UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería Electrónica

Diseño, verificación e implementación de un AWG multicanal coherente para FPGA

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Guirado Carulla, Daniel

Tutor/a: Colom Palero, Ricardo José

Cotutor/a externo: FE, JORGE DEOLINDO

CURSO ACADÉMICO: 2021/2022

Resumen

De la complejidad de los diseños digitales en el ámbito de la guerra electrónica surge la necesidad de realizar test a dichos sistemas usando señales complejas. Para generar estas señales es necesario de equipos especializados con altas prestaciones, que requieren un sistema completo para testear (antenas, hardware, sistema operativo, etc). Para poder realizar verificaciones en estados tempranos del diseño en FPGA de equipos de guerra electrónica, se va a desarrollar un AWG configurable tanto en prestaciones de señal como en número de canales y paralelizaciones de salida. De este modo se reducirá el tiempo de diseño pudiendo observar el comportamiento del diseño implementado en FPGA ante señales complejas.

Resum

De la complexitat dels dissenys digitals en l'àmbit de la guerra electrònica sorgeix la necessitat de fer test a aquests sistemes usant senyals complexos. Per generar aquests senyals és necessari equips especialitzats amb altes prestacions, que requereixen un sistema complet per testejar (antenes, hardware, sistema operatiu, etc). Per poder realitzar verificacions en estats primerencs del disseny a FPGA d'equips de guerra electrònica, es desenvoluparà un AWG configurable tant en prestacions de senyal com en nombre de canals i paral·lelitzacions de sortida. D'aquesta manera es reduirà el temps de disseny podent observar el comportament del disseny implementat a FPGA davant de senyals complexos.

Abstract

From the complexity of digital designs in the field of electronic warfare arises the need to test these systems using complex signals. To generate these signals, it is necessary to have specialized equipment with high performance, that requires a complete system to test (antennas, hardware, operating system, etc). In order to carry out verifications in early stages of the design in FPGA of electronic warfare equipment, a configurable AWG will be developed both in signal performance and in the number of channels and output parallelizations. In this way, the design time will be reduced, being able to observe the behavior of the design implemented in FPGA in the face of complex signals.

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

ADC Analogic to Digital Converter.

AMBA Advanced Microcontroller Bus Architecture.

AWG Arbitrary Waveform Generator.

AXI Advanced eXtensible Interface.

CLB Configurable Logic Block.

COMINT COMunication INTeligence.

DAC Digital to Analog Converter.

dBc decibels under carrier.

DDS Digital Direct Sintesis.

ELINT ELectronic INTeligence.

ETW Equivalent Tunning Word.

FFT Fast Fourier Transform.

FPGA Field Progammable Gate Array.

FSM Finite State Machine.

GCD Greatest Common Divisor.

GRR Gran Repetition Rate.

LFSR Linear Feedback Shift Register.

PRF Pulse Repetition Interval.

PRI Pulse Repeatition Interval.

PW Pulse Width.

RESM Radar Electronic Support Measures.

SFDR Spurious Free Dinamyc Range.

SIGINT SIGnal INTeligence.

SNR Signal to Noise Ratio.

VHDL Very high speed integrated circuit Hardware Description Language.

Parte I

Memoria

Capítulo 1

Introducción

Durante el desarrollo de un proyecto de hardware sobre FPGA las pruebas del sistema son una etapa crítica del diseño. Con la creciente complejidad de los diseños, la necesidad de poder generar los estímulos necesarios que alimenten los sistemas con la finalidad de comprobar que las funcionalidades requeridas y los requisitos demandados por los clientes se cumplen, es necesario disponer de las herramientas necesarias que generen las señales de los distintos escenarios de prueba a realizar.
En colaboración con la empresa DAS Photonics, surgió la idea de diseñar un generador de formas de onda arbitrarias, herramienta que será utilizada para poder realizar las verificaciones de diseños hardware implementado sobre FPGA con la capacidad de poder reproducir una señales de todo tipo, pudiendo así realizar pruebas de los proyectos inyectando los estímulos de señales sin usar el hardware final completo y en cualquier parte del sistema digital, consiguiendo independizar así las validaciones y verificaciones del sistema digital (FPGA más software embebido) del resto del sistema. De esta manera se elimina de los resultados de estas pruebas los posibles errores introducidos por el resto del sistema, se reduce el tiempo de validación y se obtiene una mayor cobertura a la hora de realizar las comprobaciones, ya que se puede inyectar cualquier señal conocida en el punto que se desee del sistema.

1.1. Objetivos

En este proyecto confluyen dos tipos de objetivos, los primeros son debidos a la necesidad de implementar el Ipcore del AWG como complemento con el ya disponible en los laboratorios de DAS Photonics, un Agilent N8241A, no para ser substituido, sino para poder trabajar en paralelo cuando no sea posible disponer de dicho elemento. Así será necesario estudiar el funcionamiento del AWG para que los resultados a la hora de utilizar tanto el elemento físico como el elemento diseñado como Ipcore sean los mismos.
El segundo grupo de objetivos residen en una serie de peticiones del departamento de FPGA para poder inyectar estímulos en los diseños implementados y en cualquier punto de la cadena de procesado.
Los objetivos dependientes de la estructura del AWG disponible serían:

- **Selección de tres modos de trabajo**: el modelo de AWG de Agilent dispone de varias opciones de trabajo, repetición de una de señal, o configurar la señal de salida como la secuencia

de distintos segmentos para conformar formas de onda complejas, generar pulsos, modulaciones, entre otros. El sistema a diseñar debe de poder realizar al menos la generación de señales de forma continuada y pulsadas.

- **Posibilidad de crear desfases lineales entre canales**: debido a la orientación del sector de DAS Photonics, la simulación de la misma señal desfasada en distintos canales es una opción básica para las pruebas del sistema.

En cuanto a las necesidades del departamento de FPGA para mejorar las prestaciones del ya existente AWG serían:

- **Capacidad de poder usar hasta 8 módulos del AWG en paralelo sincronizados**: actualmente el modelo N8241A disponible en el laboratorio cuenta con 2 salidas diferenciales para la generación de señales, por lo que se propone la posibilidad de implementar el sistema con más capacidad de salidas para los distintos proyectos del departamento de FPGA.
- **Generación de las formas de onda para ser guardadas en la memoria del IPcore**: la generación de los datos para ser escritos en la memoria del IPcore, los cuales serán utilizados por el mismo para generar las señales.

1.2. Objetivos específicos

Dado que el presente proyecto debe adaptarse al flujo de trabajo de los proyectos de DAS Photonics, las especificaciones técnicas vienen determinadas por las restricciones del departamento de FPGA. Las especificaciones detalladas son explicadas a continuación.

- **Un canal de entrada**: la idea detrás de este proyecto es poder instanciar dentro de la FPGA varias veces la Ipcore, para llegar a obtener la cantidad de canales de salida necesarios para cada proyecto donde se use el AWG. Por ello, el paso más sencillo sería crear la Ipcore de una simple salida, y ser instanciado tantas veces como se requiera.
- **La frecuencia de muestro máxima de 2 GHz**.
- **Frecuencia máxima de entrada de 1 GHz**: una consecuencia natural de la restricción de la frecuencia máxima de muestreo.
- **16 paralelizaciones máximas**: la frecuencia máxima de reloj interno de la FPGA es de 1.25 GHz, pero internamente los módulos trabajan con una frecuencia de reloj submúltiplo menor. Por ello, para poder alcanzar las frecuencias de muestreo requeridas, se deben de paralelizar las muestras de la generación de señales del AWG. Por ejemplo, si se trabaja con una configuración con un ADC a 1.25 GHz. pero el sistema trabaja a un cuarto de la frecuencia de reloj (312.5 MHz) el AWG para ”simular las entradas del ADC debería de necesitar paralelizar 4 muestras por flanco de reloj ($4 * 312{,}5MHz = 1{,}25GHz$) para alcanzar la tasa del ADC.
- **Posibilidad de trabajar con distintas configuraciones del reloj interno**: como se ha comentado en el apartado anterior, existen diferentes modos de configuración del reloj que emplea los módulos de la FPGA, por lo que se debe verificar que el Ipcore funciona perfectamente con las configuraciones usadas por el departamento para los proyectos actuales.

