Laboratorio 2: Introducción a los lenguajes de descripción de hardware

Jonathan Guzmán Araya, Mariano Muñoz Masís
Instituto Tecnológico de Costa Rica
Área Académica Ingeniería en Computadores
jonathana1196@gmail.com marianomm1301@gmail.com

***Abstract*—In this laboratory, we investigated digital design technologies, both in hardware, such as FPGAs, as well as in software, such as the SystemVerilog and VHDL hardware description languages, and behavior and structure design models. A 7-segment display decoder was designed in SystemVerilog in behavior model, in VHDL a full 5-bit adder in structure model, and in SystemVerilog a parameterizable N-bit counter with asynchronous reset, which were also programmed and tested in the FPGA.**

***Index Terms*—Decodificador, CPU, GPU, Flip-Flop, FPGA, HDL, SystemVerilog, VHDL.**

I. Introducción

Los lenguajes de descripción de hardware (HDL) son herramientas extremadamente importantes para los diseñadores digitales modernos. Una vez que haya aprendido SystemVerilog o VHDL, podrá especificar sistemas digitales mucho más rápido que si tuviera que dibujar los esquemas completos. El ciclo de depuración también suele ser mucho más rápido, porque las modificaciones requieren cambios de código en lugar de un complicado recableado de esquemas. Sin embargo, el ciclo de depuración puede ser mucho más largo con HDL si no tiene una buena idea del hardware que implica su código. Los HDL se utilizan tanto para simulación como para síntesis. La simulación lógica es una forma poderosa de probar un sistema en una computadora antes de convertirlo en hardware. Los simuladores le permiten verificar los valores de las señales dentro de su sistema que podrían ser imposibles de medir en una pieza física de hardware [1]. Un bloque de hardware que posea entradas y salidas es llamado módulo. Una compuerta AND, un multiplexor y un circuito de prioridad son todos ejemplos de módulos de hardware. Existen dos estilos generales en los que se describen la funcionalidad de un módulo, estos son de comportamiento y estructurales.

A. Modelo de comportamiento

Los modelos de comportamiento se utilizan ara describir el comportamiento del sistema en su totalidad, principalmente se tiene el *Modelo de Flujo de Datos*, que modela el procesamiento de los datos del sistema, y el *Modelo de Máquinas de Estado*, que modelan el sistema en función a eventos que recibe el sistema. Un ejemplo para un módulo de comportamiento se tienen una compuerta AND, la cual dependiendo de las entradas dará una salida, si cualquiera de las dos entradas es un emph0, la salida siempre será un *0*, pero cuando las dos entradas son un *1* la salida será un *1* lógico.

B. Modelo de Estructura

El Modelo de Estructura sirve para explicar los diferentes tipos de objetos de un sistema, visto desde el punto de vista de Software tenemos el *Diagrama de Clases*, básico para la construcción de cualquier programa, de la misma manera se debe modelar los circuitos electrónicos, aplicando el concepto de Diseño Modular. Un ejemplo de esto es la construcción de un sumador completo de de *1 bit*, el cual por lo general se compone de varias compuertas *XOR* para la suma y de *AND* y *OR* para el acarreo.

Un proceso importante en el uso de HDL es la síntesis lógica. Las raíces de este proceso se remontan al tratamiento de la lógica por George Boole, en lo que ahora se denomina álgebra booleana. En los primeros días, el diseño lógico implicaba manipular las representaciones de tablas de verdad como mapas de Karnaugh. Sin embargo, la evolución de componentes lógicos discretos a matrices lógicas programables aceleró la necesidad de la automatización de la síntesis lógica. La síntesis lógica e un proceso mediante el cual una especificación abstracta del comportamiento deseado de un circuito (código HDL), normalmente a nivel de transferencia de registro (RTL), se convierte en en una implementación de diseño en términos de compuertas lógicas, normalmente mediante un programa informático llamado herramienta de síntesis. Por lo que se puede decir que la síntesis lógica es un aspecto de la automatización del diseño electrónico.

