Logic blocks) que contienen circuitos de lógica combinacional y registrada que se pueden programar de forma independiente. También, contienen un conjunto de bloques de entrada y salida (I/O) que pueden ser configurados como entrada fija, salida fija o bidireccionales. Las salidas tienen capacidad para operar como de tres estados y los registros pueden ser usados para retener datos de entrada o salida. Una arquitectura general de FPGAs es mostrada en la Fig. 1. Todos los bloques lógicos y de entrada y salida pueden ser interconectados por programación para implementar virtualmente cualquier circuito. La capacidad de programar las interconexiones se logra por medio de líneas que circulan a través de los renglones y columnas en los canales entre los bloques lógicos.

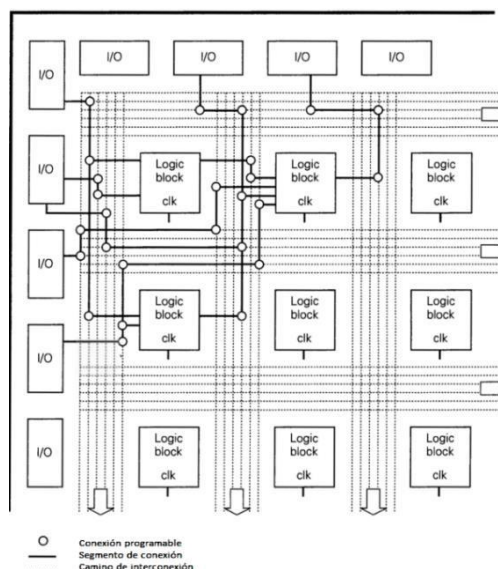


Figura 1. Arquitectura general de FPGAs.

Existen diferentes arquitecturas específicas de FPGAs y los datos que definen las conexiones programables son almacenados usando diferentes tecnologías, tales como SRAM, EEPROM, EEPROM flash y antifuse. El método de SRAM carga los datos a la RAM volátil desde una ROM externa cuando se conecta energía al dispositivo. Los métodos de EEPROM y flash trabajan de forma similar a los GAL. Tecnologías antifuse usan una conexión que está abierta hasta que se aplica un pulso de programación que causa que se pongan en corto. Es justo lo opuesto a fuse y de igual manera es reversible. Algunas FPGAs incluyen grandes bloques de memoria, otros no. Algunas usan arreglos de términos de producto para general expresiones SOP como los GAL, otros usan un enfoque de tabla de búsqueda (lookup tables, LUT). Como se puede ver, el campo de las FPGAs es diverso y cambia constantemente.

### Arquitectura de FPGAs Xilinx

La estructura básica de una FPGA está compuesta de los siguientes elementos:

- **Look-up table (LUT):** Tabla de búsqueda, este elemento se encarga de las operaciones lógicas.
- **Flip-flop:** Este elemento de registro almacena los resultados del LUT.
- **Conexiones:** Conectan elementos entre sí.
- **Pads de Entrada/Salida (I/O):** Puertos físicos que envían datos dentro y fuera de la FPGA.

La combinación de estos elementos resulta en la arquitectura básica de FPGAs mostrada en la Fig. 2. A pesar de que esta estructura es suficiente para implementar cualquier algoritmo, la eficiencia de la implementación está limitada en términos de rendimiento computacional, recursos requeridos y frecuencia de reloj alcanzable.

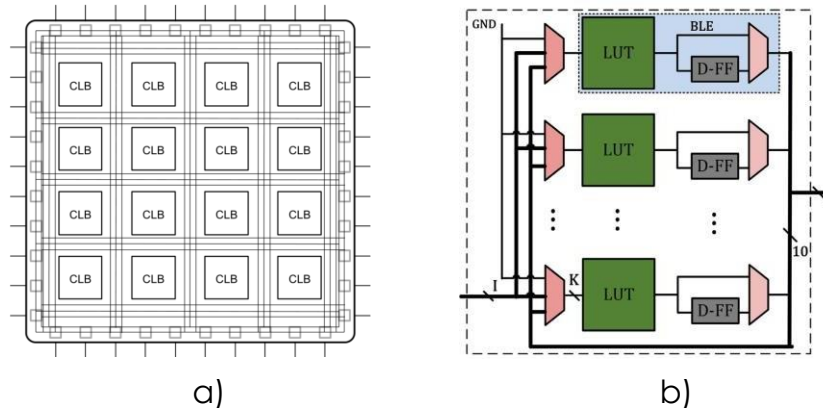


Figura 2. a) Estructura de una FPGA, b) Algunos componentes en un Configurable Logic Block (CLB).