- **La paralelización será modificable mediante generics en VHDL**: la cantidad de muestras paralelizadas a la salida de cada canal será una variable del tipo generic, modificable antes de la compilación del sistema para poder adecuar el formato de muestras en la salida del AWG con el formato de paralelización de lo datos de entrada de los módulos que lo utilicen.
- **Sincronización de hasta 8 instancias del IPcore**: una de las restricciones del departamento es la necesidad de poder usar como máximo hasta 8 canales de salida, que además deben de estar sincronizados en todo momento, dado que la diferencia de fases entre ellos es una información vital para el procesado de las señales de radar.
- **Capacidad de generar retardos en las muestras por cada canal**: para poder simular señales simples que son recibidas por una matriz de receptores, es necesario aplicar un delay a los distintos canales de forma independiente.
- **Tamaño de la palabra de salida de 16 bits con posibilidad de truncamiento**: el tamaño de los datos de salida será de 16 bits como máximo, pudiendo elegir mediante generic la cantidad de bits a la salida del módulo.
- **Uso de recursos**: dado que el generador va a ser un complemento para estimular los módulos o sistemas de la empresa DAS Photonics, este deberá ser lo más ligero posible tanto en el uso de CLB como de bloques de memoria.

1.3. Metodología

A la hora de abordar el proyecto se realizó un estudió sobre que tareas y plazos serían aconsejables para su desarrollo. En cuanto a como realizarlo, se dispuso que el diseño de dividiera en dos grandes bloques diferenciados:

- Software: la parte del proyecto necesaria para realizar las tareas de generación de los datos de las señales a guardar dentro del AWG. Esta tarea se realizará usando el lenguaje Python, y tendrá como objetivo ser la interfaz entre el usuario y el sistema.
- Hardware: en este bloque se procederá a realizar el diseño hardware del sistema. Para ello se utilizará como lenguaje descriptor VHDL, y se seguirá un esquema bottom-up: conociendo el sistema general del AWG, se dividirá la carga de trabajo en el diseño de bloques funcionales separados, los cuales tienen cada uno una entidad específica dentro del sistema global. Por ejemplo el diseño del acumulador de fase o de las memorias ram donde se aloja la información de las señales. Este enfoque permitirá realizar pruebas de verificaciones unitarias de los distintos módulos y ir interconectándolos entre si para generar el sistema global.

La empresa DAS Photonics usa el sistema de control de versiones basado en GIT llamado GitLab. Usando el método de trabajo de gitflow ([1]) se trabajará mediante la creación de discusiones para cada problema o necesidad encontrado. Dado que el proyecto es una petición del departamento de FPGA's de DAS Photonics, se realizará el diseño usando las plataformas de Xilinx, usando una placa de evaluación ZCU106 de Xilinx [2]. Aunque la mayor parte del diseño hardware sobre FPGA será realizado por el alumno, para mejorar ciertas prestaciones del diseño se usarán primitivas que la propia Xilinx ofrece como herramienta al diseñador y algún módulo ya existente en el repositorio de DAS Photonics. Para verificar los módulos de forma independiente se usará el

simulador por defecto de Vivado, el Isim junto con unos scripts de python para extraer la información necesaria si el módulo lo necesita. Para la verificación del sistema completo se realizarán una serie de baterías de pruebas con diferentes propósitos apoyados en la utilidad de Gitlab de poder realizar comprobaciones de integración continua. Cuando se abre una discusión, una rama de desarrollo es creada para trabajar sobre el problema. Dentro de la rama se pueden ejecutar los test cada vez que se realiza un commit. La configuración de cada banco de pruebas se realiza en el archivo **.gitlab-ci.yml**, donde se especifican la batería de test que se van a realizar. La información de configuración es transmitida a un archivo python, el cual usa las librerías de test Vunit ([3]) para VHDL/Verilog. Este framework realiza la tarea de compilar el código de VHDL y generar las configuraciones que ha recibido el python para transmitirlas al modelo bajo prueba.

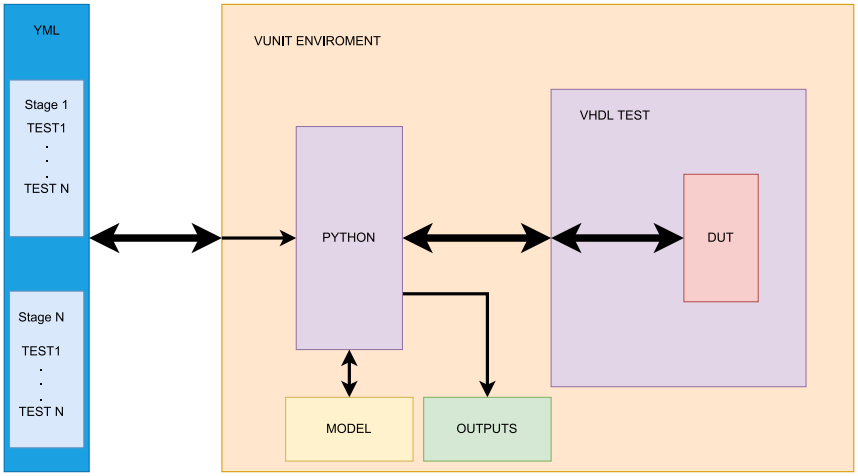


Figura 1: Esquema del test en con Vunit y Gitlab

A través de funciones de la librería en python de Vunit, se realizan las comprobaciones de las pruebas, ya sea a través de un modelo o a través de la comprobación de funcionalidades como caja negra.

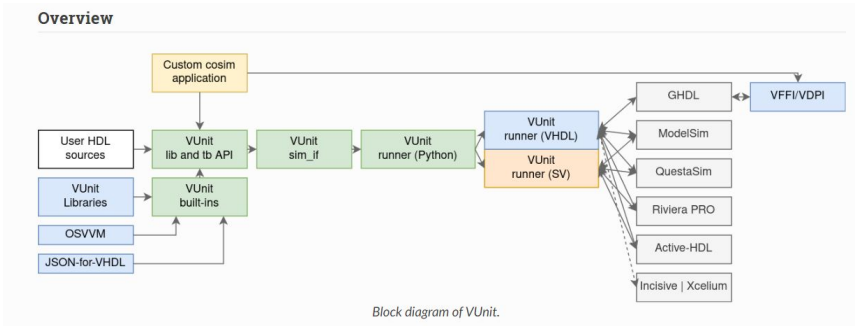


Figura 2: Diagrama de test del Vunit [3].

Cuando se ejecuta el proceso, Vunit crea La planificación de la carga de trabajo para la consecución de los objetivos del proyecto se ha realizado organizando cada tarea con las demás mediante sus precedencias, de tal forma que podemos desarrollar un diagrama de Gantt con dichas tareas organizadas temporalmente y jerárquicamente. La estimación de los tiempos para la realización de

cada tarea se ha decidido en base a la experiencia propia del alumno, junto a criterios del tutor. En la figura 3 se puede observar la estructura planificada del proyecto, con sus tareas ordenadas en un diagrama de Gantt.