Las FPGAs son el acrónimo para Field Programmable Gate Array y es una serie de dispositivos basados en semiconductores a base de matrices de Bloques Lógicos Configurables, su principal característica es que pueden ser reprogramados para un trabajo en específico [2]. Los principales componentes que posee la FPGA son terminales, Buffers, Flips – Flops, Tablas de búsqueda, Bloques de memoria, Bloques dedicados de Multiplicación, Transceptores para transmisión serie de muy alta velocidad, procesador en hardware embebido, etc [3].

La cartera de soluciones programables de Intel aborda el control determinista y la conectividad, así como la informática de borde emergente requerida para las aplicaciones de la Industria 4.0. Nuestra amplitud de soluciones y nuestro ecosistema de socios para control, conectividad, inteligencia arti-

ficial, seguridad funcional y seguridad le facilitan el diseño de sistemas que son inteligentes, seguros, eficientes y confiables.

Con las innovadoras soluciones FPGA de Intel en el corazón de sus diseños industriales, está equipado para abordar los siguientes desafíos clave:

- Adaptarse de forma rápida y rentable a los mercados finales y los estándares en evolución.
- Cumplir y aumentar los requisitos de rendimiento en todas las líneas de productos.
- Reducir el costo del desarrollo del sistema y la lista de materiales mediante una mayor integración del diseño.

Entre los más destacables esta la tecnología Cyclone, dedicada al Ethernet Industrial, al Edge Computing, los sistemas industriales de manofactura por robots, los procesos industriales dirigidos por el Cloud Computing y además usados para la predicción de mantenimientos de las maquinas utilizadas en la industria, como también de la seguridad laboral de las plantas.

El mundo actual evoluciona alrededor de la tecnología y las ventajas que esta conlleva, por lo que se depende en gran medida del crecimiento y mejora de esta para producir comodidad y satisfacción a nuestras vidas, ya sea desde el punto de vista personal o industrial (empresarial). Los dispositivos FPGA juegan un papel importante en esta faceta, por lo que algunas de sus aplicaciones más comunes son las que se encuentran en los siguientes campos de interés:

- Procesamiento de vídeos e imágenes
- Telecomunicaciones y comunicación de datos
- Servidor y nube
- Defensa y espacio
- Científico y médico

Se utiliza con frecuencia y de forma extensiva en los sistemas de comunicación para mejorar la capacidad de la red, la cobertura y la calidad del servicio, al mismo tiempo que se reduce la latencia y los retrasos, especialmente cuando se trata de manipulación de datos [4].

Intel y Xinlinx con ayuda de las FPGA han ido abriendo nichos en aplicaciones de centros de datos donde el bajo consumo de energía, el alto rendimiento y la capacidad de configuración pueden superar los desafíos de programación [5].

La industria automotriz utiliza sistemas electrónicos cada vez más complejos para ofrecer mayor seguridad y eficiencia al conductor. Sin embargo es difícil para las unidades de control (ECU) basadas en CPUy GPU mantenerse al día con la electrónica de consumo, debido a los largos ciclos de desarrollo de chips y los rigurosos estándares de confiabilidad y calidad aplicados a la industria automotriz. Es por ello que los arreglos de puertas programables (FPGA) pueden desempeñar un papel importante para llenar este vacío al proporcionar rendimiento y flexibilidad de vanguardia a los arquitectos de sistemas para personalizar sus proyectos con una estructura de circuito electrónico flexible (programable) [6].

II. Problema 1 : Decodificador 7 Segmentos

A. Desarrollo y Resultados

El decodificador es una herramienta muy utilizada en los sistemas digitales para intervenir procesos, estos tiene la característica que para una cantidad de entradas N, posee una cantidad de salidas M, donde M es menor o igual a 2^N.

Para el caso de este problema se desea transformar una entrada de 5 bits en un alfabeto de 22 símbolos.

