Universidad Nacional Autónoma de México

Facultad de Ingeniería

Reingeniería de sistema electrónico de pruebas basado en FPGA

Informe de actividades profesionales

que para obtener el título de

Ingeniero Eléctrico Electrónico

presenta

Sergio Andrés Cortés Torres

Asesor de informe

MI. Ricardo Mota Marzano

Ciudad Universitaria, Cd. Mx., 2021

Índice

Contenido **Página**

Agradecimientos

La vida es hermosa y es porque la compartimos con quienes queremos. El reconocer y agradecer el apoyo de uno o más al alcance de tus metas, permite ser una gran persona. El desarrollo del informe y el proyecto no fueron fáciles, sufrí, pero también disfruté del proceso. Todo ha valido la pena y lo seguirá siendo.

Son los principales promotores de mis sueños, gracias por confiar y creer en mí y en mis expectativas. Gracias, madre, por esas intensas tardes de estudio que a pesar de ser muy estrictas me ayudaron a formar mi método para estudiar. Gracias por las veces que tuviste que anteponer mis juntas escolares para ver como seguía, por ir a mis clases extracurriculares de deportes y arte a las que seguramente en varias ocasiones dormitabas. Gracias, padre, por esas palabras de apoyo de seguir adelante sin importar como se torne el momento; gracias por creer en mí más de lo que llegué a creer en mí mismo por mucho tiempo y gracias por el apoyo, que, aunque no fue físico hacías presencia en cada momento decisivo.

Puede que ya no estés aquí, aun así, te agradezco por todo tu apoyo. Me ayudaste a creer en mí mismo, a subir mis expectativas de vida y a no darme por vencido. Fuiste la felicidad y estabilidad que no se encuentra a menudo. Gracias por ayudarme a ver lo que el mundo tiene para mí, un lugar por el cual estoy dispuesto a alcanzar. Prometo ser la mejor versión de mí y estar ahí cuando me necesites.

Algunos llegaron desde la infancia, de los cuales no recordaba pero con los que ahora tengo fuertes lazos; otros más de la preparatoria con quienes disfrutaba de una reta de basket o halo; a mis grandes amigos de la universidad y reuniones recreativas, con los que cualquier escusa era buena para ir por un cafe o ir a rodar en moto; las tardes con los amigos que me enseñaron que la vida es un gran baile y el mundo es un salón. Todos ustedes me ayudaron en laguna parte de mi vida. Por ello les dedico estas pequeñas líneas. Quiero que sepan que estaré para vosotros en cualquier momento que me necesiten, así como estuvieron para mí.

Estimado docente, muchas gracias por su apoyo y confianza a lo largo del desarrollo de este informe, sé que no lo hubiera podido haber hecho solo. Usted fue uno de mis mejores profesores durante toda la carrera.

Ha sido un placer haber aprendido bajo tu techo desde Iniciación Universitaria, gracias por haberme permitido ser parte de tu grandiosa historia. Eres una de las responsables de que pueda culminar con este gran paso de mi vida.

Este es un momento muy especial que espero, perdure en el tiempo. Gracias a la vida por este nuevo triunfo y a todas las personas que me apoyaron y creyeron en la realización de esto.

1. Introducción

"En la era industrial el cambio era más lento. Aquello que usted aprendió en la escuela era valioso por un periodo más largo. En la era de la información, lo que usted aprende se vuelve obsoleto muy rápidamente. Lo que usted aprendió es importante, pero no tanto la rapidez con la que puede usted aprender, cambiar y adaptarse a la nueva información."[0]

Tal como lo dijo el autor de esta cita, en esta nueva era lo que uno aprende no es suficiente para estar al día, lo mismo ha pasado con la tecnología; esta ha evolucionado bastante y por ello las empresas tienen que estar a la vanguardia. Para ello tienen que poseer infraestructura de punta, actualizandose constantemente.

Para muchas organizaciones tener un equipo que diseñe circuitos en PCB y otro que diseñe su Firmware puede ser costoso y un largo tiempo de realización. Es por eso que el lenguaje de descripción de hardware (HDL) empezó a desarrollarse en 1980 por parte del Departamento de Defensa de Estados Unidos, el cual inicio un proyecto denominado Very High Speed Integrated Circuit (VHSIC); donde se especificaba que el lenguaje (VHDL) debería ser un lenguaje para diseño y descripción de hardware.

En el año 1984, una empresa llamada Xilinx introdujo el primer FPGA (tipo de circuito integrado que puede ser configurado para diferentes algoritmos después de su fabricación). Antes de que éstos existieran, se usaban docenas de circuitos integrados discretos en una placa de circuito, o incluso cientos de circuitos integrados en múltiples placas de circuito, para lograr la funcionalidad de hardware que se logra hoy en día con un FPGA.

En este informe que a continuación se presenta pretendo explicar cada etapa para cambiar la configuración del equipo para la prueba (Setup del test). Desde el aprendizaje de la función del *test* para comprender el problema, los antecedentes para darle solución y problemas con los que tuve que lidiar en el camino. Para este proyecto elegí usar tecnología FPGA por la facilidad de desarrollo y amplio uso en el futuro. Además, como lo mencioné una de sus grandes ventajas contra la programación en un microcontrolador es que elimina la necesidad de diseñar un pcb con características específicas de cada código, sus respectivas líneas de programación y alterar un circuito para el que no está diseñado (en términos de hardware).

2. Objetivo

Diseñar un sistema electrónico digital que genere los códigos de operación de un dispositivo comercial "abre puertas" de una empresa transnacional, con la finalidad de realizar pruebas de sensibilidad y selectividad de forma dinámica por medio de tecnología FPGA, sustituyendo al equipo de características fijas que actualmente se utiliza.

[0] R. Kiyosaki, 2013

3. Descripción de la empresa

La empresa fabrica abridores (soluciones de acceso) para puertas de garajes, operadores de puertas comerciales, bodegas, conjuntos residenciales, estacionamientos, sistemas de entrada de portones y otros lugares donde las personas guardan lo que más valoran.

4. Marco teórico

4.1. Sistema Ternario

También conocido como sistema numérico trinario, tiene dos tipos de representación:

1. **Balanceado,** el cual una tercia toma los valores: *-1, 0 y 1*. En este sistema balanceado en lugar de usar los valores *-1 y 1*, se emplean caracteres como *T, 0 e I* ó *-, 0 y +*.
2. **Desbalanceado,** el cual una tercia toma los valores: *0, 1 y 2*.

Cuando se habla de una base 3 por lo general se toma en cuenta el sistema Desbalanceado.

4.1.1. Conversión de base 3 a base 10

Cuando se convierte cualquier base N a base decimal, el dígito que se encuentra a la derecha es siempre el valor menos significativo y el dígito que se encuentra a la izquierda es, el valor más significativo. Una vez explicado eso, se multiplica el MSD o MSB (Dígito más significativo o Bit más significativo) por la base y el producto se suma al valor del dígito siguiente. El resutado se multiplica de nuevo por la base y el producto se suma al dígito siguente; y así sucesivamente hasta llegar al LSD o LSB (Dígito menos significativo o Bit menos significativo), de modo que el resultado de todas las operaciones es el número equivalente decimal.

Para que quede más claro se transformará un número de base 2 a base 10 y otro de base 3 a base 10.

Conversión de $(1101)_2$ a base 10:

$$(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$(1101)_2 = 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$$
$$(1101)_2 = 8 + 4 + 0 + 1$$
$$(1101)_2 = (13)_{10}$$

Conversión de $(2101)_3$ a base 10:

$$(2101)_3 = 2 \times 3^3 + 1 \times 3^2 + 0 \times 3^1 + 1 \times 3^0$$
$$(2101)_3 = 2 \times 27 + 1 \times 9 + 0 \times 3 + 1 \times 1$$
$$(2101)_3 = 54 + 9 + 0 + 1$$
$$(2101)_3 = (64)_{10}$$

4.1.2. Conversión de base 10 a base 3

Cuando se convierte de base decimal a cualquier base N se divide sucesivamente el número decimal entre la base deseada hasta que la parte entera del cociente sea 0. El número equivalente se forma agrupando todos los residuos contemplando que el último dígito (residuo) se convierte en el MSD y el primer dígito en el LSD, de modo que el resultado se lee de izquierda a derecha.

Para que quede más claro se transformará el mismo número de base 10 a base 2 y de base 10 a base 3.

Conversión de $(20)_{10}$ a base 2

20	**2**
10	0
5	0
2	1
1	0
0	1

$(20)_{10} = (10100)_2$

Conversión de $(20)_{10}$ a base 3

20	**3**
6	2
2	0
0	2
-	-
-	-

$(20)_{10} = (202)_3$

4.2. Circuitos secuenciales

Un circuito describe lógica secuencial cuando la salida depende de la entrada actual y de las entradas anteriores, o dicho de otra forma, la salida depende de la entrada y del estado del sistema. Esto introduce un nuevo elemento dentro del sistema que será la memoria, por tanto, cualquier sistema que tenga al menos una señal que ante el cambio de unas señales cambie, pero que pueda ocurrir que ante el cambio de las mismas señales conserve su valor, entonces se tratará de un sistema secuencial ya que dicha señal es un elemento de memoria. Esto nos da una pista de si un circuito será secuencial y se realizará por tanto a partir de elementos de memoria como puedan ser cerrojos o registros.

El procedimiento de diseño de los circuitos secuenciales se basa en la aplicación sucesiva de las siguientes etapas:

1. Definición del sistema y diagrama de flujo. A partir de las especificaciones de diseño se define el esquema de entradas y salidas del circuito, así como los estados de los que debe constar y las transiciones que tiene que haber entre ellos. Esto conduce al planteamiento del DIAGRAMA DE FLUJO DEL SISTEMA.