FPGAs contemporáneas incorporan los elementos básicos junto con bloques adicionales de computación y almacenamiento que incrementan la densidad computacional y la eficiencia del dispositivo. Estos elementos adicionales son:

- Memorias embebidas para almacenamiento distribuido.
- Phase-locked loops (PLLs) para configurar la FPGA a diferentes velocidades de reloj.
- Transceivers serie de alta velocidad.
- Controladores de memoria.
- Bloques de Multiplicación-Suma.

La combinación de estos elementos provee a la FPGA con la flexibilidad de implementar cualquier algoritmo de software que se ejecuta en procesadores y resulta en la arquitectura contemporánea de FPGAs que se muestra en la Fig. 3.

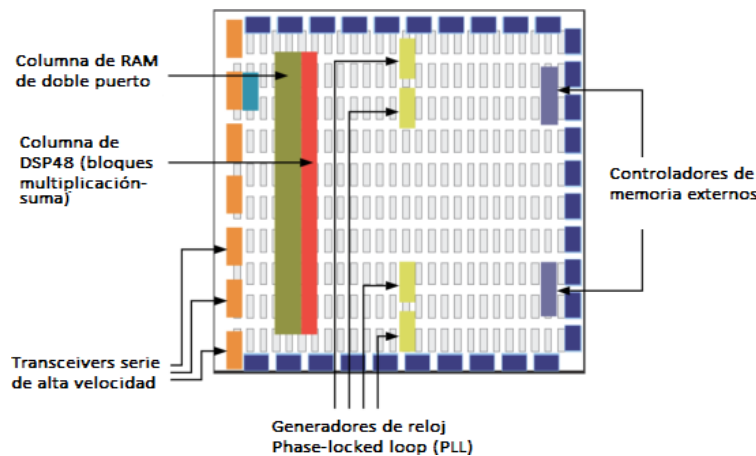


Figura 3. Arquitectura contemporánea de FPGAs.

Es el bloque de construcción básico en FPGAs y es capaz de implementar cualquier función lógica de  $N$  variables booleanas. Esencialmente, este elemento es una tabla de verdad en la cual diferentes combinaciones de las entradas implementan diferentes funciones para producir valores de salida. El límite en la tabla de verdad es  $N$ , donde  $N$  representa el número de entradas al LUT. Para el LUT general de  $N$  entradas, el número de localidades de memoria accedidas por la tabla es  $2^N$ , el cual permite que la tabla implemente  $2^N$  funciones. Un valor típico de  $N$  para FPGAs del fabricante Xilinx es 6.

## Lut

La implementación en hardware de un LUT puede ser pensada como una colección de celdas de memoria conectadas a un conjunto de multiplexores. Las entradas al LUT actúan como bits selectores en el multiplexor para seleccionar el resultado. Es importante tener esta representación en mente ya que un LUT puede ser usado tanto como un elemento de almacenamiento o como un motor de cálculo. La Fig. 4 muestra esta representación funcional del LUT.

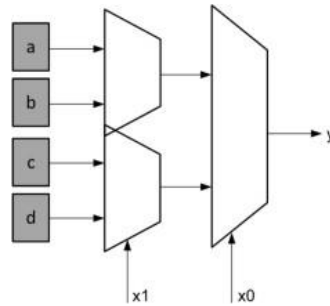


Figura 4. Representación funcional de un LUT como una colección de celdas de memoria.

## Flip-Flop

Es la unidad básica de almacenamiento dentro de una FPGA. Este elemento siempre está emparejado con un LUT para asistir en la canalización lógica y almacenamiento de datos.

## Bloque DSP

Es el bloque computacional más complejo en una FPGA, el cual se muestra en la Fig. 5. Consiste en una Unidad Aritmética y Lógica (ALU) compuesta de una cadena de tres bloques diferentes: una unidad de suma/resta, conectada a un multiplicador, que está conectado a un motor de suma/resta/acumulación final. Esta cadena permite a una sola unidad DSP implementar funciones de la forma:

$$p = a \times (b + d) + c$$

ó

$$p += a \times (b + d)$$

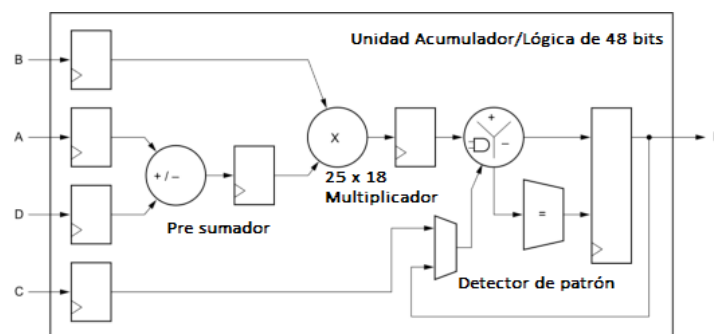


Figura 5. Estructura de un bloque DSP.