Código (Decimal)	Símbolo
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	G
17	H
18	I
19	L
20	P
21	S
22	U
Cualquier Otro	E

Fig. 1. Alfabeto Decodificador 7 Segmentos

Para representar el alfabeto anterior se diseño la tabla con las entradas y las salidas deseadas.

Posteriormente utilizando una herramienta online, se obtienen las funciones canónicas para las salidas que se esperan.

Las mismas en lenguaje de System Verilog tienen la siguiente sintaxis.

B. Análisis de Resultados

Para probar el funcionamiento del sistema, se utilizó un testbench, las pruebas se realizaron con las cadenas de bits 00000, 00001, 10000 y 01110, que según la Figura 2 corresponden a 0, 1, G y E respectivamente.

Para el test realizado, y las salidas esperadas dadas las entradas mostradas se tienen los mostrados en la Figura 6.

Con la simulación se puede apreciar que efectivamente, los valores obtenidos son los mostrados en la Tabla 2.

III. Problema 2 : Sumador 5 bits

A. Desarrollo y Resultados

Para la creación del sumador de 6 bits se utilizó la estructura básica del sumador de 1 bit, si conectamos sumadores de 1 bit en ”serie”, podremos obtener sumadores de N bits, en este

#	Alf	A	B	C	D	E	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	1	1	0	0	1	1	1	1
2	2	0	0	0	1	0	0	0	1	0	0	1	0
3	3	0	0	0	1	1	0	0	0	0	1	1	0
4	4	0	0	1	0	0	1	0	0	1	1	0	0
5	5	0	0	1	0	1	0	1	0	0	1	0	0
6	6	0	0	1	1	0	0	1	0	0	0	0	0
7	7	0	0	1	1	1	0	0	0	1	1	1	1
8	8	0	1	0	0	0	0	0	0	0	0	0	0
9	9	0	1	0	0	1	0	0	0	1	1	0	0
10	A	0	1	0	1	0	0	0	0	1	0	0	0
11	B	0	1	0	1	1	1	1	0	0	0	0	0
12	C	0	1	1	0	0	0	1	1	0	0	0	1
13	D	0	1	1	0	1	1	0	0	0	0	1	0
14	E	0	1	1	1	0	0	1	1	0	0	0	0
15	F	0	1	1	1	1	0	1	1	1	0	0	0
16	G	1	0	0	0	0	0	1	0	0	0	0	1
17	H	1	0	0	0	1	1	1	0	1	0	0	0
18	I	1	0	0	1	0	1	1	0	1	1	1	1
19	L	1	0	0	1	1	1	1	1	0	0	0	1
20	P	1	0	1	0	0	0	0	1	1	0	0	0
21	S	1	0	1	0	1	0	1	0	0	1	0	1
22	U	1	0	1	1	0	1	0	0	0	0	0	1
23	E	1	0	1	1	1	0	1	1	0	0	0	0
24	E	1	1	0	0	0	0	1	1	0	0	0	0
25	E	1	1	0	0	1	0	1	1	0	0	0	0
26	E	1	1	0	1	0	0	1	1	0	0	0	0
27	E	1	1	0	1	1	0	1	1	0	0	0	0
28	E	1	1	1	0	0	0	1	1	0	0	0	0
29	E	1	1	1	0	1	0	1	1	0	0	0	0
30	E	1	1	1	1	0	0	1	1	0	0	0	0
31	E	1	1	1	1	1	0	1	1	0	0	0	0

Fig. 2. Diseño Decodificador 7 Segmentos

a = B'C'D'E + AB'C'E + AB'DE' + A'B'CD'E' + A'BC'DE + A'BCD'E
b = AC' + AE + BDE + BCE' + B'CD'E + A'CDE'
c = AB + BCE' + BCD + ADE + ACD'E' + A'B'C'DE'
d= A'C'D'E + B'C'D'E + B'CD'E' + A'CDE + A'BC'DE' + AB'C'DE'
e=A'B'E + A'C'D'E + A'B'CD' + B'CD'E + AB'C'DE'
f = A'B'C'E + B'C'DE' + A'B'DE + A'BCD'E
g = A'B'C'D' + B'C'D'E' + AB'C'D + AB'DE' + A'B'CDE + A'BCD'E' + AB'CD'E