2. Tabla de fases inicial. En esta etapa se indica, para cada estado en el que se encuentra el sistema y cada combinación posible de entradas, cuál es el estado siguente y qué salidas le corresponde.

3. Simplificación de estados equivalentes. Se debe comprobar que no existan redundancias en los estados considerados. Para ello se comprueba si existen equivalencias entre ellos[1]. Se usa una TABLA DE IMPLICACIÓN para saber el mínimo número de estados para representar las especificaciones de diseño.

4. Codificación de estados y tabla de transiciones. La asignación de las combinaciones de bits a cada estado es arbitraria (libertad total para hacer combinaciones binarias), pero se recomienda seguir el orden del binario natural.

   Una vez hecha la codificación, se plantea la TABLA DE TRANSICIONES. Con todas las combinaciones posibles de bits de estado y entrada (estados actuales) "Q^n"; se indica cuáles son sus salidas y los correspondientes estados siguientes "Q^{n+1}".

5. Ecuaciones de salida. A partir de la tabla de transiciones, se plantea el DIAGRAMA DE KARNAUGH para hallar las expresiones lógicas de las salidas del circuito.

6. Ecuaciones de exitación. Esta etapa depende del tipo de flip-flop (de acuerdo con la tabla de funcionamiento de cada uno) que se vayan a utilizar en el diseño. De igual forma se plantea el diagrama de Karnaugh para obtener las expresiones lógicas de las excitaciones del circuito.

7. Implementación física. Una vez teniendo las ecuaciones de salida y excitación, se procede a llevar dichas expresiones lógicas a diagramas lógicos.

8. Inicialización. Una vez hecha la implementación física, hay que inicializar los flip-flops a través de sus entradas asíncronas de Set o Reset.

4.2.1. Máquina de estados

Una máquina de estados está definida por dos funciones, una calcula el estado siguiente en que se encontrará el sistema, y la otra calcula la salida. El estado siguiente se calcula, en general, en función de las entradas y del estado presente. La salida se calcula como una función del estado presente y las entradas.

Normalmente hay dos tipos de máquinas de estados; unas son las de **Mealy** donde las salidas son determinadas por el estado presente y las entradas presentes, es decir las salidas

[1] Dos estados son equivalentes si tienen la misma salida y llevan, para cada combinación de entradas, a los mismos estados siguientes o a estados siguientes equivalentes.

se producen dependiendo de cómo la máquina realiza una transición de un estado a otro. Las otras son las de **Moore** donde las salidas son independientes de las entradas, es decir, sus salidas sólo dependen del estado de la máquina.

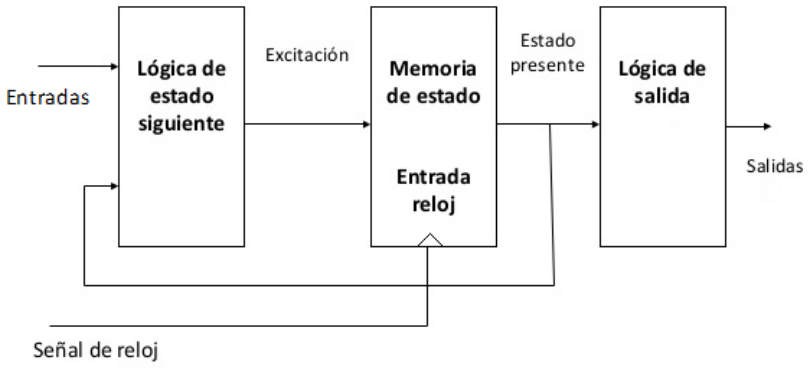

Figura 1. Máquina de estados tipo Moore.

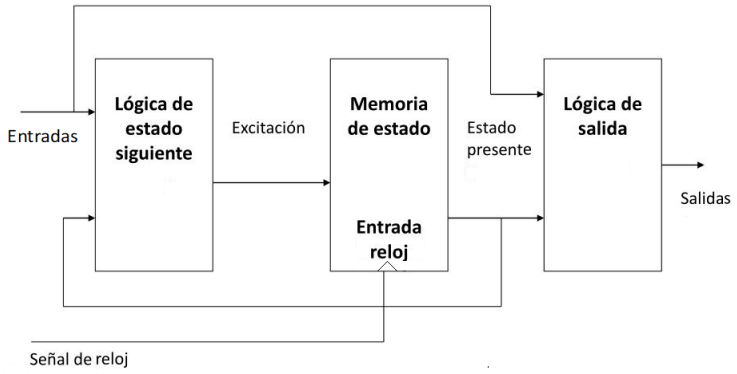

Figura 2. Máquina de estados tipo Mealy.

4.3. Microprocesador

"Un microprocesador es un circuito integrado que contiene un procesador digital completo, debe incluir como mínimo una unidad lógica aritmética y registros asociados"[2]. Esto no es suficiente para aplicaciones prácticas, el microprocesador necesita de una memoria para poder almacenar información y de periféricos para poder comunicarse con el mundo exterior. "Los microcontroladores están integrados por un procesador, una memoria y puertos de entrada y salida en un circuito integrado monolítico"[3]. Al ser dispositivos a los cuales se les puede programar una lógica, tener la capacidad de actuar como una máquina de estados, la miniaturización a la que durante décadas han sido sujetos y la dramática reducción de precio, los

[2]Davies, 2008
[3]Valvano, 2013

han colocado en casi todos los aparatos electrónicos que usamos hoy en día: computadoras personales, licuadoras, hornos de microondas, coches, tarjetas de identificación de acceso, el control remoto de la televisión, multímetros digitales, sistemas de riego caseros, teléfonos móviles, el sistema de piloto automático de un avión y sistemas de control de accesos.

4.4. FPGA

Antes de que éstos existieran, se usaban docenas de circuitos integrados discretos en una placa de circuito, o incluso cientos de circuitos integrados en múltiples placas de circuito, para lograr la funcionalidad de hardware que se puede lograr hoy con un dispositivo FPGA. Xilinx introdujo el primer FPGA en 1984, aunque no se llamaría así sino hasta que Actel los popularizara en 1988. El FPGA surgió como una evolución del CPLD (Complex Programmable Logic Device). Tanto los CPLD como los FPGA contienen un gran número de elementos lógicos programables. Podríamos decir que en un CPLD hallaríamos del orden de decenas de miles de puertas lógicas equivalentes y en un FPGA del orden de cientos de miles hasta millones de ellas.

Un FPGA (Field Programmable Gate Array) es un tipo de circuito integrado que puede ser programado para diferentes algoritmos después de su fabricación. Los actuales FPGA´s consisten en más de dos millones de celdas lógicas que pueden ser configuradas para implententar en hardware una gran variedad de algoritmos comúnmente programados en software. Aunque el flujo tradicional del diseño de FPGA es más parecido a un ASIC (Circuito Integrado de Aplicación Específica) que a un procesador; un FPGA proporciona importantes ventajas de costo en comparación con el diseño, pruebas de desempeño, rediseño y fabricación de un ASIC, ofreciendo rendimientos aceptables en la mayoría de los casos. Otra ventaja significativa entre el ASIC y el FPGA es la capacidad de reconfiguración dinámica. Este proceso, que es el mismo que cargar un programa en un procesador, puede afectar parte o totalidad de recursos disponibles en la fabricación de FPGA *(Xilinx)*.

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

1. Look up table (LUT). Las tablas de búsqueda con estructuras lógicas que almacenan valores de salida de funciones lógicas.

2. Flip Flop (FF). Este elemento de registro almacena los resultados de la LUT.

3. Interconexiones a nivel semiconductor. Estos elementos conectan elementos con otros más.

4. Entradas/Salidas. Estos puertos físicos están disponibles para intercambiar datos dentro y fuera del FPGA.

La combinación de estos elementos resulta en la arquitectura básica del FPGA. Aunque esta estructura es suficiente para la implementación de cualquier algoritmo, la eficiencia de la implementación resultante está limitada en términos de rendimieto computacional, requieren recursos y una frecuencia de reloj alcanzable.

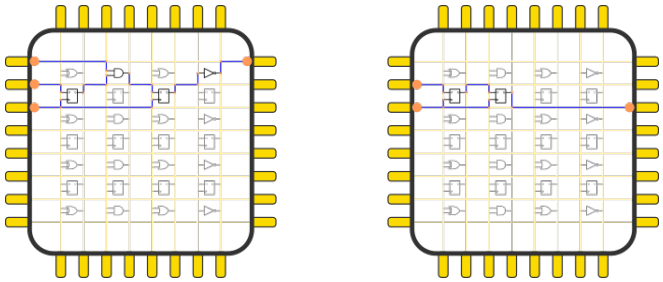

Figura 3. Estructura interna de un FPGA.

4.4.1. LUT

La LUT es el bloque de construcción básico de un FPGA 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 entradas implementan diferentes funciones para producir valores en la salida. El límite en el tamaño de la tabla de verdad es N, donde N representa el número de entradas a la LUT. Para las N-entradas de LUT, el número de localidades de memoria accedidas por la tabla es 2^N el cual permite a la tabla implementar la siguiente cantidad de funciones 2^{N^N}.

La implementación de hardware de una LUT puede considerarse como una colección de celdas de memoria conectadas a un conjunto de multiplexores. Las entradas a la LUT actúan como bits de selectores en el multiplexor para escoger el resultado dado en tiempo dado. Es importante mantener esta representación en mente, porque una LUT puede ser usada como un motor de cálculo de funciones y como un elemento para almacenar datos. La Figura 4 muestra esta representación funcional de la LUT.