### Elementos de almacenamiento

El dispositivo FPGA incluye elementos de memoria empotrados que pueden ser usados como memoria de acceso aleatorio (RAM), memoria de solo lectura (ROM) o registros de corrimiento. Estos elementos son bloques de RAM (BRAMs), bloques UltraRAM (URAMS), LUTs y registros de corrimiento (SRLs).

BRAMs pueden implementar ya sea una ROM o una RAM, la única diferencia es cuándo son escritos los datos. En una implementación de RAM, los datos pueden ser escritos y leídos en cualquier tiempo en la ejecución. En cambio, en una implementación de ROM, los datos solo se pueden leer durante la ejecución. Los datos se escriben en ROM como parte de la configuración del dispositivo y no pueden ser modificados. Los bloques UltraRAM son RAM síncronas que proveen mucha más capacidad que las BRAM.

Como se describió anteriormente, el LUT es una pequeña memoria en la cual el contenido de una tabla de verdad es escrito durante la configuración del dispositivo. Debido a su flexibilidad, pueden ser usados como bloques de memoria, comúnmente llamados memorias distribuidas. Es la memoria más rápida en el dispositivo FPGA ya que puede ser instanciada en cualquier parte de su estructura.

### FPGA y Procesadores

Un procesador, sin importar su tipo, ejecuta un programa como una secuencia de instrucciones, es decir, ejecuta una instrucción tras otra. La latencia computacional de cada instrucción no es igual entre tipos de instrucciones. Por ejemplo, dependiendo en donde se encuentre un operando, las instrucciones toman un número diferente de ciclos en completarse. Si está en la caché del procesador, la instrucción se puede completar en unas decenas de ciclos de reloj. Si está en memoria RAM DDR, las instrucciones podrían requerir cientos o miles de ciclos de reloj. Y si estuviera en el disco duro, las instrucciones tardarían aún más.

**FPGAs son estructuras de inherente procesamiento paralelo** capaces de implementar cualquier función aritmética o lógica que pueda ejecutarse en un procesador. Una FPGA también difiere de un procesador en la arquitectura de memoria y en el costo de los accesos a memoria. Los compiladores para FPGAs son capaces de organizar las memorias como múltiples bancos de almacenamiento lo más cercanos posibles al punto en donde se van a usar en el circuito. Esto resulta en un instantáneo ancho de banda de memoria, que excede por mucho al de un procesador.

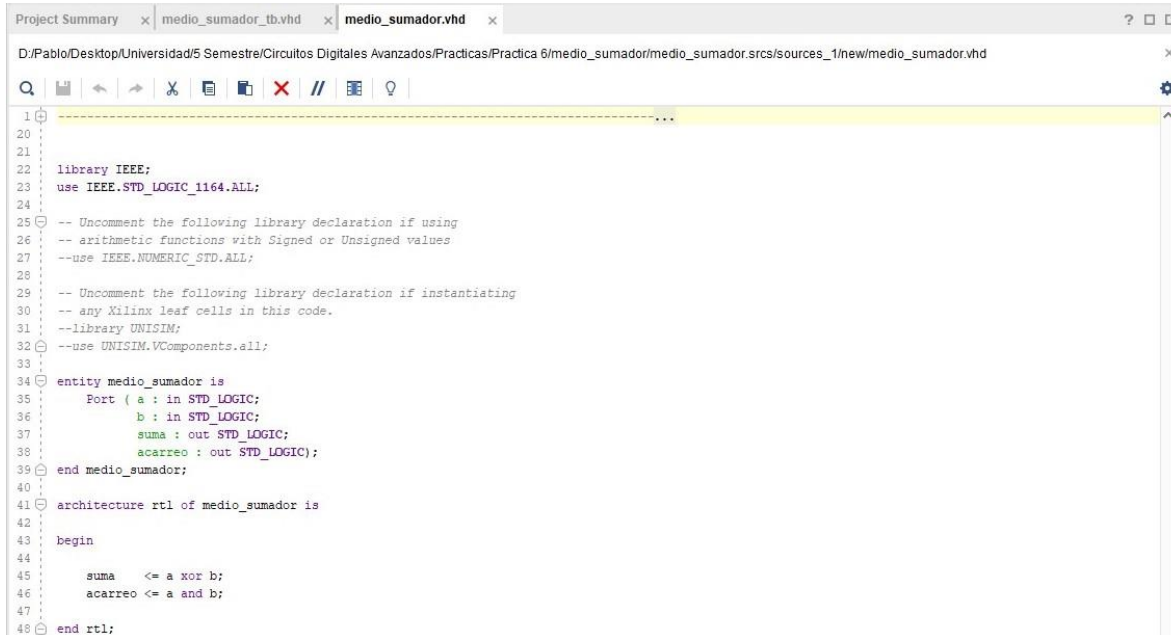
### Field Programmable Analog Array (FPAA)