Fig. 3. Funciones Canónicas

```
//a = B'C'D'E + AB'C'E + AB'DE' + A'B'CD'E' + A'BC'DE + A'BCD'E
assign seg[6] = (((~B)&(~C)&(~D)&(E))|((A)&(~B)&(~C)&(E))|
                 ((A)&(~B)&(D)&(~E))|((~A)&(~B)&(C)&(~D)&(~E))|
                 ((~A)&(B)&(!C)&(D)&(E))|((!A)&(B)&(C)&(~D)&(E)));

//b = AC' + AE + BDE + BCE' + B'CD'E + A'CDE'
assign seg[5] = (((A)&(~C))|((A)&(E))|
                 ((B)&(D)&(E))|((B)&(C)&(~E))|
                 ((~B)&(C)&(~D)&(E))|((~A)&(C)&(D)&(~E)));

//c = AB + BCE' + BCD + ADE + ACD'E' + A'B'C'DE'
assign seg[4] = (((A)&(B))|((B)&(C)&(~E))|
                 (B&C&D)|(A&D&E)|(A&C&(~D)&(~E))|
                 ((~A)&(~B)&(~C)&D&(~E)));

//d= A'C'D'E + B'C'D'E + B'CD'E' + A'CDE + A'BC'DE' + AB'C'DE'
assign seg[3] = (((~A)&(~C)&(~D)&E) | ((~B)&(~C)&(~D)&E) |
                 ((~B)&C&(~D)&(~E)) | ((~A)&C&D&E) |
                 ((~A)&B&(~C)&D&(~E)) | (A&(~B)&(~C)&D&(~E)));

//e=A'B'E + A'C'D'E + A'B'CD' + B'CD'E + AB'C'DE'
assign seg[2] = (((~A)&(~B)&E) | ((~A)&(~C)&(~D)&E) |
                 ((~A)&(~B)&C&(~D)) | ((~B)&C&(~D)&E) |
                 (A&(~B)&(~C)&D&(~E)));

//f = A'B'C'E + B'C'DE' + A'B'DE + A'BCD'E
assign seg[1] = (((~A)&(~B)&(~C)&E) | ((~B)&(~C)&D&(~E)) |
                 ((~A)&(~B)&D&E) | ((~A)&B&C&(~D)&E));

//g = A'B'C'D' + B'C'D'E' + AB'C'D + AB'DE' + A'B'CDE + A'BCD'E' + AB'CD'E
assign seg[0] = (((~A)&(~B)&(~C)&(~D))|((~B)&(~C)&(~D)&(~E))|
                 (A&(~B)&(~C)&D)|(A&(~B)&D&(~E))|((~A)&(~B)&C&D&E) |
                 ((~A)&B&C&(~D)&(~E)) | (A&(~B)&C&(~D)&E));
```

Fig. 4. Funciones Canónicas en System Verilog

```
//0 -- 0 0  0  0  0  0  1
line = 5'b00000; #10;

//1 -- 1 0  0  1  1  1  1
line = 5'b00001; #10;

//G -- 0 1  0  0  0  0  1
line = 5'b10000; #10;

//E -- 0 1  1  0  0  0  0
line = 5'b01110; #10;
```

Fig. 5. Testbench Decodificador 7 Segmentos

caso se debe realizar un sumador de 6 bits, pero dadas las limitaciones de hardware de la FPGA se diseñó de 5 bits.

De esta manera se utilizo el sumador de 1 bit para realizar la estructura básica, con un modelo de comportamiento.