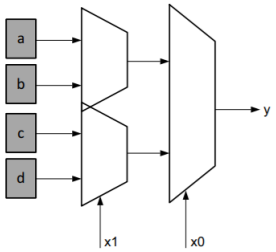

Figura 4. Representación de una LUT como una colección de celdas de memoria.

4.4.2. Flip-Flop

El flip flop es la unidad de almacenamiento básico dentro del FPGA. Este elemento siempre está en conjunto con una LUT para ayudar en la canalización lógica y almacenamiento de datos. La estructura básica de un flip-flop incluye una entrada de datos, entrada de reloj, habilitación de reloj, reinicio y salida de datos. Durante la operación normal, cualquier valor en el puerto de la entrada de datos es lanzado y pasado a la salida en cada pulso de reloj. El propósito del pin que habilita el reloj es permitir al flip-flop sostener un valor específico más de un pulso de reloj. Los nuevos datos de entrada sólo son lanzados y pasados al puerto de salida de datos cuando el borde de disparo de la señal de reloj conincide con un nivel en alto presente en la terminal de habilitación.

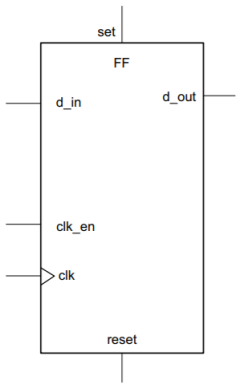

Figura 5. Estructura de un Flip-Flop.

4.5. HDL

A partir del desarrollo de circuitos integrados digitales programables con una gran cantidad de componentes lógicos y la necesidad de sistemas digitales para aplicaciones mas complejas, las herramientas de diseño tradicionales se vuelven cada vez más ineficientes y poco efectivas para lograr desarrollos adecuados, por lo tanto las empresas fabricantes de circuitos integrados desarrollaron los HDL o Lenguajes de Descripción de hardware.

Considerando que un **HDL** es una herramienta para describir la estructura y comportamiento de un sistema para una adecuada especificación, documentación y simulación antes de su realización real; se establecieron ciertas características fundamentales, los cuales son:

1. Cada elemento de diseño tiene una interfaz única y perfectamente definida, que permite conectarla a otros elementos.

2. Cada elemento tiene un comportamiento preciso y unívocamente definido, que permiten su posterior simulación.

3. La especificación de comportamiento que permite definir la operatividad puede realizarse a través de un algoritmo o de una estructura de hardware real.

4. Los diseños mantienen una estructura jerárquica, que permite descomponerlo adecuadamente.

5. Las características concurrentes, temporizadas y de sincronismo (por ej. reloj) pueden ser modeladas adecuadamente.

6. Se puede simular cualquier operación lógica y de temporización.

Existen muchas maneras de crear un HDL, y por ello el problema más importante es la normalización de estos. Actualmente los HDL de uso más generalizado son VHDL y Verilog, estandarizados por la **IEEE**.

4.5.1. VHDL

El lenguaje de descripción hardware empezó a desarrollarse en 1980 por parte del Departamento de Defensa de Estados Unidos (USDOD), el cual inicio un proyecto denominado Very High Speed Integrated Circuit (VHSIC), siendo el principal objetivo: desarrollar circuitos integrados en tecnología de 0,5 micras con muy altas prestaciones y resistencia a la radiación. Su nombre viene de VHSIC HDL (Very-High-Speed Integrated circuits Hardware Description Language).

En este proyecto participaron tres compañías: Intermetrics, Texas Instruments e IBM. Este proyecto comenzó a partir del documento que ofreció el USDOD, donde se especificaba que el VHDL debería ser un lenguaje para diseño y descripción de hardware. VHDL resulta ser un lenguaje muy sencillo a la vez que muy poderoso. Con él se pueden generar una gran variedad de descripciones de hardware que resultan en arquitecturas totalmente personalizadas que den respuesta a los requerimientos de diseño.

4.5.2. Verilog

Verilog es un lenguaje de descripción de hardware cuya finalidad es modelar sistemas electrónicos. Este lenguaje soporta el diseño, prueba e implementación de circuitos tanto analógicos, como digitales, así como, de señal mixta a diferentes niveles de abstracción. fue inventado por Phil Moorby en 1985, mientras trabajaba en Automated Integrated Design Systems, más tarde renombrada Gateway Design Automation. El objetivo de la creación de un nuevo lenguaje como Verilog era ser un lenguaje de modelado de hardware.

Cuando se diseñó Verilog se buscaba hallar un lenguaje con una sintaxis similar a la del lenguaje de programación C, de forma que resultara familiar a los ingenieros, siendo así todo más fácil cuando tuvieran que empezar a trabajar con ella y fuera rápidamente aceptada. Por lo tanto el lenguaje tiene un preprocesador muy similar al que encontramos en el lenguaje C,

y por lo tanto la mayoría de las palabras reservadas son muy parecidas, además el mecanismo de formateo en las rutinas de impresión y en los operadores del lenguaje son también similares.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity AND is
port (a,b: in std_logic;
z: out std_logic);
end AND;

architecture AND_F of AND is
begin
Process (a,b)
begin
z<= a and b;
end Process;
end AND_F;
```

(a) Código en VHDL.

```
module AND(
input a,b,
output z
);

begin
assign z=a&b;
end
end module
```

(b) Código en Verilog.

Véase la diferencia en codificación entre los dos lenguajes de descripción hardware que utilizan herramientas de síntesis: VHDL y Verilog; con los ejemplos de la puerta AND entre la codificada en VHDL y la codificada en Verilog.

4.6. Programación vs Descripción

Estos lenguajes se diferencian del lenguaje de programación C por ser un lenguaje paralelo no secuencial. Un programa realizado en VHDL puede parecer como un programa secuencial, pero hay que recordar que todo está funcionando a la vez, es decir, que se ejecuta en paralelo.

En el lenguaje VHDL encontramos comandos que poseen una mayor complejidad como son loops while, if, if-else, etc. Una diferencia que encontramos entre los lenguajes C y VHDL es que cuando se define un bucle en C el código se ejecuta una y otra vez, sin embargo, en VHDL se generan múltiples bloques idénticos de lógica, donde todos ellos procesan los datos en un ciclo de reloj (si es el caso puede ser más de un ciclo de reloj). Por ello, a la hora de utilizar el VHDL para síntesis hay que tener mucho cuidado en la codificación para que no se generen latches indeseados.

4.6.1. Ejecución de un programa en un procesador

En comparación con las arquitecturas de procesador, las estructuras que comprenden el tejido FPGA permiten un alto grado de paralelismo en la ejecución de la aplicación. La arquitectura de procesamiento personalizada generada por el compilador Vivado HLS para un programa

de software presenta un paradigma de ejecución diferente, que debe tenerse en cuenta al decidir portar una aplicación desde un procesador a un FPGA. Para examinar los beneficios del paradigma de ejecución de FPGA, esta sección proporciona una breve revisión de la ejecución del programa del procesador.

Un procesador, independientemente de su tipo, ejecuta un programa como una secuencia de instrucciones que se traduce en cálculos útiles para la aplicación de software. Esta secuencia de instrucciones son generadas por herramientas del compilador del procesador, como la colección del compilador GNU (GCC), que transforma un algoritmo expresado in C/C++ en lenguaje ensamblador (nativa del procesador). El trabajo de un compilador de procesador es tomar una función de C de la forma: $z = a + b$ y lo transforma en código ensamblador como:

$$ADD \text{ \$R1, \$R2, \$R3}$$

El código ensamblador de arriba define la operación de suma para calcular z en términos de los registros internos de un procesador. El código establece que los valores de entrada del cálculo son almacenados en los registros R1 y R2, y el resultado de la operación es almacenado en el registro R3. El código anterior no expresa todas las instrucciones necesarias para el cálculo del valor z. Este código sólo maneja la operación después que los datos han llegado al procesador. Por lo tanto, el compilador debe crear instrucciones adicionales del código ensamblador para cargar los registros del procesador con datos de la memoria central y escribir el resultado a la memoria. El código completo para el cálculo de z se encuentra debajo.

