Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

DISEÑO E IMPLEMENTACIÓN DE UN OSCILADOR SENOIDAL NUMÉRICAMENTE CONTROLADO UTILIZANDO SÍNTESIS DE ALTO NIVEL PARA UN SISTEMA EMBEBIDO LÓGICO PROGRAMABLE

Daniel Estuardo González Sierra
Asesorado por la Inga. Ingrid Salomé Rodríguez de Loukota

Guatemala, agosto de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN OSCILADOR SENOIDAL NUMÉRICAMENTE CONTROLADO UTILIZANDO SÍNTESIS DE ALTO NIVEL PARA UN SISTEMA EMBEBIDO LÓGICO PROGRAMABLE

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

DANIEL ESTUARDO GONZÁLEZ SIERRA
ASESORADO POR LA INGA. INGRID SALOMÉ RODRÍGUEZ DE LOUKOTA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, AGOSTO DE 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Jurgen Andoni Ramírez Ramírez
VOCAL V	Br. Oscar Humberto Galicia Nuñez
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADORA	Inga. María Magdalena Puente Romero
EXAMINADOR	Ing. José Antonio de León Escobar
EXAMINADOR	Ing. Armando Alonso Rivera Carrillo
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO E IMPLEMENTACIÓN DE UN OSCILADOR SENOIDAL NUMÉRICAMENTE CONTROLADO UTILIZANDO SÍNTESIS DE ALTO NIVEL PARA UN SISTEMA EMBEBIDO LÓGICO PROGRAMABLE

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 6 de septiembre de 2016.

Daniel Estuardo González Sierra

Guatemala 25 de abril de 2017

Ingeniero
Carlos Eduardo Guzmán Salazar
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Estimado Ingeniero Guzmán.

Me permito dar aprobación al trabajo de graduación titulado: **"Diseño e implementación de un oscilador senoidal numéricamente controlado utilizando síntesis de alto nivel para un sistema embebido lógico programable"**, del señor Daniel Estuardo González Sierra, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo de graduación y, yo, como su asesora, nos hacemos responsables por el contenido y conclusiones del mismo.

Sin otro particular, me es grato saludarle.

Atentamente,

Inga. Ingrid Rodríguez de Loukota
Colegiada 5,356
Asesora

Ingrid Rodríguez de Loukota
Ingeniera en Electrónica
colegiado 5356

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

Ref. EIME 21. 2017
Guatemala, 12 de MAYO 2017.

Señor Director
Ing. Francisco Javier González López
Escuela de Ingenierìa Mecànica Elèctrica
Facultad de Ingenierìa, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: DISEÑO E IMPLEMENTACIÓN DE UN OSCILADOR SENOIDAL NUMÉRICAMENTE CONTROLADO UTILIZANDO SÍNTESIS DE ALTO NIVEL PARA UN SISTEMA EMBEBIDO LÓGICO PROGRAMABLE, del estudiante Daniel Estuardo González Sierra , que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,
ID Y ENSEÑAD A TODOS

Ing. Carlos Eduardo Guzmán Salazar
Coordinador de Electrónica

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

sro

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 21. 2017.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; DANIEL ESTUARDO GONZÁLEZ SIERRA titulado: DISEÑO E IMPLEMENTACIÓN DE UN OSCILADOR SENOIDAL NUMÉRICAMENTE CONTROLADO UTILIZANDO SÍNTESIS DE ALTO NIVEL PARA UN SISTEMA EMBEBIDO LÓGICO PROGRAMABLE, procede a la autorización del mismo.

Ing. Francisco Javier González López

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 5 DE JUNIO 2,017.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 338.2017

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DISEÑO E IMPLEMENTACIÓN DE UN OSCILADOR SENOIDAL NUMÉRICAMENTE CONTROLADO UTILIZANDO SÍNTESIS DE ALTO NIVEL PARA UN SISTEMA EMBEBIDO LÓGICO PROGRAMABLE,** presentado por el estudiante universitario: **Daniel Estuardo González Sierra,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, agosto de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

/gdech

ACTO QUE DEDICO A:

Dios	Por su amor, gracia y misericordia mostrados cada día de mi vida.
Mi padre	Rodolfo, por su permanente esfuerzo, apoyo y cariño incondicional.
Mi madre	Leticia, gracias por cada momento de su vida conmigo.
Mi hermano	José, por estar en todos los momentos que lo he necesitado.
Mi abuela	Magda, por ser ejemplo de trabajo y esfuerzo, además de siempre cuidar de mí.

AGRADECIMIENTOS A:

Mis amigos	Las personas que siempre han estado ahí sin importar el momento.
Mi familia	Por su cariño y apoyo.
Mi tío	Julio, por su aprecio patentemente demostrado.
Laboratorio de Electrotecnia	Por haberme brindado la oportunidad de laborar y obtener experiencias que complementan mi formación profesional
Mi amiga	Gabriela, por su apoyo y confianza.
Mi asesora	Inga. Ingrid de Loukota, por su apoyo brindado siempre de la mejor manera posible.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

LISTA DE SÍMBOLOS

Símbolo	Significado
Hz	Hertz
Kbit/s	Kilobit por segundo
KHz	Kilohertz
Mbit/s	Megabit por segundo
MHz	Megahertz
mV	Milivoltio
mW	Miliwatt
V	Voltio
W	Watt

GLOSARIO

ARM	Máquina avanzada de conjunto de instrucciones reducido (*Advanced RISC Machine*).
Bare-Metal	Término designado a aplicaciones que funcionan en hardware sin sistema operativo.
Bit	Es la unidad básica de información en computación.
Byte	Conjunto de 8 bits que es tratado como unidad.
Eigenvector	Son vectores que, al ser transformados por un operador lineal, únicamente cambian su magnitud.
Embebido	Anglicismo sinónimo de “empotrado”.
FPGA	Arreglo de compuertas programables en campo.
Frecuencia	Número de repeticiones por unidad de tiempo.
Hardware	Conjunto de elementos físicos que constituyen un sistema informático.
Hexagesimales	Valores numéricos tomados del sistema de numeración posicional con base 16.

HLS	Es la traducción de un lenguaje de alto nivel en un archivo.
IC	Circuito integrado en una estructura de pequeñas dimensiones hecho de material semiconductor.
Inferir	Extraer conclusiones a partir de hechos o preposiciones.
Metaestabilidad	Es un estado indefinido de equilibrio débil que se resuelve en un estado de equilibrio fuerte.
NEON	Extensión de arquitectura ARM para proceso de formatos de media.
OEM	Es una organización que fabrica dispositivos de componentes comprados a otras organizaciones.
Potencia	Cantidad de trabajo que se realiza por unidad de tiempo.
Script	Archivo de órdenes o de procesamiento en lotes.
Síntesis	Es la obtención de resultados de un sistema u objeto natural a través de medios sintéticos (no naturales).
SoC	Es un circuito integrado que contiene todos los elementos de un sistema electrónico.

Software	Conjunto de elementos virtuales que componen un sistema informático.
Taxonomía	Clasificación en grupos con base en características comunes.
Vector	Es una magnitud física definida en un sistema de referencia.
Xilinx	Compañía de tecnología americana especializada en dispositivos lógicos programables.

RESUMEN

En el presente documento se muestra una propuesta para la realización de un generador de onda senoidal en tecnología lógica programable (FPGA) utilizando síntesis de alto nivel. Se da de esta manera un paso adelante en la creación de dispositivos generadores de onda reprogramables de bajo costo.

El primer capítulo es una introducción teórica a conceptos generales relacionados al proyecto y el marco de trabajo en el cual este se desarrollará. Inicialmente se definen las señales y sistemas digitales binarios, las plataformas físicas como microprocesadores, microcontroladores, FPGA, sistemas embebidos y, por último, la existencia de sistemas embebidos combinados.

El segundo capítulo desarrolla una explicación técnica de la plataforma de hardware a utilizar (Zynq) que contempla arquitectura, estructura, características principales, elementos fundamentales. Paralelo a la explicación de los componentes se desarrolla una descripción teórica de conceptos de aplicación.

El tercer capítulo describe el concepto de síntesis digital directa para la generación de formas de onda analógicas en sistemas digitales, y la metodología empleada en este sistema. Se aborda la descripción del control numérico aplicado a los osciladores, la generación de valores de onda a través del método de tabla de búsqueda y un método tradicional, el cual servirá de comparación.

El cuarto capítulo contiene tanto el diseño del sistema como la implementación del mismo. Las consideraciones del diseño, la implementación y los resultados físicos obtenidos son descritos a lo largo del mismo.

OBJETIVOS

General

Diseñar e implementar un oscilador senoidal numéricamente controlado mediante el uso de síntesis de alto nivel para un sistema embebido lógico programable.

Específicos

1. Exponer aspectos generales acerca de sistemas digitales.

2. Presentar el control numérico aplicado a la síntesis digital y los distintos métodos de síntesis.

3. Presentar el sistema embebido lógico programable ZYNQ.

4. Diseñar e implementar el sistema.

INTRODUCCIÓN

En el estudio de la ingeniería es necesaria la búsqueda de métodos y herramientas que permitan el desarrollo científico, como la investigación, para lo cual es necesario crear ambientes en los cuales las variables puedan ser controladas a manera de tener una única variable de estudio. Para ello es menester contar con equipos especializados, los cuales contribuyen a realizar esta tarea.

La generación de señales eléctricas es algo vital, tanto para las telecomunicaciones como para el tratamiento de señales, así como otras muchas ramificaciones de la electrónica en las cuales se desarrollan proyectos y aplicaciones. Las alternativas de generación y síntesis de señales en la actualidad pueden ser variadas; algunas de las más exactas y utilizadas son de forma física, es decir, por hardware, mientras otras se manejan de forma virtual; es decir, por software. Cada una de estas presenta ventajas y desventajas.

La mayoría de las soluciones por hardware son bastante costosas y algunas poco flexibles, por lo cual (en dependencia del ambiente de desarrollo) no son necesariamente la mejor opción. Ahora bien, las soluciones en software son bastante más económicas, accesibles y flexibles pero menos exactas y bastante más lentas en la generación.

Una vez tomadas en cuenta las soluciones actuales, se propone explorar un campo híbrido, el cual tenga un hardware reconFigurable y un software que se adapte a este y lo maneje. Esto se logra en plataformas de sistemas

embebidos de alta gama, FPGAs con unidades de procesamiento de bajo costo.

1. ASPECTOS GENERALES ACERCA DE SISTEMAS DIGITALES

El dominio digital es la base fundamental del diseño y desarrollo de la solución propuesta para la generación de señales; por lo tanto, se procede con una pequeña presentación de conceptos generales y habituales acerca de la concepción digital y los sistemas digitales.

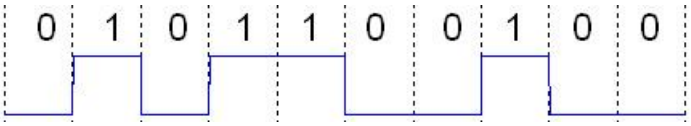
1.1. Sistema digital

Puede definirse como un conjunto de diferentes dispositivos discretos dedicados al procesamiento, tratamiento y almacenamiento de señales digitales.

1.1.1. Señal digital

Una señal digital es aquella que representa una secuencia de valores discretos con un conjunto de amplitudes; es decir, existe una *m* cantidad de niveles (en este caso de voltaje) de los cuales nuestra señal puede representar una *m* cantidad de valores, Para ese caso a la señal se le denominaría una señal digital *m*-aria. A lo largo de la historia se investigaron diferentes tipos de señales digitales sin mucho éxito hasta llegar a la utilización de únicamente dos representaciones en la señal.

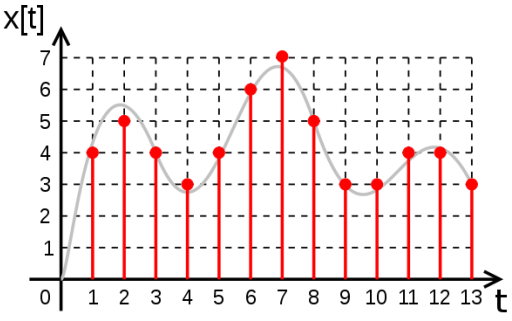
Figura 1. **Señal digital binaria**



Fuente: Señal digital. https://en.wikipedia.org/wiki/Digital_signal. Consulta: 10 de septiembre de 2016

Las señales de dos posibles valores son denominadas señales digitales binarias o señales lógicas. Son las utilizadas en sistemas digitales. Se hace la diferencia entre el concepto aplicado de las señales digitales en las cuales está basado el funcionamiento de la plataforma de desarrollo llamada ZyBo y las señales digitales que se estarán generando para sintetizar señales senoidales; esto desde la óptica proporcionada en la disciplina de procesamiento de señales en donde ya trabajamos con una señal muestreada y cuantizada pero de igual forma digital, no necesariamente binaria.

Figura 2. **Señal digital muestreada y cuantizada**



Fuente: Señal Digital. https://en.wikipedia.org/wiki/Digital_signal. Consulta: 10 de septiembre de 2016

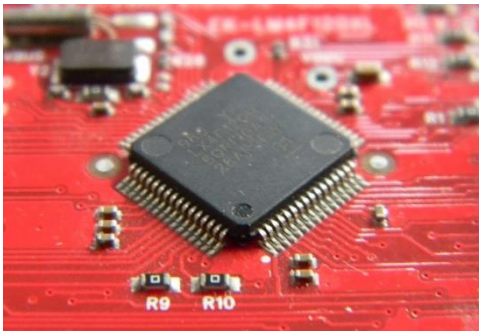
1.2. Microcontrolador

Es una pequeña computadora en un circuito integrado, también llamado *System on Chip* (SoC). Contiene un procesador núcleo, memoria, módulos de comunicación, periferias programables de entrada y salida.

Los microcontroladores fueron diseñados para construcciones embebidas en las que prevalece el principio de la miniaturización y densificación, con una integración mucho mayor del sistema en general. Contrastan con los microprocesadores utilizados en las computadoras personales para aplicaciones más generales y consistentes de varios circuitos integrados.

Son utilizados en productos y dispositivos controlados automáticamente; por lo regular, dispositivos de consumo final, algunos tan especializados como implantes médicos y otros más cotidianos como herramientas y juguetes, sin tomar en cuenta otra gran variedad de aplicaciones embebidas. Existe una gran variedad de aplicaciones gracias a la reducción del tamaño y el costo comparativo con diseños que utilizan los circuitos integrados por separado.

Figura 3. **Microcontrolador TM4C12x**



Fuente: elaboración propia.

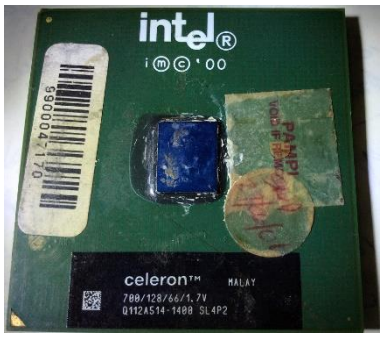
Ciertamente, estos pueden ser menos poderosos en procesamiento, frecuencia de funcionamiento, memoria interna, velocidades de transferencia de datos que los microprocesadores; sin embargo, también se encuentran en arquitecturas de 32-bits, a mayores frecuencias que las que tenían en sus inicios y con tecnologías multinúcleo, lo cual permite una gran cantidad de poderosas aplicaciones que de otro modo no podrían llevarse a cabo.

Para la programación de estos dispositivos siempre se acudió al lenguaje de máquina llamado *assembler.* Sin embargo, estos ambientes de programación han cambiado, por lo que lenguajes de programación de alto nivel como Python, java script y C++ pueden utilizarse sobre estos microcontroladores. El lenguaje más utilizado es C99 que, de hecho, será utilizado más adelante en la unidad de aplicación del SoC. Es de considerar que existen *firmwares* de intérpretes, los cuales pueden ser implementados para soporte de programación interactiva. Como ejemplos tenemos BASIC, FORTH y últimamente una tendencia de obtener un lazo de escritura, evaluación e impresión (*Read Evaluate Print Loop*) de Python en los microcontroladores.

1.3. Microprocesador

Es un procesador computacional que realiza las funciones de la unidad central de procesamiento de un computador. Este es de propósito general y contiene tanto lógica combinacional como secuencial; por lo tanto, se obtienen resultados directamente en función de las entradas y otros en función de estados anteriores.

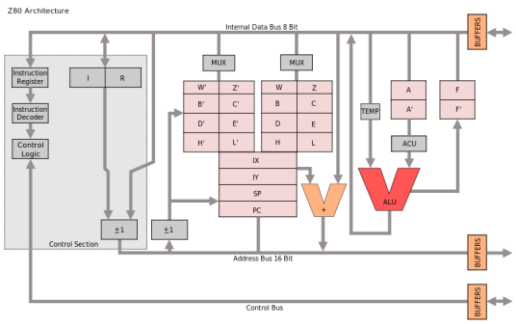
Figura 4. **Microprocesador Celeron**



Fuente: elaboración propia.

Un microprocesador es síncrono por reloj, basado en registros y electrónicamente programable. La complejidad del circuito integrado está delimitada directamente por el espacio físico; es decir, las limitantes del número de transistores que se pueden colocar en un empaquetado. Teóricamente hablando, un microprocesador puede estar conformado únicamente por una unidad aritmética lógica (ALU), banderas, las cuales son bits o registros que cambian su valor en ocurrencia de algún determinado evento; por ejemplo, que un valor llegue a cero, un número negativo o bien desbordamiento en conteo. Los registros de propósito general son donde se almacenará momentáneamente información necesaria para la realización de las operaciones, bien pueda ser este resultado de las operaciones anteriores o direcciones de memoria en las cuales se encuentra la siguiente instrucción. La lógica de control se encarga de recuperar el código de instrucción de la memoria e inicia la secuencia de operaciones que debe llevar a cabo el ALU.

Figura 5. **Arquitectura microprocesador Z80**



Fuente: Zilog Z80. https://es.wikipedia.org/wiki/Zilog_Z80. Consulta: 15 de septiembre de 2016.

Los microprocesadores existen en 4, 8, 12, 16, 32 y 64 bits; esto es el ancho de bus de procesamiento, es decir, el tamaño de los buses de datos. Los más utilizados actualmente en computadoras personales son de 64 bits; sin embargo, los de 32 bits se encuentran un campo de aplicación gigantesco en sistemas embebidos y en algunos computadores de menor capacidad.

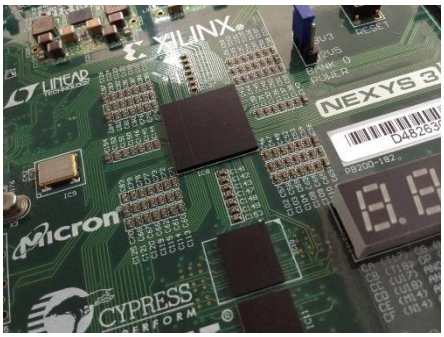
Un microprocesador necesita un set de instrucciones definido, el cual pueda seguir para realizar las operaciones más complejas a través de operaciones más sencillas. De estos existen dos: conjunto de instrucciones complejas de cómputo (CISC) y Conjunto de instrucciones reducidas de computo (RISC). CISC es un conjunto complejo, por lo que permite una mayor gama de operaciones directas entre operandos. Ahora bien, por ser más complejas dificultan el paralelismo entre operaciones. Este set es más utilizado entre los microprocesadores de alto rendimiento. A veces es necesaria la conversión a RISC separando las instrucciones complejas en instrucciones más pequeñas llamadas microinstrucciones, que puedan ser representables en RISC y permiten mayor paralelismo. RISC fue diseñado con el objetivo de posibilitar la segmentación y el paralelismo en la ejecución de instrucciones.

También existe la reducción de accesos a memoria, que evita la saturación de los buses de memoria a causa de accesos por distintas aplicaciones.

1.4. Lógica programable FPGA

Un arreglo de compuertas programable en campo (FPGA) es un circuito integrado diseñado para ser configurado después de la manufactura. Esto se puede leer muy parecido a un microcontrolador u otros tipos de dispositivos digitales, los cuales, después de la manufactura, pueden ser empleados a gusto del diseñador a través de programación. La diferencia se encuentra en referencia a la configuración; es decir, la configuración física del hardware —algo inamovible después de la manufactura en otros dispositivos— se realiza a través de lenguajes de descripción de hardware (HDL). Estos y otros métodos se estudiarán en detalle más adelante.

Figura 6. **Empaquetado FPGA**



Fuente: elaboración propia.

El FPGA contiene un arreglo de bloques lógicos programables y una red jerárquica reconfigurable de interconexiones que permite conectar estos bloques y así realizar funciones combinacionales mucho más complejas. En

estos bloques lógicos también es común encontrar elementos de memoria para funciones secuenciales o bien elementos de procesamiento de señal.

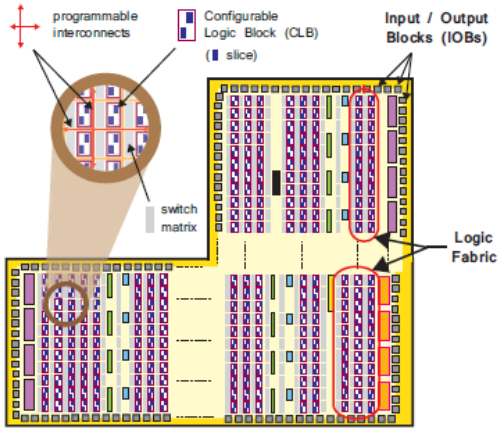
Regularmente se dice que el inicio de los dispositivos programables se dio en las memorias programables de solo lectura y, más adelante, en los dispositivos lógicos programables (PLD), los cuales dieron posteriormente paso a los FPGA. Esto a pesar de que en sus inicios no eran competitivos delante de los dispositivos por hardware como los circuitos integrados de aplicación específica (ASIC), puesto que necesitaban mucha más área y su consumo de poder era mayor, lo cual era poco rentable, en adición la falta de flexibilidad debida a limitaciones tecnológicas y de integración. En la actualidad, los FPGA se han convertido en rivales directos de ASIC y los productos estándar de aplicación específica (ASSP) mediante la reducción de la potencia consumida, aumento en la velocidad, baja en coste de materiales y una aumentada posibilidad de reconFiguración en sitio.

Pueden ser utilizadas para la resolución de problemas de computación, pues tienen un rango de aplicaciones bastante grandes. Un hecho que se puede tomar como demostración de esto es la implementación de un microprocesador descrito en HDL y configurado en FPGA. A esto se le conoce como microprocesador suave (*soft microprocessor*), el cual es capaz de resolver problemas computables. Ahora bien, dependiendo de la aplicación puede ser preferible tener algún microprocesador físico conectado a nuestra tela programable (FPGA) para mayor eficiencia en determinados cálculos y proceso de datos. Si todo esto se encuentra en un mismo empaquetado físico se le conoce como un sistema embebido (empotrado).

Entre las principales aplicaciones se encuentran procesamiento digital de señales, radio definido por software, imagen médica, reconocimiento de voz,

emulación de hardware, criptografía, entre otras. Las aplicaciones de bajo volumen, en las cuales la creación de un ASIC es útil para aplicaciones de altos volúmenes de fabricación, puede no ser la más viable en costos.

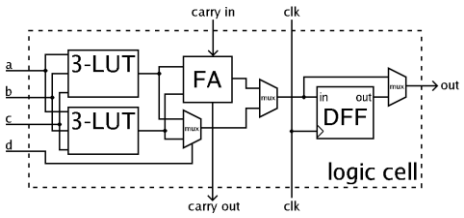
Figura 7. **Elementos de la Tela Programable**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 23.

En cuanto a arquitecturas, las más regulares consisten en arreglos de celdas lógicas, las cuales conforman bloques lógicos conFigurables (CLBs). Estos bloques tienen entradas y salidas físicas en forma de contactos (*pads*) y canales de enrutamiento entre ellos. Una ruta de interconexión es hallada entre los recursos necesarios acorde al circuito de aplicación; es decir, algunos utilizan muchas más pistas de interconexión que otros. El fabricante es el encargado de determinar la cantidad de interconexiones que el dispositivo tendrá, lo cual es una ardua tarea considerando la variedad de aplicaciones existentes. Entonces, toma como punto de partida la mayor cantidad de diseños que se ajusten en términos de tablas de búsqueda (LUT).

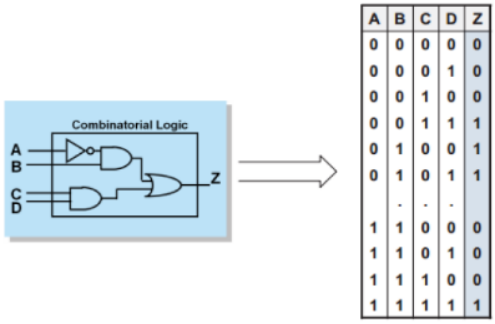
Figura 8. **Celda lógica**



Fuente: CLB. https://en.wikipedia.org/wiki/Logic_block. Consulta: 15 de septiembre de 2016

Las tablas de búsqueda son básicamente tablas que determinan qué salidas se obtendrán para las posibles entradas existentes. Por tanto, se puede decir que en un contexto de lógica combinacional es una tabla de verdad y definirá cómo la lógica combinacional interna deberá comportarse. Este comportamiento no incluye equivalencias de circuitos con retroalimentación o estados.

Figura 9. **Tabla de búsqueda**

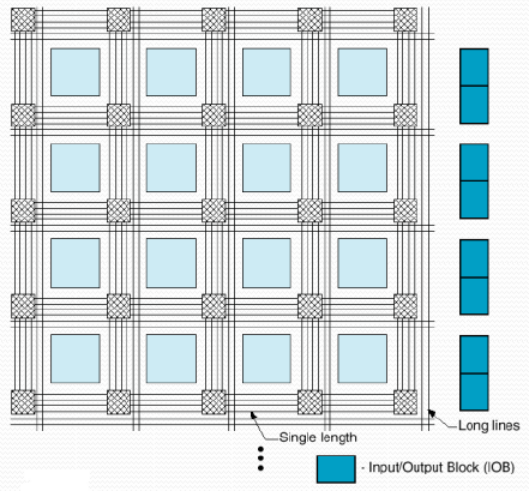


A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Fuente: MORALES, Iván. Introducción al diseño de *hardware* con FPGA utilizando VHDL. http://labelectronica.weebly.com/uploads/8/1/9/2/8192835/presentaci%C3%B3n_fpga.pdf Consulta: 20 de septiembre de 2016

Regularmente, los CLBs se encuentran compuestos por LUTs de cuatro entradas, un sumador completo, flip-flops tipo D y multiplexores. Estos bloques pueden tener modos aritméticos, lógicos puros y combinados en dependencia de los multiplexores de acarreo existentes, regularmente de la configuración que estos tengan.

Figura 10. **Enrutamiento entre CLBs**



Fuente: MORALES, Iván. Introducción al diseño de hardware con FPGA utilizando VHDL. http://labelectronica.weebly.com/uploads/8/1/9/2/8192835/presentaci%C3%B3n_fpga.pdf Consulta: 20 de septiembre de 2016

Existen bloques duros, los cuales expanden las capacidades del FPGA para incluir funcionalidades de alto nivel fijas en el silicio. Al incluir estas funciones embebidas en el silicio se reduce el área requerida e incrementa la velocidad de funcionamiento. Se puede mencionar bloques de procesamiento digital de señal (DSP), multiplicadores, procesadores embebidos, memorias embebidas o bien salidas lógicas de alta velocidad. En cuanto a sincronía se dice que toda la circuitería dentro de un FPGA es síncrona y que requiere una

señal de reloj. Los FPGAs contienen rutas globales y regionales dedicadas a las señales de reinicio (*reset*) y reloj; también contienen circuitos como bucles de enganche de fase tanto análogos como digitales (PLL, DLL) y componentes para sintetizar nuevas frecuencias de reloj. Esto se debe a que los diseños pueden ser mucho más complejos y necesitar varias frecuencias de reloj de distintas fases en diferentes partes del circuito; por tanto, es necesario crear dominios de reloj con especial cuidado en los cruces de dominio de reloj (CDC), puesto que esto puede generar meta estabilidad entre los estados de reloj a través del tiempo.

Otro aspecto muy importante a considerar es cómo definir el comportamiento de la FPGA. Esto regularmente se hace a través de un lenguaje de descripción de hardware (HDL) o algún diagrama esquemático. Cada opción tiene sus ventajas. La descripción con lenguaje es más apropiada para sistemas con grandes estructuras, permite la especificación cuantitativa de los módulos en el proceso de instanciación sin tener que crear gráficamente cada uno de ellos en forma individual. La mayor ventaja del diagrama esquemático es que permite una mejor visualización del diseño.

Después del ingreso, sea por diagrama esquemático o por lenguaje, se generará una lista de interconexiones (*netlist*) que puede ser equipada en el FPGA mediante un proceso llamado colocar y enrutar (*place and route*) regularmente realizado por un software propietario de la compañía.

El usuario valida el mapa y el resultado del proceso de colocación y enrutamiento a través de un análisis de tiempo, simulación y otros métodos de verificación. Cuando el proceso de validación está completo, un archivo binario es generado y utilizado para la conFiguración o reconFiguración del FPGA. Este

archivo es transferido al dispositivo vía interfaz serial (JTAG) o a un dispositivo de memoria interna como una EEPROM.

1.4.1. HDL

Los lenguajes de descripción de hardware más conocidos son VHDL y Verilog. Son muy parecidos con diferencias sutiles; sin embargo, pueden ser comparados a programar en lenguaje ensamblador (*assembler*) en un computador. Por lo tanto, para simplificar el diseño de sistemas complejos existen librerías con funciones complejas predefinidas y circuitos que han sido probados y optimizados para acelerar el proceso de diseño. Estos circuitos predefinidos son regularmente llamados Núcleos de propiedad intelectual (IP Cores).

Inicialmente, en un típico flujo de diseño, la descripción a nivel de transferencia de registros (RTL) en VHDL o Verilog es simulada creando pruebas llamadas *test bench* para observar los resultados del sistema. Entonces, después de que el motor sintetice el mapeo del diseño a una lista de enrutamiento (*netlist*), este es traducido a una descripción a nivel de compuertas, donde la simulación es repetida para confirmar que la síntesis fue exitosa. Finalmente, el diseño es programado en la FPGA.