Es un circuito integrado con una colección de bloques de construcción análogos interconectados, que logran una reconfigurabilidad similar a las FPGAs. Son la contraparte análoga de FPGAs. A diferencia de ellas, las FPAAs tienden a estar orientadas a tareas específicas en lugar de ser de propósito general, se usan en circuitos tales como diseño de filtros.

# DESARROLLO

## Simulación

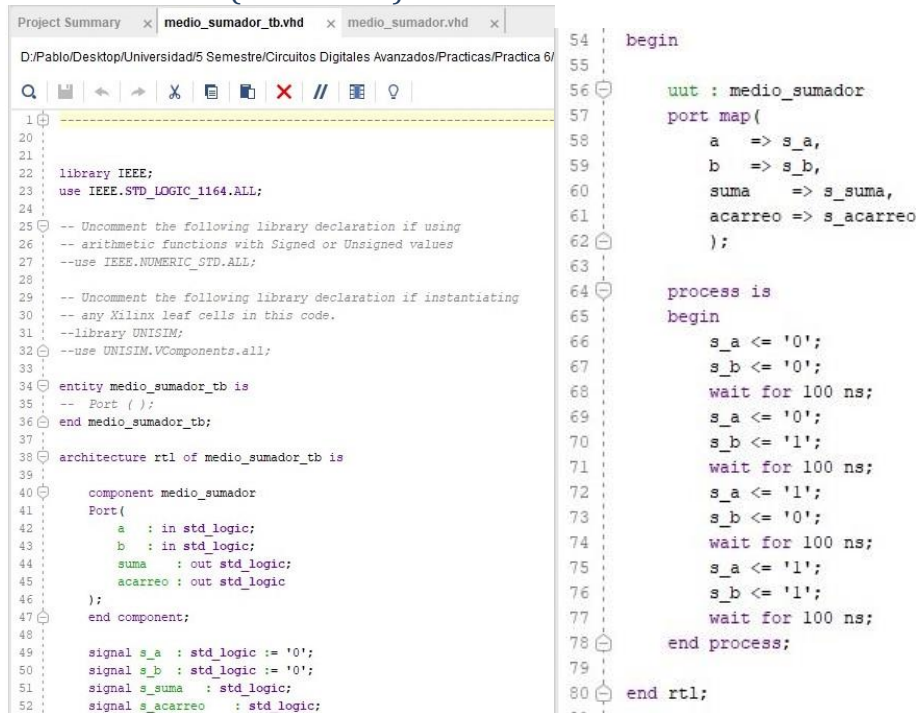
### Medio Sumador



```

1 20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity medio_sumador is
35     Port ( a : in STD_LOGIC;
36           b : in STD_LOGIC;
37           suma : out STD_LOGIC;
38           acarreo : out STD_LOGIC);
39 end medio_sumador;
40
41 architecture rtl of medio_sumador is
42
43 begin
44
45     suma <= a xor b;
46     acarreo <= a and b;
47
48 end rtl;
  