Descripción de las tareas:

1. Desc. Tarea 1: Definición del proyecto
2. Desc. Tarea 2: Estudio del estado del arte.
3. Desc. Tarea 3: Diseño y verificación del acumulador de fase.
4. Desc. Tarea 4: Diseño y verificación del generador de direcciones.
5. Desc. Tarea 5: Diseño y verificación del array de BRAMs.
6. Desc. Tarea 6: Diseño y verificación de la FSM principal.
7. Desc. Tarea 7: Diseño y verificación del generador de dithering.
8. Desc. Tarea 8: Diseño y verificación de la FSM maestro/esclavo.
9. Desc. Tarea 9: Diseño y verificación de la FSM de escritura.
10. Desc. Tarea 10: Diseño y verificación del bloque de reset.
11. Desc. Tarea 11: Diseño y verificación del bloque del detector de periodos.
12. Desc. Tarea 12: Diseño y verificación del bloque de la máscara de datos.
13. Desc. Tarea 13: Diseño y verificación de la etapa de salida.
14. Desc. Tarea 14: Diseño y verificación de la interfaz AXI4 Lite.
15. Desc. Tarea 15: Integración de los módulos.
16. Desc. Tarea 16: Verificación del funcionamiento del sistema integrado.
17. Desc. Tarea 17: Validación del sistema.
18. Desc. Tarea 18: Documentación.
19. Desc. Tarea 19: Finalización del proyecto.

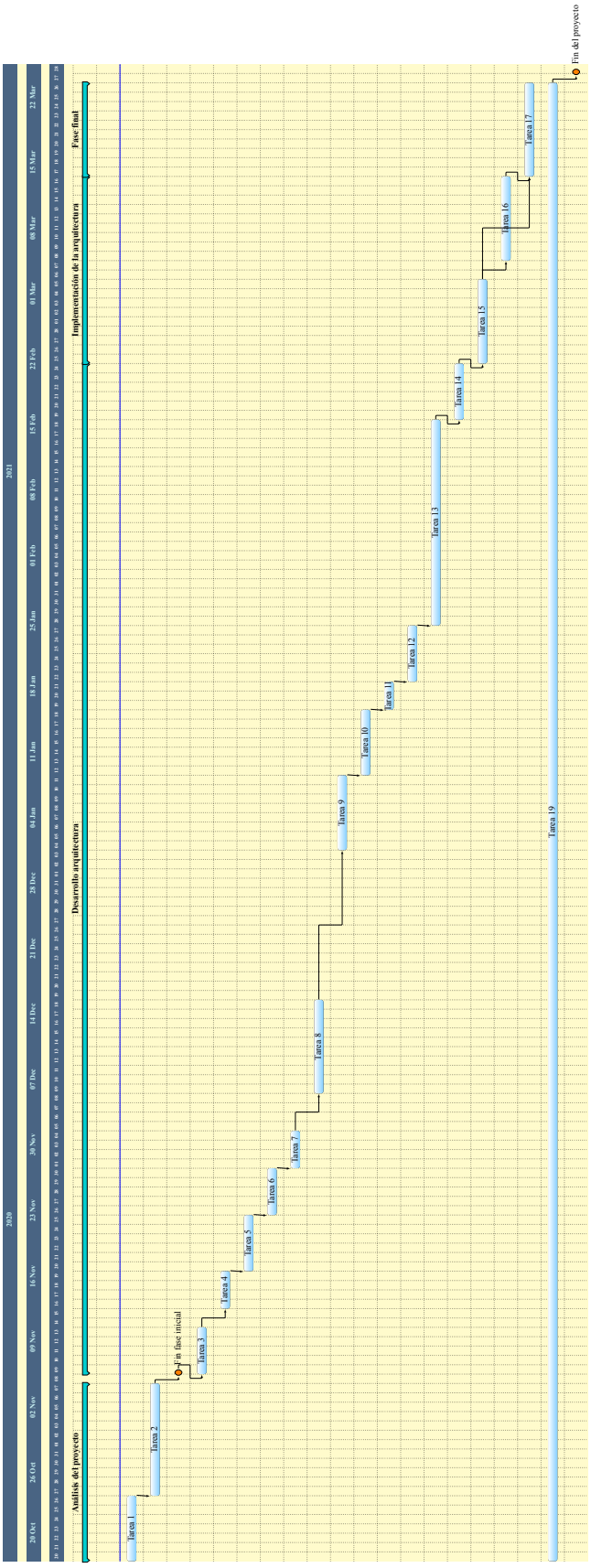


Figura 3: Diagrama de gantt del proyecto

Capítulo 2

Estado del arte

2.1. Sistemas de guerra electrónica

DAS Photonics opera en el sector de Defensa desarrollando sistemas de inteligencia de señales (SIGINT de nueva generación, proveyendo de productos para la guerra electrónica como sistemas ELINT y COMINT con adquisición de señales basada en sistemas fotónicos.

2.1.1. Sistema ELINT

En el ámbito de la guerra electrónica se conoce como inteligencia electrónica, o ELINT, a la recolección de inteligencia de señales que no sean de comunicación. Su propósito es aportar información de alertas de señales provenientes de todo tipo de radares, incluyendo los espía, control de tiro, guía de misiles etc. Los sistemas ELINT interceptan las señales y las analizan, clasificándolas según parámetros como su frecuencia, su agilidad de frecuencia, la frecuencia de repetición (PRF) y el ancho de pulso (PW). La figura 4 muestra la relación entre el PW y el PRI de una señal RF. El PW es el intervalo de tren de pulsos de la señal RF, mientras que el PRI es el intervalo en tiempo entre dos pulsos de la señal. El PRI es la inversa del PRF.

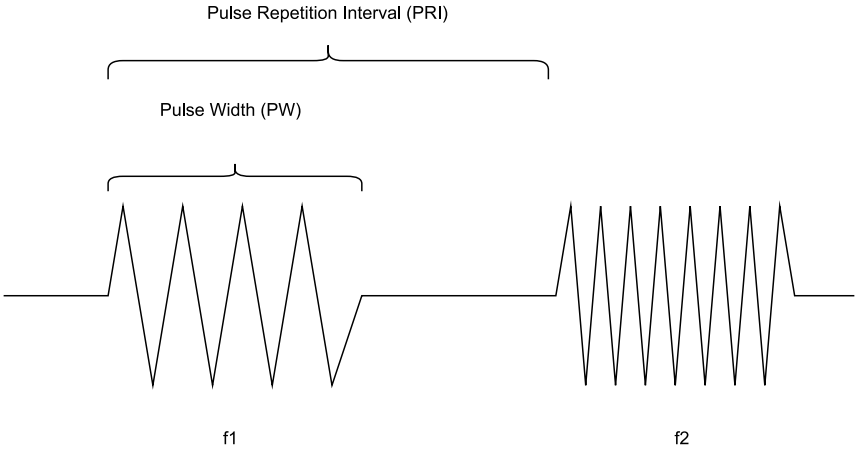


Figura 4: Características básicas de una señal RF

Normalmente con estas características es posible catalogar las distintas señales de entrada al sistema ELINT y compararlas con una biblioteca de emisores precargada en el sistema, pudiendo identificar sistemas amigos o enemigos. Los sistemas ELINT obtienen la información necesaria para poder realizar tareas de apoyo de guerra electrónica, como inhibición de señales, creación de señuelos etc, proveyendo de la información de dirección, distancia y emisor a los demás sistemas electrónicos. En el entorno actual de los sistemas militares, un uso efectivo de sistemas ELINT es vital para poder realizar las tareas asignadas.

2.1.2. Sistemas COMINT

Los sistemas COMINT realizan una operación similar a los sistemas ELINT, pero si objetivo es la obtención de información de señales de comunicación con la finalidad de obtener una ventaja sobre los movimientos y comunicaciones de los objetivos enemigos y ayudar a la ocultación de la información de las comunicaciones amigas.

2.2. Generadores de formas de onda

A la hora de diseñar un sistema electrónico o mecánico, es necesario realizar distintas pruebas para verificar el comportamiento del diseño y comprobar que las prestaciones acordadas se han conseguido. A mayor complejidad del sistema, las pruebas se complican, siendo necesario el uso de señales como entradas cada vez más complicadas. Es por ello que el sector de generadores de señales ha crecido, ampliando las posibilidades de los instrumentos, los cuales pueden ofrecer una variedad de señales enorme. Estos dispositivos, los generadores de señales, se pueden dividir en cuatro grandes familias dependiendo del tipo se señales producidas:

- Generadores de señal.
- Generadores de funciones.
- Generadores de señales arbitrarios.
- Generadores de pulsos.