1.4.2. Síntesis de alto nivel

La síntesis de alto nivel o *High Level Sintesis,* llamada HLS de aquí en adelante, comienza con una especificación del problema a alto nivel donde el comportamiento realmente no contempla aspectos como, por ejemplo, tipo de interfaces o capa de relojes digitales. HLS generalmente acepta como entrada lenguajes tales como ANSI, C, C++, Matlab. El código es analizado,

arquitecturalmente delimitado y catalogado. De esta forma se crea un nivel de transferencia de registros (RTL) en un lenguaje de descripción de hardware (HDL), el cual regularmente, después de esto, es sintetizado a un nivel de compuertas con una herramienta de síntesis lógica.

La meta principal del HLS es permitir a los diseñadores de hardware construir y verificar el mismo, permitiéndoles mejor control sobre la optimización de la arquitectura del diseño; esto mediante la idea de que el diseñador únicamente se preocupe de describir el diseño a alto nivel. Es decir, existe un mayor enfoque en el funcionamiento mientras que la herramienta realiza la implementación del RTL.

1.5. Sistema embebido

Es un sistema de aplicación específica, de pequeño tamaño, bajo costo y regularmente de funcionamiento en tiempo real. Regularmente se fabrican a escalas bastante grandes; son electrónica de consumo y, por lo tanto, también son un producto final o forman parte de un producto final: de allí el tamaño de la escala de fabricación. Los sistemas embebidos están compuestos por lo regular por alguna unidad de procesamiento, bien sea un procesador de señales (DSP), un microprocesador, un microcontrolador o cualquier unidad que aporte capacidad de cómputo y memoria, la cual puede ser interna y/o externa.

Son también comunes dispositivos de interfaz, regularmente con presentación de datos y, si es necesario, algún método de entrada, sensores, actuadores, módulos de entrada y salida, módulos de reloj y circuitería, módulo de energía; finalmente, comunicación entre todos estos módulos y partes del sistema.

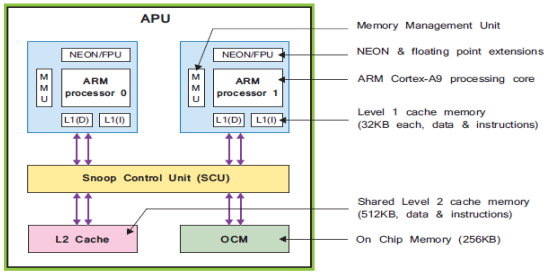
2. SISTEMA ZYNQ

2.1. Sistema de procesamiento

Todos los sistemas ZYNQ tienen la misma arquitectura, componente de un sistema de procesamiento de base, un procesador doble núcleo ARM Cortex-A9. Este es un procesador “duro”; es decir, existe como un elemento de silicio dedicado y optimizado. Se puede utilizar procesadores por software; de hecho existen aplicaciones en las cuales se utilizan ambos tipos de procesadores; los últimos son para funciones de coprocesamiento o de bajo nivel.

El sistema de procesamiento no se encuentra únicamente conformado por un procesador ARM; también cuenta con interfaces, memorias, interconexiones, circuitería de generación de reloj y demás recursos asociados para formar una Unidad de Procesamiento de Aplicación (APU). En la Figura 11 se muestra un diagrama simplificado de dicha unidad.

Figura 11. **Unidad de aplicación simplificada**

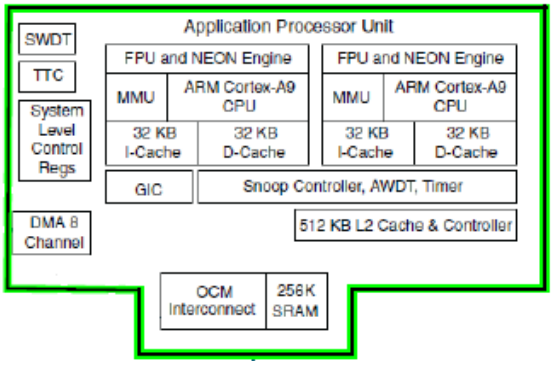


Fuente: CROCKETT, Louise. *The Zynq Book*. p. 18.

2.1.1. Unidad de procesamiento de aplicación

El sistema de procesamiento de aplicación se encuentra compuesto por dos núcleos de procesamiento ARM —cada uno con sus unidades computacionales asociadas— un motor de procesamiento de media (MPE), una unidad de punto flotante (FPU), una unidad de manejo de memoria (MMU), memoria caché de nivel 1 segmentada en dos secciones, tanto para instrucciones como para data; memoria caché nivel 2 añadido a la memoria cache L2. También existe un poco más de memoria en el chip (OCM), todo ello interconectado por la unidad de control *snoop* (SCU). En resumen, el SCU interconecta los dos núcleos, cada uno asociado a su propio MPE, MMU, FPU y caché L1 con la memoria caché L2, la OCM y la tela lógica programable (PL). A continuación un diagrama más detallado del APU.

Figura 12. **Unidad de aplicación**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 17.

2.1.1.1. ARM Cortex-A9

Debido al proceso de diseño, modelo de negocio de ARM entre otros factores externos, la documentación expuesta es aplicada y expositiva del Cortex-A9 embebido en el sistema ZYNQ. Lo anterior es a causa de la selección de algunos componentes por parte del OEM, en este caso, Xilinx, lo que incide directamente en las capacidades físicas del dispositivo.

2.1.1.2. Motor de procesamiento de media

NEON *engine* es el motor de procesamiento de media utilizado en el APU. Existe uno asociado a cada núcleo y es una extensión adicional en el procesador ARM. Provee capacidades Múltiples datos de instrucción única (SIMD) para aceleración estratégica, regularmente de media y también para algoritmos de procesamiento digital de señales. Debido a las capacidades de paralelismo de una operación para vectores de diferente tamaño, las instrucciones NEON son realmente extensiones al conjunto de instrucciones estándar de ARM, los cuales se pueden utilizar de forma explícita en la programación o bien ser inferidas por el compilador.

El SIMD es una forma de computación paralela definida en 1966 dentro de la taxonomía de Michael J. Flynn. Dicha taxonomía es una clasificación de métodos de computación que dependen del flujo de datos e instrucciones en la cual se encuentran cuatro clasificaciones principales:

- *Single Instruction Stream, Single Data Stream* (SISD)
- *Single Instruction Stream, Multiple Data Stream* (SIMD)
- *Multiple Instruction Stream, Single Data Stream* (MISD)
- Multiple Instruction Stream, Multiple Data Stream (MIMD)

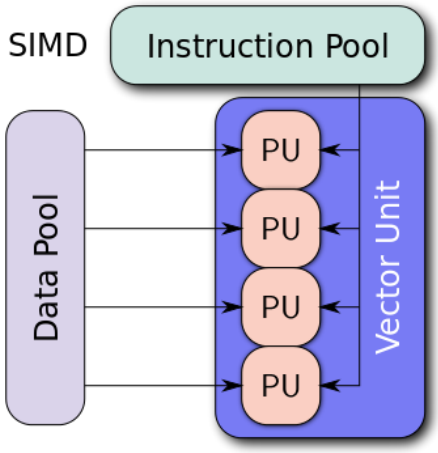
Se dice que estas clasificaciones son dependientes de los flujos de datos e instrucciones. Pueden ser múltiples o únicos, lo que sucede cuando se tiene un mismo dato al cual se realizan múltiples operaciones (MISD), como por ejemplo: multiplicarlo, sumarlo, restarlo con valores predeterminados. Esto es bastante aplicado al análisis de señal, en particular de una señal unidimensional como el sonido (sin tomar en cuenta el eje del tiempo, solo la amplitud). Simplificado esto, es como tener un solo flujo de datos y una sola instrucción también aplicable a procesamiento de señal.

También es de considerar el caso donde existen múltiples flujos de datos y de instrucciones u operaciones. Esto representa un coste computacional grandísimo, no solo en el ingreso de los datos sino también en el procesamiento de cada instrucción. Son necesarios múltiples núcleos de procesamiento regularmente funcionales de forma asíncrona e independiente para llevar a cabo estas tareas. Esto también impacta el coste económico de implementación. Un ejemplo de un sistema MIMD es el Intel Xeon Phi, el cual es toda una familia de procesadores. En la actualidad estos pueden llegar a tener hasta 72 núcleos y una frecuencia una base de operación de 1.5GHz, además de una potencia de diseño térmico (TDP) de aproximadamente 245 watts; esto es, la potencia promedio que disipa el procesador cuando funciona a su frecuencia base y todos los núcleos se encuentran activos. Estos núcleos también cuentan con memoria integrada de 16GB y diseñados para un sistema de trabajo escalable.

Ahora bien, para Instrucción única y múltiples datos (SIMD) es una forma de paralelismo computacional. Este fue creado al principio de la década de 1970 aplicado al procesamiento de vectores. Aunque hoy las arquitecturas de procesamiento de vectores ya se consideran como campo de estudio aparte, a pesar de estar basado en una única instrucción, puede procesar todos los

elementos de un vector a la vez. Es la habilidad donde un mismo valor es añadido o sustraído de un largo número de datos donde SIMD toma ventaja. Esta operación es bastante común en manejo de media, si se decidiera alterar el brillo de una imagen[1]. Operando cada vector de color RGB por un valor constante (sea añadiendo o substrayendo) este altera el brillo de la imagen.

Figura 13. **Diagrama de SIMD**



Fuente: SIMD. https://en.wikipedia.org/wiki/SIMD. Consulta: 22 de septiembre

2.1.1.3. Unidad de punto flotante

La unidad de punto flotante se define como un coprocesador matemático que forma parte de la unidad central de procesamiento. Esta permite la realización de operaciones matemáticas de forma dedicada, ya que de otra forma si estas operaciones se realizaran todas a través de instrucciones en la ALU del procesador central, llevaría una cantidad de tiempo mucho mayor, lo que afectaría el desempeño del sistema en general. Los datos de punto flotante

[1] Nota: Las imágenes se pueden ver como tres matrices representativas cada una indicando un valor a una porción de color del pixel, Rojo, Verde y Azul.

son considerados como primitivos y las operaciones con estos se vuelven recurrentes, cada vez más comunes.

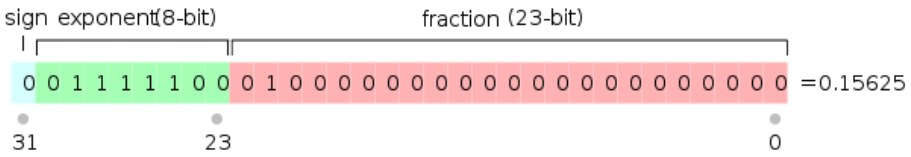
Existen diferentes tipos de precisión para el punto flotante, los más utilizados son:

- *Half-Precision*, 16 bits, 1 de signo, 5 de exponentes y 10 de fracción.
- *Single-Precision*, 32 bits, 1 de signo, 8 de exponente y 23 de fracción.
- *Double-Precision*, 64 bits 1 de signo, 11 de exponente y 52 de fracción.
- *Quadruple-Precision*, 128 bits, 1 de signo, 15 de exponentes y 114 de fracción.

Las formas de representación de punto flotante se dan en una cantidad de bits determinada (16, 32, 64, 128) que se subdivide en tres campos: un bit de signo, un campo de exponente y uno de fracción o significando. Cada uno de estos con una función determinada: el bit del signo es el encargado de representar el signo del número representado y se toma 0 para positivo y 1 para negativo. El exponente lleva una modificación a causa del hardware en el que será utilizado posteriormente; se le añade al número exponente resultante de la notación científica normalizada binaria una cantidad constante que varía únicamente entre representaciones de precisión. Para *Single-Precision* se añade 127 de forma binaria a este número. Esta suma se manifiesta como un corrimiento y es para realizar de forma más fácil comparaciones de complemento a dos, las cuales realiza el hardware que trabaja más adelante con el exponente. El último grupo de bits es la fracción o el significando, el cual es la representación del número binario de la parte fraccionaria del número decimal. Esta se toma después del primer 1 de izquierda a derecha, se

considera como implícito y lo que se escribe son los dígitos binarios, mostrados en la figura 14.

Figura 14. **Single-Precision**



Fuente: Punto flotante. https://en.wikipedia.org/wiki/Single-precision_floating-point_format. Consulta: 22 de septiembre 2016

La unidad de punto flotante del Cortex A9 es una implementación de la arquitectura de punto flotante del ARMv7. Esta es una VFPv3-D16, la cual provee bajo costo y alto rendimiento computacional. La VFPv3-D16 soporta todos los modos de direccionamiento y operaciones descritas en el Manual de referencia de Arquitectura (ARM). Entre las operaciones apoyadas se encuentra soporte completo de precisión simple y doble, así como sumar, restar, multiplicar, dividir, multiplicar y acumular, y operaciones de raíz cuadrada. La precisión doble de punto flotante no es soportada por el MPE, por lo tanto, para el cálculo es necesario utilizar el FPU pero esta no contiene modo vector ni soporte para SIMD.

Entre las aplicaciones más comunes se puede mencionar:

- En asistentes personales y teléfonos inteligentes para gráficas, compresión y descompresión de voz, interfaces de usuario, interpretación de Java y compilación *JIT*.
- En máquinas de juego para gráficas tridimensionales y audio digital.

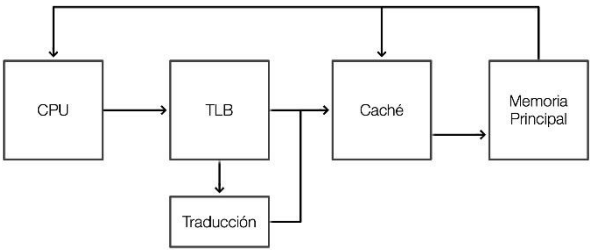
- Impresoras y periferias multifuncionales (MFP), controladores para representaciones de color de alta definición.
- Aplicaciones automovilísticas para manejo de motor.

2.1.1.4. Unidad de manejo de memoria

Es una unidad de hardware por la cual pasan todas las referencias a memoria, para que primero se realice una traducción de direcciones de memoria virtual a direcciones de memoria física. Entre otras atribuciones también se encuentran la protección de memoria, control de caché y arbitrio de bus. Regularmente se implementa como parte de la unidad central de procesamiento.

Las Unidades de manejo de memoria (MMU, *Memory Management Unit*) modernas típicamente dividen en páginas el espacio de las direcciones virtuales que son utilizadas por el procesador. Esta división se hace visible en los últimos bits de la dirección, los cuales son constantes para una determinada página y los bits más altos son la dirección dentro de esa página.

Figura 15. **Búsqueda TLB**

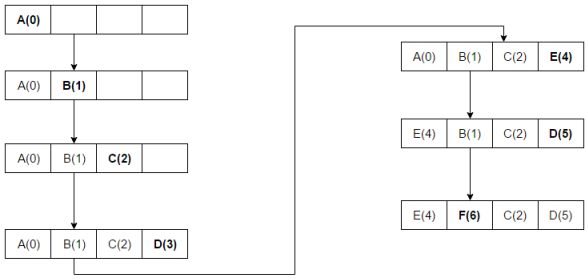


Fuente: elaboración propia

Los PTE incluyen información acerca de modificación, acceso y tipos de procesos: la información de modificación se guarda en *Dirty Bit*, el cual es un bit asociado a un bloque de memoria e indica si este ha sido modificado o no, y si la modificación ha sido guardada en almacenamiento. Este bit es activado cuando el procesador escribe (modifica) en el bloque de memoria.

La información de accesos se tiene en el bit acceso (*accessed bit*), el cual es muy útil para la implementación de un algoritmo de los menos utilizados recientemente (*Least Recently Used,* LRU) aplicado al reemplazo de las páginas, a manera de tener únicamente las más utilizadas de forma reciente. LRU es realmente una familia de algoritmos de caché (se denominan así ya que implementan una política de reemplazo de caché), que reemplazan los accesos que se han tenido a páginas con los accesos más recientes y les proporciona una clasificación de prioridad mayor. Mientras mayor sea este número, menos probabilidades tienen de ser reemplazados.

Figura 16. **Ejemplo de LRU**



Fuente: LRU. https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU. Consulta: 23 de septiembre de 2016

La información del tipo de proceso, sea en modo de usuario o de supervisor, puede leer o modificar y si debe ser guardado en caché o no. Los PTEs pueden prohibir el acceso a una página virtual cuando no existe una localidad física en RAM para la página virtual. Regularmente, en estos casos el MMU avisa al procesador del error y es tarea del sistema operativo manejar esta excepción, algunas veces trata de buscar algún espacio libre en RAM para hacer un nuevo PTE relacionado a la dirección virtual inicial. Otra situación ocurrente es cuando no existe espacio en RAM disponible para crear este nuevo PTE y, por lo tanto, es necesario escoger alguna página existente llamada “víctima” para que, utilizando algún algoritmo de reemplazo, la “víctima” sea guardada en disco y la nueva dirección virtual sea funcional en el espacio físico de la página “víctima”. Este proceso es conocido como paginación.

Existen otra clase de errores o señales de error que pueden generar una MMU, por ejemplo: condiciones de error de acceso ilegal, errores de página no válida. Esto son causados por accesos a memoria ilegales o inexistentes, los cuales conducen a errores de bus o de segmentación.

2.1.1.5. Memoria caché

En la forma más general, el término “caché” en informática hace referencia a un componente de software o hardware que guarda datos para que las futuras peticiones de estos datos sean servidas mucho más rápido.

De forma más específica, la caché es un tipo de memoria de acceso rápido que funciona como un intermediario entre el microprocesador y la memoria de acceso aleatorio. Crea y almacena copias de los datos más recientemente utilizados, reduciendo los accesos a la memoria principal y el tiempo de acceso a datos.

Cuando el procesador accede por primera vez a un dato, lo primero que hace es copiar éste a caché para reducir el tiempo de acceso al mismo, ya que le puede ser útil múltiples veces a lo largo de una operación; así mismo, si el procesador necesita leer o escribir alguna locación de memoria, siempre verifica que no exista una copia de este dato en la caché. Debido a este uso tan demandante, a la cantidad de datos que se manejan y al tamaño reducido de la memoria caché, es necesario manejar políticas de reemplazo. Estas se aplican mediante algoritmos de reemplazo de datos, entre los que se puede mencionar:

- Algoritmo de Bélády
- Primero adentro primero fuera (*First In First Out,* FIFO)
- Último adentro primero fuera (*Last In First Out,* LIFO)
- Menos usado recientemente (*Least Recently Used,* LRU)
- Más usado recientemente (*More Recently Used,* MRU)
- Pseudo-LRU (PLRU)
- Remplazo Aleatorio (*Random Replacement,* RR)
- Usados con menos frecuencia (*Least-Frecuently Used,* LFU)

- Caché de reemplazo adaptable (*Adaptive Replacement Cache,* ARC)
- Reloj con reemplazo adaptable (*Clock Adaptive Replacement,* CAR)
- Algoritmo Multi Cola (*Multi Queue, MQ*)

En la memoria caché los datos se alojan en diferentes niveles dependiendo de la frecuencia de uso que tengan. Estos niveles son:

- Memoria caché nivel 1 (L1): se encuentra en el núcleo del microprocesador. Se accede a datos importantes y de uso frecuente; puede tener un tamaño de 256 KB y se divide en dos segmentos: el caché de datos (*Data Cache)* que almacena los datos más frecuentemente utilizados y el caché de instrucciones (*Instruction Cache),* que almacena las instrucciones más frecuentemente utilizadas.
- Memoria caché nivel 2 (L2): forma parte del procesador mas no se encuentra en su núcleo. Almacena datos de uso frecuente, es más lenta que la L1 pero más rápida que la RAM. Se divide en dos segmentos: caché exclusivo, que elimina los datos solicitados de L2, y el caché inclusivo, en el cual los datos solicitados permanecen en L2.
- Memoria caché nivel 3 (L3): genera una copia de L2, es más lenta que esta pero más rápida que la RAM. Su trabajo es agilizar el acceso tanto a datos como a instrucciones que no se localizaron en L1 y L2.

En el procesador que se trabajará se encuentra una memoria caché L1 para cada núcleo, la cual consta de 32 KB para datos y 32 KB para instrucciones; una memoria caché L2 de 512 KB, sin caché L3 existente puesto que se recurre a una extensión directa de memoria SRAM y un segmento de memoria en chip (*On Chip Memory,* OCM).

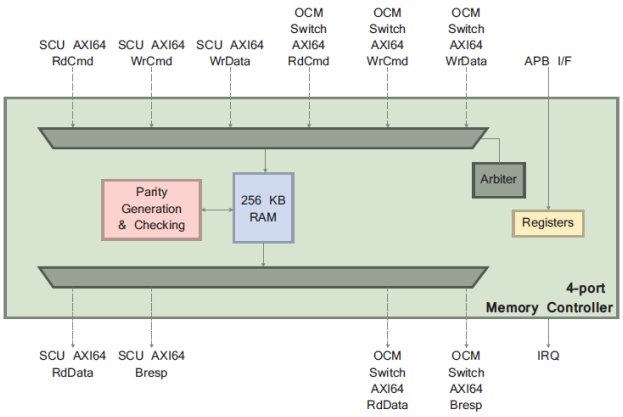
2.1.1.6. Memoria en chip

La memoria en chip (OCM) se utiliza regularmente en sistemas en chip. Son memorias rápidas en velocidad pero de pequeña capacidad; no son tan rápidas como las caché L1 pero buscan su lugar al ser mucho más rápidas que la memoria RAM principal. En este dispositivo se cuenta con una memoria estática de acceso aleatorio (*Static Random Access* Memory, SRAM) de 256 KB, 128 KB de memoria únicamente de lectura (*Read Only* Memory, ROM) e interconexiones mediante un controlador a la interconexión central, a la lógica programable a través de puertos de alto desempeño y, de forma interna, a la unidad controladora de registro furtivo.

Es en esta ROM donde la memoria de inicio (BootROM) reside. El OCM soporta dos puertos esclavos AXI 3.0 de 64 bits; uno es dedicado al acceso de CPU/ACP a través del APU SCU y el otro es compartido por todos los buses maestros entre el sistema de procesamiento y la lógica programable. Al implementar la RAM con doble ancho (128 bits) se habilita un alto rendimiento en la lectura y escritura en AXI para acceso a memoria RAM. Ahora bien, para hacer uso de esta mejora en el rendimiento, las aplicaciones del usuario deben utilizar direcciones alineadas a 128 bits y ráfagas AXI pares.

El BootROM no es visible al usuario, ya que está reservado para uso exclusivo de los procesos de inicio. El BootROM es el primer software en correr en el APU, se ejecuta en el primer núcleo mientras el segundo núcleo realiza la instrucción de esperar un evento (*Wait for event*, WFE).

Figura 17. **Diagrama de bloques OCM**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 209.

2.1.1.7. Unidad controladora de registro

La unidad controladora de registro furtivo (*Snoop Control Unit,* SCU) interconecta los dos procesadores Cortex-A9 al sistema de memoria a través de interfaces extensibles avanzadas (*Advanced Extensible Interface,* AXI) y tiene las siguientes funciones:

- Mantener la coherencia de los datos del caché entre los procesadores del Cortex-A9.
- Iniciar los accesos AXI a memoria de L2.
- Arbitrar entre las solicitudes de acceso a L2 de los procesadores.
- Manejar accesos del puerto acelerador de coherencia (*Acceleration Coherency Port,* ACP).

El SCU del Cortex-A9 no soporta manejo por hardware de coherencia del caché de instrucciones.

El SCU cuenta con registros para información, configuraciones, restricciones y atributos, los cuales son:

- Registro de control
- Registro de configuración
- Registro de control de poder del CPU
- Registro de estado seguro
- Registro de filtrado de dirección de inicio
- Registro de filtrado de dirección de fin
- Registro de control de acceso al SCU
- Registro de control de acceso no seguro al SCU

Una de las principales tareas del SCU es velar por los accesos a memoria caché y el OCM, la coherencia y repetición de los datos entre estas y los accesos de forma externa que se puedan tener por parte de la lógica programable a través del puerto acelerador de coherencia (*Accelerator Coherency Port,* ACP)

2.1.1.8. Puerto acelerador de coherencia

El puerto acelerador de coherencia (*Accelerator Coherency Port,* ACP) es un puerto esclavo opcional AXI de 64 bits. Puede ser conectado a periféricos AXI maestros no cacheables tales como accesos directos a memoria (*Direct Memory Access,* DMA), o bien dispositivos que aprovechen la cualidad de tener acceso directo a la memoria caché.

Esta interfaz AXI esclava compatible con la especificación de Arquitectura avanzada de bus para microcontrolador (*Advanced Microcontroller Bus Architecture 3,* AMBA 3) ubicada en el SCU, provee un punto de interconexión para un rango de sistemas maestros para que, por razones de desempeño, consumo energético o simplificaciones de software, estos dispositivos maestros sean interconectados directamente al núcleo del procesador Cortex-A9.

Los aspectos más importantes a considerar de ACP en cuanto a modo de funcionamiento son:

- Peticiones ACP: las peticiones de lectura y escritura llevadas a cabo en el ACP se comportan de diferente manera en dependencia si la petición es coherente o no.
- Reloj de interfaz ACP: aunque es una interfaz AXI, el ACP establece relaciones de ratio de “medio reloj” entre el reloj AXI y el reloj de la SCU. Esto quiere decir que solo relaciones de ratio que tengan como resultado números enteros son soportados.
- Limitaciones del ACP: el ACP se encuentra optimizado en transmisión para tramas de la longitud establecida por los datos del caché y soporta una amplia variedad de peticiones AXI AMBA 3. Este encuentra dos tipos de limitaciones: las de desempeño que tienen como causa la optimización para transferencias que igualan las peticiones coherentes del procesador A9; las limitaciones por funcionalidad que tienen origen con tres tipos de transferencia AMBA 3 AXI no soportados relacionados a la exclusividad de transacciones de memoria coherente, transacciones de bloqueo total y transacciones optimizadas de memoria coherente cuando los patrones de bytes (*Byte Strobes*) no han sido especificados.

2.1.2. Interfaces externas del PS

La comunicación entre el sistema de procesamiento (*Processing System,* PS) y la variedad de interfaces externas existentes se realiza a través de entradas y salidas multiplexadas (*Multiplexed Input/Output,* MIO), lo cual provee una flexibilidad de conexión de 54 pines, dejando a entera libertad el enrutamiento entre cada pin y periferia.

También existe otro medio de comunicación entre las periferias y los pines llamada entrada y salida multiplexada extendida (*Extended Multiplexed Input/Output,* EMIO). Es muy parecida al MIO pero no es un camino directo entre la periferia y los pines; más bien atraviesa y comparte los recursos de entrada y salida de la región Lógica programable (*Programmable Logic,* PL). El EMIO se utiliza cuando una extensión más allá de los 54 pines es necesaria o si se desea interconectar alguna de estas periferias con algún bloque de hardware implementado en el PL.

La lista de las periferias incluidas en el sistema de procesamiento (*Processing System*, PS) se da a continuación:

- Interfaz periférica serial (*Serial Peripheral Interface*, SPI)
- Circuito interintegrado (*Inter-Integrated Circuit,* I2C)
- Bus CAN (*Controller Area Network,* CAN)
- Receptor transmisor asíncrono universal (*Universal Asynchronous Receiver Transmitter,* UART)
- SD (*Secure Disk*)
- USB (*Universal Serial Bus*)
- GigE (*Ethernet*)

- Entrada y salida de propósito general (*General Purpose Input/Output*, GPIO)

Se tiene dos periferias de cada tipo, excepto los GPIO, de los cuales se tiene cuatro bancos, cada uno de 32 bits.

2.1.3. Programación sistema de aplicación

El flujo de trabajo en un sistema en chip (*System on Chip*, SoC) Zynq-7000 permite la creación de aplicaciones de software en un ambiente unificado de herramientas xilinx. También se pueden utilizar herramientas de terceros, algunas muy conocidas como *GNU Compiler Toolchain* y algún entorno de desarrollo integrado (*Integrated Development Enviroment,* IDE) basado en Eclipse.

De forma general, las herramientas de software necesarias para el desarrollo y depuración de aplicaciones de software son descritas en los siguientes subtítulos.

2.1.3.1. Software IDE

Es un entorno de desarrollo integrado para aplicaciones de software que se ejecuten en el sistema de procesamiento. La principal misión del IDE es proporcionar los servicios necesarios para facilitar al desarrollador su tarea de forma integral. Existen elementos comunes en estos y son:

- Editor de código fuente
- Herramientas de construcción
- Depurador o *Debugger*

- Auto completado (*Intelli Sense*)
- Memoria de variables
- Compilador o intérprete

2.1.3.2. GNU-Toolchain

El *GNU-Toolchain*[2] es un conjunto de proyectos de software producidos en el marco del proyecto GNU. Estos proyectos contienen herramientas de desarrollo de software, las cuales son vitales para el desarrollo de sistemas operativos y aplicaciones de software para los mismos. El sistema de procesamiento forma finalmente parte de una unidad de aplicación, así que es probable que finalmente se utilice alguna opción basada en un sistema operativo acorde al procesador. Dentro de esta suite de proyectos se encuentran los siguientes:

- GNU Make
- Colección de compiladores GNU (*GNU Compiler Collection*, GCC)
- GNU Binutils
- Depurador GNU (*GNU Debugger*, GDB)
- Constructor de sistemas GNU (*GNU build system*)

Esta es la parte del software encargada de convertir el código fuente de la aplicación en un programa ejecutable. Entre las herramientas se utilizará una versión modificada de esta suite enfocada principalmente en arquitectura ARM.[3]

[2] GNU. *Toolchains*. http://elinux.org/Toolchains. Consulta: Agosto 2016.
[3] ARM, Developer. *ARM Embedded Toolchain*. https://developer.arm.com/open-source/gnu-toolchain/gnu-rm. Consulta: Noviembre 2016.

2.1.3.3. Depurador JTAG

JTAG es un sistema de hardware existente tanto en microprocesadores como microcontroladores, el cual permite realizar pruebas tanto de pines como de tarjetas de circuito impreso (*Printed Circuit Board*, PCB). Este sistema dentro del SoC permite también el monitoreo de cada pin a una determinada frecuencia, lo cual es útil para saber el estado físico de su funcionamiento sin necesidad de mediciones externas.