```

### Medio Sumador (Test Bench)



```

1 20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity medio_sumador_tb is
35     -- Port ( );
36 end medio_sumador_tb;
37
38 architecture rtl of medio_sumador_tb is
39
40     component medio_sumador
41     Port(
42         a : in std_logic;
43         b : in std_logic;
44         suma : out std_logic;
45         acarreo : out std_logic
46     );
47 end component;
48
49 signal s_a : std_logic := '0';
50 signal s_b : std_logic := '0';
51 signal s_suma : std_logic;
52 signal s_acarreo : std_logic;
53
54 begin
55
56     uut : medio_sumador
57     port map(
58         a => s_a,
59         b => s_b,
60         suma => s_suma,
61         acarreo => s_acarreo
62     );
63
64     process is
65     begin
66         s_a <= '0';
67         s_b <= '0';
68         wait for 100 ns;
69         s_a <= '0';
70         s_b <= '1';
71         wait for 100 ns;
72         s_a <= '1';
73         s_b <= '0';
74         wait for 100 ns;
75         s_a <= '1';
76         s_b <= '1';
77         wait for 100 ns;
78     end process;
79
80 end rtl;
  
```



## Medio Sumador (Simulación)

Name	Value	0.000 ns	100.000 ns	200.000 ns	300.000 ns	400.000 ns
s_a	0					
s_b	0					
s_suma	0					
s_acarreo	0					

## Sumador Completo

```

Project Summary | x | sumador_completo_tb.vhd | x | sumador_completo.vhd | x
D:\Pablo\Desktop\Universidad\5 Semestre\Circuitos Digitales Avanzados\Practicas\Practica 6\sumador_completo\sumador_completo.srcs\sources_1\new\sumador_completo.vhd

1
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity sumador_completo is
35     Port ( a : in STD_LOGIC;
36           b : in STD_LOGIC;
37           c : in STD_LOGIC;
38           suma : out STD_LOGIC;
39           acarreo : out STD_LOGIC);
40 end sumador_completo;
41
42 architecture rtl of sumador_completo is
43 begin
44
45     suma <= a xor b xor c;
46     acarreo <= (a and b) or ((a or b) and (c));
47
48 end rtl;

```

## Sumador Completo (Test Bench)

```

D:\Pablo\Desktop\Universidad\5 Semestre\Circuitos Digitales Avanzados\Practicas\Practica 6\
1
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx leaf cells in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity sumador_completo_tb is
35     Port ( );
36 end sumador_completo_tb;
37
38 architecture rtl of sumador_completo_tb is
39
40     component sumador_completo
41     Port(
42         a : in std_logic;
43         b : in std_logic;
44         c : in std_logic;
45         suma : out std_logic;
46         acarreo : out std_logic
47     );
48 end component;
49
50 signal s_a : std_logic := '0';
51 signal s_b : std_logic := '0';
52 signal s_c : std_logic := '0';
53 signal s_suma : std_logic;
54 signal s_acarreo : std_logic;
55
56 begin
57
58 uut : sumador_completo
59 port map(
60     a => s_a,
61     b => s_b,
62     c => s_c,
63     suma => s_suma,
64     acarreo => s_acarreo
65 );
66
67 process is
68 begin
69     s_a <= '0';
70     s_b <= '0';
71     s_c <= '0';
72     wait for 100 ns;
73     s_a <= '0';
74     s_b <= '0';
75     s_c <= '1';
76     wait for 100 ns;
77     s_a <= '0';
78     s_b <= '1';
79     s_c <= '0';
80     wait for 100 ns;
81     s_a <= '0';
82     s_b <= '1';
83     s_c <= '1';
84     wait for 100 ns;
85     s_a <= '1';
86     s_b <= '0';
87     s_c <= '0';
88     wait for 100 ns;
89     s_a <= '1';
90     s_b <= '0';
91     s_c <= '1';
92     wait for 100 ns;
93     s_a <= '1';
94     s_b <= '1';
95     s_c <= '0';
96
97     wait for 100 ns;
98     s_a <= '1';
99     s_b <= '1';
100     s_c <= '1';
101     wait for 100 ns;
102 end process;
103 end rtl;

```

## Sumador Completo (Simulación)

Name	Value	0.000 ns	100.000 ns	200.000 ns	300.000 ns	400.000 ns	500.000 ns	600.000 ns	700.000 ns	800.000 ns
s_a	0									
s_b	1									
s_c	0									
s_suma	1									
s_acarreo	0									

## Investigación

1. Describa detalladamente en que consiste cada una de las etapas, que reciben y que producen como salida.

### Síntesis:

El proceso síntesis, verifica la sintaxis del código y analiza la jerarquía del diseño, lo cual asegura que el diseño este optimizado para la arquitectura de diseño que ha seleccionado. El netlist resultante se guarda en un archivo NGC para Xilinx Synthesis Technology o en uno EDIF para Precision o Synplify.