Los generadores de señal suelen ser los más simples, son capaces de crear señales sinusoidales o cuadradas, en las que se pueden variar la frecuencia y la amplitud, y si disponen de más de un canal, pueden modificar el desfase entre ambos.
Los generadores de funciones son una evolución de los anteriores, pudiendo reproducir señales periódicas como triangulares, cuadradas, rampas, dientes de sierra o ruido, pero todas ellas de un catálogo predefinido, pudiendo modificar variables como en los generadores de señal, y en algunos casos incluso poder hacer modulaciones de amplitud o frecuencia. Además pueden llegar a tener la opción de simular estándares de comunicación.
Los generadores de señal arbitrarios, o AWG por sus siglas en inglés, son generadores de funciones con el añadido de poder reproducir señales no solo de un catálogo predefinido, sino formas de onda definidas por el usuario.
Por último, los generadores de pulsos están especializados en generar señales pulsadas, con un alto nivel de modificación de los parámetros asociados a este tipo de señales, normalmente cuadradas, como son los tiempos de on y off, su amplitud, los tiempos de subida y bajada entre otros.

2.3. Generadores de formas de onda arbitrarios

Para poder realizar pruebas en los sistemas, normalmente se han usado distintos generadores de estímulos para introducir las señales deseadas, perfectas e ideales, desde generadores de ondas cuadradas, triangulares, senoidales, etc. Con el aumento de la complejidad de los sistemas digitales se ha necesitado no solo probar los modelos con señales perfectas, sino poder ponerlos a prueba introduciendo unos estímulos con ruido o condiciones no ideales, formas de onda, por lo general, no periódicas y complejas. Por ello se empezó a diseñar generadores que pudieran emular estímulos no ideales ni periódicos, es decir, que pudieran ser programados para crear cualquier tipo señal que pudiera ser guardada. Los AWG surgieron por primera vez durante los años 80 como resultado de esta necesidad, un instrumento cuya salida es una señal programable, con distintos tipos de modos de funcionamiento y modulaciones. Con este tipo de generadores ahora se pueden probar sistemas o subsistemas con entradas más arbitrarias, pudiendo obtener resultados de verificación más acordes a la realidad del entorno de funcionamiento del producto sin tener que esperar al sistema completo para realizar test, reduciendo el tiempo de desarrollo y acelerando la detección de posibles fallos o comportamientos no deseados en modelos digitales complejos.
Como todo instrumento, los AWG tienen ciertas características básicas que determinan su funcionalidad, dejando al técnico encargado de realizar los test la posibilidad de elegir el generador que mejor se acople a las características necesarias para las pruebas. Algunas de las características son intrínsecas de la arquitectura del AWG, otras a su rendimiento, pero las especificaciones más usuales suelen ser:

- Frecuencia de muestreo: se puede pensar en los AWG como el instrumento que realiza la función inversa a un osciloscopio digital, es decir, en vez de almacenar una forma de onda en un buffer proveniente de muestrear una señal a través de un convertidor analógico digital, el flujo se invierte: a partir de una memoria donde se guarda una forma de onda se envía a un convertidor digital analógico para generar la señal. Una de las implicaciones es que está sujeto a la limitación del teorema de muestreo de Nyquist, donde la frecuencia máxima de la señal que se puede generar debe ser menor a la mitad de la frecuencia de muestreo.

- Tamaño de la memoria: la forma de onda completa se guarda en una memoria de tamaño finito. El tamaño de la memoria influye directamente en la frecuencia máxima y mínima de salida de la señal.

- Resolución vertical: esta especificación es el número de bits del DAC, por ejemplo si la resolución vertical es de N bits, el DAC podrá generar hasta 2^N niveles de de salida. Cuanto más resolución vertical, más detallada será la señal en el eje vertical.

- Ancho de banda analógica: corresponde al ancho de banda de salida del AWG y se relaciona con la respuesta del generador.

- Número de canales: existen AWG con diferentes cantidades de canales, uno, dos, cuatro, etc. Cada canal puede ser independiente entre los otros, o sincronizarse entre si, elegir el desfase entre canales o elegir distintas frecuencias de muestreo, dependiendo de la complejidad del modelo de AWG.

- Características de la señal de salida: existen modelos con la capacidad de controlar distintas características sobre la salida, tal como nivel de DC, amplitud, ruido, etc. La relación entre

estás características suelen estar interconectadas, por ejemplo, subir la amplitud de la señal o su offset suele conllevar una pérdida del ancho de banda o de resolución vertical por ejemplo.

Se pueden catalogar los AWG según diversas características expuestas anteriormente, por ejemplo usando la resolución vertical, o la frecuencia de muestreo, pero comúnmente se suelen diferenciar según la arquitectura usada. Keysight Technologies propone en [4] una clasificación basada en cinco arquitecturas generales:

- True arb AWG.
- DDS AWG.
- Interpolating DAC AWG.
- Pseudo-interleaving AWG.
- True Form AWG.

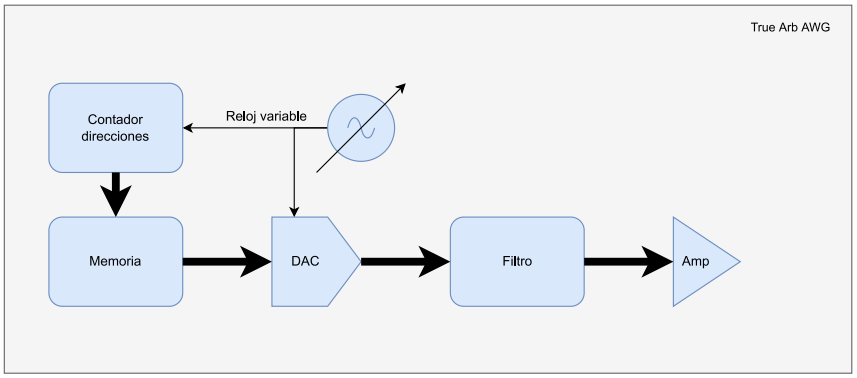


Figura 5: Arquitectura del True Arb AWG

El generador True arb se presenta en la figura 5 donde se pueden apreciar los elementos básicos, una fuente de reloj variable, un contador fijo de direcciones, una memoria donde se almacenan las muestras de la señal, un conversor digital-analógico, un filtro y un amplificador. El funcionamiento sería el siguiente: el contador y el DAC funcionan a la misma tasa de reloj, cuya frecuencia es posible variar dentro de unos rangos permitidos por el AWG, el contador realiza la operación de incrementar en uno la salida, generando una señal en forma de rampa, hasta que llega al final de cuenta, normalmente la profundidad de la memoria. La salida del contador es directamente la dirección de acceso a la memoria, generando así que se lea toda la memoria de inicio a fin de forma cíclica, con un periodo entre muestras igual al periodo del reloj. El DAC realiza la operación de conversión entre la forma de onda almacenada en la memoria digitalizada al domino analógico, y posteriormente se usan filtros para regenerar la señal. En la arquitectura basada en True Arb se leen todas las muestras almacenadas en la memoria y variando la frecuencia del reloj se consigue variar la frecuencia de la señal de salida. Esto tiene sus ventajas y sus inconvenientes, las ventajas son que la señal para el rango de frecuencias permitido se reproduce lo más fiel posible a lo guardado en la memoria. A cambio la lógica para cambiar la frecuencia del reloj suele ser compleja y realizar saltos de frecuencia para la señal de salida no suele ser instantáneo, requiere cierta cantidad de

ciclos de reloj, además de un posible cambio en las muestras almacenadas en la memoria. Este tipo de generadores arbitrarios suelen ser simples si la fuente del reloj es externa, por lo que son muy comunes en el mercado.