Esta interfaz regularmente se utiliza para reprogramar el dispositivo con el cual se esté trabajando, pero fue creada con un propósito ligeramente distinto por el grupo de acción de prueba conjunta (*Joint Test Action Group*, JTAG). La idea era realizar pruebas de los dispositivos en cada uno de sus pines, almohadillas (*pads*) o esferas y la PCB en la cual era instalados sin necesidad de utilizar un método de externo de mesura. Esta interfaz permite la lectura individual de cada pin en tiempo real (en dependencia de la tasa de funcionamiento). Será útil al momento de comparar el código para ser ejecutado y el estado físico del dispositivo, es decir, la instrucción que se ejecuta y el resultado obtenido.[4]

2.1.4. Kit de desarrollo de software

El kit de desarrollo de software (*Software Development Kit*, SDK) de Xilinx provee el ambiente en el cual aplicaciones completamente funcionales puedan ser creadas, compiladas y depuradas. Es decir, completamente funcionales a causa de todos los pasos necesarios a seguir para el desarrollo de estas aplicaciones, ya que el hardware varía según sea programada la PL. Por lo

[4] IEEE Computer Society. *IEEE Standard Test Access Port and Boundary-Scan Architecture*, Revision IEEE Std. 1149.1-1990. http://fiona.dmcs.pl/~cmaj/JTAG/JTAG_IEEE-Std-1149.1-2001.pdf. Consulta: Noviembre 2016.

tanto, obtener una aplicación funcional conlleva más herramientas y pasos que otras plataformas sin PL.

El SDK incluye un *GNU toolchain* modificado que, de igual forma, contiene un compilador GCC, depurador GDB, utilidades y librerías. También dentro del SDK se incluye un depurador JTAG, un programador flash, *drivers* para los bloques de propiedad intelectual (*Intellectual Property*, IP) de Xilinx, además de paquetes de apoyo a tarjetas (*Board Support Package*, BSP) *bare-metal* y librerías intermedias (*middleware*) para funciones específicas de aplicación.

El SDK tiene como funcionalidades:

- Manejo del proyecto
- Error de navegación
- Entorno de edición y compilación de C/C++
- Generación automática *makefile*
- Entorno integrado para depuración y perfilado de dispositivos embebidos.
- Funcionalidades añadidas a través de extensiones como código fuente y control versiones.

A causa de la flexibilidad que proporciona el sistema ZYNQ es necesario, al diseñar, tomar en cuenta muchos aspectos de arquitectura tanto del sistema físico como la arquitectura del sistema de software que se estará ejecutando. Se tomarán estas para ubicar cuáles son las que se utilizarán mediante el flujo de trabajo facilitado por las herramientas de desarrollo proporcionadas por

Xilinx. Dado este flujo de trabajo, algunas de estas decisiones pueden ser obviadas y, de igual manera, obtener un sistema funcional.

2.1.5. Consideraciones arquitecturales de software

En el sistema ZYNQ hay un procesador doble núcleo ARM Cortex-A9. Una de las consideraciones a determinar es el proceso múltiple, si este se llevará a cabo de forma asimétrica (*Asymmetric Multiprocessing*, AMP) o simétrica (*Symmetric Multiprocessing*, SMP)

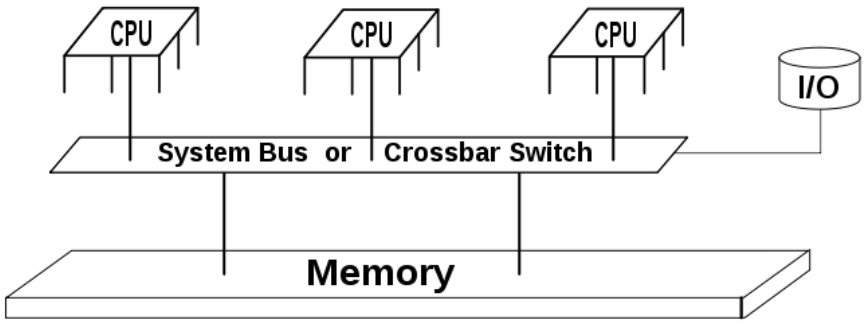
2.1.5.1. Multiproceso simétrico

En el modelo de multiprocesamiento simétrico cada procesador del sistema multiprocesador es idéntico; todos están interconectados a una sola memoria compartida, tienen acceso a todas las entradas/salidas y ejecutan una sola imagen de un sistema operativo, en el cual el programador de tareas (*Scheduler*) del sistema operativo se encarga de asignar los procesos a cada procesador.

Usualmente, cada procesador tiene asociada una memoria privada de alta velocidad llamada caché. La interconexión entre los procesadores puede darse utilizando buses, interruptores de barra transversal o malla de conexión en el chip. Aunque la dificultad en la escalabilidad del SMP se encuentra en el ancho de banda y el consumo de potencia de las interconexiones entre procesadores, memorias y arreglos de memoria no volátil tales como discos, regularmente estas dificultades de interconexión pueden ser solventadas con diseños de malla pero sacrifican en gran manera la facilidad en programación. Esto porque la arquitectura requiere de modos de programación, uno para los CPU como tal y otro para la interconexión entre ellos. Un solo lenguaje debería no solamente

encargarse de la división de trabajo sino de la comprensión de localidad en memoria, algo complicado para diseños de malla.[5]

Figura 18. **Diagrama de bloques SMP**



Fuente: SMP. https://en.wikipedia.org/wiki/Symmetric_multiprocessing. Consulta: 2 de noviembre de 2016

Este es un modelo bastante eficiente cuando el sistema operativo seleccionado satisface los requerimientos de sistema. El sistema operativo utiliza el poder de procesamiento de múltiples procesadores de forma automática y, por lo tanto, es consecuentemente transparente para el usuario. Sin embargo, se puede especificar determinada tarea para ser realizada por determinado procesador, manejar las interrupciones por cualquier procesador libre y designar un procesador como maestro para inicializar el sistema y los demás procesadores.

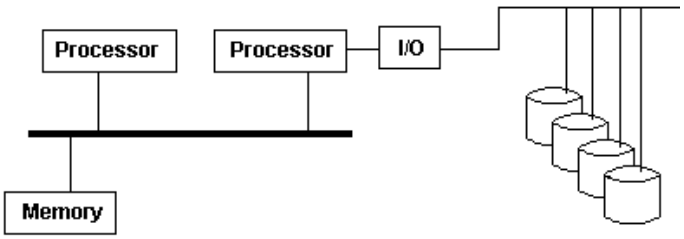
[5] KARAM, Lina et al. *Trends in Multi-Core DSP platforms*, http://users.ece.utexas.edu/ ~bevans / papers/ 2009 / multicore /MulticoreDSPsForIEEESPM Final.pdf. Consulta noviembre 2016.

2.1.5.2. Multiproceso asimétrico AMP

Es un modelo de procesamiento en el que cada procesador, en un sistema multiprocesador, ejecuta una imagen de sistema operativo diferente mientras comparten la misma memoria física. Estas imágenes pueden ser del mismo sistema operativo. Son ejecutadas de manera independiente; sin embargo, es mucho más típico encontrar diferentes imágenes de distintos sistemas operativos para este modelo de procesamiento.

Este fue el primer método para manejar multiprocesamiento antes de que el SMP existiera. En buena medida, la utilización de este modelo es separar el acceso a dispositivos de cada núcleo dependiendo del sistema operativo que cada uno esté ejecutando. Un sistema operativo formal como Linux permite la comunicación al mundo exterior mediante redes y capas elevadas de comunicación. Esto puede realizarse en capas más bajas pero tomaría más tiempo, además de ser únicamente útil si se realiza un sistema muy personalizado que también lo haría incompatible con los demás sistemas. Ahora bien, un sistema operativo más pequeño y ligero (en código) puede ser mucho más eficiente respecto a memoria y operaciones en tiempo real.

Figura 19. **Diagrama de bloques AMP**



Fuente: AMP. https://en.wikipedia.org/wiki/Asymmetric_multiprocessing. Consulta: 3 de noviembre de 2016.

Un ejemplo típico sería tener Linux ejecutándose como sistema operativo primario acompañado de un sistema operativo mucho más pequeño como FreeRTOS o un *Bare-Metal System* ejecutándose en otro núcleo. La división de dispositivos entre procesadores es un elemento crítico. Al momento de diseñar el sistema se deben tomar en cuenta aspectos como la dedicación y disponibilidad de los dispositivos a su respectivo procesador. El controlador de interrupciones está diseñado para ser compartido con múltiples procesadores; por lo tanto, un procesador es designado como maestro del control de interrupciones por que los inicializa. Finalmente, la comunicación entre procesadores es un elemento clave que permite a ambos sistemas operativos y a la división de dispositivos ser efectivos. Puede lograrse de muchas y diferentes formas incluyendo interrupciones entre procesadores, memoria compartida o bien envío de mensajes.

2.1.6. Consideraciones de sistema operativo

2.1.6.1. Sistemas *Bare-Metal*

Se refiere a un sistema de software que no contiene un sistema operativo. Dichos sistemas de software no poseen tantas características como las que se pueden encontrar en un sistema operativo; estos requieren un mayor rendimiento por parte del procesador y el sistema de software en sí, tienden a ser menos determinísticos. La brecha causada por los requerimientos de rendimiento del sistema operativo es cada vez menor, ya que la velocidad de procesamiento en sistemas embebidos es cada vez mayor. Por lo tanto, la selección de un sistema base *bare-metal* se da por la necesidad de determinismo en el sistema, cálculos realizados en tiempo real y complejidad del sistema.

2.1.6.2. Linux

Es un sistema operativo de código abierto. Puede encontrarse de forma gratuita o vendida por distribuidores en diferentes formas (se les llama distribuciones). También puede ser compilado desde los repositorios de código abierto. Este sistema operativo no es inherentemente un sistema en tiempo real.

Es un sistema operativo completamente funcional y toma ventaja de la unidad de manejo de memoria y esto, consecuentemente, lo convierte en un sistema operativo protegido, además de ofrecer capacidades de SMP para tomar ventaja de múltiples núcleos.

2.1.6.3. Sistema operativo en tiempo real

2.2. Lógica programable

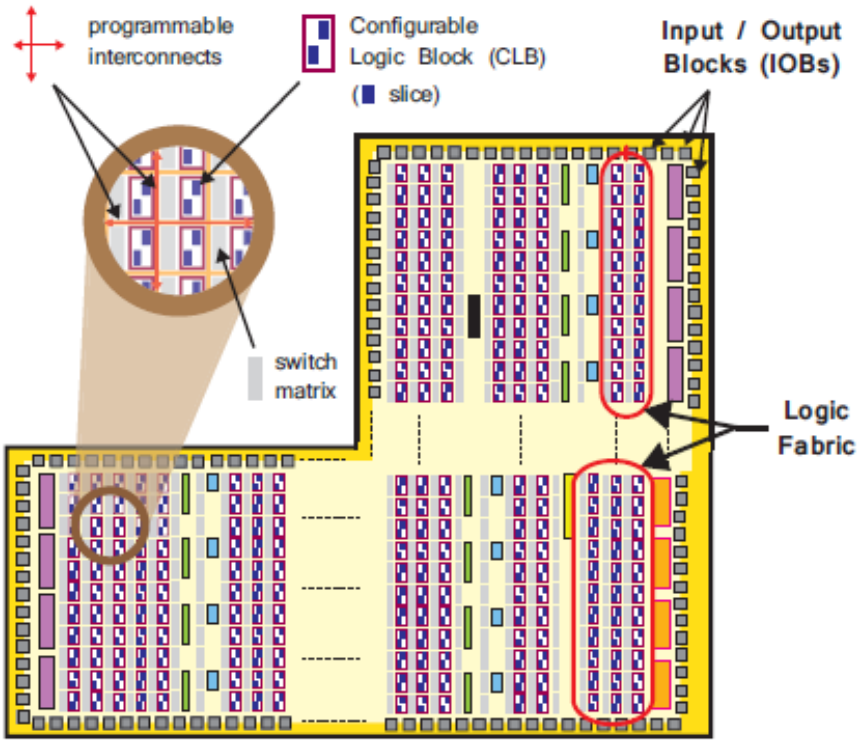
La parte PL del dispositivo Zynq se muestra en la Figura 20, con varias características resaltadas. El PL está compuesto principalmente por tejido lógico FPGA de propósito general, constituido por segmentos y bloques lógicos configurables (CLBs). También existen bloques de entrada / salida (IOB) para la interconexión (todos estos son todos términos específicos de Xilinx).

2.2.1. Tejido lógico

Este es, de hecho, la parte predominante del todo el sistema en chip. Se le llama también PL (*Programmable Logic*) y está compuesto de tejido lógico FPGA de propósito general el cual, a su vez, está compuesto de una serie de

elementos constitutivos. Se hace la aclaración que la terminología utilizada es específica de Xilinx y de los FPGAs desarrollados por ellos.

Figura 20. **Tejido lógico y sus elementos constituyentes**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 23.

- *Configurable Logic Block* (CLB): son agrupaciones pequeñas y regulares de elementos lógicos que se encuentran dispuestos en un arreglo bidimensional en el PL, conectado a otros recursos similares a través de interconexiones programables. Cada uno de estos se encuentra posicionado junto a una matriz de interruptores y contienen dos porciones o rebanadas llamadas *Slices,* como se muestra en la Figura 21.

- *Slice*: una subunidad dentro del CLB, que contiene recursos para implementar circuitos de lógica secuencial y combinacional. Como se indica en la Figura, los *Slices* están compuestos por cuatro tablas de búsqueda (*Look Up Tables*), 8 *Flip-Flops* y algunos otros elementos lógicos.

- *Look Up Table* (LUT): un recurso bastante flexible, capaz de implementar una función lógica de hasta seis entradas, una pequeña memoria solo de lectura (*Read Only Memory*), una pequeña memoria de acceso aleatorio (*Random Access Memory*) o bien un registro de desplazamiento (*Shift Register*). Las LUTs pueden ser combinadas para formar funciones lógicas, memorias o registros de desplazamiento mucho más grandes.

- *Flip-Flop* (FF): un elemento secuencial que implementa un registro de 1-bit, con funcionalidad de reinicio. Uno de los FFs puede ser opcionalmente utilizado para implementar un cerrojo (*latch*).

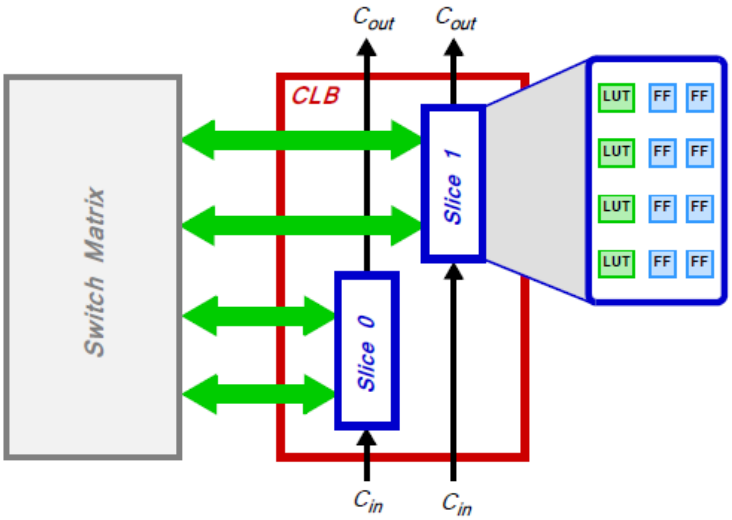
- *Switch Matrix*: una matriz de interruptores se encuentra siempre situada a la par de un CLB. Provee infraestructura de rutas flexibles para realizar conexiones entre los elementos internos del CLB o de un CLB a otro recurso en el PL.

- *Carry Logic*: los circuitos aritméticos requieren señales intermedias para propagarse entre las porciones (*Slices*) adyacentes, y esto se logra a través de la lógica de transporte. Esta comprende una cadena de rutas y multiplexores para enlazar porciones (*Slices*) en una columna vertical.

- *Input/Output Blocks* (IOBs): proveen interconexión entre recursos lógicos del PL y las "almohadillas" físicas del dispositivo usadas para conectar con el circuito externo. Cada IOB puede manejar una señal de entrada o salida de

un bit. Los IOBs se encuentran generalmente alrededor del perímetro del dispositivo.

Figura 21. **Composición interna de un CLB**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 24.

A pesar de la importancia de cada uno de estos elementos no es completamente necesario tener todo el conocimiento respecto a ellos, ya que las herramientas encargadas de hacer la síntesis también manejan la interconexión de los recursos, aunque se recomienda conocimiento general.[6]

2.2.2. DSP y BRAM

En adición al tejido general, existen dos componentes de propósito especial que son: bloques de RAM para requerimientos densos de memoria, y

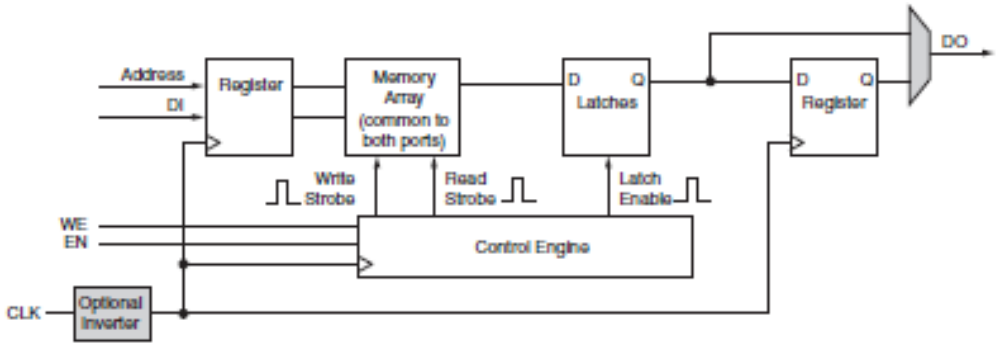
[6] Xilinx Inc., *7 Series FPGAs ConFigurable Logic Block User Guide*. http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf. Consulta: Noviembre 2016.

porciones (*Slices*) DSP48E1 para obtener alta velocidad en operaciones aritméticas. Ambos recursos se encuentran integrados en un arreglo lógico que forma una columna, empotrados en el tejido lógico regular.

Se localiza muy cercano un recurso del otro ya que el poder de procesamiento intensivo y el almacenamiento de datos son operaciones estrechamente relacionadas.

Cada bloque de RAM puede almacenar hasta 36Kb de información y puede configurarse como una RAM de 36Kb, o dos RAM independientes de 18Kb. El tamaño de palabra predeterminado es 18 bits, y en esta configuración cada RAM comprende 2048 elementos de memoria. La memoria puede ser moldeada de tal manera que pueda contener un mayor número de elementos más pequeños y no únicamente los de tamaño del bus nominal. Entonces, por ejemplo, en vez de tener 2048 elementos de 18 bits cada uno podemos tener 4096 elementos de 9 bits cada uno.

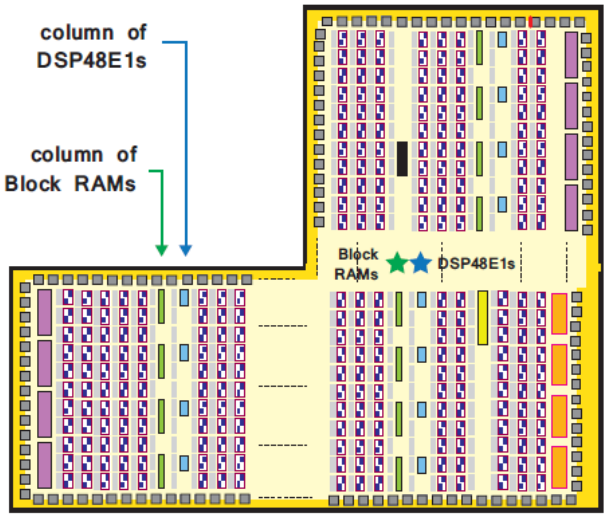
Figura 22. **Diagrama lógico de un puerto BRAM**



Fuente: Xilinx Inc. *7 Series FPGAs Memory Resources*. p. 20.

Se habla de requerimientos de memoria densos ya que un bloque de RAM se traduce a una amplia cantidad de data que puede ser almacenada en un pequeño espacio físico, dentro de un elemento de memoria dedicado y optimizado. Otra alternativa es realizar una RAM distribuida, la cual es construida con las tablas de búsqueda (LUTs) que se encuentran en el tejido lógico. Ahora bien, un número significativo de estas tablas de búsqueda es requerido para crear una memoria de tamaño comparable. Estas tablas de búsqueda se encuentran dispersas a lo largo del tejido lógico; por lo tanto, el área de implementación tiene resultados negativos en la implementación final puesto que esta sufre restricciones de tiempo y rendimiento, a causa del incremento de lógica utilizada y de retardos de rutina.[7]

Figura 23. **Ubicación física de DSP y BRAM**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 26.

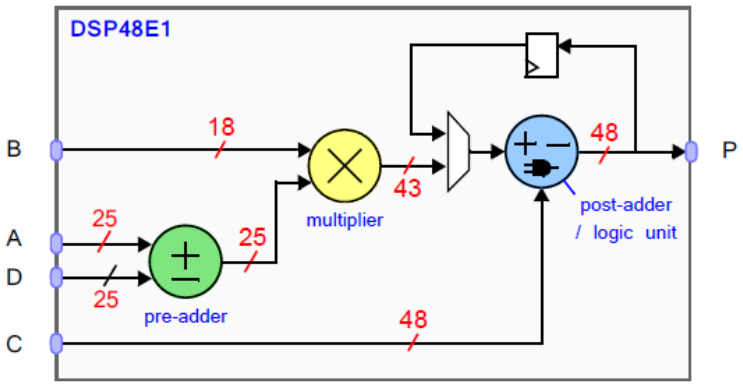
[7] Xilinx Inc., *7 Series FPGAs Memory Resources*. http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf. Consulta: Diciembre 2016.

Las tablas de búsqueda (LUTs) en el tejido lógico pueden ser utilizadas como operadores aritméticos de cualquier longitud arbitraria. Sin embargo, estas son mucho más aplicables como operadores aritméticos de longitud de datos corta, puesto que con longitud de datos larga puede tener una gran huella en el área utilizada y, por lo tanto, tener problemas de enrutamiento y frecuencias de reloj por debajo de lo óptimo.

Los DSP son porciones (*slices*) especializados para la implementación de aritmética de alta velocidad de señales de medio o largo ancho de palabra. Son recursos de silicio completamente dedicados que se componen de un presumador/sustractor, un multiplicador y un sumador/sustractor con una unidad lógica donde el máximo del ancho de palabra aritmética se encuentra marcado.

El DSP48E1 proporciona una mayor flexibilidad y utilización, mejora de la eficiencia de aplicaciones, reducción del consumo total de energía y aumento de la frecuencia máxima. El alto rendimiento permite a los diseñadores implementar múltiples operaciones en un solo DSP48E1 utilizando métodos de multiplexación de tiempo. Dicha multiplexación también soporta la alternación dinámica de computación. Esta función puede ser cambiada de ciclo en ciclo con base en los requerimientos. Varias computaciones simultáneas son posibles al utilizar 12 o bien todos los operadores aritméticos. Esto se logra a través de una entrada de modo operación (OPMODE), la cual conFigura los multiplicadores internos. Esta no se muestra completamente en el diagrama de la Figura 24; sin embargo, determina la funcionalidad aritmética que se implementa. En este espacio también existe la capacidad de hacer procesamiento SIMD al implementar dos o cuatro operaciones más cortas de adición, sustracción o acumulación de 24 o 12 bits, respectivamente.

Figura 24. **Diagrama de capacidades aritméticas del DSP48E1**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 27.

Las longitudes de palabras aritméticas estándar marcadas en la Figura 24 son adecuadas para la mayoría de requerimientos, pero también pueden ampliarse al combinar varios DSP48E1 si es necesario. La aritmética compleja se puede realizar, de nuevo al combinar DSP48E1. Las longitudes de palabra también son adecuadas para implementar la aritmética de punto flotante.

Como resultado de estas propiedades, los DSP48E1 son adecuados para una variedad de aplicaciones en el procesamiento de señales y más. Uno de sus usos más comunes es implementar filtros de Respuesta de impulso finito simétricos, que se usan comúnmente en tratamiento digital de señales (DSP) y comunicaciones digitales. El presumador asegura que cada DSP48E1 pueda implementar dos derivaciones (*taps*) del filtro, y filtros enteros pueden ser formados con cascadas de DSP48E1, sin la necesidad de utilizar lógica general

del PL. Esto proporciona una implementación de alto rendimiento y altamente eficiente para los cálculos fundamentalmente importantes en DSP.[8]

2.2.3. GPIO

A toda la construcción de entrada y salida de propósito general (*Input Output Block*, IOB) del dispositivo Zynq se le conoce de forma colectiva como *SelectIO*, y se organizan en bancos de 50 IOB cada uno. Cada IOB contiene una almohadilla (*pad*), que proporciona la conexión física al mundo exterior para una única señal de entrada o salida.

Los bancos de entrada o salida se categorizan como Alto Rendimiento (HP) o Alto Rango (HR) y soportan una gran variedad de estándares de I/O y voltajes. Las interfaces HP están limitadas a voltajes de 1.8V y se utilizan normalmente para interfaces de alta velocidad como memoria y otros chips, mientras que las interfaces HR permiten voltajes de hasta 3.3V y satisfacen una mayor variedad de estándares I/O. Soporta también conexiones de par diferencial así como las de una sola conexión[9].

2.3. Interfaces PS y PL

En los párrafos anteriores se ha demostrado el atractivo del sistema Zynq no únicamente por las propiedades de los sistemas que lo conforman, sino en las cualidades de comunicación entre ellas para crear sistemas completos integrados y completos. El factor clave en este aspecto es un conjunto de

[8] Xilinx Inc. *7 Series DSP Slice User Guide*. http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf. Consulta: Diciembre 2016.

[9] Xilinx Inc. *7 Series FPGAs SelectIO Resources*. http://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf. Consulta: Diciembre 2016.

conexiones AXI altamente especificadas, formando puentes entre las partes existentes. Además de estos existen otros tipos de conexiones entre la PS y la PL, en particular las EMIO.

2.3.1. Estándar AXI, interconexiones e interfaces

AXI, que proviene de Advanced eXtensible Interface, cuya versión actual es AXI4, forma parte del estándar abierto ARM AMBA® 3.0. Muchos dispositivos y bloques IP producidos por fabricantes y desarrolladores externos se basan en este estándar. AMBA es un estándar abierto para la conexión y gestión de bloques funcionales en un sistema en chip (SoC). Facilita el desarrollo de primera mano de los diseños multiprocesador con un gran número de controladores y periféricos.

- Interfaz de concentrador coherente (*Coherent Hub Interface*, CHI)
- Extensiones de coherencia AXI (*AXI Coherency Extensions*, ACE)
- Interfaz extensible avanzada (*Advanced eXtensible Interface*, AXI)
- Bus avanzado de alto rendimiento (*Advanced High-Performance Bus*, AHB)
- Bus periférico avanzado (*Advanced Peripheral Bus*, APB)
- Bus avanzado de rastreo (*Advanced Trace Bus*, ATB)

El estándar AMBA fue originalmente desarrollado por ARM para su uso en microcontroladores. La primera versión fue publicada en 1996. Desde entonces, la norma ha sido revisada y extendida, y ahora ARM lo describe como "El estándar *de facto* para la comunicación en chip".[10] Ahora el enfoque es aplicado a SoC, incluyendo los que son basados en FPGA o bien en el caso de Zynq, el cual es un dispositivo que incluye tela programable. De hecho, AXI4 es

[10] ARM. *Amba Open Specification*. http://www.arm.com/products/system-ip/amba-specifications. Consulta: Diciembre 2016.

la tecnología de intercomunicación óptima para su uso entre arquitecturas de FPGA.

El soporte para AXI en las herramientas Xilinx existe y fue incluido por primera vez en el ISE Design Suite 12.3. Tiempo después este fue incluido de forma extensiva en Vivado Design Suite. Es importante recordar que Xilinx jugó un papel muy importante en la definición y detalle del estándar AXI4.

Los buses AXI se pueden usar con flexibilidad y en sentido general se utilizan para conectar el procesador con otros bloques IP en un sistema embebido. De hecho hay tres tipos de AXI4, cada uno de los cuales representa un protocolo de bus diferente. La elección del protocolo de bus AXI para una conexión determinada depende de las propiedades deseadas de esa conexión.

- AXI4: este tipo provee enlaces de memoria mapeada y el máximo desempeño; una dirección de memoria es enviada seguido de un cúmulo de datos, el cual puede ser de hasta 256 palabras de datos.

- AXI4-Lite: es un enlace simplificado que soporta únicamente una transferencia de data por conexión; es decir, no existe un cúmulo de datos el cual se ha enviado después de la dirección sino únicamente uno. AXI4-Lite también funciona a base de mapeo de memoria.

- AXI-Stream: se utiliza para transacciones de alta velocidad de data, la cual necesita soporte de modo ráfaga de tamaño ilimitado; en este no existe mecanismo de dirección amiento y este bus es el mejor para el flujo directo de datos entre fuente y destino.