Los valores corresponden a: c_{in}, es el acarreo de entrada, a y b, son los valores que se desean sumar, c es el resultado de esta suma y c_{out} es el acarreo de salida debido a un exceso de bits. Dados los datos de la Figura 7 se plantean las funciones canónicas que se aprecian en la Figura 8.

Estas funciones son escritas en VHDL, y utilizadas para hacer el modulo principal del programa. Mediante el acople de módulos se logra un sumador de 5 bits, con bandera de acarreo de salida.

Además de ello, se diseña un modulo para representar los valores obtenidos en la FPGA.

B. Análisis de Resultados

Al igual que en el problema anterior para analizar las entradas, se realiza en testbench.

Dada la cantidad de pares de sumas que se pueden realizar se van a analizar únicamente un par.

Primeramente tenemos 0 + 1, que en binario sería 00000 + 00001 y que como resultado se esperaría un 00001, para representar la salida de este programa se utilizaron 7 segmentos para representar cada uno de los bits de salida.

Si un 7 segmentos tiene como salida ”1111001”, simboliza una salida en alta, o un 1 lógico, pero si tiene como salida ”1000000” simboliza una salida en bajo o un 0 lógico.

Por tanto la operación anterior debería dar como resultado el siguiente arreglo de 7 segmentos

- seg5 = 1000000
- seg4 = 1000000
- seg3 = 1000000
- seg2 = 1000000
- seg1 = 1000000
- seg0 = 1111001

A lo cual se obtuvo como resultados lo mostrado en la Figura 9.

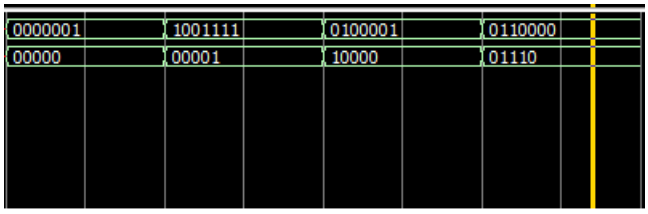

Fig. 6. Resultados de la Simulación

c_in	a	b	c_out	c
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig. 7. Diseño de un Sumador de 1 bit

$$c\_out = BC + AC + AB$$
$$c = A'B'C + A'BC' + AB'C' + ABC$$

Fig. 8. Funciones Canónicas Sumador 1 bit

El resultado es el esperado. Las pruebas para demostrar el correcto funcionamiento se realizaron mediante pruebas físicas con la FPGA y mostraron los valores esperados.

IV. Problema 3 : Contador Parametrizable N bits

A. Desarrollo y Resultados

Un contador es un circuito en el que sus salidas siguen una secuencia fija, que cuando acaba o es reiniciado vuelve a comenzar, el conteo puede ser sobre pulsos de reloj o bien pueden originarse por una fuente externa y ocurrir en intervalos fijos o aleatorios, además el número de salidas limita el número máximo que este puede contar. Además cabe recalcar que los contadores son circuitos secuenciales por lo tanto se crean con flip-flops, que bien pueden ser de tipo D, T, J-K o también pueden crearse a partir de compuertas lógicas. Para este problema se solicita la creación de un contador asincrónico parametrizable de n bits, un contador de n bits puede contar desde 0 hasta 2^{n-1}. El conteo se realiza mediante cambios en la entrada de los fip-flops, de manera que estos

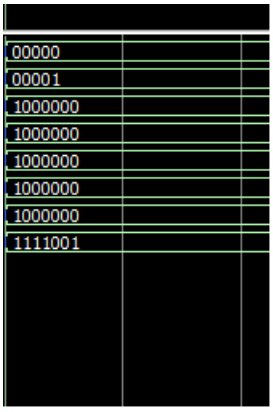

Fig. 9. Test Bench Sumador 5 bits

modifican sus estados dando lugar a un nuevo valor de salida, pero cuando la entrada no varía, los flip-flops conservan su estado presente. Existen dos grupos de contadores: sincrónicos y asincrónicos. Los contadores sincrónicos tienen un reloj interno, mientras que los asincrónicos no.