La arquitectura del generador basado en DDS se muestra en la figura 6 donde se puede ver que comparte cierto parecido con el True Arb en la parte final, incluyendo la memoria donde se almacena la señal, el DAC, los filtros y el amplificador.
Las dos principales diferencias están en la generación de las direcciones de memoria y el uso de un reloj de frecuencia fija. Para las direcciones de memoria se usa un acumulador de fase, el cual es un contador cuyo paso puede ser variable. La salida del generador depende de un ratio entre la frecuencia de muestreo, el valor del paso del acumulador de fase y el número de muestras guardadas en la memoria. Esto simplifica mucho la lógica del AWG, pero conlleva inconvenientes:

- Para una memoria dada y una frecuencia de muestreo fija, la salida depende directamente del paso aplicado al acumulador de fase, por lo es posible que no todas las muestras guardadas en la memoria sean utilizadas para construir la forma de onda de salida. Esto implica que puede no representarse la señal de forma fiel para todos los valores de paso. Una solución a este problema sería aumentar el tamaño de las muestras almacenadas, aumentando la memoria, pero esta elección tiene un límite de diseño.

- Adicionalmente puede darse el caso de *jitter* en la salida, producido por variaciones en las direcciones de memoria a leer en cada iteración de un periodo de la señal.

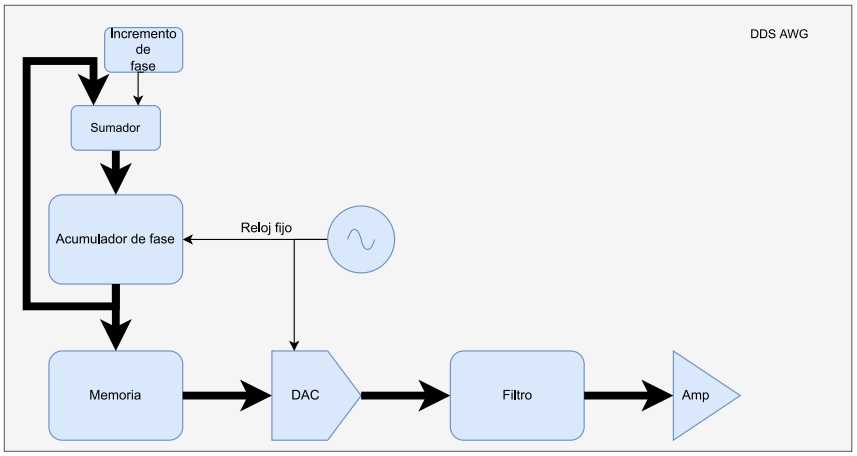


Figura 6: Arquitectura del DDS AWG

Los AWG interpoladores son parecidos a la arquitectura True Arb (figura 7), pero añadiendo dos elementos, el primero un divisor del reloj variable para el contador de direcciones, y el segundo bloque un interpolador entre la memoria y el DAC. Al usar una memoria con frecuencia más baja que la del DAC implica que la posición de las imágenes en el espectro que sean múltiples de la frecuencia del DAC están más alejadas que la componente espectral de la señal debido a la memoria, haciendo más sencillo filtrar las componentes debidas al DAC. Por ello la principal ventaja de trabajar con una arquitectura con interpolador frente a la True Arb reside en la mejora de la calidad de la señal.

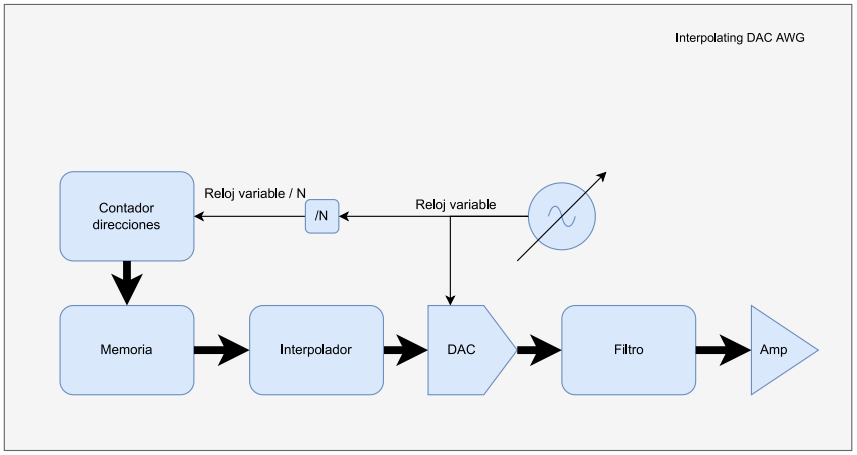


Figura 7: Arquitectura del AWG interpolador

Los generadores Pseudo-interleaving, esquema en la figura 8, son el resultado de combinar dos canales de procesado a la vez por separado, logrando así doblar la frecuencia de muestreo. Comúnmente se almacenan las muestras pares e impares en dos memorias distintas. Un reloj variable alimenta al contador de direcciones, que actúa sobre ambas memorias a la vez, cuya salida van a dos DAC diferentes, pero uno de ellos tiene un desfase de medio ciclo del reloj de muestreo variable. Cada salida de los DAC produce el mismo espectro, salvo para las imágenes alrededor de la frecuencia de los DAC, imágenes que tienen la misma forma pero fases invertidas. Al sumarse ambas señales procedentes de los convertidores, las imágenes con fase invertida cerca de la frecuencia de los DAC se cancelan, dejando un espectro similar a como si tuviéramos un solo camino de procesado con un solo convertidor digital analógico.

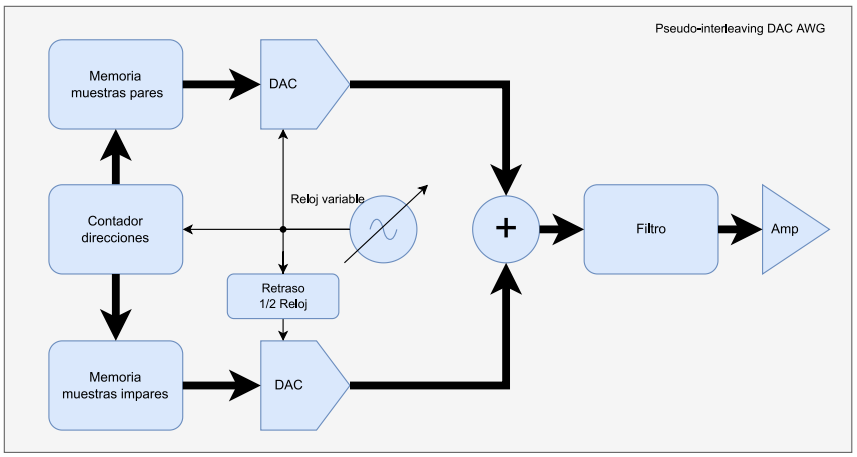


Figura 8: Arquitectura del AWG pseudo-interleaving

La arquitectura True form es la combinación de la arquitectura basada en DDS y la interpoladora. Un esquema de este tipo de generador se puede ver en la figura 9, donde se observa el acumulador de fase del DDS y el bloque interpolador entre la memoria y el DAC.

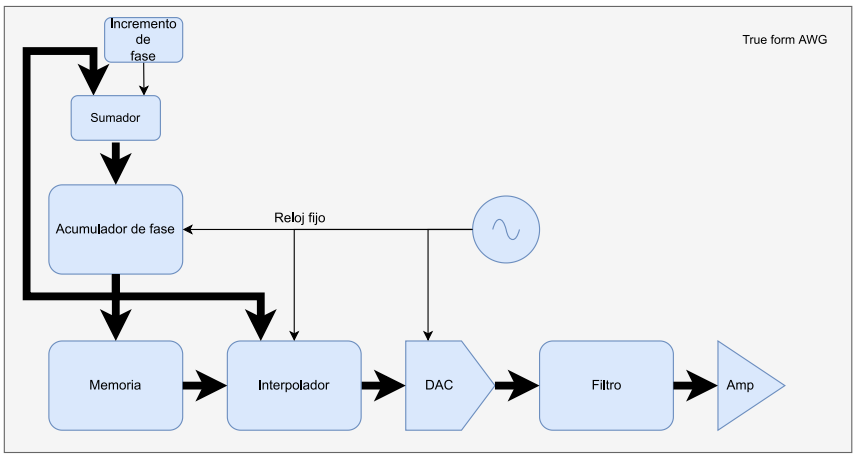


Figura 9: Arquitectura del AWG True form

La principal desventaja de la arquitectura DDS aquí se soluciona con el bloque interpolador, ya que aunque el acumulador de fase no generará todos los índices de la memoria para pasos superiores a uno, la interpolación generará las muestras intermedias entre dos muestras de la memoria. De esta forma tenemos las ventajas de la arquitectura DDS y las ventajas de la arquitectura interpoladora.