```
LD   a, $R1
LD   b, $R2
ADD  $R1, $R2, $R3
ST   $R3, c
```

El código muestra que a pesar de ser una simple operación de suma entre dos valores, resulta en múltiples instrucciones de ensamblador. La latencia computacional de cada instrucción no es igual en todos los tipos de instrucciones. Por ejemplo, dependiendo de la localidad de a y b, las operaciones *LD* toman diferentes números de ciclos de reloj para ser completadas. Si los valores están en el caché del procesador, estas operaciones de carga se completan dentro de pocas decenas de ciclos de reloj. Si los valores están en la memoria principal de velocidad de datos doble (DDR), las operaciones tardan entre cientos y miles de ciclos de reloj para completarse. Si los valores están en un disco duro, las operaciones de carga tardan más en completarse. Ésta es la razón por la cual los ingenieros de software con acceso a caché tardan mucho tiempo reestructurando sus algoritmos para aumentar el espacio de localidad de datos en memoria para aumentar la tasa de aciertos de caché y disminuir el tiempo de procesador por instrucción.

4.6.2. Ejecución de un programa en un FPGA

El FPGA es un tejido de procesamiento inherentemente paralelo capaz de implementar cualquier función lógica y aritmética que pueda ejecutarse en un procesador. La principal diferencia es que el compilador del FPGA es usado para transformar descripciones de software en RTL (Register Transfer Level), no se ve obstaculizado por las restricciones de un caché y espacio de memoria unificado. El cálculo de z es compilado en varias LUTs requeridas para lograr el tamaño del operando de salida. Por ejemplo, asuma que en el programa del software original las variables a, b y z son definidas con datos de tipo *short*. Este tipo, que define un contenedor de datos de 16 bits, es implementado como 16 LUTs[4].

Las LUTs usadas para el cálculo de z son exclusivas a esta operación. A diferencia de un procesador, donde todas las operaciones comparten la misma ALU, una implementación de FPGA crea instancias de conjuntos independientes de LUTs para cada operación en el algoritmo de software.

Además de asignar una única LUT como recurso por cálculo, el FPGA difiere de un procesador en la arquitectura de memoria y el costo de accesos a la memoria. En una implementación de FPGA, el compilador organiza memorias dentro de múltiples bancos de almacenamiento lo más cerca posible al punto de uso en la operación. Esto resulta en un ancho de banda instantáneo en memoria, que por mucho excede las capacidades de un procesador.

Con respecto al rendimiento computacional y ancho de banda de memoria, el compilador ejecuta las capacidades del FPGA a través de los procesos de programación, canalización y flujo de datos.

El proceso de programación es el proceso de identificación de datos y control dependientes entre diferentes operaciones para determinar cuando se ejecutará cada uno. Esto permite al compilador agrupar operaciones en el mismo ciclo de reloj y ajustar el hardware que permita la superposición de funciones llamadas.

La canalización es una técnica de diseño digital que permite al diseñador evitar la dependencia de datos e incrementar el nivel de paralelismo en la implementación de un algoritmo de hardware.

El flujo de datos es otra técnica de diseño digital, que es similar a la canalización. El objetivo del flujo de datos es expresar el paralemismo a un nivel de **coarse-grain**. En términos de ejecución de software, esta transformación aplica a la ejecución paralela de funciones dentro de un solo programa.

[4]Como regla general, 1 LUT es equivalente a 1 bit de cálculo.

5. Antecedentes del proyecto

Una de las preocupaciones principales cuando se empieza un proyecto de rediseño es conservar la calidad y funcionalidad del producto. Sin embargo, en este proyecto se hace un rediseño para realizar una prueba, que a su vez también debe garantizar los mismo estándares. El *setup* actual ha dado muy buenos resultados de prueba ya que ha mostrado un muy buen desempeño. Por otra parte, se decidió modificar el *setup* para que no solo unos cuantos puedan realizar las pruebas, sino que sea más intuitivo y menos tardado.

Las pruebas realizadas a las unidades dentro de las fábrica son una parte crítica durante la aprobación de los GDO´s (Abridores de puerta de garaje). Dentro de las pruebas realizadas, unas indispensables son las de SENSIBILIDAD y SELECTIVIDAD. Si no se cubren y aprueban en su totalidad dichos tests, el producto queda expuesto a riesgos de no cumplir con las normas de **FCC** y certificación **UL** con la función de proteger lo que el usuario desea.

5.1. Sensibilidad

"Es la mínima magnitud en la señal de entrada requerida para producir una determinada magnitud en la señal de salida, dada una determinada relación señal/ruido, u otro criterio especificado."[5]

"Es la potencia mínima de activación necesario de un tablero (receptor) para reconocer los códigos enviados por el transmisor en una frecuencia deseada."[6]

5.2. Selectividad

"Es una medida del rendimiento de una radio receptora para responder sólo a la señal de radio que está sintonizado y rechazar otras señales cercanas en frecuencia. Normalmente se mide como un cociente en decibeles (dB)."[7]

"Es la capacidad del tablero (receptor) de detectar una única frecuencia, en este caso la mandada por el transmisor, y el de rechazar todas las demás."[6]

5.3. Diagrama de bloques y Setup de las pruebas a realizar

A continuación se presenta el diagrama de bloques que está conformado de la caja negra (black box), que es un pulsador normalmente abierto (clicker) modificado junto con su interfaz, el generador de señales y el tablero de prueba (también puede ser el GDO).

[5]Gilmore, 2003. Definición general del término.
[6]Definición específica para las pruebas en la empresa.
[7]Besser, 2003. Definición general del término.

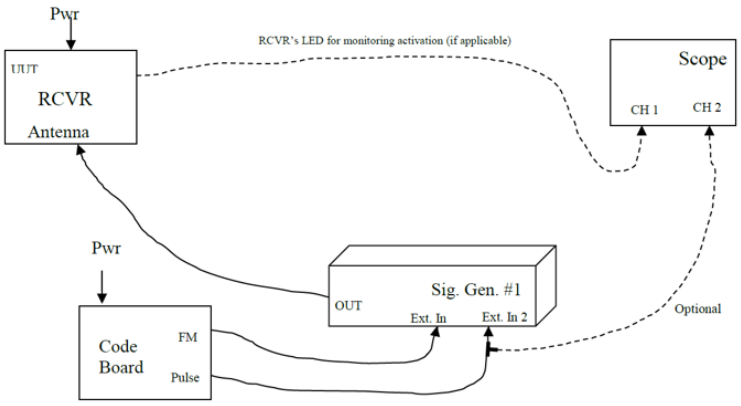

Figura 6. Diagrama de bloque de la prueba a realizar.

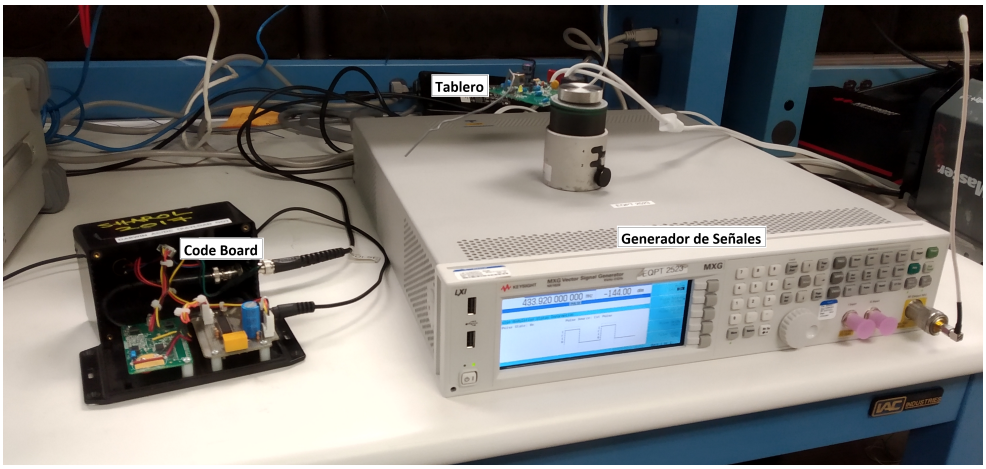

Figura 7. *Setup* Frankenstein para hacer pruebas de sensibilidad y selectividad.

El procedimiento para realizar la prueba de sensibilidad[8] es el siguiente:

Cuando uno presiona el pulsador del *Code Board* se mandará un código, dicho código entrará por la terminal Ext2 del generador de señales y este modulará el mensaje a una frecuencia dada y una potencia (mínima) para comunicarse con el tablero. Estos últimos dos parámetros (frecuencia y potencia) se varían para verificar que el tablero diseñando esté en buenas condiciones.

Para la prueba de selectividad[8] , el procedimiento es el siguiente:

Una vez conocida la potencia mínima a la cuál se transmite con éxito un código al tablero, se varían la potencia y la frecuencia, con el fin de saber cuáles son las frecuencias mínima y máxima en las cuales todavía hay comunicación transmisor-receptor.

[8]Estos procedimientos se aplican a todos los modelos de *GDO* y códigos de la empresa.

6. Definición del problema

Dentro de las pruebas para validar una unidad de la empresa, dos de las más importantes son sensibilidad y selectividad. Para ello se utiliza un *setup* construido por los ingenieros de esta empresa conformado por un pulsador obsoleto (en fabricación y venta), el cual su hardware es modificado. A su vez se programa un microprocesador con algunos de los códigos para activar los *GDO´s*. Para realizar las pruebas de sensibilidad de cada código es necesario reprogramar la memoria flash del microprocesador; proceso el cual es tardado y tedioso. Además que dicho setup cuenta con un gran tamaño y tiene un aspecto desordenado (Frankenstein). Cabe mencionar que el costo de fabricación de dicho setup es bastante caro.