### Implementación:

El proceso de implementación consiste en tres pasos:

#### Traducción:

Este proceso es un paso importante en el proceso de implementación ya que fusiona todas las “netlist” de entrada, las restricciones de diseño del archivo NGC o EDIF y genera un archivo NGD (Native Generic Database [base de datos genérica nativa de Xilinx]) que describe el diseño lógico reducido a primitivas de Xilinx.

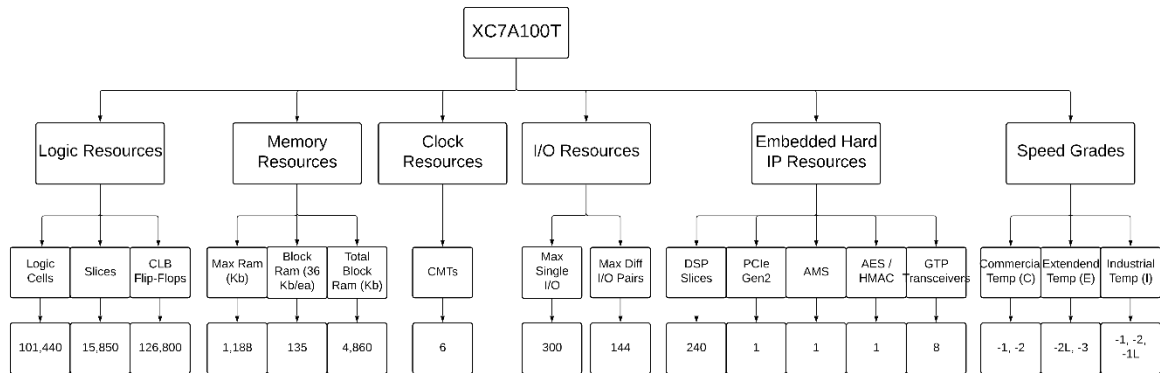
#### Mapeo:

Este proceso mapea toda la lógica definida por un archivo NGD en elementos FPGA, como CLB e IOB. El diseño de salida es un archivo de descripción de circuito nativo (Native Circuito Description, NCD) el cual representa físicamente el diseño asignado a los componentes en XILINX FPGA.

#### Ruteo de componentes:

El proceso de ruteo de componentes toma el archivo NCD mapeado y describe varios procesos en los que los elementos del “netlist” se ubican físicamente y se asignan a los recursos físicos de la FPGA.

2. Realice un mapa conceptual donde indique las características y capacidades del dispositivo FPGA que se usó en esta práctica.



3. ¿Qué es un archivo test bench y por qué es importante realizarlo?

Un test bench son piezas de código HDL que se utiliza durante la simulación FPGA o ASIC. Este código le permite proporcionar un conjunto documentado y repetitivo de estímulos que es portátil a través de diferentes simuladores. La simulación es importante ya que genera la posibilidad de revisar el diseño y asegurarse que hace lo que se espera que haga. Se debe intentar crear todas las condiciones posibles de entrada para verificar cada caso del proyecto.

## CONCLUSIONES

### **Gómez Cárdenas Emmanuel Alberto:**

En esta práctica aprendimos a programar y simular los FPGAs con la utilización del programa Vivado. Gracias a la práctica realizada logramos entender más el funcionamiento y la utilidad de los FPGAs, en específico del XC7A100T.

## REFERENCIAS

ISE Design Suite. (2021). Retrieved 27 May 2021, from

<https://www.xilinx.com/products/design-tools/ise-design-suite.html>

fpga4fun.com - FPGA software 5 - FPGA synthesis and place-and-route. (2021).

from <https://www.fpga4fun.com/FPGAsoftware5.html>

Implementation Overview for FPGAs. (2021). from

[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx11/ise\\_c\\_i\\_mplement\\_fpga\\_design.htm](https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_i_mplement_fpga_design.htm)

Mouser Electronics, Inc. México. (2021). from

<https://www.mouser.mx/new/xilinx/xilinx-artix-7-fpgas/https://www.nandland.com/articles/what-is-a-testbench-fpga.html>.

Running the Map Process. (2021). from

[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx11/pp\\_p\\_process\\_mapping.htm](https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/pp_p_process_mapping.htm)

Running the Synthesis Process. (2021). from

[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx10/isehelp/pp\\_p\\_process\\_synthesize.htm](https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/pp_p_process_synthesize.htm)

Translate Process. (2021). From

[https://www.xilinx.com/support/documentation/sw\\_manuals/xilinx10/isehelp/pp\\_n\\_process\\_translate.htm](https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/pp_n_process_translate.htm)