2.4. Fundamentos del DDS

La síntesis digital directa (DDS por sus siglas en inglés) es una técnica digital para generar señales capaces de controlar sus características como la frecuencia o la fase de salida usando un reloj de frecuencia fija. Actualmente existen una multitud de distintos diseños de DDS en el mercado, con arquitecturas especificas dependiendo de ciertas especificaciones que se requieran alcanzar, pero todos tienen en común una serie de bloques:

- Acumulador: en esencia es un contador de módulo N cuyo paso es variable mediante lo que se conoce como tunning word o paso. La finalidad de este bloque es generar las direcciones necesarias para el siguiente bloque, la memoria donde se almacenan las muestras de la señal a reproducir.

- Memoria o bloques Fase/Amplitud: se trata de un bloque que contiene la memoria con las muestras de la señal. Para adaptar la salida del acumulador a las direcciones disponibles de la memoria suele truncar los bits más altos del acumulador.

- DAC: la salida de la memoria alimenta a un DAC el cual convierte la información digital de la señal a analógica.

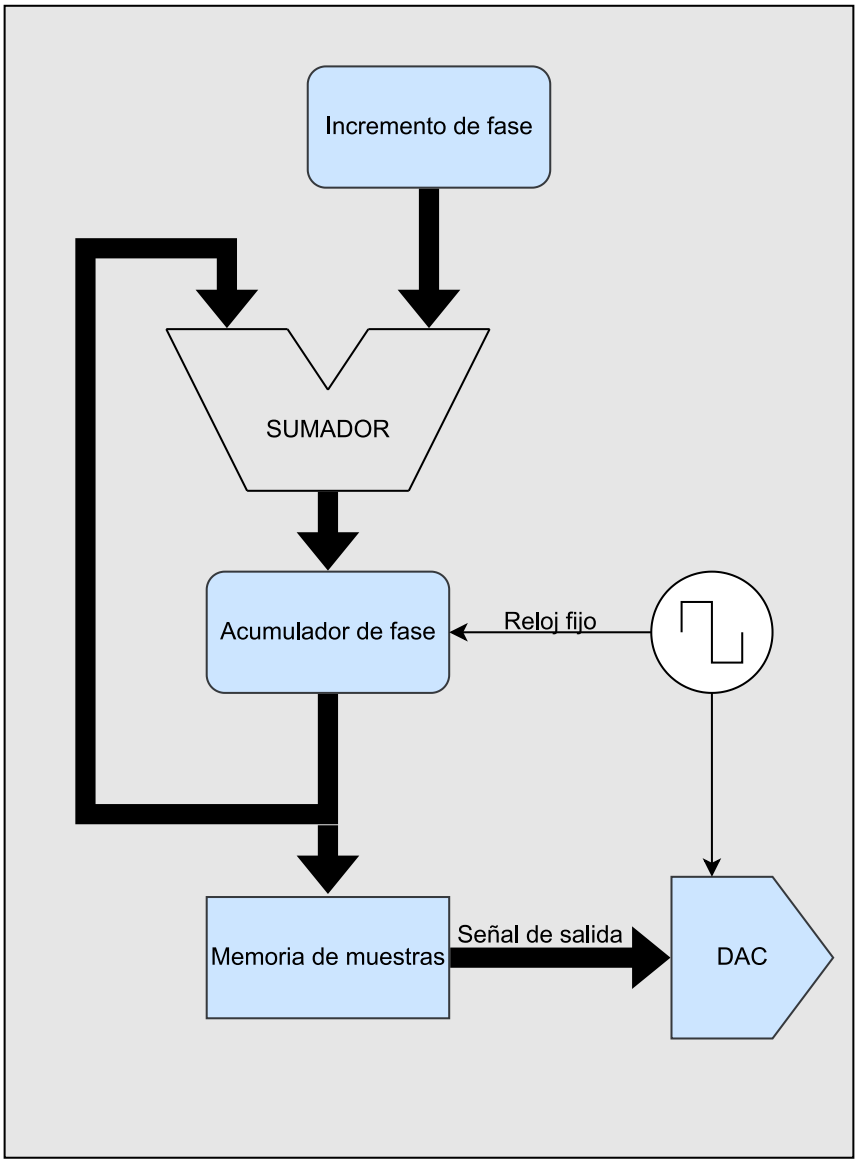


Figura 10: Arquitectura básica de un DDS

La ecuación general que relaciona la frecuencia de salida del DDS con las variables que pueden ser modificadas es la siguiente:

$$f_{output} = \frac{K \cdot f_{clk}}{2^N} \tag{1}$$

Siendo:

- f_{output} es la frecuencia de salida.
- K es el paso del acumulador.
- f_{clk} es la frecuencia de trabajo del acumulador.
- N es la profundidad de la memoria a leer.

Como se desprende de la ecuación 1, la frecuencia de salida es directamente proporcional al valor K, el incremento del acumulador de fase a la frecuencia del reloj f_{clk} y inversamente proporcional al tamaño de la memoria. Se puede visualizar el funcionamiento de un DDS como si se rotase un vector en un circulo y cada rotación tuviera como incremento un salto de fase que depende del paso del acumulador (figura 11).

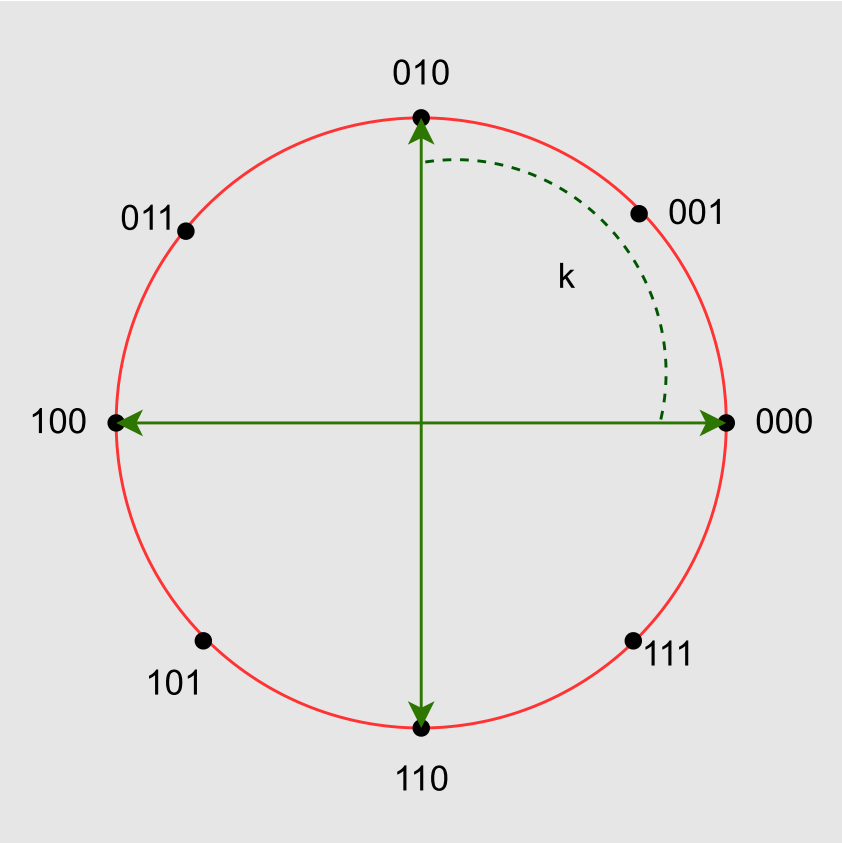


Figura 11: Círculo de fase de un DDS de módulo 8

La figura 11 muestra como sería la salida del acumulador de N=3 y k= 2, por lo que usando la ecuación 1 la frecuencia de salida en función de la frecuencia del reloj sería:

$$f_{output} = \frac{2 \cdot f_{clk}}{2^3}$$
$$f_{output} = \frac{f_{clk}}{4}$$

2.4.1. Espectro de salida del DDS

Un DDS se puede entender como la operación inversa de muestrear una señal analógica, donde la frecuencia de muestreo debe de ser al menos el doble de la frecuencia de la señal a muestrear, tal y como explica el teorema de Nyquist. Para el caso del DDS también se debe de cumplir dicho teorema, dado que a la hora de generar una señal de frecuencia dada, al menos debe de ser utilizado un reloj del sistema del doble. Es por ello que si en la ecuación 1 substituimos la frecuencia de salida

f_{output} por $f_{clk}/2$ obtenemos que el valor máximo del paso del acumulador debe de ser la mitad de la profundidad de la memoria donde se almacenan las muestras. La señal de salida entonces está limitada a $f_{clk}/2$ y generará una serie de imágenes que se pueden obtener según la frecuencia de reloj de muestreo y de la frecuencia fundamental de la señal como $f_{clock} \pm f_{out}$. Por ejemplo, si tenemos una frecuencia de muestreo de 312,5MHz y una frecuencia de salida de la señal de 39 MHz, entonces se generaran imágenes de la fundamental en :

Imágenes de la fundamental en la salida del DDS

Imagen	Frecuencia (MHz)
1^r	273,5
2º	351,5
3º	586
4º	664
5º	898,5

En la figura 12 se puede observar la distribución de las imágenes de la tabla anterior en el espectro.

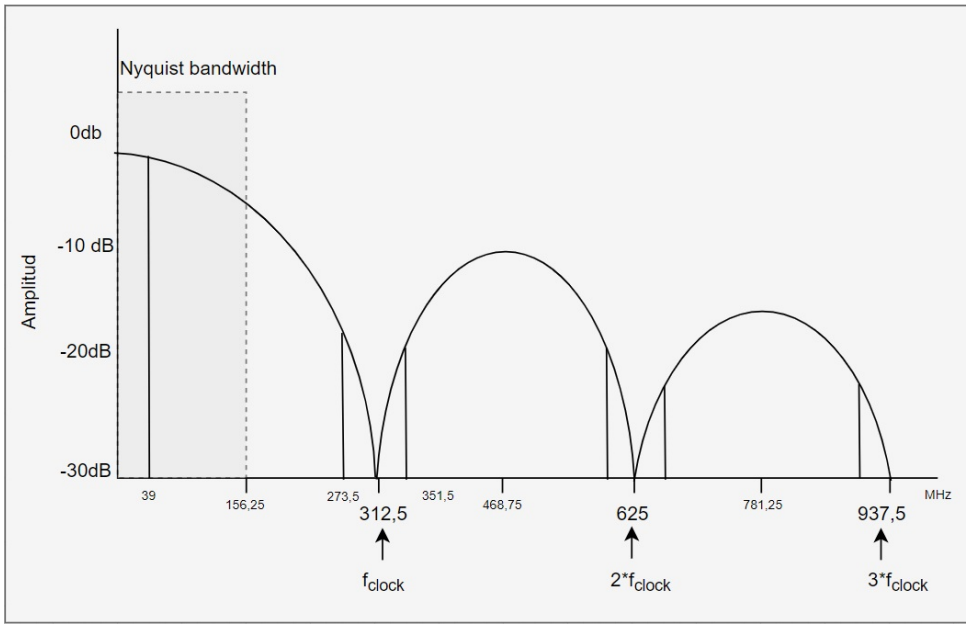


Figura 12: Espectro de salida del DDS para una señal de frecuencia 39 MHz

2.4.2. Control del DDS

Dos de los parámetros sobre los que se tiene control a la hora de usar un DDS son:

- La frecuencia de salida: este parámetro viene determinado como ya se ha explicado por el paso del acumulador. La velocidad a la que un DDS puede realizar saltos de frecuencia viene

impuesto por la velocidad de carga del nuevo valor del paso. Esta operación normalmente se suele realizar cargando todos los bits del paso en paralelo, para aumentar la velocidad de salto de frecuencia y evitar transiciones indeseadas si se carga en serie.

- El desfase de la señal: este parámetro viene determinado por la cantidad de muestras de la memoria y del tamaño del DAC, por ejemplo, para un DAC de 5 bits cada muestra está desplazada 11,25º, mientras que para uno de 12 bits este desfase entre muestras será de 0.09º aproximadamente. Esto quiere decir que con una memoria de 5 bits de profundidad la resolución de fase que podemos conseguir es de 11,25º.

2.4.3. Ruido del DDS

Un DDS ideal sería aquel cuya memoria tuviera infinitas muestras y una palabra de cada muestra también infinita, pero al tener un tamaño finito de muestras y una palabra de la memoria también finita nos encontramos que la salida es una aproximación de la ideal, y por lo tanto tiene cierto error de cuantificación. En general se puede dividir el ruido de cuantificación en dos, el ruido de fase y el ruido de amplitud. El ruido de fase es debido a la profundidad de la memoria (N), el valor del paso del acumulador (P) y el tamaño del acumulador (A). Normalmente en el diseño de los DDS se suelen elegir tamaños de acumulador grandes, del orden de 32 bits o más. La elección del tamaño del acumulador suele decidirse por dos razones principalmente. La primera razón reside en que a mayor tamaño del acumulador la frecuencia que se puede obtener es más pequeña, aumentando el rango de frecuencias que el DDS puede abordar. La segunda razón es más una cuestión de eficiencia del hardware elegido, ya que se suelen elegir valores que sean compatibles con los sumadores disponibles en el hardware, por ejemplo de 32 o 48 bits. El problema de usar acumuladores tan grandes es la cantidad enorme de LUTs necesarias para direccionar la salida del acumulador, por ejemplo si usamos 48 bits como tamaño, necesitaríamos 281.474.976.710.656 LUTs, algo impensable. Es por esta razón que para mantener un acumulador con un tamaño tan grande se suele truncar los bits de mayor peso para acomodarlos al tamaño de la memoria, que suele tener entre 5 a 14 bits de direcciones. Por ejemplo, si usamos el contador de 48 bits y una memoria de 10 bits, entonces los 38 bits más bajos serán ignorados. El problema de truncar la salida del acumulador de fase es que introduce un error que se traduce en el espectro de salida como ruido de fase. Se puede explicar este efecto usando el círculo de fase como el de la figura 11. Siguiendo el ejemplo de la figura, tenemos una memoria con una profundidad de 5 bits y supongamos que el acumulador es de 7 bits, la resolución del acumulador es de $360/2^7$ grados mientras que la resolución de la memoria está truncada y es de $360/2^5$ grados. Si elegimos un paso del acumulador 5 cada ciclo de reloj la salida del acumulador aumentará en 5 unidades. En la figura 13 se puede observar los cinco primeros pasos del ejemplo. El círculo exterior corresponde a los saltos del acumulador, cada uno de ellos con un desfase de 2,81º mientras que el círculo interior en rojo corresponden a los saltos truncados de las direcciones de la memoria. Durante el primer salto se llega a $5 \cdot 2,81$ º, es decir 14,05º, pero el salto del círculo interior se queda entre el primer y segundo salto, por lo que existe una diferencia de $1 \cdot 2,81$ º (2,81º). El segundo salto del acumulador vuelve a sumar otros 14,05º y el error ahora es de $2 \cdot 2,81$ º. En el tercer salto el error será de $3 \cdot 2,81$ º, mientras que en cuarto salto el error es de 0º. A partir del quinto vuelve a repetirse la serie descrita.