Es por eso que surgió la necesidad de diseñar y construir un setup que reemplazara el actual. Que tuviera el mismo propósito pero ocupando un espacio más pequeño, que sea más fácil de manipular, que tenga una guía de usuario más completa y principalmente más barato y reproducible para pruebas en un futuro. Utilizando tecnología FPGA.

La opción que se propuso fue desarrollar el nuevo sistema bajo tecnología FPGA, aprovechando los conocimientos asimilados durante mis estudios en la Facultad de Ingeniería de la UNAM.

El *setup* debe demostrar que exista el mínimo retardo en el tiempo de respuesta entre la pulsación del FPGA y el *GDO*. Esto es necesario para que se pueda garantizar el test con seguridad y se pueda seguir con las demás pruebas. Si el tiempo de retardo es grande significará que a pesar de que el código es el adecuado, no proporciona la garantía del producto.

7. Metodología utilizada

La forma de trabajo **Agile** fue pensada en 2001 por un grupo de desarrolladores buscando mejorar su productividad y aumentar la calidad de sus productos. Es un proceso de administración de proyectos, es iterativo, exige colaboración entre los miembros del equipo, requiere de una comunicación constante y efectiva, además de estar constantemente evaluando los resultados. En proyectos tradicionales cada miembro del equipo desarrolla una parte y la entrega al siguiente, es revisado y liberado al final, tratando al problema por separado y sin una realimentación rápida que permita corregir errores que surjan en el camino o agregar nuevas funcionalidades. En cambio, cuando se trabaja con Agile se va haciendo un seguimiento constante del trabajo y se administra en pequeños incrementos. La planeación, los requerimientos y los resultados son flexibles y por ello es posible reaccionar a los cambios, sean los requerimientos, los tiempos o los recursos, con gran velocidad.

Una metodología que comparte los principios de realimentación y mejora continua es el uso de un **Kanban** (cartelera en japonés). Implementado por Toyota a final de la década

de los años 40s. Kanban copia el modelo de los supermercados donde lo que hay en los estantes es justo la demanda calculada de los consumidores, optimizando la cantidad de artículos disponibles y ventas, asegurando que lo que los clientes buscan siempre estará disponible. A esta forma de administración el inventario se le conoce como justo a tiempo (JIT por sus siglas en inglés Just In Time). Su uso en grupos de ingeniería aprovecha la idea central de la optimización de recursos y la comunicación entre las diferentes partes sobre las necesidades actuales o futuras del proyecto. El Kanban es la herramienta visual de esta metodología.

Los grupos que trabajan de manera Agile no están restringidos a una serie de pasos o etapas, pueden decidir cuáles son las que mejor se adaptan a su ritmo y forma de trabajo. Mientras que un Kanban básico sólo tiene 4 etapas (pendientes, en progreso, en revisión y terminado) en el desarrollo del proyecto se identificaron 5 etapas:

1. **Creación**. Se asignó al responsable del proyecto (único integrante), el rol total a desarrollar (setup, códigos, interfaz), y los tiempos aproximados de entrega.

2. **Definición**. Se establecieron claramente los objetivos del proyecto. La meta fue puesta y se comenzó a discutir cual sería la forma en que se llevarían a cabo los códigos y conexión entre los equipos. Los tiempos fueron puestos y la complejidad del proyecto fue analizada para ajustar los objetivos y metas a entregas realistas. Pensamos en casos especiales o situaciones en las que una reacción particular, es decir, diferentes condiciones a las de operación normal, debía existir.

3. **Desarrollo**. Empieza una etapa de aprendizaje sobre los códigos existentes a diseñar, cuanto tiempo es necesario para dominar estos y como será la implementación. Una visión general del sistema ya está definida y comienza a tomar forma. Los primeros códigos son diseñados y puestos a prueba para encontrar las oportunidades de mejora. Se empieza a trabajar continuamente con el nuevo código identificando diferencias y similitudes con el actual, puede ser que el resultado final no tenga elementos de esta primera forma, pero será la base sobre la cual se construyan las mejoras.

4. **Implementación**. Comienzan las pruebas funcionales del *setup* (código e interfaz) y se prueba la compatibilidad entre ellas. Los objetivos son evaluados para ver si ya han sido cumplidos y se les hacen los ajustes necesarios para alcanzar las metas puestas. Se revisan que cumplan casos especiales y las excepciones que puedan surgir durante las operaciones.

5. **Presentación**. Una presentación oral fue preparada con los detalles más relevantes del funcionamiento del *setup* en un alto nivel de abstracción. Se le dio prioridad a la forma en la que el proyecto fue presentado y lo que era más importante destacar. Además, se preparó una demostración funcional del sistema.

8. Participación profesional

Inicialmente mi tarea consistió en entender y conocer las bases de los códigos de la empresa y replicar dos de ellos; así como los procesos de producción que están involucrados en la construcción de este módulo. Para ello fue necesario que aprendiera a realizar las siguientes acciones.

8.1. Familiarización con la gestion de la empresa

SAP se utiliza principalmente para la gestión de materiales en el proceso de manufactura. Esto implica una descripción de todos los materiales que conforman una unidad final. Esto se especifica en un listado de materiales (LM). Además, se cuenta con documentos (Instrucciones de Manufactura) que especifican claramente cómo deben ser los procesos de manufactura. Por otra parte, SAP sirve a distintas áreas además de ingeniería, como lo es compras y calidad. Compras utiliza este sistema para costear todas las partes. Calidad tiene que asegurarse que efectivamente se construya un producto de acuerdo con la LM.

En **Windchill** se encuentra toda la información de la planta: procesos, dibujos, diseños, sistemas y partes utilizados en la planta. Además, tiene la función de gestionar los cambios de ingeniería.

Esto fue muy importante porque se pudo basar en la documentación que aparece en Windchill para poder adaptar el proyecto al objetivo. Y se pudo usar SAP para conocer el *status* del material para realizar la interfaz.

8.2. Familiarización con el código

Se empezó por definir las limitaciones impuestas al diseño y modelando el comportamiento del sistema en un nivel alto de abstracción. Se decidió dividir en dos partes el proyecto. La primera fue replicar los códigos en un FPGA y la segunda realizar la correcta comunicación entre el FPGA y el Generador de Señales por medio de una Interfaz.

8.2.1. Código A

El código A [9] también llamado Billion Code está basado en el sistema trinario representado en pares de bits **BP** (*Bit pair*) que consta de un mensaje compuesto de cuatro bloques. El primer bloque contiene la primera sección de información **FF** (*First Frame*) [10] que dura 40 ms, con 10 dígitos; cada uno contiene 4 bits (dos BP) representando al sistema trinario.