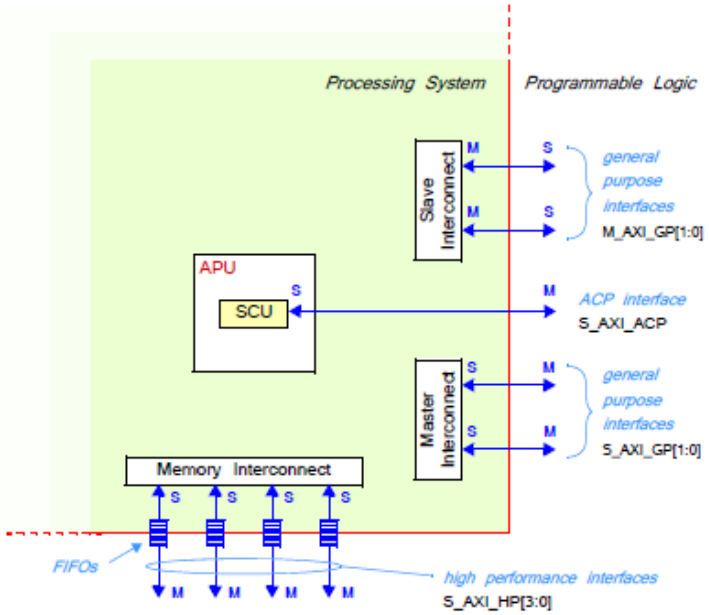
Como método de aclaración de la memoria es utilizado en las descripciones anteriores y se emplea para confirmar su significado. Si en un protocolo de memoria una dirección es especificada en conjunto con la transacción por el dispositivo maestro; es decir, dice cuál es la acción a realizar, si escribirá o leerá y la dirección y memoria en la cual debe realizar esa acción, esta dirección corresponde al espacio en memoria del sistema. En el caso de AXI4-Lite, soporta únicamente una transferencia de datos por transacción. La palabra de datos entonces es escrita o leída en la dirección especificada; en el caso de AXI4, debido a que trabaja en modo ráfaga, se especifica la dirección primeramente y después la cantidad de datos que serán transmitidos. Ahora bien, es deber del dispositivo esclavo calcular las direcciones para las demás palabras de datos. Esto es lo que lo convierte en una interconexión mucho más compleja que la anterior.

También se da la necesidad de definir y separar brevemente dos importantes conceptos: la interconexión y la interfaz, ambos existentes para el AXI:

- Interconexión: una interconexión, en el contexto AXI, es básicamente un *switch* que administra y dirige el tráfico entre las diferentes interfaces conectadas. Existen muchas interconexiones entre el PS conectadas directamente al PL y algunas otras que son exclusivas para el uso interno de la unidad de procesamiento. Las conexiones entre estas interconexiones se forman utilizando también interfaces AXI.
- Interfaz: es una conexión de punto a punto para el transporte de datos, direcciones y señales de inicio de conexión (*handshake signals*) entre dispositivos maestros y esclavos dentro del mismo sistema.

La principal interfaz entre el PS es a través de un conjunto de nueve interfaces AXI, cada una de las cuales compuesta por múltiples canales. Estos crean conexiones dedicadas entre el PL y las interconexiones del PS. En el diagrama que se muestra en la Figura 25, todas las interfaces son específicamente conectadas a interconexiones AXI, las cuales se reciben dentro del PS.

Figura 25. **Estructura de interfaces e interconexiones AXI**



Fuente: CROCKETT, Louise. *The Zynq Book*. p. 32.

La figura 25 muestra la interconexión central, la cual se encuentra dentro de la unidad de procesamiento; sin embargo, este incluye diagramas de las interconexiones e interfaces disponibles. Cuenta además con un sumario de las interfaces, las cuales se muestran con flechas que indican la dirección en la que existe flujo de datos y una descripción corta de cada interfaz. Además está escrita la letra mayúscula M en donde se encuentra el maestro, y la letra

mayúscula S donde se encuentra el esclavo. De acuerdo con la convención, el maestro está en control del bus e inicia las transacciones, mientras el esclavo responde. Los tipos y roles de interfaces son:

- AXI de propósito general: es un bus de datos de 32 bits, el cual es muy útil para un promedio de comunicación media y baja entre el PL y el PS. La interfaz es directa y no incluye ningún dispositivo de *buffering*. Existen cuatro tipos de interfaces de propósito general en total, en donde el PS es el maestro de dos y el PL el maestro en las otras dos.
- AXI *Coherency Port*: es una única conexión asíncrona entre el PL y el SCU en el APU, con un ancho de bus de 64 bits. Este puerto es utilizado para lograr coherencia entre los cachés del APU y los elementos del PL. El maestro es el PL.
- *High-Performance Port*: los cuatro puertos de alto rendimiento de interfaces incluyen *buffers* tipo FIFO para acomodar los datos en ráfaga en comportamiento de lectura y escritura, soportar una comunicación de alta tasa de transferencia entre el PL y los elementos de memoria en el PS. El ancho de los datos es cualquiera 32 o 64 bits y el PL puede ser maestro de las cuatro interfaces.

2.4. Interfaces EMIO

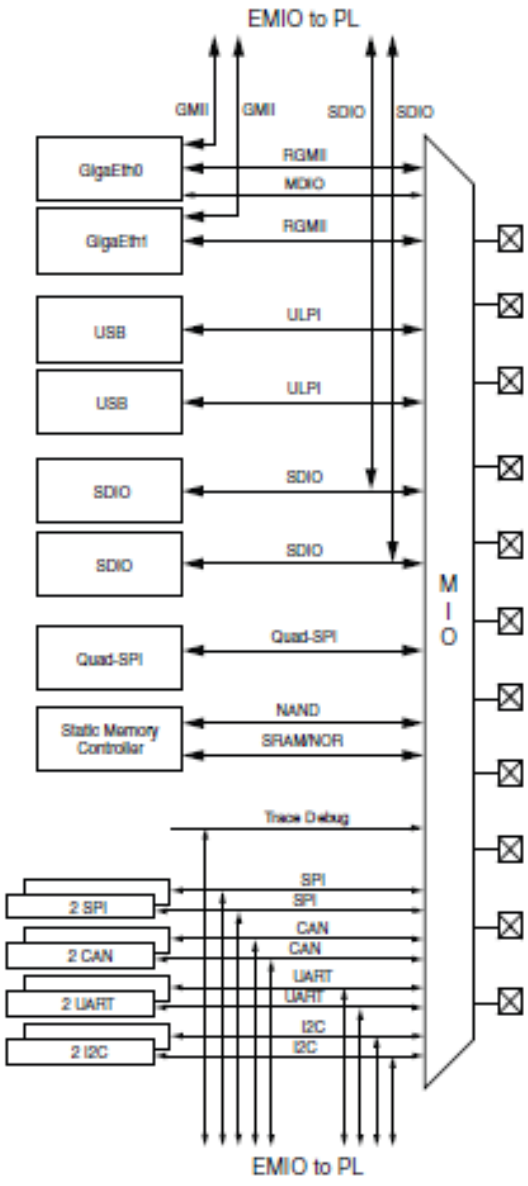
Muchas conexiones del PS pueden ser interconectadas a través de interfaces externas y de ahí a IOBs o pines de entrada o salida. A esto se le conoce como MIO extendido o EMIO.

Esto implica la transferencia de señal entre dos dominios, lo cual puede lograrse a través de un simple conjunto de conexiones. Consecuentemente, no

todas las interfaces MIO son soportadas en EMIO, y algunas, si no son, tienen capacidades reducidas.

Las condiciones están arregladas en dos bancos de 32 bits cada uno. En muchos casos las interfaces a través de EMIO pueden ser conectadas directamente a cualquier externo en el PL, especificándolo en la inclusión de entradas apropiadas en el archivo de restricciones. En este modo, EMIO puede proveer 64 entradas adicionales y 64 salidas con sus habilitadores de salida correspondientes. Otra opción es utilizar esta extensión de interfaces para comunicar el P.S. con un bloque periférico que se encuentre en el PL.

Figura 26. **Interconexiones EMIO**



Fuente: Xilinx Inc. *Zynq 7000 Overview*.
https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.
Consulta: 5 de noviembre de 2016

Otras señales que cruzan la frontera entre PS-PL incluyen temporizadores de protección (*watchdog timers*), señales de reinicio (*reset*), interrupciones y señales de interfaces DMA.

3. SÍNTESIS DIGITAL

3.1. Síntesis digital directa

Es una técnica utilizada para bloques de procesamiento de datos digital como medio generador de señales de salida con frecuencia y fase flexible, referenciada a una frecuencia fija. Esta técnica es regularmente utilizada en hardware como medio de generación de señales en forma digital para obtener productos que sean competitivos en costo, funcionalidad, alto rendimiento y pequeño tamaño como una alternativa a los sintetizadores analógicos que tienen agilidad en frecuencia.

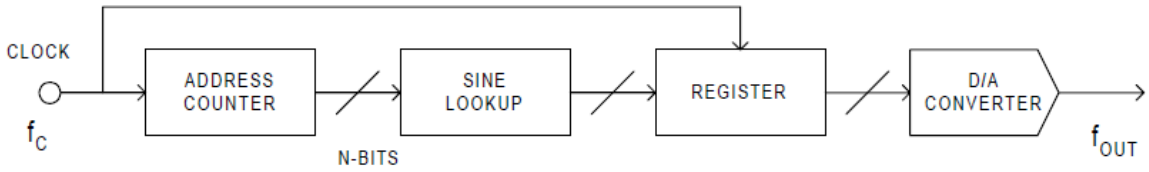
En su forma más simple, un sintetizador digital directo puede ser implementado con un reloj de referencia, un oscilador numéricamente controlado (*Numerically Controlled Oscillator*, NCO) y un convertidor digital a análogo; un contador de direcciones, una memoria programable de solo lectura (*Programmable Read Only Memory*, PROM) y, en el caso que se desee, una solución extendida, un convertidor Digital-Análogo puede ser interconectado de tal manera que se obtenga un sistema DDS.

El reloj de referencia proporciona una base de tiempo estable para el sistema y la exactitud en frecuencia del sintetizador digital directo (*Direct Digital Synthesizer*, DDS). La exactitud se refiere a que proporciona la frecuencia a la que el NCO producirá en su salida (de tiempo discreto) una versión cuantificada de la forma de onda de salida deseada (por ejemplo: sinusoide), cuyo periodo es controlado por la palabra digital o palabra de sintonización (*Tuning Word*) contenida en el Registro de control de frecuencia. La forma de onda digital

muestreada se convierte en una forma de onda analógica por el DAC. El filtro de reconstrucción de salida rechaza las réplicas espectrales producidas por la retención de orden cero inherente al proceso de conversión analógica y suaviza la señal.

Con base en lo anterior, suponga el caso donde la información digital de la amplitud correspondiente a un ciclo de una onda senoidal se encuentra almacenada en la PROM; entonces, la memoria de lectura se comporta como una tabla de búsqueda (*Look Up Table*, LUT). El contador de direcciones cuenta y accede a cada una de las locaciones de memoria de la PROM y el contenido obtenido de cada una de las locaciones es presentado a un convertidor digital-análogo de alta velocidad. El convertidor genera una señal senoidal en respuesta a la data digital proveniente de la PROM hacia la entrada del convertidor.

Figura 27. **Sintetizador digital directo simple**



Fuente: Analog Devices Inc. *A Technical Tutorial on Digital Signal Synthesis*. p. 6.

3.1.1. Ventajas

- Cambio extremadamente cambio entre frecuencias al sintonizar la frecuencia de salida.

- La arquitectura digital elimina la necesidad de sistemas manuales de sintonización y ajustes asociados a la edad y variación de temperatura de los componentes en soluciones análogas.
- La interfaz de control digital de la arquitectura DDS facilita el entorno donde los sistemas puedan ser remotamente controlados.

3.1.2. Desventajas

- En una estructura muy sencilla (Figura 27), la flexibilidad de sintonización de frecuencia es inexistente.
- Las estructuras análogas poseen mejor fidelidad en la mayoría de casos.
- La dispersión temporal, también conocida como *jitter,* puede afectar la salida.[11]

3.2. Oscilador numéricamente controlado

Se busca construir un dispositivo que sea capaz de sintetizar una señal senoidal y generar en ésta cambios en frecuencia, únicamente modificando un parámetro numérico simplificado.

3.2.1. Control numérico

El control numérico es un sistema de automatización regularmente utilizado para el control de sistemas mecánicos. La idea es que una máquina sea capaz de realizar una determinada tarea sin necesidad de operación humana, sino mediante la ejecución de instrucciones lineales escritas en un dispositivo de almacenamiento. Para la ejecución es necesaria la existencia de

[11] Analog Devices Inc., *A Technical Tutorial on Digital Signal Synthesis*. http://www.ieee.li/pdf/essay/dds.pdf. Consulta: Enero 2017.

un procesador que interprete dichas instrucciones para después comunicarlas a los actuadores pertinentes que desarrollan la tarea.

El principio de funcionamiento para los sistemas mecánicos parte de un sistema de coordenadas en el cual se especificarán los movimientos de la herramienta final. Para efectuar los movimientos en el sistema de coordenadas es necesario trabajar con la relación entre los ejes de coordenadas de la máquina; esto regularmente se hace utilizando un programa ejecutado por computadora. el cual calcula y traduce instrucciones en movimiento. La idea es la simplificación del control mediante la matemática implicada en los cálculos programados en la unidad ejecutora; así es como nace el Control Numérico Computarizado (CNC)

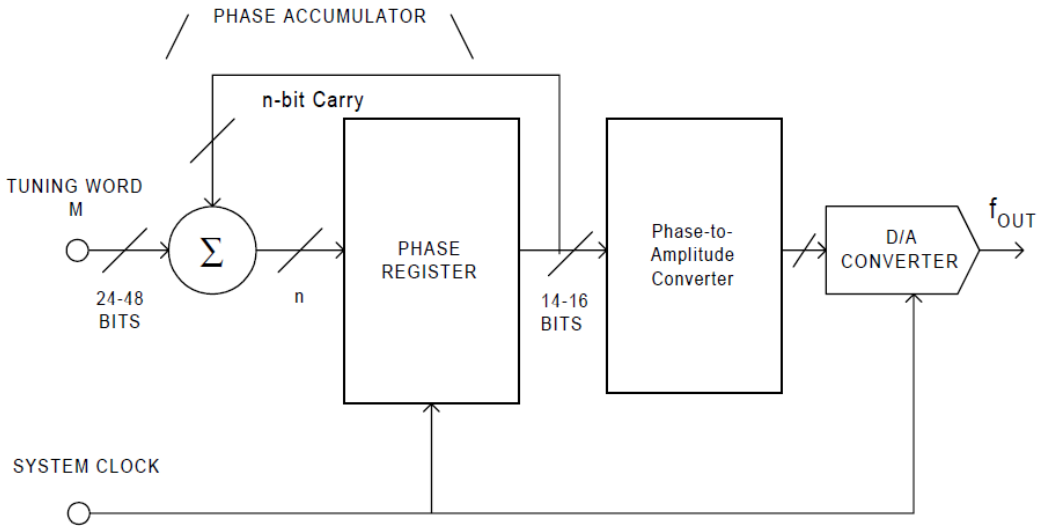
El concepto de control numérico aplicado a síntesis digital hace referencia a la capacidad de poder modificar las características de un generador de onda mediante el cambio de parámetros numéricos simplificados. Esto se logra en los osciladores numéricamente controlados, valiéndose de conceptos tales como tablas de búsqueda, rueda de fase, acumulador de fase y la reconstrucción de una señal.

3.2.2. Oscilador controlado

Un oscilador controlado es aquel que puede alterar sus parámetros de generación de onda a través de algún medio, sea virtual, eléctrico, mecánico o de otro tipo. Puede alterar amplitud, frecuencia o fase en función de una determinada variable, la cual es controlada por el usuario en cualquiera de los medios mencionados. La idea es obtener flexibilidad instantánea sin sacrificar precisión.

Aplicado a este experimento se utiliza un oscilador, el cual es controlado de forma numérica; es decir, consta de una entrada que es un número entero que permite variar la frecuencia de salida. Para controlar de forma numérica directa el oscilador es necesaria la inclusión de un acumulador de fase, el cual se encarga de proveer un equivalente al vector de rotación lineal y esto reemplaza el contador de direcciones lineal poco flexible, como se muestra en la figura 28.

Figura 28. **Sistema DDS con acumulador de fase**



Fuente: Analog Inc. *Fundamentals of DDS Technology*. p. 6.

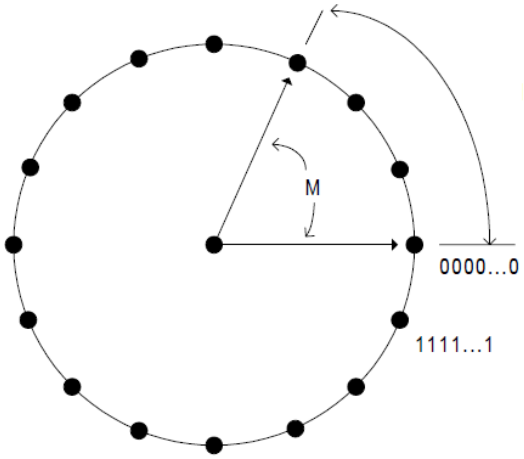
3.3. Círculo de fase

Debido a que el objetivo es generar ondas periódicas en el tiempo, por definición una onda periódica es la que se repite continuamente cada cierto tiempo (T) llamado período. Por lo tanto, toda la información existente se puede obtener en un solo período, y replicarla a lo largo del tiempo al momento de la reconstrucción de la señal.

Como se trabaja en un marco digital que se encuentra en el dominio del tiempo discreto, se partirá de un conjunto finito S de N muestras obtenidas del muestreo de un periodo de una señal analógica. El conjunto S puede ser visto como un vector circular que puede ser recorrido de forma continua para reconstruir la señal, y así sintetizar una señal analógica semejante a la que se originaron las muestras.

Al tomar el conjunto de muestras S como un vector circular, permite ser utilizado como un círculo de fase; para esto es mejor visualizarlo de forma gráfica, como se muestra en la Figura 29. Cada punto en el círculo de fase corresponde a un punto de amplitud equivalente en una onda senoidal, mientras el vector rota alrededor del círculo de fase se visualiza una onda senoidal correspondiente generada en la salida.[12]

Figura 29. **Círculo de fase**

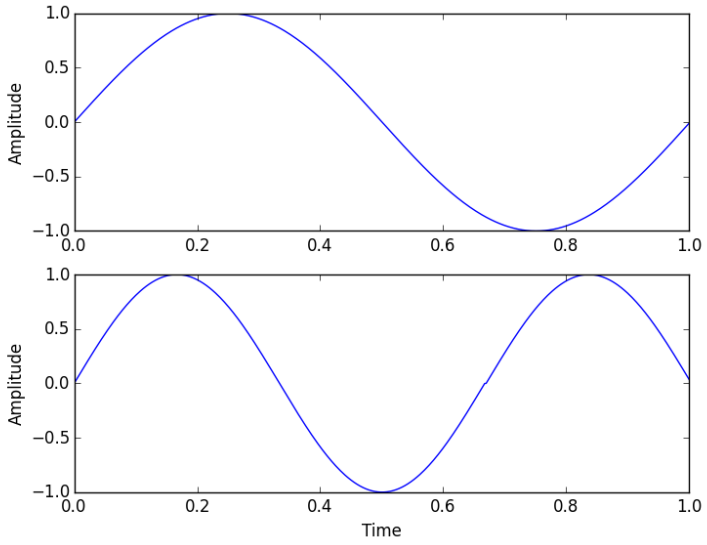


Fuente: Analog Devices Inc. *Fundamentals of DDS Technology*. p. 7.

[12] Analog Inc., *Fundamentals of DDS Technology*. ftp://ftp.analog.com/pub/cftl/DDS%20Tutorial/DDS%20Section%201.pdf. Consulta: Enero 2016.

Cuando se recorre el vector circular S a una velocidad constante y haciendo un salto entre M muestras también constante, se obtiene la misma forma de onda pero con menos muestras y, por ende, menor resolución. Se obtiene una misma forma de onda con menos muestras y una menor longitud de onda que, al momento de la reconstrucción a una tasa de un reloj de referencia constante incide en la salida, con una mayor frecuencia. En el recuadro superior de la Figura 30 se utiliza un paso entre muestras M igual a 10; es decir, cada 10 puntos en el círculo de fase se toma una muestra de un vector de muestreo S de 4096 muestras. En el segundo recuadro se observa una onda senoidal tomada del mismo conjunto de muestras S pero con un paso entre muestras de 15. Se consigue que existan menos muestras tomadas para un período de la onda; por lo tanto, su longitud de onda es menor y su frecuencia, mayor.

Figura 30. **Comparación de círculo de fase**



Fuente: elaboración propia, empleando Python.

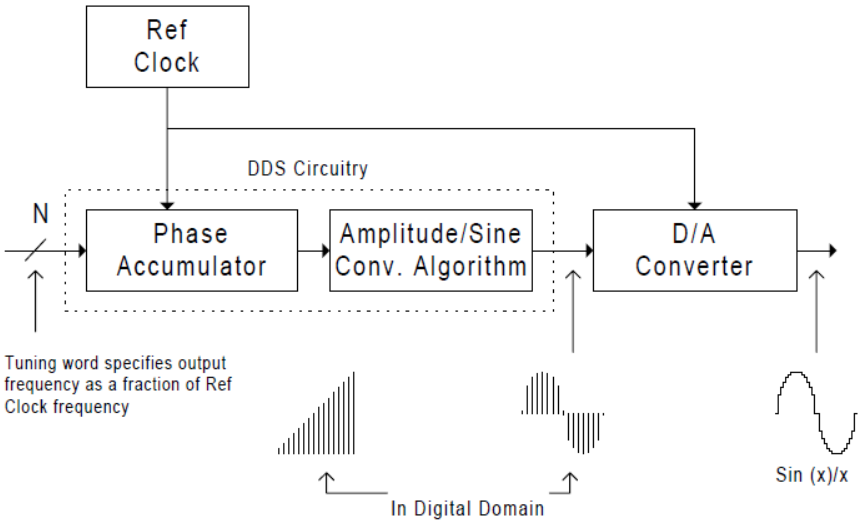
De esta forma es como se logra realizar saltos en frecuencia con un conjunto de muestras, en función de únicamente un parámetro numérico llamado palabra de sintonización (*Tunning Word*).

3.4. Muestreo de salida de un DDS

El muestreo de salida hace referencia al conjunto de muestras que conformarán la señal sintetizada a la salida de un DDS, como estas muestras son obtenidas, generadas o calculadas. También se hace referencia a cómo serán trabajadas puesto que hay diferentes tipos de objetos que puedan representar estos valores. Se pueden representar como números binarios enteros para una determinada cantidad de bits o bien como enteros positivos y negativos o con números de punto flotante.

En el estudio de este proyecto únicamente se considera la generación de ondas senoidales; sin embargo, en el bloque (a la derecha del acumulador de fase) donde se encuentra el algoritmo de conversión a amplitud, en la Figura 31, se puede desarrollar otro algoritmo que realice la conversión a amplitudes de otras formas de onda no senoidales.

Figura 31. **Flujo de señal en un sistema DDS**



Fuente: Analog Inc. *Fundamentals of DDS Technology*. p. 8.

Las muestras son generadas a través de software matemático o de procesamiento de señales, y capturadas en primitivos de software como formato de punto flotante con signo. El manejo de las muestras en punto flotante permite obtener valores normalizados, los cuales pueden ser escalados y fácilmente procesados por el convertidor de digital a análogo a utilizar, particularmente en casos donde se utilicen interfaces de audio como codecs.

3.5. Formas de generación de onda

Existen diferentes formas de generar ondas senoidales en hardware digital.

3.5.1. Cálculo de la función seno

El cálculo de la función seno es un método utilizado principalmente en la generación de ondas senoidales generadas en computadora. Se efectúa con la utilización de representaciones en punto flotante de forma apropiada en el coprocesador dedicado a esto. En el procesador específico de aplicación se utilizan diferentes algoritmos de aproximación, regularmente en un esquema de Taylor, interpolación o CORDIC, ya que las funciones seno y coseno pueden ser estimadas como:

Figura 32. **Aproximación matemática de función seno**

$$\sin(\pi x/2) = 1{,}57063x - 0{,}64323x^3 + 0{,}07271x^5,$$
$$\cos(\pi x/2) = 0{,}9994 - 1{,}22279x^2 + 0{,}22399x^4,$$

Fuente: Sine Wave Generator. http://www.kanyevsky.kpi.ua/Studentam/labexercise%201.pdf. Consulta: 10 de enero de 2017.

Estos cálculos tienen un margen de error pequeño (menor al uno por ciento) y sin importar la frecuencia, no existe pérdida de fidelidad en la síntesis. Sin embargo, el problema es la complejidad de los cálculos y esto sí impacta en el tiempo de generación, poniendo un límite a las frecuencias altas: para que el tiempo de cálculos sea menor, la potencia computacional debe ser mucho mayor y generar un alza en los recursos utilizados, en la potencia disipada y en el costo de los elementos. [13]

[13] KANYEVSKY Lab. *Sine Wave Generator*. http://www.kanyevsky.kpi.ua/Studentam/labexercise%201.pdf. Consulta: Febrero 2017.

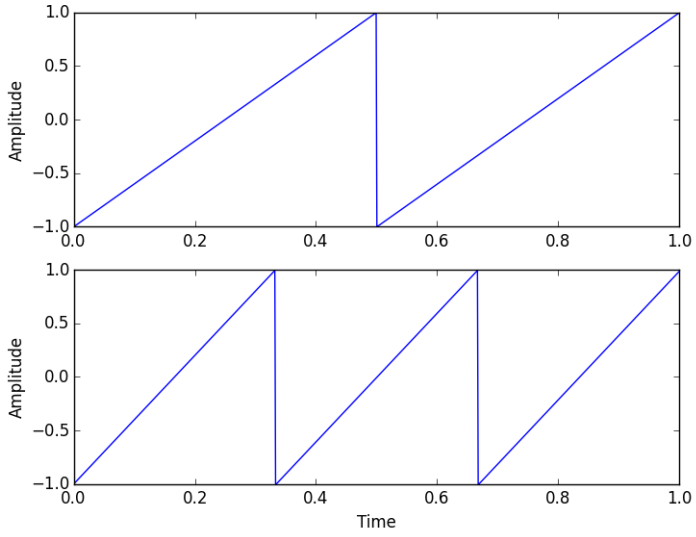
3.5.2. Tabla de búsqueda (LUT)

Los generadores a base de tablas de búsqueda son los más simples y los más utilizados. La idea es emplear los datos que ya se han computado con anterioridad y almacenarlos en una tabla. Los valores se generan con algún método utilizando las calculaciones complejas con puntos flotantes de un período de la onda senoidal y después son almacenados. Al momento de ser necesitados, únicamente se busca la ubicación de la celda deseada en la tabla y se obtiene el valor que se necesita de la onda.

Realizar los cálculos computacionales complejos y guardar los resultados permite disminuir en gran manera el costo computacional en la implementación, lo que hace este sistema mucho más ligero en recursos, potencia disipada y costos mediante un intercambio de memoria en la cual se almacenan los datos de la tabla.

Este método se vuelve más general y transparente respecto a las formas de onda generadas puesto que los valores muestreados almacenados pueden ser de otra forma de onda que no necesariamente sea un seno. Por ejemplo, un diente de sierra y el mismo principio del círculo de fase es aplicable, como se demuestra en la figura 33.

Figura 33. **Círculo de fase aplicado a una señal no senoidal**



Fuente: elaboración propia, empleando Python.

Las muestras fueron generadas con ayuda de Scipy, una librería numérica científica que permite el cálculo aproximado de muestras de diferentes formas de onda en punto flotante con módulo menor que 1.

3.6. Aplicaciones

Un DDS puede controlar la frecuencia y la fase del estímulo con una excelente resolución, así como reemplazar todo el circuito discreto. No se requieren componentes externos para el control de frecuencia. El DDS también tiene la flexibilidad de controlar la fase de salida con una resolución de 10 bits.[14] Con esto se refiere a circuitos integrados fabricados especialmente por gigantes tecnológicos que se dedican a esto como Analog Devices, entre otros. Sin

[14] MURPHY, Eva. *DDS Applications*. http://www.eetimes.com/document.asp?doc_id=1156511. Consulta: Febrero 2017.

embargo, algunas de las características mencionadas permanecen en el concepto intrínseco del DDS; así, sin importar la forma de implementación, siguen siendo ventajas.

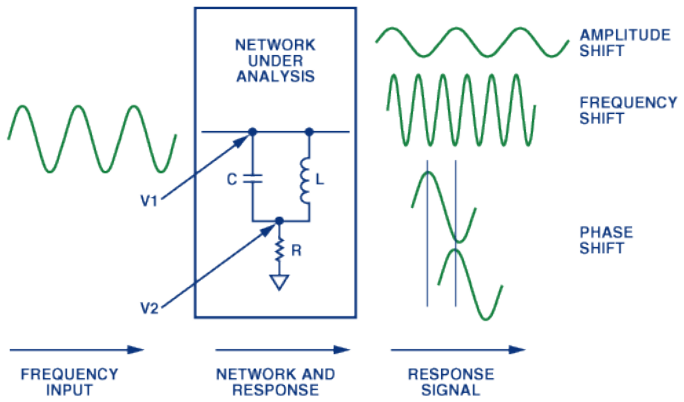
3.6.1. Generador de señales en análisis de redes

Esta clase de aplicación implica estimular un circuito o sistema con frecuencias de amplitud y fase conocidas, y analizar las características de la respuesta para proporcionar información clave del sistema.

Debido a que se conoce la forma de onda estimulante se observan los cambios que hay en amplitud, frecuencia y fase, los cuales son determinados por la respuesta de la red objeto de estudio. Dado que la red bajo análisis regularmente es lineal, la salida puede ser representada con los mismos elementos vectoriales característicos (eigenvectores) que ingresaron, por lo que seguirán siendo una suma de senos y cosenos.

El proceso permite obtener la respuesta de la red. Dicha respuesta en frecuencia es muy útil para calcular la salida de la red eléctrica a una determinada frecuencia sin necesidad de realizar el experimento, puesto que se tiene la medición indirecta de forma matemática.

Figura 34. **Prueba de respuesta**

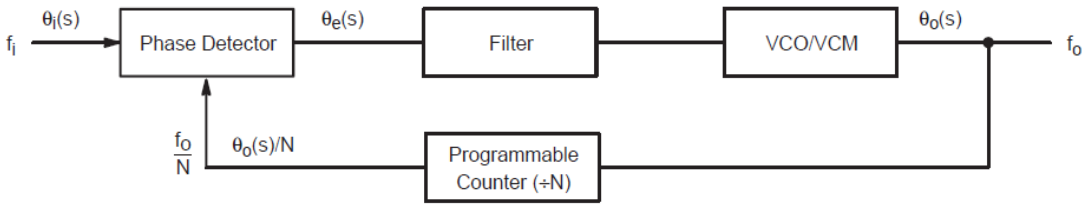


Fuente: MURPHY, Eva. *Control Waveforms in Test, Measurement and Communication*. p. 1.