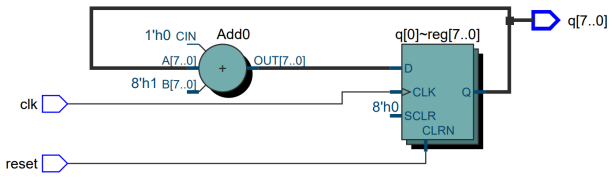

Fig. 10. Contador 8 bits

El contador desarrollado se puede observar en la Figura 10, este es un contador de parametrizable de 8 bits. Se desarrolló una tabla que se muestra en la Figura 11 con los números en binario y su representación decimal para un contador de 6 bits, aunque cabe destacar que funciona para el contador de 2 y 4 bits, únicamente tomando la cantidad de bits correspondiente.

Q5	Q4	Q3	Q2	Q1	Q0	DEC
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	2
0	0	0	0	1	1	3
0	0	0	1	0	0	4
0	0	0	1	0	1	5
0	0	0	1	1	0	6
0	0	0	1	1	1	7
0	0	1	0	0	0	8
0	0	1	0	0	1	9
0	0	1	0	1	0	10
0	0	1	0	1	1	11
0	0	1	1	0	0	12
0	0	1	1	0	1	13
0	0	1	1	1	0	14
0	0	1	1	1	1	15
0	1	0	0	0	0	16
0	1	0	0	0	1	17
0	1	0	0	1	0	18
0	1	0	0	1	1	19
0	1	0	1	0	0	20
0	1	0	1	0	1	21
0	1	0	1	1	0	22
0	1	0	1	1	1	23
0	1	1	0	0	0	24
0	1	1	0	0	1	25
0	1	1	0	1	0	26
0	1	1	0	1	1	27
0	1	1	1	0	0	28
0	1	1	1	0	1	29
0	1	1	1	1	0	30
0	1	1	1	1	1	31
1	0	0	0	0	0	32
1	0	0	0	0	1	33
1	0	0	0	1	0	34
1	0	0	0	1	1	35
1	0	0	1	0	0	36
1	0	0	1	0	1	37
1	0	0	1	1	0	38
1	0	0	1	1	1	39
1	0	1	0	0	0	40
1	0	1	0	0	1	41
1	0	1	0	1	0	42
1	0	1	0	1	1	43

Fig. 11. Tabla representación binaria y decimal de los números

Posteriormente se realizó un testbench para contadores de 2, 4 y 6 bits para comprobar si los datos simulados corresponden a los teóricos, los resultados de este se encuentran en la Figura 12.

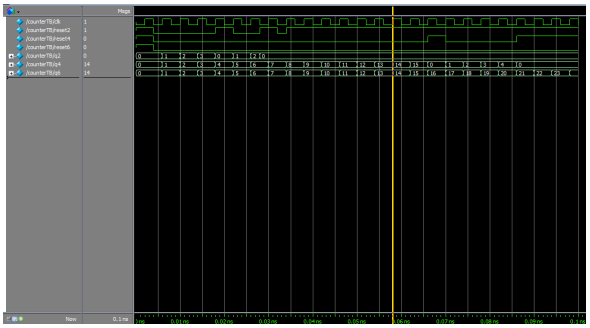

Fig. 12. Resultados obtenidos en el testbench

B. Análisis de Resultados

Al igual que en los problemas anteriores se realizó un testbench para comprobar el correcto funcionamiento de las entradas y salidas de los contadores de 2, 4 y 6 bits, como se dislumbra en la Figura 12.

En la Figura 13 se observa como al realizar el primer conteo los datos mostrados por la salidas de los contadores de 2, 4 y 6 bits corresponden con los previstos en la tabla de la Figura 11.