[9] Es indispensable saber que los frames de este código siempre serán los mismos en todas las transmisiones.
[10] Tramo de datos binarios con longitud variable.

El segundo y cuarto bloque no tiene información alguna, es un simple espacio en blanco (**BT**) que permite al emisor leer la información a tiempo y correctamente. Pero cabe destacar que el primer BT tiene una duración de 37 ms, mientras que el cuarto BT 39 ms.

El tercer bloque contiene la segunda sección de información **SF** (*Second Frame*) que dura 40 ms al igual que el FF, y cada dígito contiene 4 bits (dos BP) representando al sistema trinario. Hay que considerar que ambos *frames* son diferentes a pesar de que duran lo mismo, cada *frame* contiene información específica que le permite al receptor saber cuál transmisor es el que quiere interactuar con él. Solo manda dos mensajes por cada pulsación, pero si se deja presionado aumentan los mensajes.

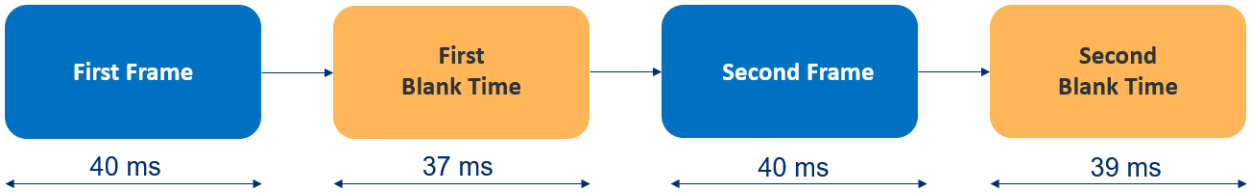

Figura 8. Estructura del Código A.

8.2.2. Código E/F

El código E [11] también llamado Rolling Code está basado en el sistema trinario (al igual que el código A) representado en BP que consta de un mensaje compuesto de cuatro bloques. El primer bloque contiene el FF que dura 31 ms y cada dígito contiene 2 bits (un BP) representando al sistema trinario. Además de eso, arroja cuatro mensajes por cada pulsación, pero si lo dejamos presionado la cantidad de mensajes incrementa.

El segundo y cuarto bloque no tienen nada de información, son simples BT que permiten al emisor leer la información a tiempo y correctamente. En este código, los BT duran lo mismo con un tiempo de 69 ms.

El tercer bloque contiene la segunda sección de información **SF** (*Second Frame*) que dura 31 ms al igual que el FF, y cada dígito contiene 2 bits (un BP) representando al sistema trinario. Igual que en el código A, ambos frames son diferentes a pesar de que duran lo mismo, cada frame contiene información específica que le permite al receptor saber cual transmisor es el que quiere interactuar con él.

Existen tres tipos de código E: PayLoad 00, 01 y 10. Todos tienen la misma estructura básica, sin embargo, cada uno de ellos tiene duración distinta. Los tres cambian sus frames en cada mensaje. El proyecto se enfocó únicamente en el Código **E PayLoad 00**.

[11] Es indispensable saber que los frames de este código siempre cambian en cada mensaje de transmisión.

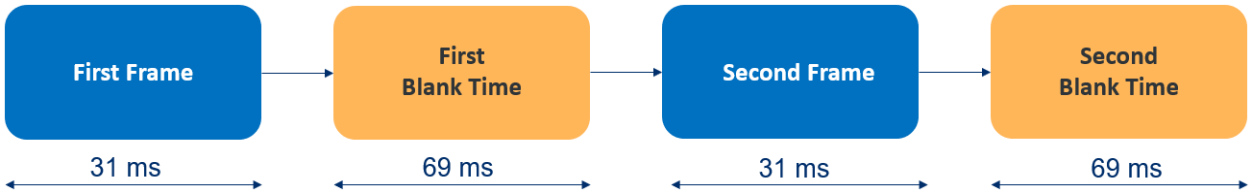

Figura 9. Estructura del Código E.

8.3. Diagrama de bloques

Para poder analizar más fácil las etapas de comunicación para transmitir los códigos a un tablero se decidió hacer un diagrama de bloques de lo que será el *Setup* (SIMBA) completo.

En primer lugar (izquierda) tenemos el *Black Box* que generará los códigos deseados. Esta etapa está representada por el FPGA. Después (en medio) tenemos una interfaz que nos ayudará a conectar el *Black Box* con el generador de señales respetando las especificaciones de ambos equipos. Y finalmente (derecha) está el generador de señales que transmitirá los códigos con los parámetros específicos.[12]

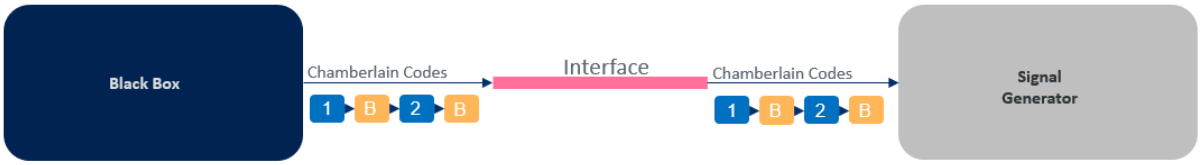

Figura 10. Diagrama de bloques del proyecto.

8.4. Especificaciones de equipos

Antes de pasar con las especificaciones del Generador de Señales y el FPGA (*Black Box*), se debe considerar que como elemento extra es necesario una fuente de corriente directa para alimentar la interfaz del setup, específicamente al OpAmp. La fuente debe suministrar 6 V DC. Más adelante se muestra la obtención de dicho voltaje.

Cada uno de los equipos usados en SIMBA tienen parámetros que necesitan ser respetados para la conexión con otros equipos. Ya que de no seguirlos se corre el riesgo de un mal o nulo funcionamiento del equipo, gasto innecesario y elevado y tiempo desperdiciado. Fue por ello que se dedicó buena parte del proyecto a cumplir estas especificaciones para realizar la interfaz. Las *Datasheet* (hoja de datos) de los equipos de interés fueron del Generador de Señales y del *Black Box*.

[12]Cada código trabaja a una frecuencia específica.

Table 1. Absolute Maximum Ratings for Cyclone V Devices—*Preliminary*

Symbol	Description	Minimum	Maximum	Unit
V_{CC}	Core voltage and periphery circuitry power supply	–0.5	1.35	V
V_{CCPGM}	Configuration pins power supply	–0.5	3.75	V
V_{CC_AUX}	Auxiliary supply	–0.5	3.75	V
V_{CCBAT}	Battery back-up power supply for design security volatile key register	–0.5	3.75	V
V_{CCPD}	I/O pre-driver power supply	–0.5	3.75	V
V_{CCIO}	I/O power supply	–0.5	3.9	V
V_{CCA_FPLL}	PLL analog power supply	–0.5	3.75	V
V_{CCH_GXB}	Transceiver high voltage power	–0.5	3.75	V
V_{CCE_GXB}	Transceiver power	–0.5	1.35	V
V_{CCL_GXB}	Transceiver clock network power	–0.5	1.35	V
V_I	DC input voltage	–0.5	3.80	V
V_{CC_HPS}	HPS core voltage and periphery circuitry power supply	–0.5	1.35	V
V_{CCPD_HPS}	HPS I/O pre-driver power supply	–0.5	3.75	V
V_{CCIO_HPS}	HPS I/O power supply	–0.5	3.9	V
$V_{CCRSTCLK\_HPS}$	HPS reset and clock input pins power supply	–0.5	3.75	V
V_{CCPLL_HPS}	HPS PLL analog power supply	–0.5	3.75	V
I_{OUT}	DC output current per pin	–25	40	mA
T_J	Operating junction temperature	–55	125	°C
T_{STG}	Storage temperature (No bias)	–65	150	°C

Figura 11. Especificaciones de salida de corriente del FPGA Cyclone V.

Inputs and Outputs

Front panel connectors	
Ext 1	External AM/FM/PM #1 input; nominal input impedance is 50 Ω/600 Ω/1M Ω, nominal; damage levels are ± 5 V
Ext 2	External AM/FM/PM #2 input; nominal input impedance is 50 Ω/600 Ω /1M Ω, nominal; damage levels are ± 5 V

Figura 12. Especificaciones de Ext1 del Generador de Señales N5181B Analog.

Con las tablas directamente obtenidas de las *datasheet* de los equipos queda claro que la interfaz no debe superar una corriente de salida en el FPGA a 40 mA, es decir, la interfaz no debe exigir más de 40 mA al FPGA. Mientras que la entrada Ext1 no tolera un voltaje mayor a los 5 V DC. Por lo tanto, la señal de salida tiene que ser controlada con valores de 0 V DC (0 lógico) y 3.3 V DC [13] (1 lógico).

[13]Salida del FPGA.

8.5. Interfaz

Para realizar la interfaz, que es el intermediario entre FPGA - Generador, se tuvo que diseñar un pequeño circuito que respetase los parámetros de ambos equipos. Es por eso que se tuvieron varios diseños de circuitos terminando al fin con un arreglo que se muestra a continuación.

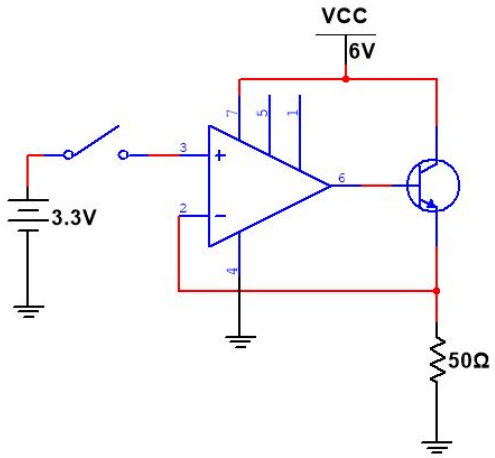

Figura 13. Circuito de interfaz.

Para controlar la corriente de entrada a la intefaz, pareció una buena idea incluir al inicio del circuito un Amplificador operacional (Op Amp) por su alta impedancia de entrada. El Op Amp tiene una configuración de "seguidor". Sin embargo, para poder aumentar la corriente de salida se agregó un transistor tipo NPN con una configuración tipo "emisor común". Entre ambas configuraciones se coloca como *High Compliance Current Sink*.

8.5.1. Amplificador Operacional

El Amplificador operacional que se utilizó fue el **LM358** porque en su *datasheet* su ancho de banda de ganancia unitaria es de 1 MHz, opera bien en un rango de 0 ^{o}C a 70 ^{o}C bastante similar al clásico LM741, pero mejor por tener dos Op Amps, además de que es un CI muy usado en la planta. Cabe destacar que aunque existen mejores OpAmps, recordar que una de las metas del proyecto es trabajar a bajos costos y con el material disponible.