3.6.2. Referencia modificable para un PLL

Los lazos de seguimiento de fase (*Phase Locked Loop*, PLL) son circuitos ampliamente utilizados hoy en muchas aplicaciones de telecomunicaciones porque pueden sincronizar un oscilador con otro. Entre muchas otras bondades, tienen una arquitectura básica, como lo demuestra la figura 35.

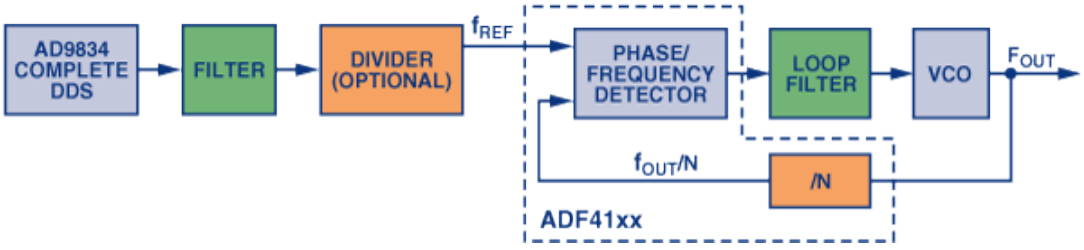
Figura 35. **Lazo de seguimiento de fase (PLL)**



Fuente: Motorola. *Phase-Locked Loop Design Fundamentals*.

En las modificaciones en el reloj de referencia donde se añade un sistema DDS a un PLL que funciona como sintetizador de frecuencia como una solución híbrida, la resolución del DDS mejora la sintonización del sistema a un nivel que únicamente utilizando un PLL no sería posible.[15] Dicha modificación se muestra en la figura 36.

Figura 36. **DDS como generador de frecuencia de referencia**



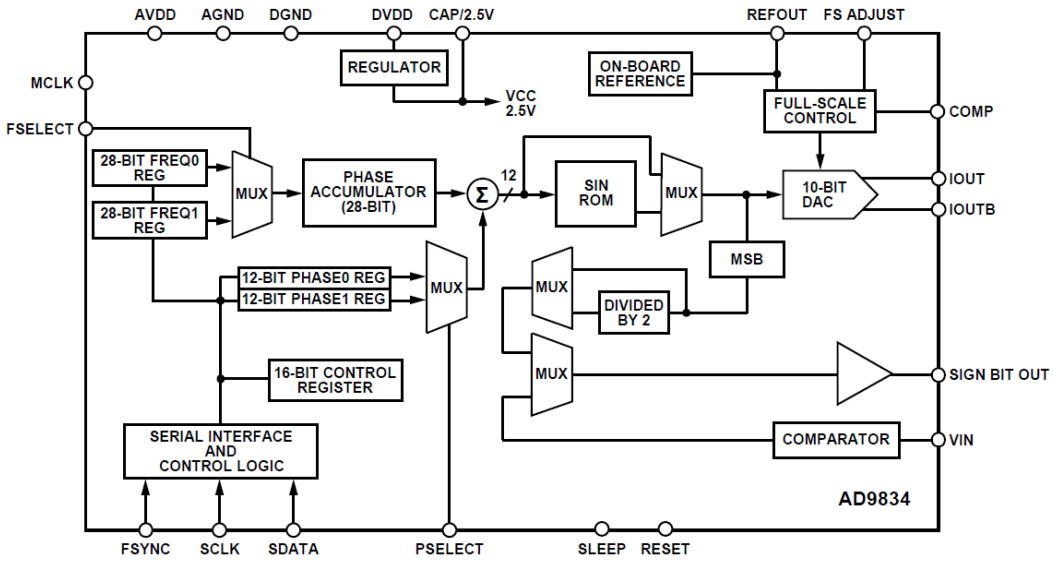
Fuente: MURPHY, Eva. *Control Waveforms in Test, Measurement and Communication*. p. 2.

El bloque AD9834 hace referencia a un circuito integrado, el cual es un sistema DDS completo de 20mW de potencia y una frecuencia máxima de salida de hasta 37.5MHz, como se muestra en la figura 37.[16]

[15] MURPHY, Eva. *DDS Controls Waveforms in Test, Measurement and Communications*. http://www.analog.com/media/en/analog-dialogue/volume-39/number-3/articles/dds-controls-waveforms-in-test.pdf. Consulta: Febrero 2017.

[16] Analog Inc., "Datasheet AD9834". http://www.analog.com/media/en/technical-documentation/data-sheets/AD9834.pdf. Consulta: Marzo 2017

Figura 37. **Diagrama lógico interno de un AD9834**



Fuente: Datasheet AD9834. http://www.analog.com/media/en/technical-documentation/data-sheets/AD9834.pdf. Consulta: 15 enero 2017.

El DDS con la palabra de sintonización de 28 bits permite que la frecuencia de referencia sea estrechamente sintonizada, lo que resulta en un ajuste fino de la frecuencia de salida mucho más conveniente que el uso de un PLL fraccional.[17]

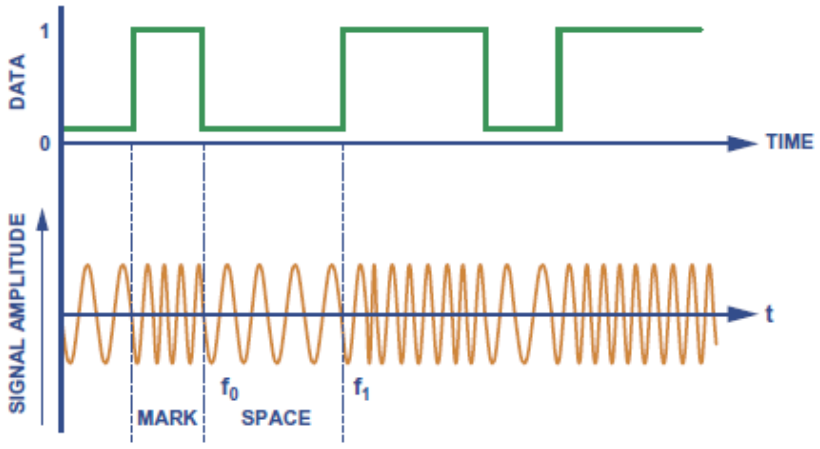
3.6.3. Codificación de datos por FSK

Puesto que la especialidad de los dispositivos DDS es el ajuste y cambios de frecuencia y fase, se vuelven especialmente útiles en la codificación de modulación de frecuencia y fase en una señal portadora para transmisión de datos.

[17] Analog Inc., *Analog Devices*. http://www.analog.com/en/analog-dialogue/articles/dds-controls-waveforms-in-test.html. Consulta: Marzo 2017.

La codificación binaria a través de cambios en frecuencia (*Frequency Shift Keying*, FSK) es una de las formas más simples de codificación de datos. Los datos son transmitidos cambiando la frecuencia de una señal portadora continua de una a otra entre dos frecuencias discretas. Se da así la forma de una operación binaria para lograr representar los dos estados discretos de decisión de una señal binaria. De esta manera es posible transmitir los datos requeridos con una mayor resistencia al ruido e interferencia, ya que la amplitud deja de ser el parámetro de decisión. Un ejemplo de esto se hace visible en la figura 38.

Figura 38. **Codificación de cambios en frecuencia (FSK)**

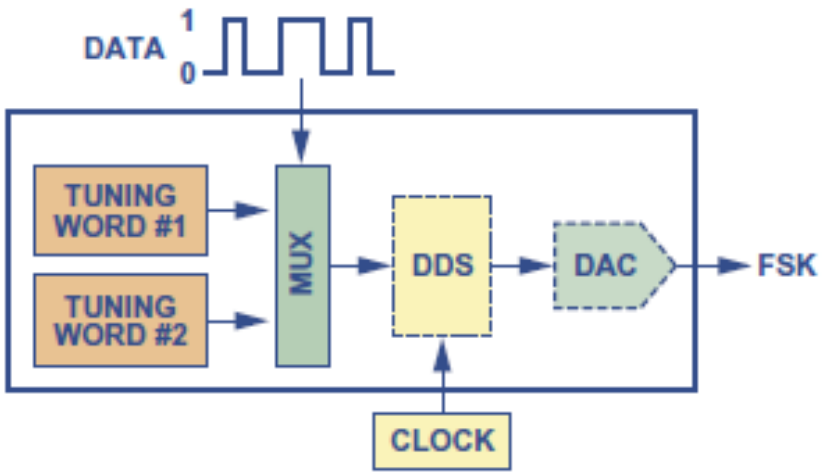


Fuente: MURPHY, Eva. *Control Waveforms in Test, Measurement and Communication*. p. 3.

Este esquema de codificación es de implementación fácil a través de un DDS que utiliza la palabra de sintonización como el parámetro de cambio. Entonces se tienen dos valores binarios, los cuales son elegibles por medio de un multiplexor. Cada valor binario representa una frecuencia diferente en la salida del DDS; entonces, para realizar el cambio de frecuencia, basta con

cambiar el bit de selección en el multiplexor, acorde al esquema[18] de la figura 39.

Figura 39. **Diagrama de bloques codificación FSK**



Fuente: MURPHY, Eva. *Control Waveforms in Test, Measurement and Communication*. p. 3.

También es posible la implementación de otras codificaciones utilizando DDS, por ejemplo: la codificación por cambio de fase (*Phase Shift* Keying, PSK) en donde la frecuencia es constante y lo que cambia es la fase de la onda. La variante más sencilla del esquema es el PSK binario (*Binary Phase Shift Keying*, BPSK) que utiliza únicamente dos fases de señal 0° y 180°, cada una representa un evento binario correspondiente.

[18] MURPHY, Eva. *DDS Controls Waveforms in Test, Measurement and Communications*. http://www.analog.com/media/en/analog-dialogue/volume-39/number-3/articles/dds-controls-waveforms-in-test.pdf. Marzo 2017.

4. DISEÑO E IMPLEMENTACIÓN DEL MODELO

4.1. Diseño del sistema

El foco central de este trabajo es la implementación de un oscilador numéricamente controlado en tejido lógico de propósito general, utilizando síntesis de alto nivel en un sistema embebido lógico (ZYNQ). Ahora bien, para lograr esto, es necesario tener una plataforma de hardware. ZyBo fue la plataforma seleccionada para este proyecto puesto a consideración del autor. Dicha plataforma cumple con los requisitos necesarios en la implementación del diseño.

Para efectos demostrativos del proyecto en la plataforma y del diseño necesario para crear sistemas mucho más complejos se decidió orientar la implementación del oscilador a audio, lo cual afecta de cierta forma el manejo de datos, la generación muestral y el método de conversión análogo-digital a causa del hardware a utilizar.

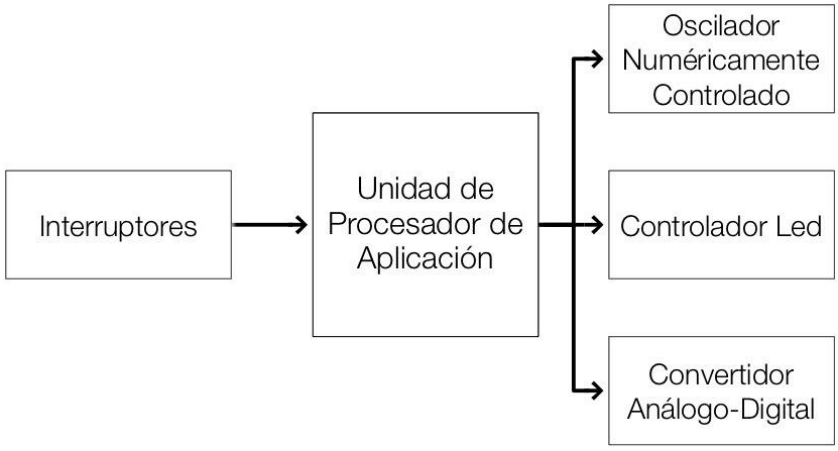
En una recapitulación rápida se necesita:

- Muestras aptas para su utilización en tablas de búsqueda e interpretables por el sistema de hardware encargado de hacer la conversión entre analógico y digital.
- Módulo NCO que genere los valores de amplitud en función de la palabra de sintonización.
- Interconexiones entre módulos que permitan comunicación y transferencias de datos.

- Método demostrativo externo de cambio instantáneo de frecuencia.
- Sistema de conversión análogo-digital.

Tomando en cuenta las descripciones del proyecto anteriormente mencionadas, se procede a elaborar un bosquejo en bloques (ver figura 40).

Figura 40. **Diagrama de bloques del sistema (simplificado)**



Fuente: elaboración propia.

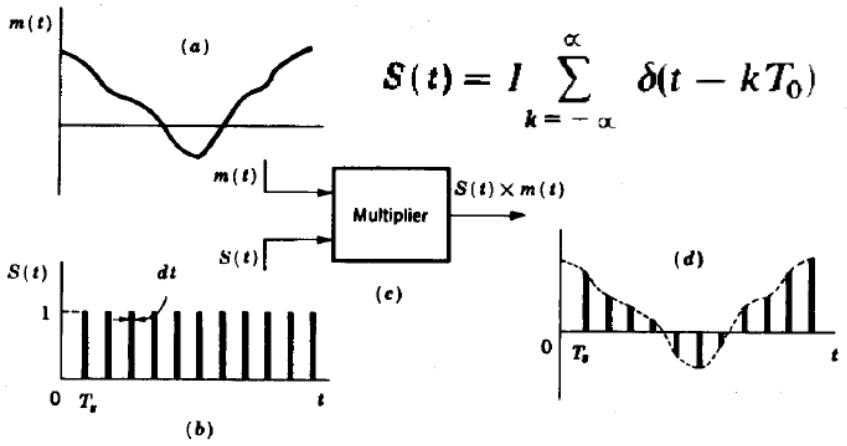
4.1.1. Arreglos muestrales

Los arreglos muestrales son los vectores en los cuales se almacenan los valores secuenciales de amplitud de una forma de onda específica mediante un proceso de muestreo, el cual puede ser de forma numérica mediante cálculos de aproximaciones de los modelos matemáticos de dicha onda en determinados puntos discretos.

Idealmente, el muestreo de forma matemática es la multiplicación de una función de tren de impulsos con la función matemática de la forma de onda

deseada, sea esta una representación de Fourier o un modelo matemático directo como una función seno o coseno. La figura 41 muestra de forma gráfica como esto sería visible y la forma matemática que debería tener la función S(t).[19]

Figura 41. **Muestreo con tren de impulsos**

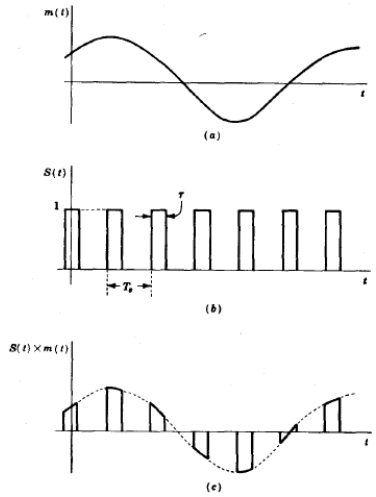


Fuente: TAUB, Herbert. *Principles of Communication Systems*. p. 186

Al momento, es imposible obtener un muestreo ideal de forma física con un tren de impulsos, puesto que su ancho tiende a 0. Los sistemas siempre necesitan algún tiempo de funcionamiento, por más pequeño que sea, convirtiendo los impulsos en pulsos cuadrados, como se muestra en la figura 42.

[19] TAUB, Herbert. *Principle of Communication System*. p. 186.

Figura 42. **Muestreo realizado con tren de pulsos cuadrados**

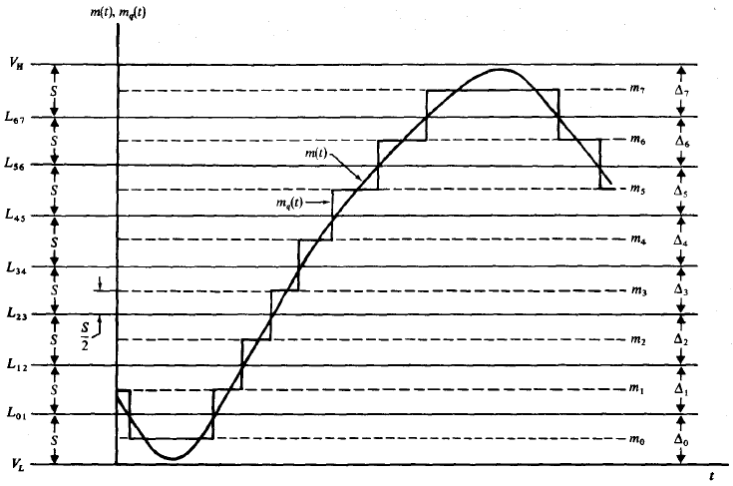


Fuente: TAUB, Herbert. *Principles of Communication Systems*. p. 198.

El tiempo requerido para hacer la toma de la muestra genera errores que introducen distorsión en la señal, debido a que el proceso de distinción de la amplitud percibida por el sistema tiene niveles de decisión. Cada determinada distancia entre valores llamados pasos se genera un error inicial en la toma de la muestra. En la figura 43 se muestran dos señales: una, continua en el tiempo m(t), de la cual se quiere obtener las muestras, y la escalonada $m_q(t)$, que es la versión muestrada y cuantizada de m(t).[20] A simple vista se puede denotar que existe una diferencia (error).

[20] TAUB, Herbert. *Principle of Communication System*. p. 205.

Figura 43. **Operación de muestreo y cuantización**



Fuente: TAUB, Herbert. *Principles of Communication Systems*. p. 206.

Debido a que en la reconstrucción de la señal es necesario sostener una muestra por un determinado tiempo, se genera otro tipo de error conocido como sostenimiento de orden cero (*Zero Order Hold*) y se modela matemáticamente,[21] como se muestra en la figura 44.

Figura 44. **Espera de datos orden cero**

$$h(t) = \sum_{k=0}^{\infty} x(kT)\ [u(t-kT)] - u(t-(k+1)T)]$$

Fuente: Tymerski, Richard. ECE 452/552 Guide.
http://web.cecs.pdx.edu/~tymerski/ece452/Chapter3.pdf. Consulta: 2 de febrero de 2017.

Los errores mencionados son corregibles con un método llamado sobremuestreo, en el cual la frecuencia de muestreo es mucho mayor que la

[21] TYMERSKI, Richard. *ECE 452/552*. http://web.cecs.pdx.edu/~tymerski/ece452/Chapter3.pdf. Consulta: Marzo 2017.

frecuencia muestreada. Mucho más que el necesario cumplimiento del teorema de muestreo de Nyquist-Shannon, el cual indica que la frecuencia de muestreo debe ser al menos el doble de la frecuencia de la señal a ser muestreada.

4.1.2. Generación de arreglos

La generación de los arreglos muestrales se realiza a través de software computacional que permita el manejo de librerías matemáticas para generación de valores aproximados de ondas, bien sean senoidales o de otro tipo, en función del tiempo. Los valores obtenidos de estos programas son útiles sin importar la aplicación que se desee, puesto son valores con módulo no mayor a 1; por lo tanto, son completamente escalables.

Los valores generados se encuentran entre -1 y 1. Es necesario tener la resolución correcta para poder distinguir entre cada punto sin dar paso a la repetición de valores. Es decir, sería fácil si estos intervalos de dos unidades únicamente se subdividieran en 3 partes; solo se necesitaría un decimal para poder identificar claramente los puntos generados (ejemplo: [-1, -0.5, 0, 0.5, 1]). Ahora bien, no se subdivide en 3 partes sino en 1023, 2047 o 4095 partes y así obtener la cantidad de muestras necesarias.

La resolución de los decimales depende en gran manera de la forma de onda que se intente generar, puesto que algunas tienen secciones donde la pendiente es bastante baja y, por lo tanto, hay un menor cambio de amplitud en función del tiempo. Esto puede generar valores muy cercanos que puedan ser difíciles de diferenciar. Para ondas senoidales y de diente de sierra que se trataron experimentalmente, se toma un criterio inicial de 14 decimales; sin embargo, se puede llevar a cabo hasta con 6 decimales.

La generación de muestras senoidales se realizó con el entorno proporcionado por la importación de *pylab,* el cual es una forma de importar de manera fácil y cuantitativa funciones de los modulos *pyplot* y *numpy.* Estos son componentes de la librería Matplotlib, la cual se recomienda instalar al momento de instalar Scipy. Ahora bien, la razón principal de la existencia de este entorno es su trabajo en consolas interactivas como ipython y no únicamente el desarrollo de programas estáticos (*Scripts*).[22]

Figura 45. **Código generador de muestras senoidales**

```
from pylab import *

t = linspace(0, 1, 4096)
s = sin(2 * pi * t)

savetxt('test.txt', s, fmt='%f', newline=',\n')
```

Fuente: elaboración propia, empleando Python IDLE.

Se considera hacerlo en Scipy, el cual "Es un ecosistema basado en Python de software de código abierto para matemáticas, ciencias e ingeniería"[23]. Puesto que es software de código abierto, funciona bajo licencia GNU y permite no incurrir en gastos innecesarios. Además, cuenta con más funcionalidades como ploteo, empaquetado de sonido, modificación rápida, interpretación, exportación archivos de texto, entre otras.

[22] Pylab. *Pylab Github*. https://scipy.github.io/old-wiki/pages/PyLab. Consulta: Marzo 2017.

[23] Scipy. *ScipyWeb*. https://www.scipy.org/. Consulta: Marzo 2017.

Figura 46. **Código generador de muestras de señal diente de sierra**

```
from scipy import signal
import numpy as np

t = np.linspace(0, 1, 4096)
s = signal.sawtooth(2 * np.pi * t)

np.savetxt('test.txt', s, fmt='%f', newline=',\n')
```

Fuente: elaboración propia, empleando Python IDLE

4.1.3. Diseño NCO

El diseño del oscilador numéricamente controlado (*Numerically Controlled Oscillator*, NCO), hará uso de los diagramas y arquitecturas utilizados.

Debido a que el enfoque del proyecto es la utilización de síntesis de alto nivel (*High Level Synthesis*, HLS), no se plasmará la abstracción de funcionamiento obtenida previamente de forma directa en VHDL o Verilog, sino que se crearán funciones y variables en algún lenguaje de alto nivel (C++). Este ayudará a transmitir la idea funcional a un motor sintetizador capaz de generar el equivalente de este modelo funcional tanto en VHDL (lenguaje de descripción) como a un modelo RTL. En este proceso se necesitan dos archivos principales, uno llamado “fuente” (*Source*) y el otro llamado “banco de prueba” (*Test Bench*).

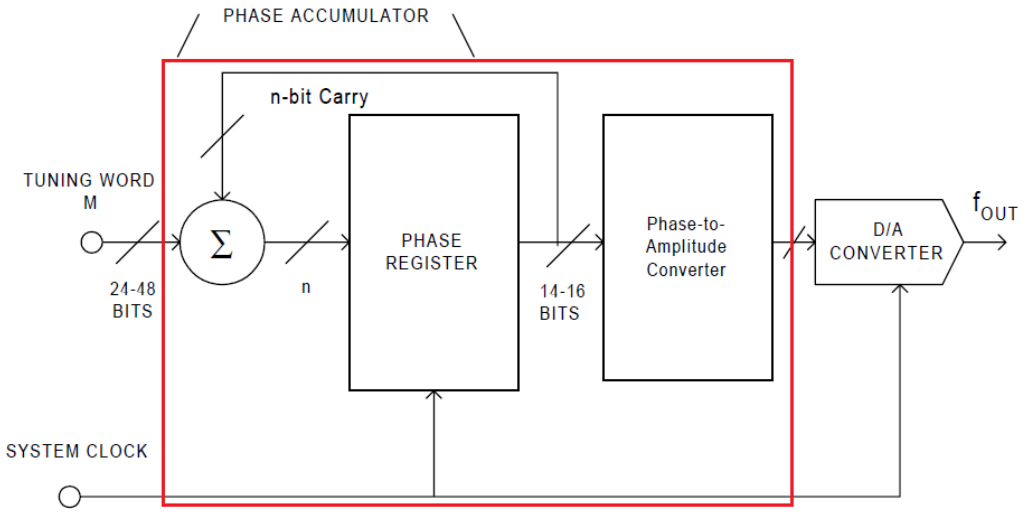
En el archivo fuente se describe en una función principal; es decir, que internamente tiene como valores almacenados, variables y funciones internas que desarrollan un determinado proceso, entradas y salidas del módulo.

El archivo banco de prueba establece una instancia del módulo que se programó en el archivo fuente. En esta instancia se pone a prueba el módulo y se estimula con entradas y salidas de un valor específico, para así poder mesurar el comportamiento de la unidad en prueba (*Unit Under Test*, UUT) y compararlo con lo que se necesita. A través de este comportamiento, el programa en el cual se desarrollará la solución de síntesis de alto nivel (*High Level Synthesis*, HLS) determinará las diferentes RTL que puedan ser la posible solución.

El diseño del NCO partirá del diagrama mostrado en la Figura 47. Sin embargo, esta solución parte desde la palabra de sintonización (*Tunning Word*) hasta la salida del convertidor de fase a amplitud, como lo demuestra el rectángulo rojo. Por tanto, hay tres partes internamente importantes: sumatoria de acarreo, registro de rase y convertidor de fase a amplitud.

Al observar con atención el recuadro de definición se aclara que los pines externos del bloque NCO son la palabra de sintonización, reloj de sistema y salida de amplitud. Los dos primeros son entradas y el último, salida.

Figura 47. **Sistema DDS, diagrama base del NCO**



Fuente: Analog Devices Inc. *Fundamentals of DDS Technology*. p. 6.

Cada uno de los bloques internos será sustituido por alguna estructura programática. El convertidor de fase a amplitud es sustituido por un arreglo, el cual contiene todos los valores generados previamente; un vector de 4096 posiciones conteniendo números de punto flotante ocupa este bloque. El Registro de fase es una variable, la cual almacena un número entre 0 y 4096; por lo tanto, se establece de 12 bits que, por defecto, al momento de existir un *overflow* regrese al valor cero. Este es el argumento ingresado al vector que alberga las amplitudes. La Sumatoria de acarreo es la suma del registro de fase actual y la palabra de sintonización, lo cual es sustituido por una suma entre variables.

Cuando todo se encuentra junto, se tiene una variable de entrada a la función, que es la palabra de sintonización. Este valor se suma a un valor inicial cero, lo que fija el valor del paso entre las muestras del vector de amplitudes. A esta misma variable se le estará sumando el valor del paso de

forma constante a manera de recorrer de forma selectiva cada cierta cantidad de muestras el vector de amplitudes. Dichas amplitudes son almacenadas en una variable puntero, la cual se actualiza instantáneamente al último valor obtenido del vector.

El reloj del sistema no es una consideración que se deba tomar en el diseño, puesto que el puerto es automáticamente añadido de forma optimizada por el software de desarrollo. La conexión al reloj físico es elección del diseñador, mas se recomienda conectarlo al reloj central de la plataforma de desarrollo. La figura 48 genera un archivo VHDL de aproximadamente 273 líneas de código.

Figura 48. **Código NCO**

```
ap_ufixed<16,12> temp = 0.0;
ap_ufixed<12,12> addr;

void nco (ap_fixed<16,2> *muestra, ap_ufixed<16,12> paso){

    temp+=paso; // Acumulador.

    addr = ap_ufixed<12,12>(temp); // Dirección del vector

    *muestra = seno_lut[(int)addr]; // muestra de salida
}
```

Fuente: elaboración propia, empleando Vivado HLS.

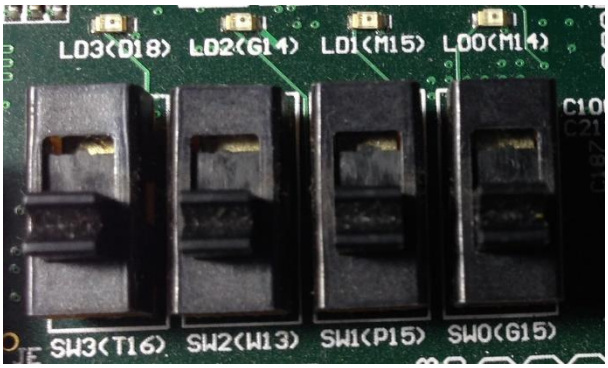
Las interfaces que se observan en el boque son el valor de entrada, que es la palabra de sintonización también llamado paso; el valor de salida, que es la amplitud de la onda, y el reloj del bloque. Ahora bien, debido de que la comunicación entre todo el sistema con el procesador de aplicación es a través de protocolo AXI —el cual es de lectura y escritura— únicamente observaremos dos interfaces externas: una conexión al reloj del sistema y una interfaz AXI-Lite

mediante la cual recibiremos el paso entre muestras y enviaremos las muestras de amplitud.

4.1.4. Interfaces de control

La interfaz de control del usuario es el método mediante el cual el usuario del sistema interactuará para realizar los cambios determinados al sistema. Debido al fin demostrativo del sistema se utilizará una interfaz física de la tarjeta de desarrollo. Se utilizará cuatro interruptores, los cuales representarán un número binario de 4 bits; este número será recibido por el procesador de aplicación y después enviado al NCO como el valor de la palabra de sintonización. Por tanto, estos botones permitirán cambiar la frecuencia de salida.

Figura 49. **Interruptores físicos**



Fuente: elaboración propia.

La idea de utilizar interfaces físicas parte de la facilidad al momento de demostrar; si bien es cierto no es la interfaz que utiliza todos los bits de la palabra de sintonización (paso), es un excelente ejemplo del funcionamiento de un sistema completo implementado en la plataforma, donde hay módulos de

control y de generación instanciados en el tejido lógico unidos al procesador de aplicación.