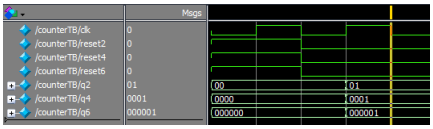

Fig. 13. Resultados obtenidos en el testbench

De la misma manera, en la Figura 14 al cambiar la salida a números decimales como se observa en la Figura 15 y contrastarlos con la tabla de la Figura 11 se comprueba el correcto funcionamiento de los contadores de 4 y bits, por lo que el desarrollo del contador parametrizable de n bits es correcto.

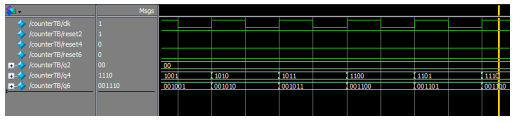

Fig. 14. Resultados obtenidos en el testbench

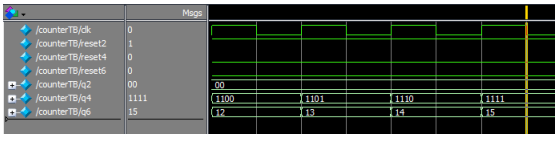

Fig. 15. Resultados obtenidos en el testbench

V. Conclusiones

Para desarrollar soluciones debemos seguir una serie de pasos definidos para obtener resultados esperados, por lo general el inicio de un proyecto esta basado en la creación de un diseño o modelo que pueda representar el comportamiento que tendrá en un futuro la aplicación que estamos diseñando. En este caso para los problemas 1 y 2, se utilizó el modelo de comportamiento, para determinar la solución óptima que se adapte mejor a nuestras necesidades, mediante la creación de una tabla de verdad y la utilización de Mapas de Karnaugh.

Seguido de este paso, se procedió a realizar la solución que sigue los lineamientos de diseño, para esta fase, se tenia una idea del funcionamiento general de ambos programas, sin embargo, no fue hasta que se creo el ”prototipo” de la aplicación que realmente se obtuvo lo deseado.

Por ende el utilizar las metodologías y herramientas de diseño, nos ayudan a utilizar eficientemente el tiempo, y por ende aumentar las ganancias que podamos percibir de nuestros proyectos.

Realizar un testbench de auto-chequeo es de vital importancia para comprobar que el diseño implementado funciona correctamente o verificar errores y solucionarlos antes de realizar su conexión a la FPGA, permitiendo que encontrar dicho fallo sea menos engorroso.

La FPGA es una herramienta útil en el diseño de módulos de hardware para circuitos simples, y para poder probar su viabilidad, y correcto funcionamiento, ya que el probarlos mediante un self-check sin duda alguna es importante, pero el poder comprobar su funcionamiento a nivel de hardware utilizando la FPGA representa mejor la importancia de los lenguajes de descripción de hardware y su infinidad de aplicaciones posibles mediante el uso de estas dos herramientas.

References

[1] D. Harris and S. Harris, *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[2] J. López. (2020, Sep.) Qué es un FPGA: características y utilidad de este tipo de componente. Hardzone. [Online]. Available: https://hardzone.es/reportajes/que-es/fpga-caracteristicas-utilidad/

[3] M. C. Sisterna. FIELD PROGRAMMABLE GATE ARRAYS(FPGAS). Departamento de Electrónica y Automática. [Online]. Available: http://dea.unsj.edu.ar/sisdig2/FieldProgrammableGateArrays_A.pdf

[4] HardwareBee. FPGA Advantages and Most Common Applications. HardwareBee. [Online]. Available: https://hardwarebee.com/fpga-advantages-common-applications-today/

[5] K. Freund. (2018, Jul.) An Update On Data Center FPGAs. Forbes. [Online]. Available: https://www.forbes.com/sites/moorinsights/2018/07/20/an-update-on-data-center-fpgas/?sh=4d0f53af46c2

[6] M. D. P. Emilio. (2017, Aug.) Maurizio Di Paolo Emilio. Electronic Design. [Online]. Available: https://www.electronicdesign.com/markets/automotive/article/21805452/use-fpgas-to-streamline-automotive-electronics