8.5.2. Transistor

El Transistor que se utilizó fue el **2n2222a NPN** porque en su *datasheet* su $V_{CE(sat)} = 0.3$ V si $I_C = 150$ mA y $I_B = 15$ mA. También $V_{BE(sat)} = 1.2$ V si $I_C = 150$ mA y $I_B = 15$ mA.

Esto nos garantiza una buena construcción de interfaz sin riesgos a esperar resultados terribles.

8.5.3. Análisis del circuito

La interfaz consistía de un solo transistor 2n2222a NPN configurado como emisor común. Los parámetros del circuito se calcularon de la siguente forma:

Con $V_{in} = 3.3$ V, $V_{BE} = 0.7$ V, $V_{Rc} = 3$ V y proponiendo $R_c = 5.1\ k\Omega$

$$i_C = \frac{v_{Rc}}{R_c} = \frac{3}{5.1 \times 10^3} \Longrightarrow \underline{\underline{\boxed{i_C = 588.23\ \mu A}}}$$

I. Con $\beta = 35$

$$R_b = \frac{v_{in} - v_{BE}}{i_B} = \frac{\beta}{i_C}(v_{in} - v_{BE}) = \frac{35}{588.23 \times 10^{-6}}(3.3 - 0.7) \Longrightarrow \underline{\underline{\boxed{R_b = 154.70\ k\Omega}}}$$

$$i_B = \frac{v_{in} - v_{BE}}{R_b} = \frac{3.3 - 0.7}{154.70 \times 10^3} \Longrightarrow \underline{\underline{\boxed{i_B = 16.81\ \mu A}}}$$

i) Con $i_B = 16.81\ \mu A$ y $\beta = 35$

$$i_C = \beta i_B = 35(16.81 \times 10^{-6}) \Longrightarrow \underline{\underline{\boxed{i_C = 588.35\ \mu A}}}$$

$$V_c = R_c i_C = (154.70 \times 10^3)(588.35 \times 10^{-6}) \Longrightarrow \underline{\underline{\boxed{V_c = 3\ \text{V}}}}$$

ii) Con $i_B = 16.81\ \mu A$ y $\beta = 300$

$$i_C = \beta i_B = 300(16.81 \times 10^{-6}) \Longrightarrow \underline{\underline{\boxed{i_C = 5.04\ mA}}}$$

$$V_c = R_c i_C = (154.70 \times 10^3)(5.04 \times 10^{-3}) \Longrightarrow \underline{\underline{\boxed{V_c = 25\ V}}}$$

II. Con $\beta = 300$

$$R_b = \frac{v_{in} - v_{BE}}{i_B} = \frac{\beta}{i_C}(v_{in} - v_{BE}) = \frac{300}{588.23 \times 10^{-6}}(3.3 - 0.7) \Longrightarrow \boxed{R_b = 1.326\ M\Omega}$$

$$i_B = \frac{v_{in} - v_{BE}}{R_b} = \frac{3.3 - 0.7}{1.326 \times 10^6} \Longrightarrow \underline{\underline{\boxed{i_B = 1.97\ \mu A}}}$$

i) Con $i_B = 1.97\ \mu A$ y $\beta = 35$

$$i_C = \beta i_B = 35(16.81 \times 10^{-6}) \Longrightarrow \underline{\underline{\| \; i_C = 68.95 \; \mu A \; \|}}$$

$$V_c = R_c i_C = (1.326 \times 10^6)(68.95 \times 10^{-6}) \Longrightarrow \underline{\underline{\| \; V_c = 0.352 \; V \; \|}}$$

ii) Con $i_B = 1.97 \; \mu A$ y $\beta = 300$

$$i_C = \beta i_B = 300(16.81 \times 10^{-6}) \Longrightarrow \underline{\underline{\| \; i_C = 591 \; \mu A \; \|}}$$

$$V_c = R_c i_C = (1.326 \times 10^6)(591 \times 10^{-6}) \Longrightarrow \boxed{V_C = 3.01 \, V}$$

Con estos resultados, se ve que la salida del colector del transistor tiene que ser lo más parecido a V_{in}, por lo que tenemos dos opciones. Una de ellas es cuando $\beta = 35$ y $i_B = 16.81$ μA. La segunda es cuando $\beta = 300$ y $i_B = 1.97$ μA. Se decidió entre el segundo valor de $R_b = 1.326$ $M\Omega$ por tomar el valor más extremo de β.

Para conocer el valor de V_{CC} se calcula con la ley de Ohm.

$$R_c = \frac{V_{cc} - V_{Rc}}{i_C} \Rightarrow V_{cc} = R_c i_C + V_{Rc} = V_C + V_{Rc} \Longrightarrow \boxed{V_{cc} = 6 \, V}$$

Con estos valores de componentes justificados, se comenzó a simular el circuito. Funcionó como se esperaba. Sin embargo, cuando se construyó físicamente el voltaje de salida V_c variaba. Así que se cambiaron los resistores a diferentes valores comerciales, se cambió el transistor e incluso se modificó el valor de V_{cc}.

Sin embargo, ninguno de esos cambios funcionó. Así que se recordó que se podría usar un Amplificador Operacional a la entrada de la base del transistor, ya que la entrada del operacional tiene una alta impedancia, y su configuración para mantener el voltaje debería ser un seguidor. El OpAmp 741 era adecuado para este trabajo, pero no se encontraba disponible en la empresa. Así que primero se tuvo que revisar en *SAP* qué OpAmp estababa disponible en *stock*. Después revisar sus *datasheet* para saber cuál podría cumplir la tarea. Fue cuando se encontró que el OpAmp 358 era muy similar al OpAmp 741.

Revisando la *datasheet* de este Operacional, se encontró una configuración similar a la propuesta, llamada **High Compliance Current Sink**. Con esta nueva configuración no hizo falta colocar resistor alguno. Aún así, se incluyen los cálculos matemáticos de la obtención de estos componentes para apreciación del lector.

8.6. Adaptación a los casos que puedan causar errores

La adaptación de los códigos **E** y **F** consistió en estudiar y entender la máquina de estados para poder ver los posibles casos en que pudieran fallar los códigos. Tales como el tratar de cambiar la dirección de la flecha del motor del GDO sin que se detenga o no haga caso, vincular correctamente por primera vez al GDO con el *setup* sin oprimir varias veces y mantener presionado el botón del FPGA. Del ánalisis se pudo definir que necesitaba enviar la secuencia del código (códigos E y F) al mismo tiempo que se actualizaban los valores del siguiente mensaje.

8.7. Ahorros

Las consideraciones tomadas para la reducción de costos que se incluyeron fueron:

1. Eliminación del costo fabricando un PCB para cada código.
2. Reducción del número de componentes para la interfaz.
3. Dejar de usar un viejo *clicker* obsoleto y costoso.

En el sector industrial, el uso indispensable de algún *setup* para realizar pruebas, procesos, etc. debe ser fácil y rápido. Es por ello que escoger o diseñar el equipo adecuado para trabajar es una tarea que lleva tiempo y análisis. Es de suma importancia conocer las tolerancias que pueden ser permitidas en los componentes (corriente máxima, voltaje, potencia, etc.).

Con este proyecto no solo se ahorra en componentes sino que también en tiempos, debido a que no es necesario hacer modificaciones de hardware (PCB) ni "flashear" varias veces el microcontrolador para acceder a otro código. Los tiempos se pueden reducir hasta en un 68 % lo que significa que ese tiempo de sobra se puede emplear haciendo otras actividades.

8.8. Prueba funcional

Para poder validar el funcionamiento correcto del proyecto la intención fue vincular el FPGA con un GDO y poder controlarlo a voluntad con algún código que pudiese manejar el módulo.

Para hacer esto necesitamos conectar el nuevo *Setup* (SIMBA) de acuerdo con el diagrama ya antes mencionado. Los pasos a realizar fueron:

1. Conectar la salida del FPGA (el que llevará el código de la empresa) al tercer pin del Op Amp.
2. Unir la salida de la interfaz (Emisor) al cable que irá a la entrada (Ext1) del Generador de Señales.

3. Alimentar la interfaz con ± 6 V DC.
4. Configurar el generador de Señales con las especificaciones correctas.
5. Encender el GDO y programarlo para vincular con un nuevo transmisor.
6. Vincular el FPGA con el GDO.
7. Una vez verificada la conexión, realizar una prueba (abrir/cerrar) al equipo.

Si se desea probar con algún otro GDO con el mismo código, lo único que se tiene que hacer es programar el GDO para vincularse con algún transmisor. Y si se desea probar otro código, se necesita cargar el código al FPGA desde la computadora y configurar el generador de señales a la nueva frecuencia con la que trabaja el código.

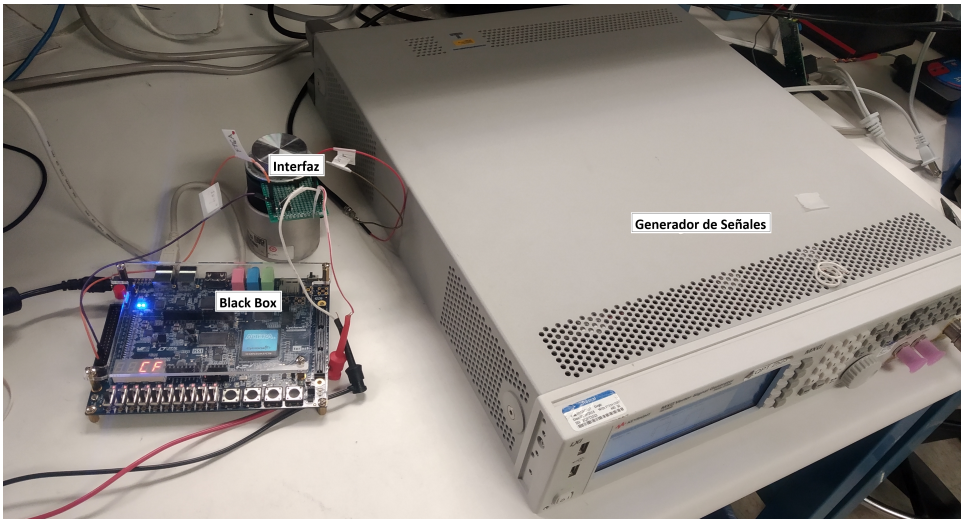

Figura 14. Setup de SIMBA.

9. Resultados y aportaciones