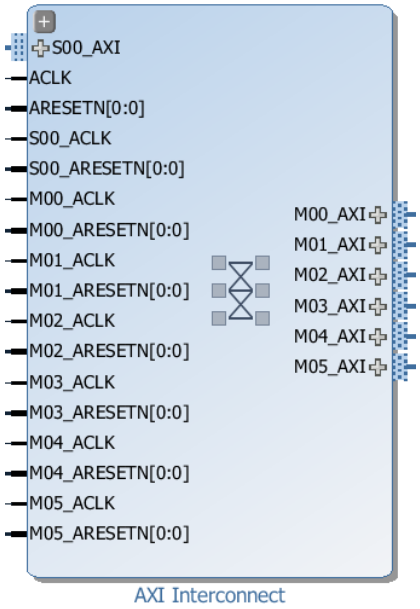
El controlador de los interruptores es una interfaz de rápida construcción basada en el estándar AXI-lite. Existen herramientas de rápida generación de estas interfaces y finalmente es un módulo solo de lectura. Se interpreta como un registro de lectura que devuelve al procesador de aplicación (PS) un valor de 32 bits. No importa que el valor a leer sea de máximo 4 bits; la interfaz más pequeña concebible es de 32 bits; por lo tanto, los demás bits serán ceros.

4.1.5. Interconexiones

Para las interconexiones entre módulos se utilizará el estándar AXI a través de una periferia de control por parte del procesador de aplicación instanciada en el área del tejido lógico que se encarga de las interconexiones AXI. En esta se conectan todos los módulos que se necesiten utilizar: módulo de interruptores, de leds, de audio, módulo NCO.

La periferia AXI se encarga del manejo de datos entre los módulos y el procesador de aplicación multiplexando la comunicación, cada una con su determinado reloj asíncrono y reinicio también asíncrono. La mayoría de las interfaces a conectar en cuanto a módulos son esclavos AXI-Lite. En la mayoría de las interconexiones en dependencia del nivel de personalización es posible utilizar herramientas de automatización, que permitan tanto la instanciación de las interfaces necesarias como la interconexión con los módulos. La idea es evitar al usuario realizar una misma tarea de forma repetitiva y así aumentar la productividad.

Figura 50. **Ejemplo interconexión AXI por bloques**



Fuente: elaboración propia, empleando Vivado Software 2015.4.

4.1.6. Sistema de conversión digital-análogo

Es la conversión final de los valores digitales obtenidos del NCO a una señal analógica capaz de ser escuchada en una bocina o con audífonos. En este caso se utilizará el códec de audio existente en la plataforma de hardware ZyBo, el cual es diseñado y fabricado por Analog Devices Inc. Este dispositivo será el encargado final de entregar la señal análoga reconstruida. El dispositivo SSM2603 es un códec de audio estéreo de bajo poder y alta calidad, regularmente utilizado en aplicaciones portátiles de audio digital, con una pareja de estéreo de amplificadores de ganancia programable (*Programmable Gain Amplifier*, PGA), líneas de entrada y una línea de micrófono monoaural. Contiene además dos convertidores de análogo a digital (*Analog-Digital*

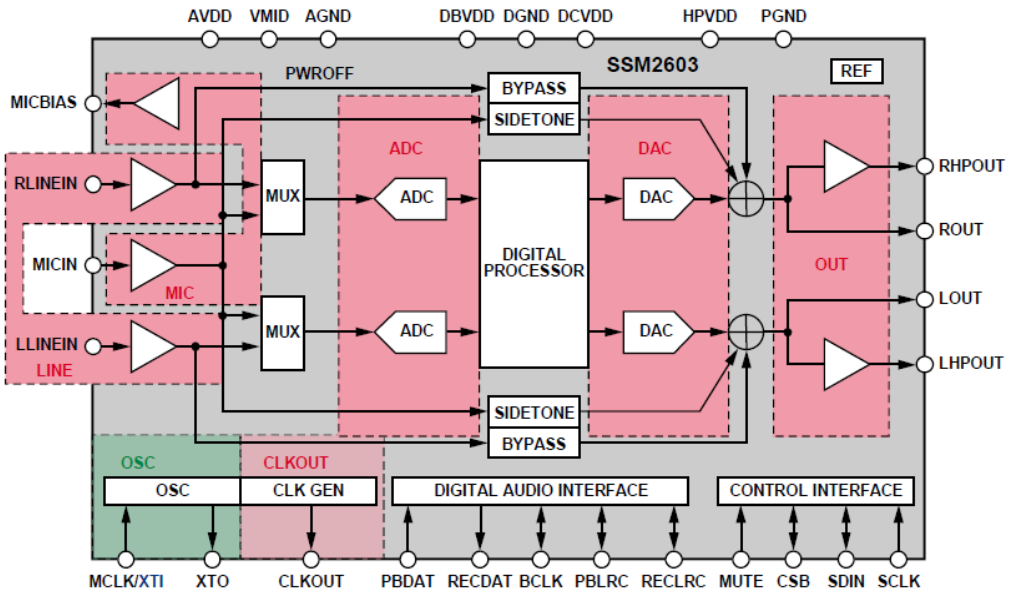
Converter, ADC) de 24 bits cada canal, y dos convertidores de digital a análogo (*Digital-Analog Converter*, DAC) de 24 bits cada canal.

Soporta diferentes frecuencias de reloj maestro dependiendo del modo en el cual el dispositivo se encuentre operando: modo USB, en el cual acepta relojes de 12 MHz y 24 MHz; modo de sobremuestreo (*oversampling*) estándar 256 f_s o 384 f_s basado en relojes como 12.288 MHz y 24.576 MHz, además de muchas tasas de muestreo de audio como 96 kHz, 88.2 kHz, 48 kHz, 44.1 kHz, 32 kHz, 24 kHz, 22.05 kHz entre otras menores.

La interfaz de comunicación de audio entre el dispositivo de conversión digital-análoga y el dispositivo ZYNQ utilizará un estándar dedicado a la transmisión serial de audio digital entre dispositivos, desarrollado por Phillips Semiconductors a finales de 1985 para ser publicado finalmente en febrero de 1986.

El control del dispositivo se da, como lo muestra la Figura 51, a través de la interfaz de control que consta de cuatro pines: MUTE, CSB, SDIN, SCLK. Cada uno de los pines de la interfaz de control tiene una función específica. MUTE es el encargado de silenciar la salida de controlador en función de una entrada binaria; CSB es un pin encargado de determinar la dirección de comunicación del dispositivo; SDIN y SCLK son un par de pines, los cuales forman parte de un estándar de comunicación entre circuitos integrados llamado *Inter-Integrated Circuit*, más comúnmente referido como I2C.

Figura 51. **Diagrama interno SSM2603**



Fuente: SSM2603 Datasheet. http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2603.pdf. Consulta: 17 de febrero de 2017

4.1.6.1. I2S

Debido a que hoy gran parte del manejo de audio en los dispositivos de uso diario es a través de procesadores digitales de audio y después transferidos a otros dispositivos digitales y/o circuitos integrados, se encuentra la necesidad de estructuras de comunicación estandarizada tanto para el equipo como para quien manufactura del circuito integrado y así incrementar la flexibilidad. Esto lleva a la fabricación del bus de comunicación de sonido entre circuitos integrados (*Inter-IC Sound*, I2S).

La idea principal es que el bus tenga que lidiar únicamente con data, mientras otras señales como subcodificación y control son transferidas de forma separada a manera de eliminar procesos posteriores a la recepción de los datos

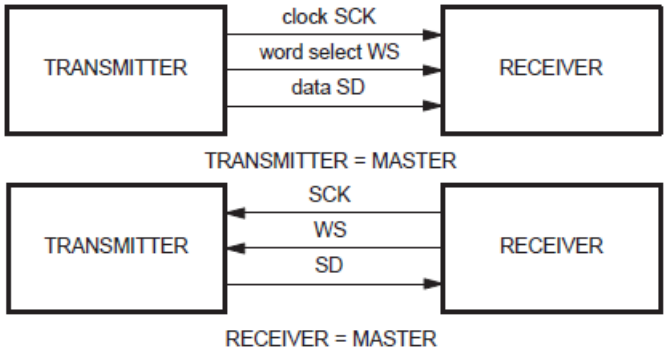
y hacer la transferencia lo más directa posible. I2S es un bus serial de tres líneas consistentes en una línea de datos que transmite los datos de dos canales multiplexados en el tiempo, una línea de selección de palabra encargada de delimitar la multiplexación y una línea de reloj.

Entre sus principales aplicaciones se encuentran procesadores digitales de audio, cintas de sonido digital, sonido de TV digital, entre otros. Ahora, las aplicaciones directas de forma electrónica son las siguientes:

- Convertidores análogo-digital
- Convertidores digital-análogo
- Procesadores digitales de señal
- Correctores de errores para disco compacto
- Correctores de errores en grabación de audio
- Filtros digitales
- Interfaces digitales de entrada/salida

Las líneas de comunicación son tres, cada una con una tarea específica; sin embargo, existen diferentes modos de conexión entre los dispositivos dependiendo de quien sea el maestro y el esclavo en la conexión. En la Figura 52 se muestran dos posibles formas de conexión entre dos dispositivos.

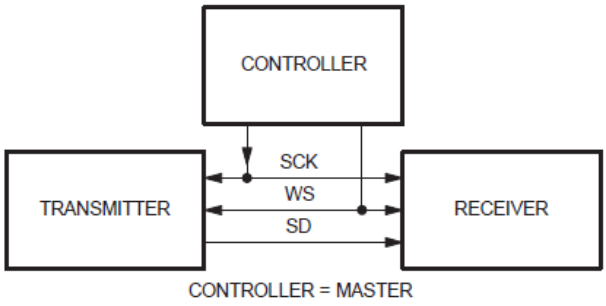
Figura 52. **Diagrama de bloques conexión I2S directa**



Fuente: Phillips Semiconductors, *I2S Bus Specification*. https://web.archive.org/web/20060702004954/http://www.semiconductors.philips.com/acrobat_download/various/I2SBUS.pdf. Consulta: 17 de febrero de 2017.

La tercera forma de conexión es mediante un controlador que arbitre entre líneas de transmisión de datos y selección de palabra bidireccionales, como se muestra en la Figura 53.

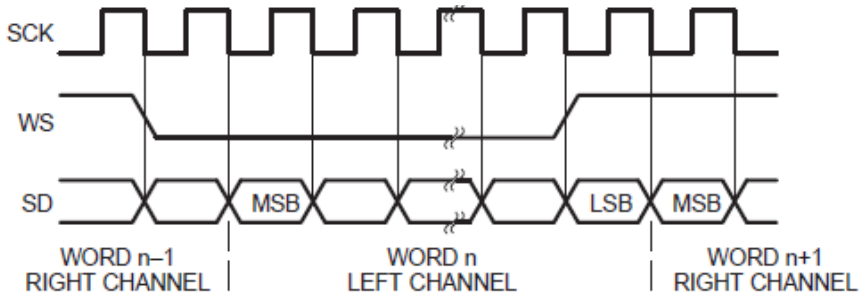
Figura 53. **Diagrama de bloques conexión I2S mediante controlador**



Fuente: Phillips Semiconductors, *I2S Bus Specification*. https://web.archive.org/web/20060702004954/http://www.semiconductors.philips.com/acrobat_download/various/I2SBUS.pdf. Consulta: 17 de febrero de 2017.

Independientemente de la forma de conexión que exista entre los dispositivos, es necesario respetar los diagramas de temporización que establece el protocolo para cada una de las líneas utilizadas. Como es visible en la Figura 54, el reloj serial controla el ancho del bit en la línea de datos y, por lo tanto, también el ancho en los estados de la línea de selección de palabra.[24]

Figura 54. **Diagrama de tiempo en la comunicación I2S**



Fuente: Phillips Semiconductors, *I2S Bus Specification*. https://web.archive.org/web/20060702004954/http://www.semiconductors.philips.com/acrobat_download/various/I2SBUS.pdf. Consulta: 17 de febrero de 2017.

4.1.6.2. I2C

*Phillips semiconduct*ors desarrolló un bus bidireccional simple de dos líneas, eficiente para control entre circuitos integrados. Este es llamado Inter-IC o bus I^2C. Solo dos líneas del bus son requeridas: una línea de data serial llamada SDA y una línea de reloj serial llamada SCL. El estándar se caracteriza por ser completamente serial, bidireccional, orientado a 8 bits y con múltiples modos de funcionamiento, el cual da flexibilidad en los rangos de velocidad de transferencia que permite el estándar.

[24] Phillips Semiconductors. *I^2S Bus Specification*. https://web.archive.org/web/20060702004954/http://www.semiconductors.philips.com/acrobat_download/various/I2SBUS.pdf. Consulta: Marzo 2017.

Los modos de transferencia son los siguientes:

- Modo estándar (*Standard-mod*e), puede llegar hasta 100kbit/s.
- Modo rápido (*Fast-mode*), puede llegar hasta 400kbit/s.
- Modo rápido+ (*Fast-mode Plus, Fm+*), puede llegar hasta 1Mbit/s.
- Modo alta velocidad (*High-Speed mode*), puede llegar hasta 3.4Mbit/s.
- Modo ultra rápido (*Ultra-Fast Mode, uFM*), es un modo de transferencia unidireccional capaz de llegar hasta 5Mbit/s.[25]

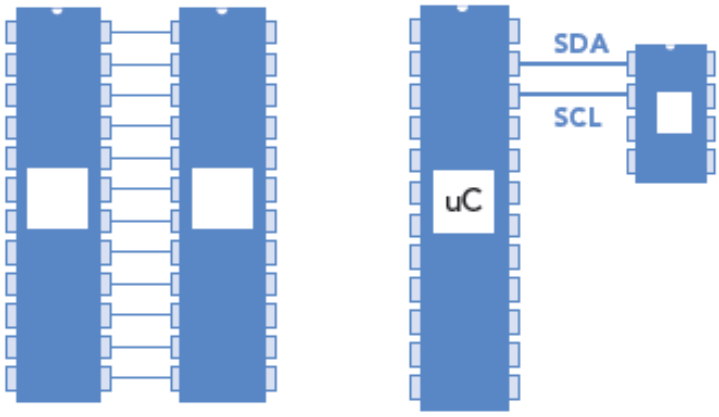
El bus I2C es un estándar *de facto* en la industria, ya que es utilizado en arquitecturas varias tales como:

- Bus de administración de sistema (*System Management Bus*, SMBus).
- Bus de administración de potencia (*Power Management Bus*, PMBus).
- Plataforma inteligente de administración de interfaz (*Intelligent Platform Management Interface*, IPMI).
- Canal de datos de visualización (*Display Data Channel*, DDC).
- Arquitectura avanzada de computación de telecomunicaciones (*Advanced Telecom Computing Architecture*, ATCA).

Anterior a la existencia de este estándar, la comunicación entre circuitos integrados era regularmente realizada en paralelo, lo cual podía ser relativamente más rápido. Sin embargo, de igual forma era poco eficiente en el uso de los recursos físicos ya que limitaba el número de pines disponibles, como muestra la Figura 55 la conexión entre dos circuitos integrados.

[25] NXP Semiconductors. *I^2C-bus Specification and User Manual*. http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: Marzo 2013.

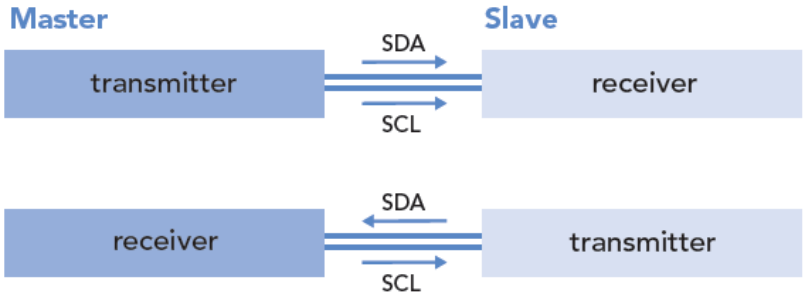
Figura 55. **Conexión paralelo vs. conexión I²C**



Fuente: NXP Semiconductors. *I²C-bus Solutions 2014*.
http://www.nxp.com/documents/leaflet/75017540.pdf. Consulta: 17 de febrero de 2017.

En las conexiones existe un maestro, el cual se encarga en todo momento de proporcionar el reloj serial de la comunicación. Lo único que cambia es la dirección del flujo de información, como se puede apreciar en la figura 56.

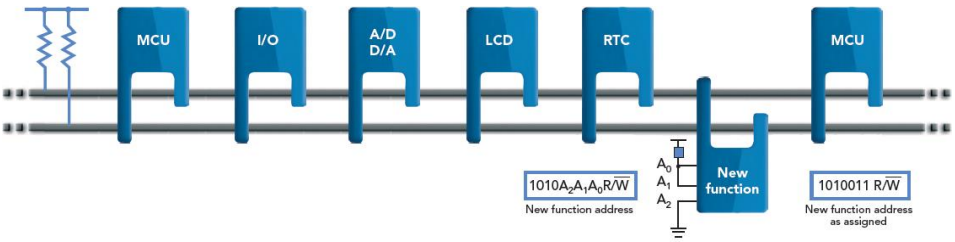
Figura 56. **Diagrama de bloques conexión I²C**



Fuente: NXP Semiconductors. *I²C-bus Solutions 2014*.
http://www.nxp.com/documents/leaflet/75017540.pdf. Consulta: 17 de febrero de 2017

Al momento de tener múltiples esclavos en una conexión, con la capacidad de identificar a cada uno de forma independiente de los demás, se puede observar una topología de comunicación entre los dispositivos. La forma de identificar los dispositivos es mediante una dirección física única para cada uno. En algunos casos es modificable por hardware si se desea conectar más del mismo tipo de dispositivos y que no tengan la misma dirección. Como se puede observar en la Figura 57, una dirección base de 4 bits y los últimos tres modificables de forma física, esto es muy útil puesto que se pueden conectar hasta 15 iguales en el mismo puerto, lo que acrecienta la topología sin necesidad de protocolo, interfaz o procesamiento más elaborado.

Figura 57. **Diagrama de múltiple conexión I^2C**

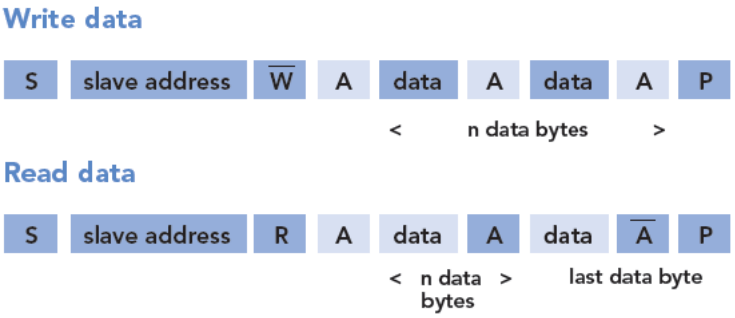


Fuente: NXP Semiconductors. *I^2C-bus Solutions 2014*.
http://www.nxp.com/documents/leaflet/75017540.pdf. Consulta: 17 de febrero de 2017

La forma estándar del protocolo establece que para escribir datos debe iniciarse la transmisión con una condición de inicio “S”, la dirección del dispositivo esclavo al cual se le comunicarán los datos, la instrucción de lectura o escritura (R/W), un acuse de recibo, los datos a ser escritos, seguido de una condición de parada. Debido a que la transmisión es orientada a 8-bits si se pueden transmitir n bytes (donde un byte es un paquete de 8 bits) pero serán separados por un acuse de recibo enviado por el dispositivo esclavo para cada byte. La lectura de datos se realiza de la misma forma, con la única variante

que los datos son enviados por el dispositivo esclavo y los acuses de recibo son enviados por el dispositivo maestro. Al final de la transmisión se concede cuando el maestro envía un acuse de recibo negado seguido de una condición de parada,[26] como lo muestra la figura 58.

Figura 58. **Protocolo de transferencia I^2C**

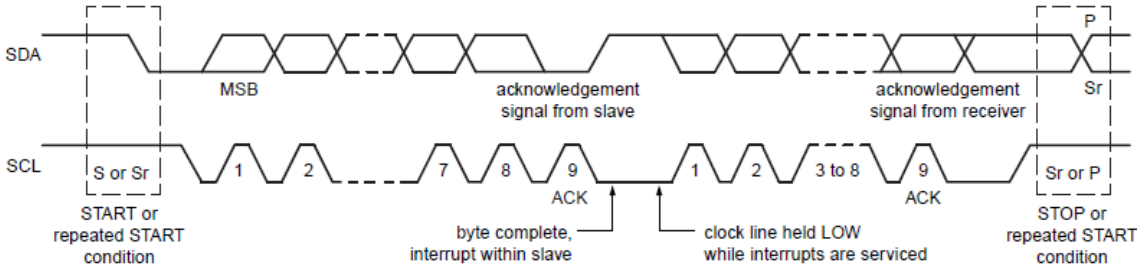


Fuente: NXP Semiconductors. *I^2C-bus Solutions 2014*.
http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: 17 de febrero de 2017

Los diagramas temporales con los que se debe cumplir para que el estándar sea funcional son los mostrados en la figura 59. Se debe tener cuidado con las condiciones de inicio repetido, puesto existen tanto dispositivos esclavos como módulos maestros que no soportan este modo.

[26] NXP Semiconductors, *I2C-bus solutions 2014*. http://www.nxp.com/documents/leaflet/75017540.pdf. Consulta: Marzo 2017.

Figura 59. **Diagrama de tiempo en la comunicación I^2C**



Fuente: NXP Semiconductors, *I^2C-bus Specification and User Manual*. http://www.nxp.com/documents/user_manual/UM10204.pdf. Consulta: 18 de febrerode 2017

El módulo I^2C a utilizar en la implementación del diseño se encuentra en la unidad de aplicación, por lo cual es controlado a través de librerías existentes creadas por Xilinx. Es programado desde el SDK de Xilinx en el programa principal que se ejecuta en el procesador de aplicación.

4.1.6.3. Programación del códec

La programación I^2C del códec se realiza utilizando Xiic.h y Xiicps.h, archivos tipo *header* que son *drivers* diseñados para un dispositivo I^2C, sea maestro o esclavo. Este se llama sobre la instancia del módulo IIC_0 existente en el APU.

Este *driver* es un sistema operativo en tiempo real (*Real Time Operating System*, RTOS) e independiente del procesador; funciona únicamente con direcciones físicas y cualquier necesidad de administración dinámica de

memoria, memoria virtual o control de cache será satisfecha por la capa superior a este *driver.*[27]

La programación del códec se hace cambiando los valores que por defecto tienen los registros de control. Estos se muestran en la figura 60, acorde a los valores requeridos en dependencia de las funciones requeridas.

Figura 60. **Tabla de registros de control de SSM2603**

Reg.	Address	Name	D8	D7	D6	D5	D4	D3	D2	D1	D0	Default
R0	0x00	Left-channel ADC input volume	LRINBOTH	LINMUTE	0	LINVOL[5:0]						010010111
R1	0x01	Right-channel ADC input volume	RLINBOTH	RINMUTE	0	RINVOL[5:0]						010010111
R2	0x02	Left-channel DAC volume	LRHPBOTH	0	LHPVOL[6:0]							001111001
R3	0x03	Right-channel DAC volume	RLHPBOTH	0	RHPVOL[6:0]							001111001
R4	0x04	Analog audio path	0	SIDETONE_ATT[1:0]		SIDETONE_EN	DACSEL	Bypass	INSEL	MUTEMIC	MICBOOST	000001010
R5	0x05	Digital audio path	0	0	0	0	HPOR	DACMU	DEEMPH[1:0]		ADCHPF	000001000
R6	0x06	Power management	0	PWROFF	CLKOUT	OSC	Out	DAC	ADC	MIC	LINEIN	010011111
R7	0x07	Digital audio I/F	0	BCLKINV	MS	LRSWAP	LRP	WL[1:0]		Format[1:0]		000001010
R8	0x08	Sampling rate	0	CLKODIV2	CLKDIV2	SR[3:0]				BOSR	USB	000000000
R9	0x09	Active	0	0	0	0	0	0	0	0	Active	000000000
R15	0x0F	Software reset	Reset[8:0]									000000000
R16	0x10	ALC Control 1	ALCSEL[1:0]		MAXGAIN[2:0]			ALCL[3:0]				001111011
R17	0x11	ALC Control 2	0	DCY[3:0]				ATK[3:0]				000110010
R18	0x12	Noise gate	0	NGTH[4:0]					NGG[1:0]		NGAT	000000000

Fuente: SSM2603 Datasheet. http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2603.pdf. Consulta: 18 de febrero de 2017

El código encargado de la programación del códec se muestra en la figura 61, donde los nombres verbosos dentro del paréntesis son variables con los números hexagesimales equivalentes a las direcciones de la tabla de registros mostrada. Del lado derecho se muestra el valor binario que deben tomar esos registros para la funcionalidad deseada.

[27] Xilinx Inc. *Xilinx Device Drivers Documentation.* http://www.xilinx.com/ise/embedded/xilinx_drivers.pdf. Consulta: Marzo 2017.

Figura 61. **Código de programación SSM2603**

```
AudioWriteToReg(R15_SOFTWARE_RESET,                0b000000000);
usleep(75000);
AudioWriteToReg(R6_POWER_MANAGEMENT,               0b000110000);
AudioWriteToReg(R0_LEFT_CHANNEL_ADC_INPUT_VOLUME,  0b000010111);
AudioWriteToReg(R1_RIGHT_CHANNEL_ADC_INPUT_VOLUME, 0b000010111);
AudioWriteToReg(R2_LEFT_CHANNEL_DAC_VOLUME,        0b001111001);
AudioWriteToReg(R3_RIGHT_CHANNEL_DAC_VOLUME,       0b001111001);
AudioWriteToReg(R4_ANALOG_AUDIO_PATH,              0b000010010);
AudioWriteToReg(R5_DIGITAL_AUDIO_PATH,             0b000000111);
AudioWriteToReg(R7_DIGITAL_AUDIO_I_F,              0b000001110);
AudioWriteToReg(R8_SAMPLING_RATE,                  0b000000000);
usleep(75000);
AudioWriteToReg(R9_ACTIVE,                         0b000000001);
AudioWriteToReg(R6_POWER_MANAGEMENT,               0b000100010);
```

Fuente: elaboración propia.

4.1.7. Frecuencia de salida teórica

Basados en la teoría de operación expuesta en la parte de síntesis digital y del diseño de NCO, la frecuencia de salida es una función de tres factores principales: la frecuencia del reloj de referencia, el número de bits en el acumulador de fase (función B) y el valor de incremento de fase (cambio en theta), como se observa en la figura 61.

Figura 62. **Ecuación frecuencia de salida**

$$f_{\text{out}} = \frac{f_{\text{clk}} \Delta\theta}{2^{B_{\Theta(n)}}} \text{ Hz}$$

Fuente: DDS v5.0. https://www.xilinx.com/support/documentation/ip_documentation/dds.pdf. Consulta: 19 de febrero de 2017.

La capacidad teórica del módulo, suponiendo que el funcionamiento únicamente depende del NCO, se tiene en dos momentos: la frecuencia más baja obtenible es cuando el valor del incremento de fase es la unidad; y la frecuencia más alta posible depende de la cantidad de muestras en la tabla de búsqueda, ya que a menor muestras utilizadas mayor distorsión de la señal. Por lo tanto, el valor más alto de incremento de fase en el ejemplo será 125 para así obtener 32 muestras por período. La generación funciona hasta con 15 muestras por período. Sin embargo, se utilizarán los valores más conservadores posibles.

Con los datos anteriores se obtiene una frecuencia de salida mínima de 24.4 kHz y una frecuencia máxima de 3.05 MHz. La resolución en frecuencia es la menor frecuencia de salida cuando el incremento en fase iguala la unidad.

La frecuencia de salida en esta aplicación no depende únicamente de las tres variables mencionadas con anterioridad, ya que por ser un sistema más complejo se presentan otros factores delimitantes. Algunos de ellos son la limitación física de los interruptores en el incremento de fase a únicamente tener un valor máximo 15; el reloj de referencia utilizado por el DAC, la multiplexación de los datos realizada en el canal de entrada al DAC.

La información proporcionada por la hoja de datos respecto a las consideraciones de reloj se establecen en la Figura 63. El reloj maestro (MCLK) de 12.288 MHz se genera en el PS y se interconecta de forma directa al códec. En la Figura 63 se puede observar que dadas las especificaciones del códec, la frecuencia de transmisión de muestra (BCLK) es finalmente MCLK dividido en un factor de 4. Se tiene así para obtener una frecuencia de muestreo de 48 kHz, y como resultado un reloj BCLK de 3.072 MHz.

Figura 63. **Ecuación frecuencia de salida**

MCLK (CLKDIV2 = 0)	MCLK (CLKDIV2 = 1)	ADC Sampling Rate (RECLRC)	DAC Sampling Rate (PBLRC)	USB	SR[3:0]	BOSR	BCLK (MS = 1)[1]
12.288 MHz	24.576 MHz	8 kHz (MCLK/1536)	8 kHz (MCLK/1536)	0	0011	0	MCLK/4
		8 kHz (MCLK/1536)	48 kHz (MCLK/256)	0	0010	0	MCLK/4
		12 kHz (MCLK/1024)	12 kHz (MCLK/1024)	0	0100	0	MCLK/4
		16 kHz (MCLK/768)	16 kHz (MCLK/768)	0	0101	0	MCLK/4
		24 kHz (MCLK/512)	24 kHz (MCLK/512)	0	1110	0	MCLK/4
		32 kHz (MCLK/384)	32 kHz (MCLK/384)	0	0110	0	MCLK/4
		48 kHz (MCLK/256)	8 kHz (MCLK/1536)	0	0001	0	MCLK/4
		48 kHz (MCLK/256)	48 kHz (MCLK/256)	0	0000	0	MCLK/4
		96 kHz (MCLK/128)	96 kHz (MCLK/128)	0	0111	0	MCLK/2

Fuente: DDS v5.0. https://www.xilinx.com/support/documentation/ip_documentation/dds.pdf. Consulta: 19 de febrero de 2017.

Se procede a calcular el valor máximo de frecuencia de salida ocurrente cuando todos los interruptores de la plataforma ZyBo se encuentran activados, ya que son 4 y representan un número binario de 4 bits. El mayor número representable es 15. En la ecuación 1 se muestra el acumulador de fase, que es de 12 bits; el aumento máximo de fase es de 15 muestras y la frecuencia de referencia 3.072 MHz. Por lo tanto se obtiene una frecuencia de máxima de salida teórica de 11.250 MHz.

$$B_{\Theta(\mathrm{n})} = 12\mathrm{bit},\ \ \Delta\theta_{max} = 15,\ \ \ f_{clk} = 3.072\mathrm{MHz}$$

$$f_{out_{max}} = \frac{f_{clk} * \Delta\theta_{max}}{2^{B_{\Theta(n)}}}\mathrm{Hz} = 11{,}250\mathrm{Hz} \quad (1)$$

La resolución de frecuencia teórica se calcula para obtener valores posibles de incerteza y conocer los cambios esperados en función del paso entre muestras. En la ecuación 2 se muestra el cálculo para los parámetros actuales.

$$B_{\Theta(\mathrm{n})} = 12\mathrm{bit},\ \ \ f_{clk} = 3.072\mathrm{MHz}$$

$$f_{out_{max}} = \frac{f_{clk}}{2^{B_{\Theta(n)}}}\text{Hz} = 750\text{Hz} \quad (2)$$

La frecuencia de salida teórica esperada con cotas de incerteza permite contemplar posibles errores generados por cambios inesperados del paso entre muestras. Ahora bien, no contempla errores experimentales como incerteza de relojes, entre otros. En la ecuación 3 se muestra el cálculo puramente teórico. Según los modelos encontrados en las referencias bibliográficas, este modelo debe ser ajustado más adelante, dados los errores de implementación y ser comparado con los datos obtenidos de forma experimental.

$$f_{out_{max}} = (11{,}250 \pm 750)\text{Hz} \quad (3)$$

4.2. Implementación del sistema

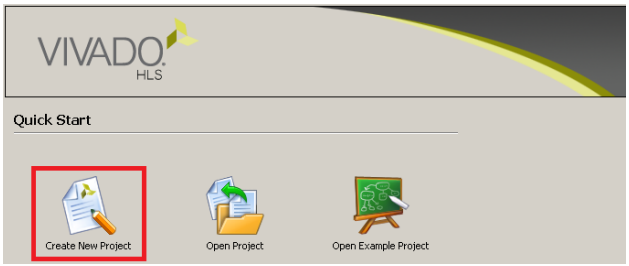
La implementación del sistema es llevar al mundo real con las herramientas y teorías estudiadas, mediante los métodos propuestos, una solución al problema inicialmente planteado.

4.2.1. Creación de solución HLS

Para la creación de un bloque de propiedad intelectual (*Intellectual Property,* IP) que posteriormente pueda ser agregado al diseño por bloques de sistemas más complejos, es necesario tomar la abstracción programática realizada en la sección 4.1.3 de este documento y agregar las interfaces de control pertinente, así como realizar las simulaciones correspondientes de verificación.

Para la creación se usará el software Vivado™ HLS versión 2015.4. Se ejecuta el software y se procede a crear un nuevo proyecto, al cual se le asigna un nombre y ubicación específica.

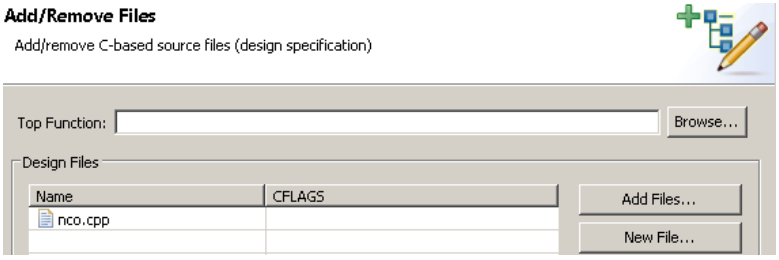
Figura 64. **Creación nuevo proyecto**



Fuente: elaboración propia

Se procede a importar los archivos fuente con base en los cuales se generará la solución. Estos son dos archivos: uno fuente del módulo y otro fuente de prueba del módulo.

Figura 65. **Importación de archivos fuente**

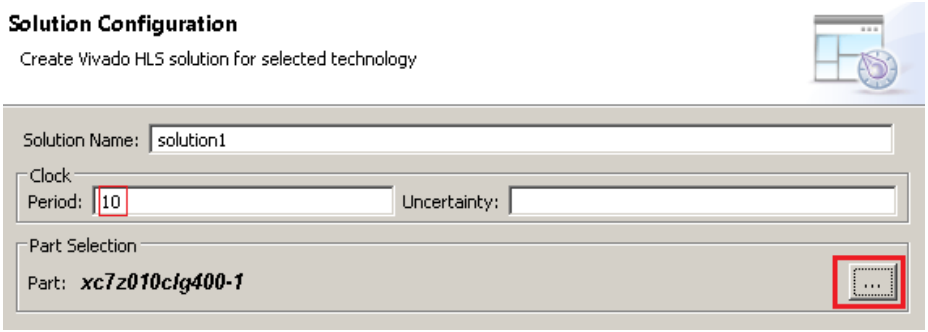


Fuente: elaboración propia

El programa solicitará al usuario seleccionar la parte para la cual será creada la solución de alto nivel. El modelo y familia del SoC que se utilizará es

el que se debe buscar. Para este caso particular, es la parte número XC7Z010CLG400-1. Luego se selecciona un período en el reloj de 10; la interfaz de usuario no lo especifica pero son 10 nano segundos, para que el módulo pueda trabajar con un reloj de 100 MHz.

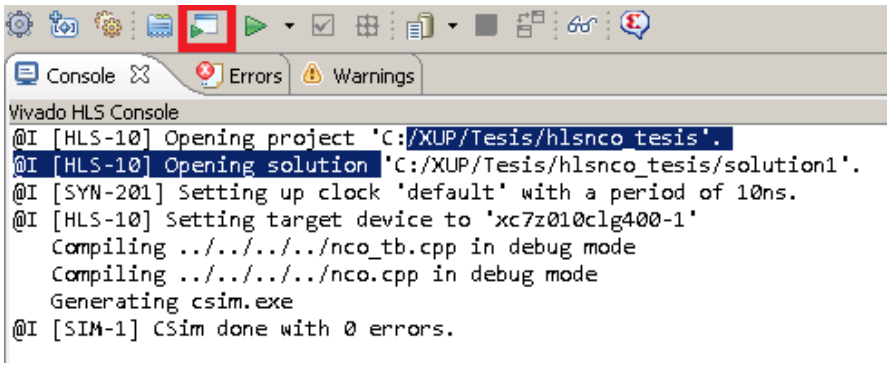
Figura 66. **Selección de parte y periodo**



Fuente: elaboración propia

Se ejecuta una simulación para corroborar y validar la coherencia entre la programación realizada en el archivo fuente del módulo y el comportamiento esperado de este en el archivo fuente de prueba (test bench), así como correcciones de sintaxis que puedan ser halladas en ambos archivos. Para iniciar la simulación se debe hacer clic en el botón indicado en la Figura 67 y analizar el contenido que posteriormente se obtendrá en la consola. Este mismo contenido se encuentra en un archivo creado llamado "_csim.log".

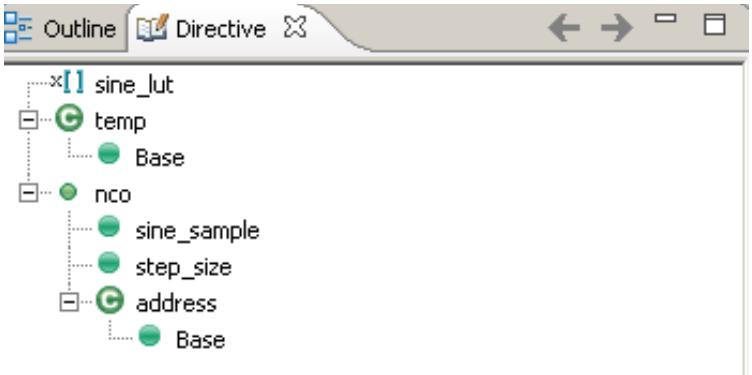
Figura 67. **Simulación de archivos fuente**



Fuente: elaboración propia

Se procede a agregar las interfaces pertinentes al módulo mediante comandos constructivos llamados directivas. Para esto se selecciona la lengüeta del archivo fuente del módulo como activa (que sea el archivo en el cual se encuentre trabajando); al lado derecho se encontrará un recuadro como el que se muestra en la Figura 68, en el cual se insertarán las directivas adecuadas.

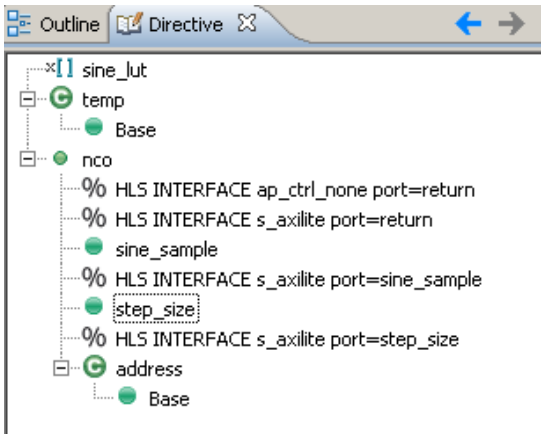
Figura 68. **Panel de directivas**



Fuente: elaboración propia

A continuación, se crean las directivas respectivas en cada uno de los puertos y registros a utilizarse. Estas se crean haciendo clic derecho sobre el puerto o variable deseada, y clic izquierdo sobre la opción insertar directiva (Insert Directive). En el cuadro de diálogo emergente se selecciona que se desea una interfaz y qué tipo; estas se eligen en listas que se despliegan en el menú de diálogo. Para la interfaz AXI-LITE debe seleccionarse la opción “s_axilite” y para la interfaz de remoción de señales innecesarias de control “ap_ctrl_none”. Se debe seleccionar tres interfaces AXI-LITE: una para la función “nco”, para la cual también se debe crear una interfaz de remoción de señales innecesarias de control; una para la variable “sine_sample” y una para la variable “step_size”.

Figura 69. **Panel de directivas aplicadas**

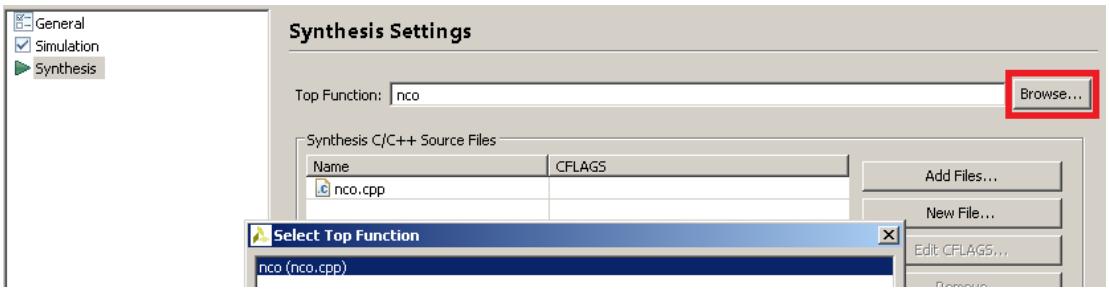


Fuente: elaboración propia

Cuando se programa en VHDL es necesario establecer un “top” que es el nivel más alto de un diseño jerárquico. Esto define los puertos del módulo y la arquitectura interna con base en instanciaciones lógicas de submódulos y su interconexión entre ellos. En HLS es necesario hacerlo pero estableciendo una

función programática como “top”. Esto se puede realizar al hacer clic en “Project” en la barra de menú principal; luego otro clic en “Project Setting” y en el cuadro de diálogo seleccionar “Synthesis”. En este cuadro aparecen todos los archivos fuente agregados o creados. Se hace clic en la opción “Browse”, lo que permitirá seleccionar de una lista de funciones existentes la que será el nivel más alto (top) en la jerarquía del diseño.

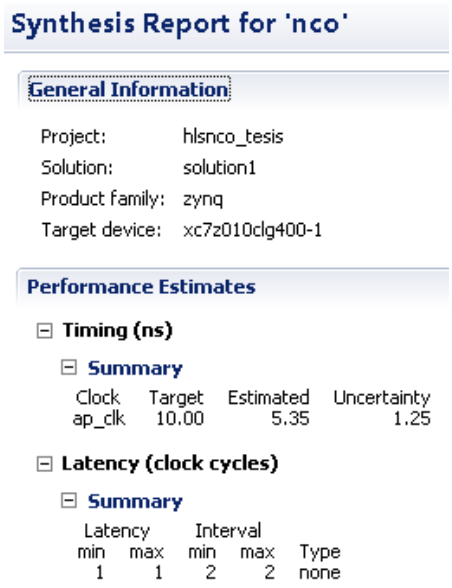
Figura 70. **Selección de función *top***



Fuente: elaboración propia

Se realiza la síntesis en C haciendo clic en la opción “*Solution*” que se localiza en la barra principal de herramientas; luego, al colocar el puntero encima de la opción “*Run C Synthesis*” se despliega un menú, en el cual se debe hacer clic sobre “*Active Solution*”.

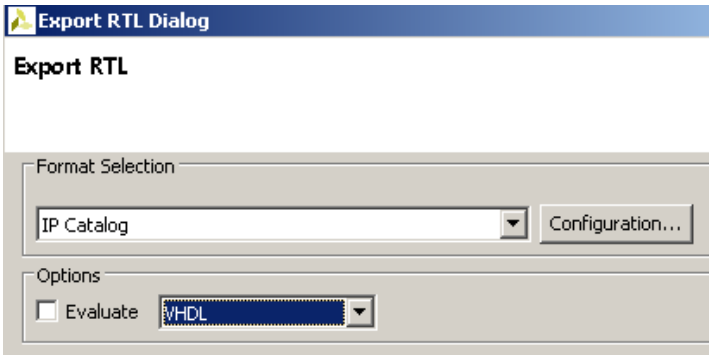
Figura 71. **Reporte de síntesis**



Fuente: elaboración propia

Se debe exportar el RTL creado en el paso anterior después de la síntesis. Para esto se hace clic en el menú "*Solution*" que se encuentra en la barra principal de herramientas y luego, clic en la opción "*Export RTL*". Se selecciona el formato de catálogo IP, ya que es el predilecto de las herramientas Xilinx y por convención se establece que el código de descripción de hardware sea VHDL. Puede ser Verilog o bien mixto.

Figura 72. **Menú de exportación de RTL**

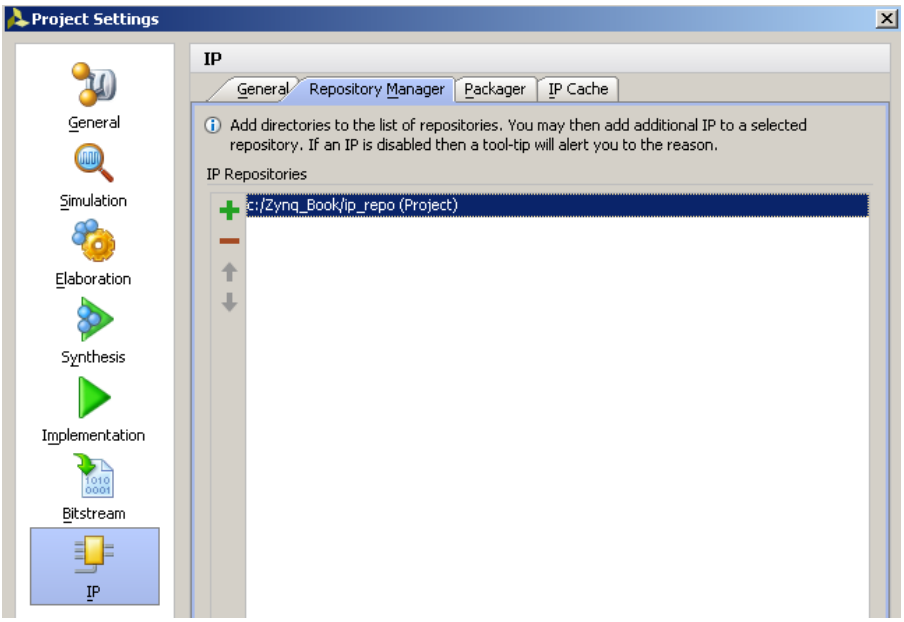


Fuente: elaboración propia

4.2.2. Repositorios IP

Es necesario crear un repositorio en el cual se encuentren todos los bloques IP que se utilizarán en la construcción del sistema más adelante. Esto se puede hacer desde la barra principal de herramientas. Se da clic en "*Tools*" luego en "*Project Settings*". En el cuadro de diálogo emergente se selecciona IP. Allí es donde se pueden agregar o quitar directorios, en donde se encuentren los bloques IP.

Figura 73. **Menú de repositorios IP**



Fuente: elaboración propia

4.2.3. Creación de sistema con integrador IP

A pesar de que el diseño RTL permite el mayor nivel de personalización, la aceleración de desarrollo de sistemas más inteligentes requiere niveles de automatización más allá de este. Xilinx entrega un ambiente de desarrollo enfocado en SoC, y céntrico tanto en sistemas como IP a manera de "taclear" los principales puntos de ralentización en la integración e implementación del sistema.[28]

Provee un flujo de desarrollo con las siguientes características:

- Gráfico

[28] Xilinx Inc. *Vivado IP integrator: Accelerated time to IP creation and integration*. https://www.xilinx.com/publications/prod_mktg/vivado/Vivado_IP_Integrator_Backgrounder.pdf. Consulta: Abril 2017.

- Basado en TCL
- Corrección en construcción
- Orientado a sistemas
- Enfocado en IP

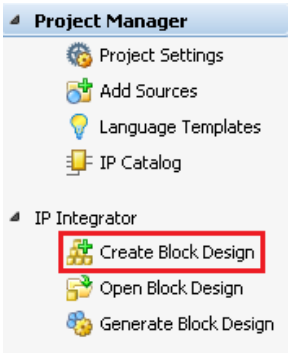
Con un flujo de trabajo enfocado en abstracciones más elevadas como el enfoque en sistema y el enfoque en bloques de propiedad intelectual (IP), ya de por sí se puede esperar una disminución en el tiempo necesario para crear e integrar, ya que estas abstracciones permiten el manejo de los módulos de forma genérica y facilitan múltiples aplicaciones de forma más inmediata.

4.2.4. Diseño por bloques

Es una de las mayores ventajas de este flujo de trabajo: proveer un inmenso catálogo IP que puede ser compartido a un equipo de diseño, división o compañía. El diseño de sistemas a partir de bloques nace de la tecnología analítica de colocación y ruteo creada para el fácil diseño de sistemas, ya que cualquier bloque IP se encuentra a un clic de distancia, inclusive los más complicados como *microblaze,* el cual es un IP de microprocesador, además de sistemas de procesamiento preconfigurados para los sistemas ZYNQ AP.

En el menú lateral llamado "*Flow Navigator*" se encuentran las opciones de ejecución necesarias durante el proceso de diseño y desarrollo de una plataforma de hardware. Al hacer clic donde se muestra en la figura 74 se invoca al integrador IP a crear un nuevo diseño por bloques, al cual se le debe nombrar.

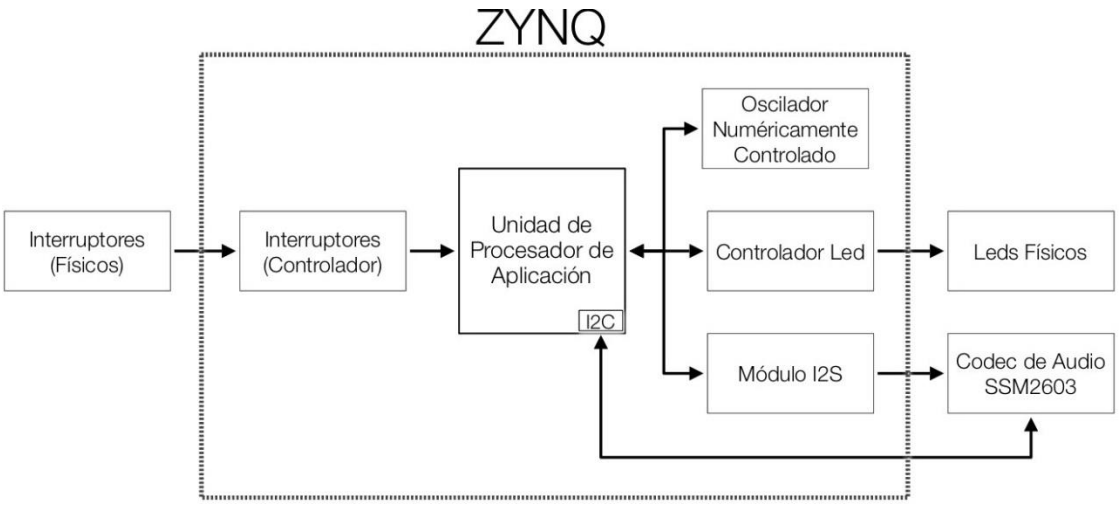
Figura 74. **Creación por bloques**



Fuente: elaboración propia

La creación del diagrama de bloques se hace ajustando los recursos disponibles. El que se concibió como solución posible al problema planteado en la sección 4.1, en la figura 75 se muestra como una versión más completa de este planteamiento, que servirá como guía al proceso de creación del sistema. La parte interna del recuadro con línea punteada es la parte del sistema que será implementada dentro del SoC ZYNQ, tanto en PS como en PL.

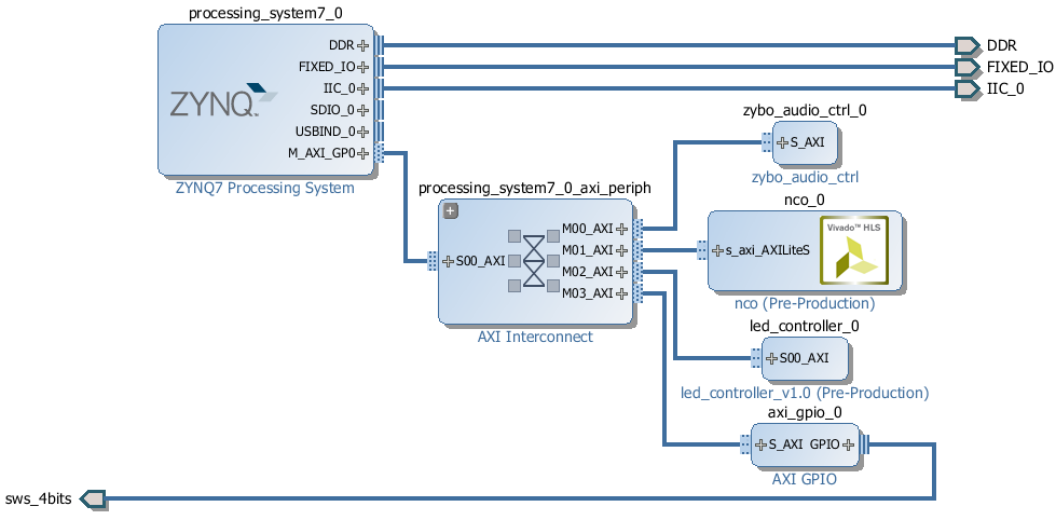
Figura 75. **Diagrama de bloques del sistema completo**



Fuente: elaboración propia

Una versión minimizada del diseño a implementar en el SoC visto en el integrador IP se muestra en la figura 76. Se le llama minimizada debido a que sólo muestra las conexiones de interfaz, y como es posible observar, contiene todos los elementos necesarios: una unidad de procesamiento desde el cual se controla directamente la conexión I2C a través del bus IIC_0, un sistema de interconexión AXI, el controlador de audio I2S, el módulo NCO realizado en HLS, un controlador led, y un controlador GPIO para los interruptores.

Figura 76. **Diagrama de bloques minimizados integrador IP**



Fuente: elaboración propia

4.2.5. Validación del diseño

La validación del diseño es un proceso de verificación de conexiones en el que se hace evidente las conexiones erróneas existentes o bien las que se debieron realizar; sin embargo, si el usuario no lo hizo, es necesario un proceso de corrección en construcción, lo que aumenta las probabilidades de un diseño funcional. Se basa en un grupo de normas estandarizadas pero flexibles en cuanto al diseño, las cuales se denominan reglas de restricción de diseño (*Design Rules Constraint*, DRC). La validación se realiza desde la barra principal de menú en el submenú "*Tools*" haciendo clic en la opción "*Validate Design*" o presionando la tecla F6. De este proceso se espera una respuesta afirmativa de la no existencia de errores en el diseño. Como se muestra en la figura 77, es posible avanzar.

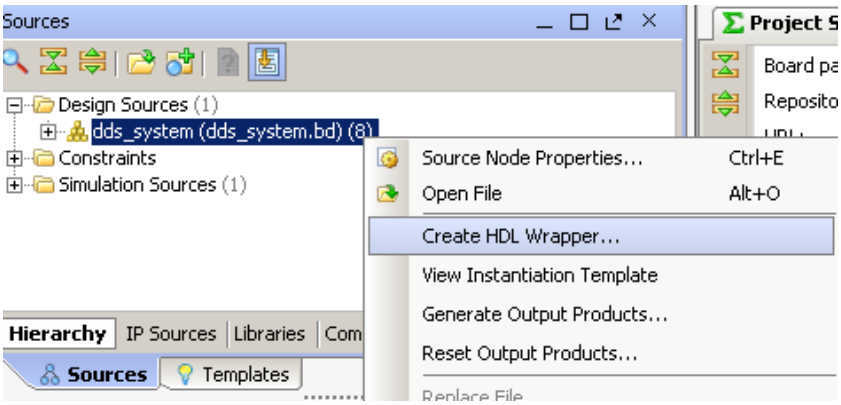
Figura 77. **Mensaje de validación exitosa**

Fuente: elaboración propia

4.2.6. Generación de archivo contenedor

El archivo contenedor es el archivo VHDL o Verilog que fue generado con base en el diseño de bloques. Finalmente es un archivo *top* en un diseño jerárquico; es decir, en este se crean instancias de todos los bloques IP que fueron colocados en el diseño. Este archivo contenedor (*Wrapper File*) es el que se sintetizará en los siguientes pasos. Los puertos de conexión que pueda tener son los que se deben incluir en archivo de restricciones XDC. Tome en cuenta la capacidad del software para trabajar con leguajes de descripción de hardware mixto, así que es posible que alguno de los archivos contenga instancias creadas en Verilog aunque el archivo contenedor esté hecho en VHDL. En la figura 78 se demuestra cómo iniciar la creación del archivo contenedor haciendo clic izquierdo en el diseño de bloques y para luego seleccionar la opción "*Create HDL Wrapper*" de la lista emergente.

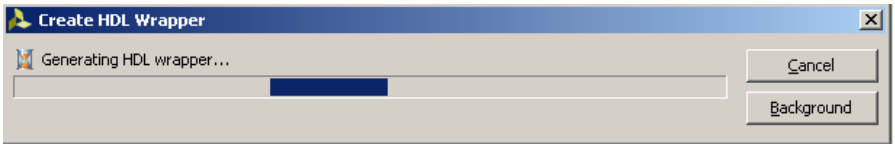
Figura 78. **Ejecutar creación de archivo contenedor**



Fuente: elaboración propia

Al iniciar la creación del archivo contenedor, el sistema preguntará si se desea tener una copia para ediciones posteriores por parte del usuario o si se desea permitir al software realizar actualizaciones automáticas. Como normalmente no se desean hacer modificaciones posteriores al sistema como tal, es recomendable permitir al software realizar las actualizaciones del archivo de forma automática; de esta forma, también se previenen problemas posiblemente causados por manejo de versiones y copias indeseadas.

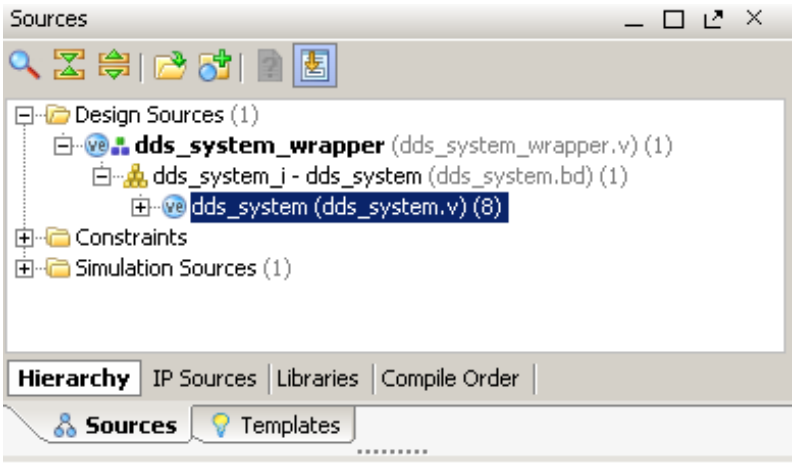
Figura 79. **Mensaje de creación de archivo contenedor**



Fuente: elaboración propia

Cuando el archivo contenedor ha sido creado, la jerarquía del sistema de bloques cambia y coloca el contenedor (*wrapper*) en la parte más alta, seguido del diseño en bloques del sistema; después, el archivo de descripción que contiene todas las instancias, como se muestra en la figura 80.

Figura 80. **Archivo contenedor**



Fuente: elaboración propia

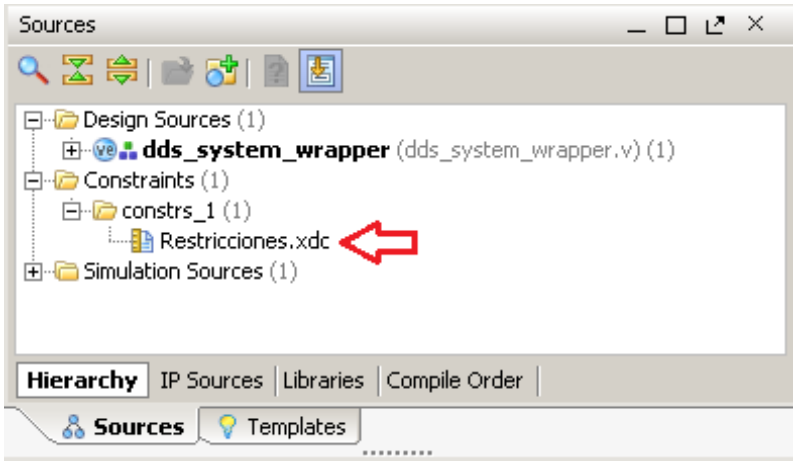
4.2.7. Restricciones de síntesis

Las restricciones de síntesis son aquellas que le dicen al software sintetizador dónde debe realizar las conexiones pertinentes a los puertos en el diseño superior del archivo contenedor. Estos son los puertos que se conectarán a algún pin de salida del SoC. En estos archivos de restricción se le hace saber al software a cuál pin debe conectarlo, con qué estándar de voltaje, si es algún bus determinado o si solamente es un pin de uso general.