A lo largo de mi estancia en esta empresa, comprendí los procesos de problemas de *sustaining*; desde que llega a la plataforma *backlog* (pendiente) de **Jira** hasta que el problema queda hecho *done* (hecho).

Para realizar la descripción de hardware de los códigos, hice uso de mis conocimientos previos en diseño digital, programación y electrónica. Además, fueron necesarias bastantes horas de estudio e investigación acerca del lenguaje de descripción VHDL para un mejor uso del FPGA.

Logré comprender los códigos A y E (que se me asignaron) en tiempo y forma de la manera más eficaz posible, dejando comentarios en cada uno de los archivos del IDE del FPGA. Cabe mencionar que solo tuve un par de días para realizar la descripción de hardware

de un código más de la lista (código F).

Logré comunicar de manera física el *Black Box* con el Generador de Señales sin problema gracias a la interfaz entre ellos respetando así los parámetros de ambos equipos.

Gracias a la comunicación para el nuevo *setup*, hubo una exitosa transmisión entre **SIMBA** y un *GDO* de la compañía. Con esto se demostró que sí es posible migrar del setup viejo (con un tipo de tecnología) a uno nuevo, usando tecnología FPGA con la misma eficiencia y a bajos costos. Dejando así paso para que en un tiempo futuro se puedan agregar los demás códigos de la empresa.

10. Conclusiones

El objetivo se cumplió en su totalidad, los costes para el armado del nuevo *setup* (SIMBA) fueron bastante bajos en comparación con el setup anterior ahorrando al rededor de $ 20,300 dlls. La eficiencia de ambas pruebas se mantuvo. Y la ejecución y ensamble del *setup* fueron más sencillas y rápidas.

Este proyecto me exigió una capacidad de aprendizaje rápida, además de una base sólida de conocimientos técnicos/teóricos para poder juntar las partes que harían que el proyecto funcionara. Entre los nuevos conocimientos que adquirí están el uso de plataformas tales como SAP, Windchill y JIRA; el uso de un FPGA "Altera" y su descripción en Lenguaje VHDL; y el uso de un instrumento llamado *Logic Analyzer* (analizador lógico) de Saleae, con el cual me apoyaba para observar la señal de los códigos.

Para los códigos E y F (códigos definitivos), tuve que comprender cómo operaba el código trinario representado en pares de bits, como ordenar dichos bits, actualizarlos y como hacer que cambien en cada mensaje sin que se viera afectado el tiempo de transmisión.

Aunque mi proyecto no tuvo la finalidad de producción para salir al mercado, me aseguré de que tuviera la calidad para sustituir al *setup* pasado; que su capacidad de manufactura (en caso que se quiera o requiriera replicar) fuera diseñado por los recursos disponibles en la planta; y por último, contemplar el costo que, se debió considerar para cubrir requisitos de los dos puntos anteriormente mencianados.

Mi experiencia en la empresa fue única. Cada día aprendía de la empresa, del sitio (Nogales) y de mí mismo. No sólo desarrollé habilidades profesionales, sino también personales. Me di cuenta de que los conocimientos y experiencias adquiridas en la Universidad ayudan bastante. No obstante, hay cosas en las que uno no está preparado entrando al ambiente laboral; es por lo que agradezco haber tenido la oportunidad de ser parte de este programa.

11. Bibliografía

- Pardo Carpio, F., & Boluda Grau, J. (1999). VHDL. Madrid: Ra-ma.
- Cirovic, M. M. (1979). Electrónica fundamental: Dispositivos, Circuitos y Sistemas.
- Blakeslee, D. (1972). The Radio Amateur's Handbook.
- Gilmore, R., & Besser, L. (2003). Practical RF Circuit Design for Modern Wireless Systems: Active Circuits and Systems, Vol. 1; Artech House.
- Agile design processes and guidelines | Atlassian. (2018). Atlassian. Retrieved 30 April 2018, from https://www.atlassian.com/agile/design
- Kanban - A brief introduction | Atlassian. (2018). Atlassian. Retrieved 30 April 2018, fromhttps://www.atlassian.com/agile/kanban
- $http://www1.frm.utn.edu.ar/tecnicad1/_private/Apuntes/VHDL.pdf$
- $https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf$
- $http://oa.upm.es/48931/1/TFG_ROSA_SANZ_FERNANDEZ.pdf$
- $https://www.mobilefish.com/developer/iota/iota_quickguide_tutorial.html$
- $http://jagarza.fime.uanl.mx/general/presentaciones/notas.pdf$
- $https://copro.com.ar/Selectividad_(electronica).html$
- $https://www.keysight.com/us/en/assets/7018-03380/data-sheets/5991-0038.pdf$
- $https://class.ece.uw.edu/271/peckol/doc/DE1-SoC-Board/Datasheet/FPGA/cyclone5_datasheet.pdf$
- $https://www.mouser.mx/pdfdocs/C5G_User_Manual.PDF$
- $http://www.ti.com/lit/ds/symlink/lm358-n.pdf?\&ts=1589576589834$
- $https://www.mouser.com/datasheet/2/149/PN2222A-371983.pdf$
- $http://redeya.bytemaniacos.com/electronica/tutoriales/PDF/vhdl.pdf$
- $https://www.profesores.frc.utn.edu.ar/electronica/tecnicasdigitalesi/pub/file/AportesDelCudar/Maquinas\%20de\%20Estado\%20MC\%20V5.pdf$

12. Glosario

→ ASIC. Application Specific Integrated Circuit (Circuito Integrado de Aplicación Específica).

→ Billion Code. Código usado en el código A (también llamado trillion code).

→ Black Box. Caja negra representada por el FPGA que representa el primer bloque del diagrama de bloques de SIMBA.

→ BP. Bit pair (par de bit) representa el código trinario de esta forma.

→ BT. Blank Time que se encuentra entre cada frame (información).

→ Clicker. Otra manera de referirse a un transmisor.

→ Datasheet. Hoja de especificaciones de algún componente.

→ Firmware. Código que puede ser programado en un lenguaje de alto o bajo nivel. Un ejemplo de firmware es el control remoto de la televisión que emite diferentes códigos a través del LED infrarrojo que tiene dependiendo del botón que haya sido presionado.

→ Frame. Tramo de datos binarios de longitud variable.

→ FF. First Frame (primera estructura)de cada mensaje de cualquier código.

→ FPGA. Field Programmable Gate Array (Matriz de Puertas Lógicas Programable en Campos).

→ Glitch. Error, problema o malfuncionamiento que, al no afectar negativamente al rendimiento, no puede considerarse un fallo, sino más bien una característica no prevista.

→ GDO. Garage Door Opener.

→ Hardware. Se refiera a los componentes electrónicos de un sistema.

→ LUT. Look Up Table (Tabla de búsqueda).

→ Logic Analyzer. Un analizador lógico es un instrumento que captura los datos de un circuito digital y los muestra en pantalla de modo similar a un osciloscopio, con la diferencia de que se pueden visualizar señales de múltiples canales.

→ Rolling Code. Código usado en los códigos E y F.

→ Setup. Organización de componentes o equipos para una determinada tarea.

→ SIMBA. Nombre del proyecto.

→ Sustaining. Área de mantenimiento, en este caso electrónica.

→ SF. Second Frame (segunda estructura) de cada mensaje de cualquier código que siempre va entre dos blank times.

→ SM. State Machine (máquina de estados).

13. Anexos

A continuación se muestran los diagramas de bloques y diagramas de estados de cada código.

13.1. Diagramas de Bloques

13.1.1. Código A

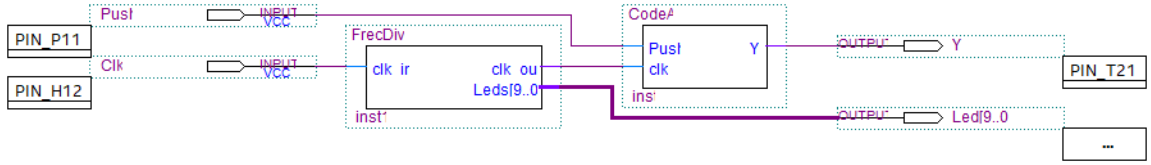

Figura 15. Diagrama de Bloques de código A.

13.1.2. Códigos E y F

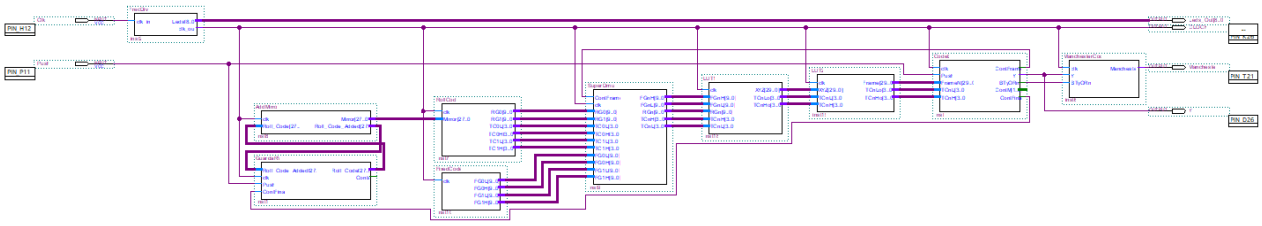

Figura 16. Diagrama de Bloques de códigos E y F.

13.2. Diagramas de Estados

13.2.1. Código A

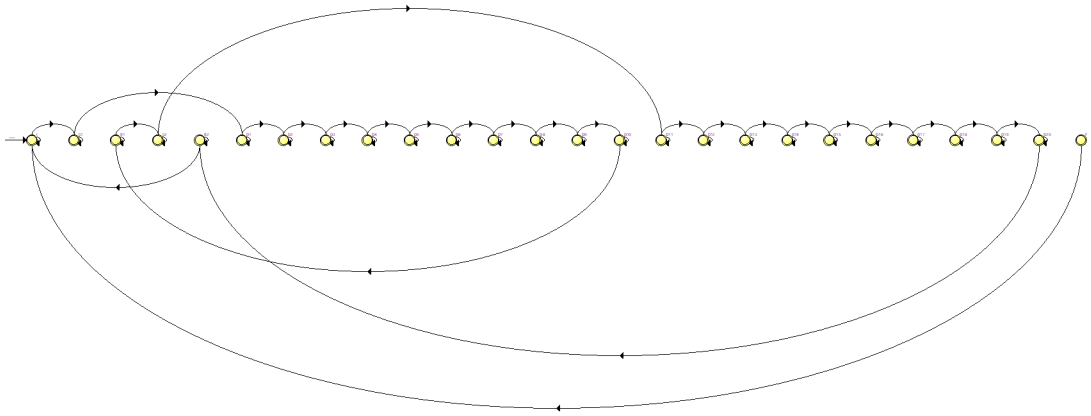

Figura 17. Diagrama de Estados de código A.

13.2.2. Códigos E y F

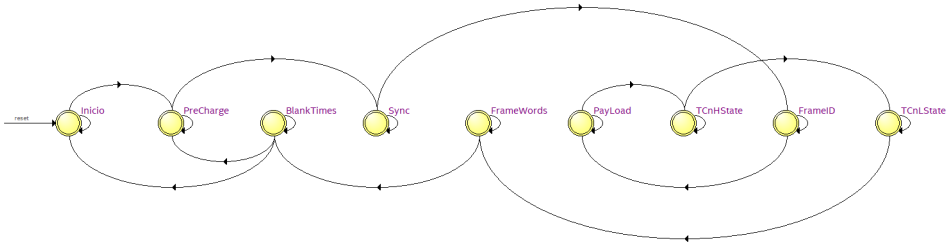

Figura 18. Diagrama de Estados de códigos E y F.