Los archivos Xilinx de restricciones de diseño (*Xilinx Design Constraints*, XDC) son los que se utilizan en Vivado Suite en comparación del archivo de

restricciones del usuario (*User Constraints File*, UCF), utilizado en versiones de ISE Design Suite. Esto se debe a que los XDC están basados en el estándar de restricciones de diseño de sinopsis (*Synopsys Design Constraints*, SDC), es cual es de los más utilizados y probados en la industria.

Figura 81. **Ubicación archivo XDC**

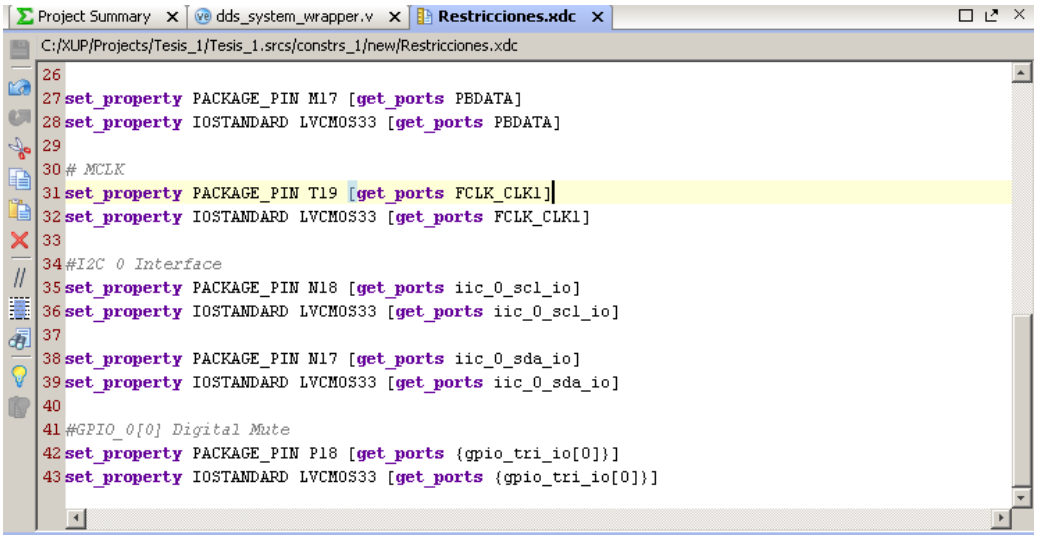


Fuente: elaboración propia

El archivo XDC no está lleno de cadenas de texto que después se procesen posteriormente por Vivado, sino que son comandos formulados acorde a la semántica establecida por la consola TCL. Son de formato genérico; por lo tanto, pueden ser interpretados por cualquier otra consola TCL y no necesariamente la de Vivado.[29] En la figura 82 se muestra el contenido el archivo de restricciones utilizado en este diseño.

[29] Xilinx Inc. *User Guide: User Constraints*. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug903-vivado-using-constraints.pdf. Consulta: Abril 2017.

Figura 82. **Archivo de restricciones XDC**

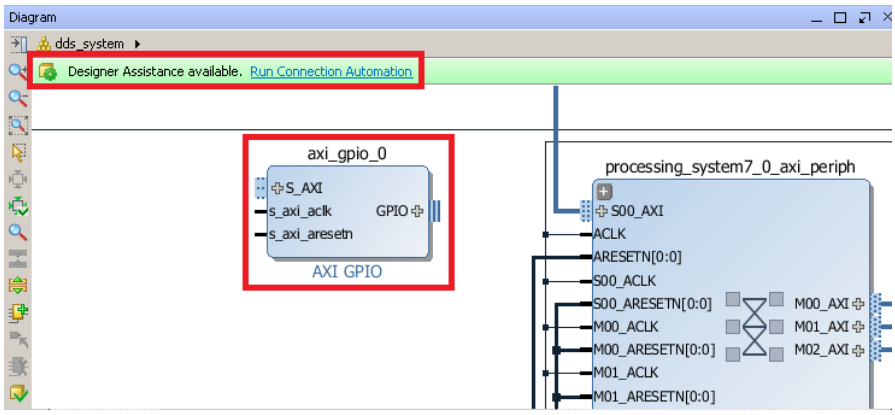


Fuente: elaboración propia

4.2.8. Interconexión del sistema

La interconexión del sistema se realiza mediante un sistema automatización de conexión con asistentes de diseño, los cuales notifican al usuario cuando un proceso es automatizable a través de alguna herramienta existente en el flujo de trabajo, en aras de la automatización. Esto evita la engorrosa tarea de crear desde cero las interfaces que, de por sí, son estándar en la implementación de cualquier sistema. Dichas interfaces pueden llegar a ser muy complicadas dependiendo de las velocidades y modos de transferencia; por lo tanto, es muy apreciable la automatización de esta tarea en tiempo de desarrollo e implementación.

Figura 83. **Asistente de automatización de conexión**

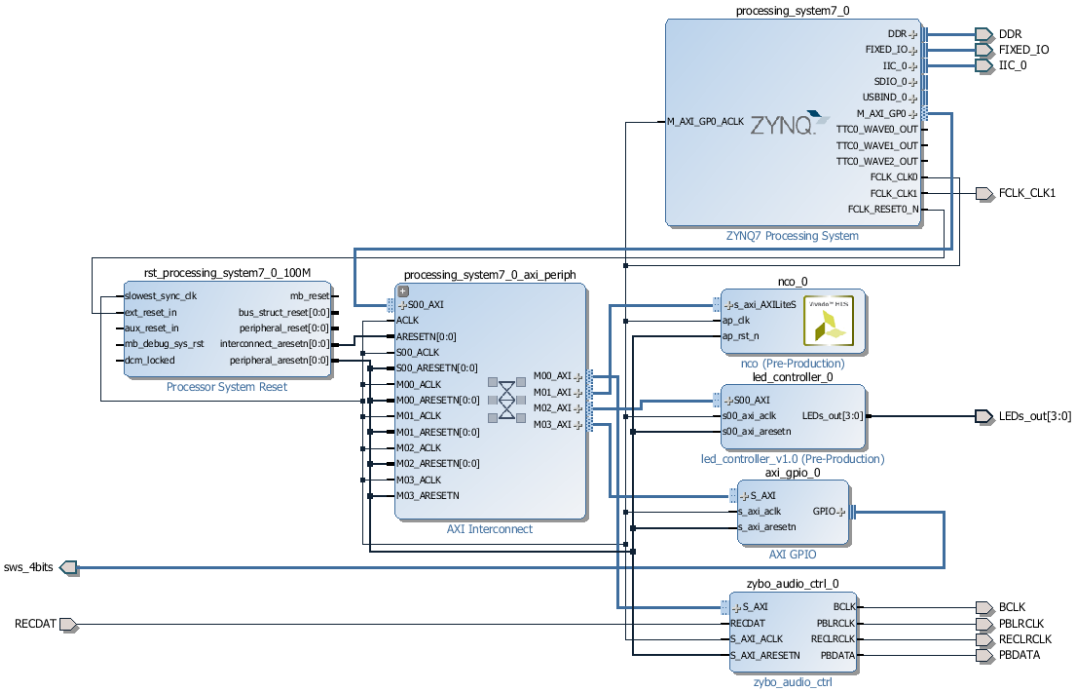


Fuente: elaboración propia

En la figura 84 se puede observar el diagrama de bloques completo con las interconexiones pertinentes, los dispositivos de multiplicación y arbitraje entre los módulos que intentan conectarse, así como los submódulos de reinicio, entre otros. Los módulos de la unidad de procesamiento son preconfiguraciones de inicio que habilitan determinadas funcionalidades, buses, relojes, conexiones internas, etc.

Los submódulos de control de intercomunicaciones también pueden (en dependencia de la complejidad de la interfaz) estar encargados de la generación de señales de control en puertos seriales, así como de la anulación de estas señales en caso no sean necesarias al módulo. Es una cuestión que los asistentes de automatización toman entre sus atribuciones únicamente con la especificación del módulo IP.

Figura 84. **Diagrama de bloques con interconexiones**



Fuente: elaboración propia

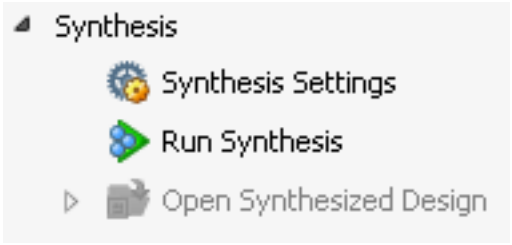
4.2.9. Síntesis

La síntesis es uno de los procesos más directos existentes en todo el flujo de trabajo. Es un proceso analítico de colocación y conexión de los recursos lógicos en la FPGA a manera de sintetizar los comportamientos descritos en la programación a través lenguajes de descripción de hardware (HDL).

Es recomendable, antes de pasar a la síntesis, ejecutar simulaciones de comportamiento (*Behavioral*) y realizar un análisis del nivel de transferencia de registros (*Register Transfer Level*, RTL) antes de llegar directamente al proceso de síntesis, ya que puede arrojar información útil en el proceso de diseño. Otra metodología de trabajo es la obtención de información sobre iteraciones del

diseño en implementaciones, en las cuales se puede sintetizar en una etapa más temprana pero siempre se puede recurrir a estas herramientas en caso de encontrar errores.

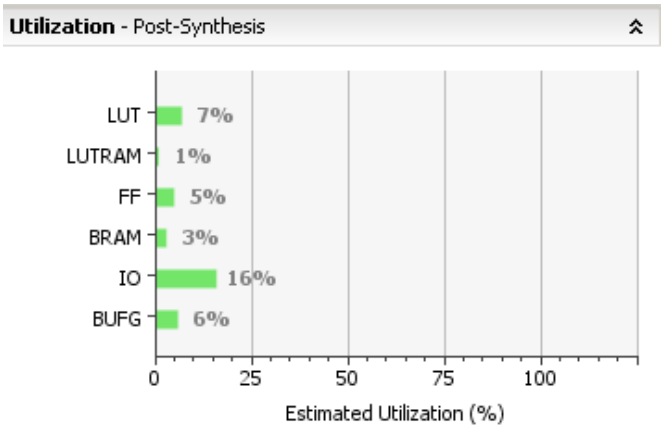
Figura 85. **Ejecutar síntesis**



Fuente: elaboración propia

Al ejecutar la síntesis y terminar será visible la información sobre el consumo de recursos, tiempo que tomó la síntesis y sobre qué proyecto se realizó, etc.

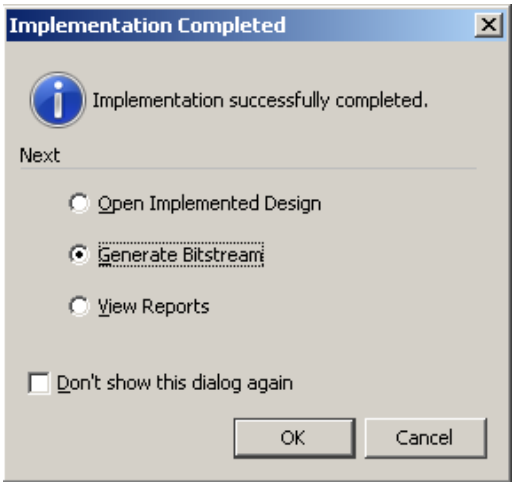
Figura 86. **Sumario de la síntesis**



Fuente: elaboración propia

Acto seguido de la síntesis se realiza la implementación, tomando en cuenta los archivos de restricción aplicados al dispositivo físico y, posteriormente, la escritura del archivo *bitstream.* Este es utilizado por la FPGA para programarse a su encendido.

Figura 87. **Cuadro de diálogo escritura de archivo *bitstream***

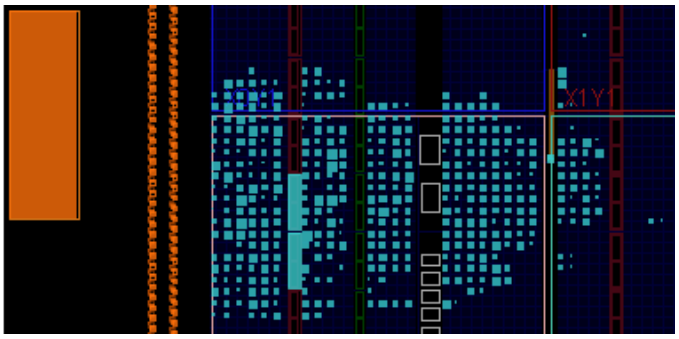


Fuente: elaboración propia

Como último paso antes de iniciar el desarrollo de las aplicaciones, debe seleccionarse la opción "*Open Implemented Design*". Esta abre una representación de la implementación del hardware sobre un mapa de piso del PL en nuestro SoC, como se muestra en la figura 88.

Se muestra de forma gráfica la utilización de recursos que se mostraba anteriormente en un gráfico de barras, pero como este tomaría lugar de forma física en el FPGA.

Figura 88. **Implementación del dispositivo**



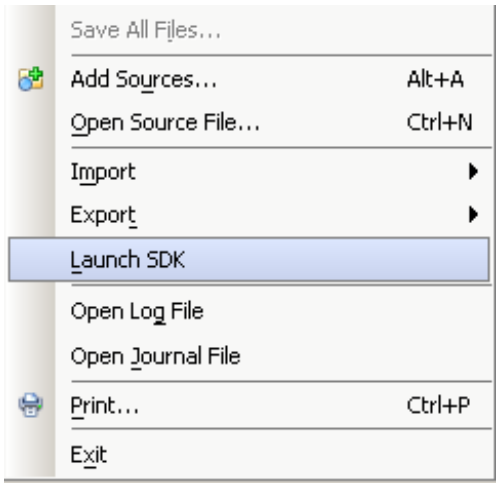
Fuente: elaboración propia

4.3. Creación de aplicación en SDK

4.3.1. Información de la plataforma de hardware

Importar la información de la plataforma de hardware con la cual se está trabajando engloba tanto la información de la tarjeta de desarrollo ZyBo como al hardware sintetizado en su interior y a todo el sistema contenedor de la arquitectura; por lo tanto, se debe hacer desde donde esto fue concebido. En Vivado, en la barra de menú principal, al hacer clic sobre la opción “File”, se localiza la opción “Launch SDK” que no solamente ejecuta el software SDK sino también exporta el hardware con el cual se estuvo trabajando. El hardware implementado debe encontrarse abierto en ventana para que se exporte de forma predeterminada.

Figura 89. **Menú de exportación al SDK**



Fuente: elaboración propia

Después de preguntar si se desea lanzar el SDK sobre el proyecto local (que sí se desea) se inicia el programa del SDK.

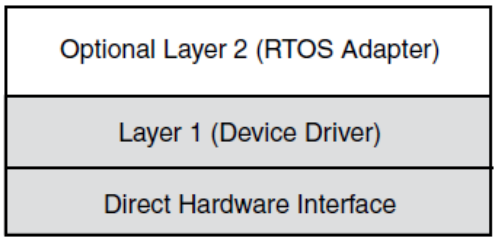
Figura 90. **Pantalla de inicio del SDK**

Fuente: elaboración propia

4.3.2. Creación del paquete de soporte

El objetivo principal de esta sección es la creación de la aplicación *Bare-metal* en un paquete de soporte para la tarjeta, pero son igualmente importantes muchas de las tareas, exceptuando las decisiones arquitecturales del diseño de sistema y la programación. La estructura regular de los *drivers* utilizados en esta clase de aplicaciones se muestra en la Figura 91. Las interfaces directas de hardware no añaden una función llamada "*Overhead*" sino que son implementadas a través de un conjunto de manifiestos de constantes y macros.[30]

Figura 91. **Arquitectura del driver de dispositivo *Bare-Metal***



Fuente: elaboración propia

Para crear un nuevo BSP únicamente se debe seleccionar "*File*" en la barra de menú principal; luego "*New*" y finalmente "*Application Project*". Aquí también debe agregarse los repositorios pertinentes y dependencias adecuadas en función de los archivos fuente que se desee añadir.

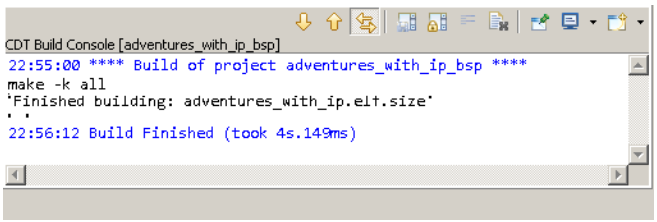
[30] Xilinx Inc. *Zynq-7000 All Programmable SoC Software Development Guide*. https://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf. Consulta: Abril 2017.

Es también posible la utilización de BSP de terceros fabricantes, como por ejemplo FreeRTOS u otros; sin embargo, por motivos de automatización en el proceso y para evitar las complicaciones que puedan desencadenar el añadir variables inciertas al usuario, es mejor utilizar las herramientas proporcionadas por el propio fabricante.

Al terminar la programación acorde a los *drivers* obtenidos en la creación del sistema de hardware y en la importación de fuentes, resta hacer un proceso de depuración con las herramientas proporcionadas, tanto para errores de programación como errores semánticos y de escritura. La depuración es uno de los pasos finales antes de la compilación; sin embargo, en este flujo de trabajo, debido a que no son programas de gran tamaño, la compilación se vuelve un proceso rápido y, por lo tanto, se puede realizar múltiples veces sin afectar el tiempo de desarrollo.

La compilación del proyecto es el proceso mediante un programa llamado compilador, el cual traduce un programa escrito en un lenguaje de programación a otro lenguaje diferente.[31] Regularmente estos son utilizados para traducir a lenguaje máquina, un lenguaje más entendible y rápidamente asimilable por cualquier computador

Figura 92. **Consola al momento de la compilación**



Fuente: elaboración propia

[31] CLOCKSIN, William. *Clause and Effect*. p.93.

4.4. Implementación y prueba

La implementación se realiza mediante el sistema de hardware desarrollado en la primera parte, la cual da como resultado un archivo *bitstream.* Este sirve para programar la FPGA con los arreglos lógicos requeridos, un sistema de aplicación para el núcleo ARM interno, las interconexiones entre los diferentes módulos existentes, las formas referenciales existentes entre los módulos para comunicación, así como las librerías programáticas (*drivers*) para su correcta utilización desde el procesador de aplicación.

La segunda parte es la encargada de unificar todo lo creado con anterioridad en aplicaciones con lenguajes de alto nivel, donde finalmente se realizan las pruebas de funcionamiento pertinentes mediante los procesos de depuración en los paquetes de desarrollo de software proporcionados por el fabricante (SDK), y mesuras realizadas con instrumentación de laboratorios y osciloscopios de una tasa de muestreo de 100 MHz marca RIGOL modelo D1152.

4.4.1. Programar el dispositivo y correr la aplicación

La programación de la FPGA se realiza desde el SDK en la barra mostrada en la Figura 93.

Figura 93. **Herramienta de programación FPGA**

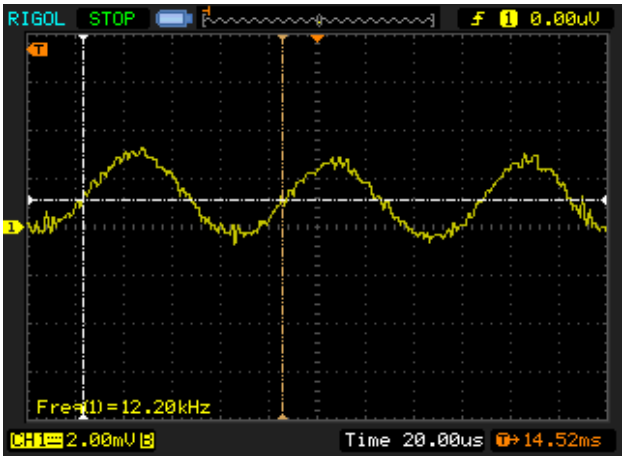
Fuente: elaboración propia

La programación se realiza a través de la interfaz JTAG existente en la tarjeta de desarrollo. JTAG no es una interfaz dedicada únicamente a la programación del dispositivo sino de propósito de prueba general; sin embargo, es utilizada tanto para la programación del FPGA como para la programación de procesador de aplicación (ARM)

4.4.2. Frecuencia de salida experimental

La frecuencia experimental mesurada es de 12.2 kHz con una amplitud de 4 mV de voltaje pico a pico (V_{p-p}). Los voltajes obtenidos en esta medición son bastante pequeños, ya que son de señales especiales para audífonos. Por tanto, lo que la potencia final busca disipar es de mili watts; para esto es necesario que tanto los componentes de voltaje como de corriente sean pequeñas. Además existe un pequeño factor de atenuación proporcionado por las características del códec de audio a frecuencias altas para protección de usuario.

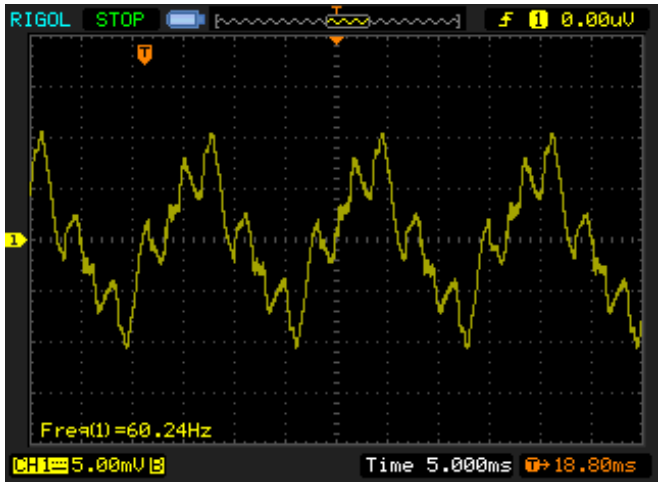
Figura 94. **Frecuencia experimental**



Fuente: elaboración propia

También de forma experimental es necesario tomar en cuenta el ruido que pueda existir por la interconexión de dispositivos, por los dispositivos conectados a la red y el ruido de la transmisión de energía. Todos estos son factores que atenúan una señal de procesamiento tan pequeña. En la Figura 95 se muestra el ruido de 60 Hz que de por sí existía en la red en la que se realizaron las mediciones. Mucho de este ruido logra ser suprimido mediante correctas conexiones a tierra y métodos paliativos de ruido existentes en el circuito de generación como en el instrumento de medición.

Figura 95. **Ruido de red eléctrica**



Fuente: elaboración propia

CONCLUSIONES

1. La exposición de los aspectos generales en sistemas digitales es la base de construcción de abstracciones más elevadas que permiten acelerar los procesos en ingeniería.

2. El control numérico simplifica los parámetros de control del sistema de síntesis digital.

3. La creación de hardware específico permite la implementación de sistemas operativos en tiempo real.

4. La creación de un sistema en síntesis de alto nivel disminuye el tiempo de diseño e implementación.

5. El códec de audio funciona como un filtro paso bajo el cual, a determinada frecuencia, presenta una mayor atenuación.

6. Los dominios de reloj deben de ser cuidadosamente seleccionados puesto pueden crear malfuncionamientos en el sistema.

7. En las mediciones siempre es necesario considerar el ruido de la red eléctrica, ya que puede alterar la obtención de datos al punto de destrucción de información.

RECOMENDACIONES

1. Obtener los paquetes de software más actualizados, puesto que en cada iteración permiten un mayor número de funciones y dan estabilidad en los procesos.

2. Actualizar las tarjetas de desarrollo y estar pendiente de la actualización de los recursos en hardware como los sistemas en chip que puedan ser fabricados en el futuro.

3. La interconexión entre dispositivos de medición (osciloscopio) y sistemas de captura (computador) al momento de realizar las mediciones puede causar ruido e interferencia.

4. Continuar el presente trabajo para el desarrollo de dispositivos generadores de onda de bajo costo, los cuales puedan ser utilizados en laboratorios.

BIBLIOGRAFÍA

1. Analog Devices Inc. *A Technical Tutorial on Digital Signal Synthesis.* [en línea]<ftp://ftp.analog.com/pub/cftl/DDS%20Tutorial/DDS%20Outline.pdf>. [Consulta: 9 de enero de 2017.]

2. Analog Devices Inc. *AD9834 Datasheet*. [en línea] <http://www.analog.com/media/en/technical-documentation/data-sheets/AD9834.pdf>. [Consulta: 10 de enero de 2017.]

3. Analog Devices Inc. *Application Note AN237*. [en línea] <http://www.analog.com/media/cn/technical-documentation/application-notes/AN-237_cn.pdf>. [Consulta: 10 de enero de 2017.]

4. Analog Devices. *Direct Digital Synthesis (DDS) Controls Waveforms in Test, Measurement, and Communications*. [en línea] <http://www.analog.com/en/analog-dialogue/articles/dds-controls-waveforms-in-test.html>. [Consulta: 10 de enero de 2017.]

5. Analog Devices. *Direct Digital Synthesis Enables Digital PLLs.* [en línea] <http://defenseelectronicsmag.com/site-files/defenseelectronicsmag.com/files/archive/rfdesign.com/mag/707RFDF2.pdf>. [Consulta: 10 de enero de 2017.]

6. Analog Devices Inc. *SSM2603 Low Power Audio Codec Datasheet*. [en línea] <http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2603.pdf>. [Consulta: 3 de febrero de 2017.]

7. Analog Devices Inc. *Technical Tutorial on Digital Signal Synthesis*. [en línea]<ftp://ftp.analog.com/pub/cftl/DDS%20Tutorial/DDS%20Outline.pdf>. [Consulta: 17 de enero de 2017.]

8. ARM. *ARM Architectures, Processors, and Devices*. [en línea] <http://infocenter.arm.com/help/topic/com.arm.doc.dht0001a/DHT0001A_architecture_processors_and_devices.pdf>. [Consulta: 18 de septiembre de 2016.]

9. ARM. *Cortex™-A9 Floating-Point Unit Technical Reference Manual*. [en línea] <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0408g/DDI0408G_cortex_a9_fpu_r3p0_trm.pdf>. [Consulta: 5 de octubre de 2016.]

10. CROCKETT, Louise, et al. *The Zynq Book*. 1a ed. Glasgow: Strathclyde Academic Media, 2014.11. p. 50.

11. IEEE. *IEEE Std 1149.1™-2001*. [en línea] <http://fiona.dmcs.pl/~cmaj/JTAG/JTAG_IEEE-Std-1149.1-2001.pdf>. [Consulta: 17 de octubre de 2016.]

12. Intel. *Intel® Xeon Phi Processor™ Your Path to Deeper Insight*. [en línea]<http://www.intel.com/content/dam/www/public/us/en/docume

nts/product-briefs/high-performance-xeon-phi-coprocessor-brief.pdf>. [Consulta: 20 de diciembre de 2016.]

13. KARAM, J., LINA, et al. *Trends in multi-core dsp platforms*. Austin, Texas: IEEE Signal Processing Magazine, Special Issue on Signal Processing on Platforms with Multiple Cores, Nov. 2009. p. 28.

14. Motorola. *Application Note Phased-Locked Loop Design Fundamentals.* [en línea] <http://lens.unifi.it/ew/dwl.php?dwl=bm90ZXMvUExMX0Z1bmRhbWVudGFsLnBkZg==&mtyp=application/pdf>. [Consulta: 19 de enero de 2017.]

15. NXP Semiconductors. *I2C-bus Solutions 2014*. [en línea] <http://www.nxp.com/documents/leaflet/75017540.pdf>. [Consulta: 8 de febrero de 2017.]

16. NXP Semiconductors*. I^2C-bus specification and user manual.* [en línea] <http://www.nxp.com/documents/user_manual/UM10204.pdf>*.* [Consulta: 9 de febrero de 2017]

17. Phillips Semiconductors. *I^2S bus specification*. [en línea] <https://web.archive.org/web/20060702004954/http://www.semiconductors.philips.com/acrobat_download/various/I2SBUS.pdf>. [Consulta: 4 de febrero de 2017]

18. TAUB, H., & Schilling, D. *Principles of Communication Systems*. 1a edición. Nueva York [u.a.]: McGraw Hill, 1986.12 p.

19. Xilinx Inc. *7 Series DSP48E1 Slice.* [en línea] <https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf>. [Consulta: 15 de octubre de 2016.]

20. Xilinx Inc. *7 Series FPGAs Configurable Logic Block.* [en línea] <https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf>. [Consulta: 15 de octubre de 2016.]

21. Xilinx Inc. *7 Series FPGAs Memory Resources.* [en línea] <https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf>. [Consulta: 12 de octubre de 2016.]

22. Xilinx Inc. *Designing IP Subsystems Using IP Integrator*. [en línea] <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug994-vivado-ip-subsystems.pdf>. [Consulta: 2 de febrero de 2017.]

23. Xilinx Inc. *Using NEON for Parallel Data Processing*. [en línea] <https://www.xilinx.com/Attachment/53775/Neon_Introduction_for_Avnet_training.pdf>. [Consulta: 2 de enero de 2017.]

24. Xilinx Inc. *Vivado Design: Using Constraints.* [en línea] <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug903-vivado-using-constraints.pdf>. [Consulta: 1 de febrero de 2017.]

25. Xilinx Inc. *Zynq-7000 All Programmable SoC Overview Product Specification.* [en línea]

<https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf>. [Consulta: 7 de noviembre de 2016.]

26. Xilinx Inc. *Zynq-7000 All Programmable SoC Software Developers Guide*. [en línea] <https://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf>. [Consulta: 21 de diciembre de 2016.]

27. Xilinx Inc. *Zynq-7000 All Programmable SoC Technical Reference Manual.* [en línea] <https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf>. [Consulta: 6 de noviembre de 2016.]