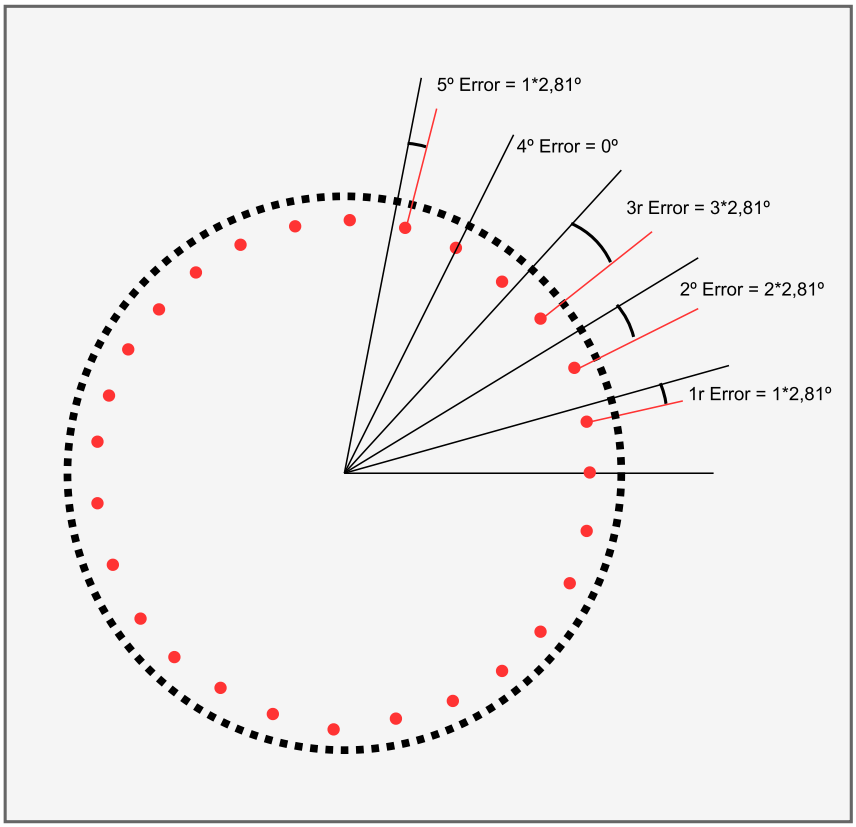


Figura 13: Error de fase por truncamiento

La salida del acumulador truncada da como resultado errores de amplitud a la salida de la memoria de forma periódica, provocando la aparición de espurios en el espectro de salida. Durante el proceso de generación de señal de un DDS el tamaño del acumulador y el tamaño truncado que genera las direcciones de memoria son fijos, por lo que es el paso del acumulador el que genera los errores de fase. Si la memoria es de N bits, el paso del acumulador tiene un rango acotado de valores que variará entre 1 y 2^{N-1} para cumplir el teorema de Nyquist, como se explica en 2.4.1. No todos los valores del paso del acumulador generan espurios, como se explica en [5], los valores del paso que generan los niveles máximos de espurios siguen la siguiente regla:

$$GCD(T, 2^{(A-P)}) = 2^{(A-P-1)} \tag{2}$$

Donde T es el valor del paso del acumulador, A es el tamaño en bits del acumulador y P el tamaño truncado. De forma gráfica y siguiendo el ejemplo anterior con A=7 y P=5, el patrón de los pasos del acumulador que generan el máximo nivel de espurios sería el siguiente:

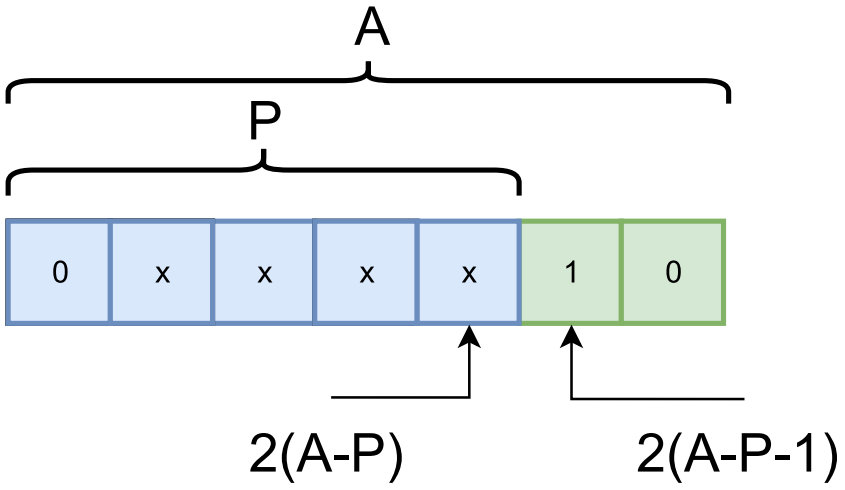


Figura 14: Patrón del paso para la igualdad 2

Es decir, todos aquellos valores para el que la posición A-P-1 valga uno seguido de todo ceros serán aquellos valores que generen el nivel máximo de espurios a la salida. Si la diferencia de A-P es mayor que 4, entonces el SFDR asociado a estos espurios cuando la fundamental es 0 dB se puede calcular de forma aproximada usando:

$$SFDR = -6,02 \cdot P + 3,92 \ (dB) \tag{3}$$

Mientras que los valores del paso que no producen espurios satisfacen:

$$GCD(T, 2^{(A-P)}) = 2^{(A-P)} \tag{4}$$

Donde el patrón quedaría como:

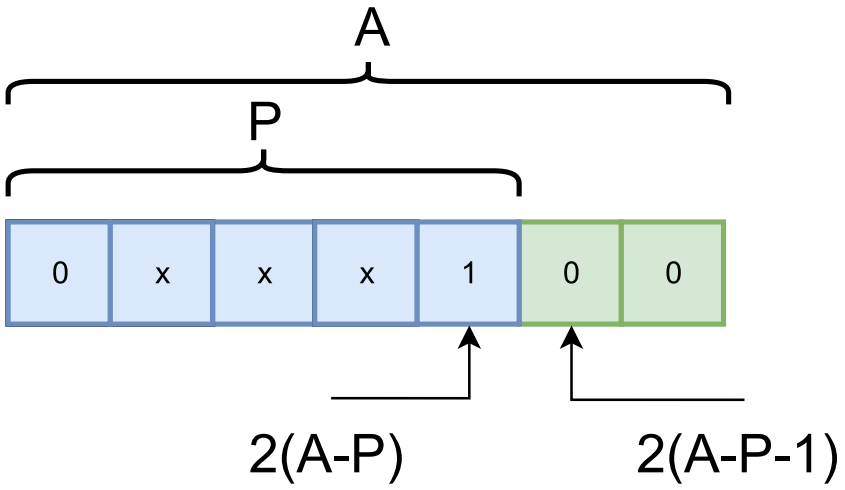


Figura 15: Patrón del paso para la igualdad 4

Lo que nos dice que todos aquellos valores de paso que tengan como mínimo un uno en la posición A-P y todo ceros en las posiciones de 0 a A-P-1 no generarán espurios en el espectro de la señal de salida. Todos los valores que no estén en las dos opciones mencionadas tendrán unos valores intermedios entre no espurios y -6,02P +3,92 dBc.
En el caso de que la señal de salida tenga espurios, podemos aproximar la distribución de estos en el espectro. Como se muestra en la figura 13 cada ciclo de reloj el acumulador incrementa su valor con el paso, el cual en el ejemplo era 6. Independientemente del valor del paso, después de una serie de ciclos el valor del acumulador volverá a valer su valor inicial y repetirá la secuencia otra vez. Este ciclo repetitivo se conoce como Gran Ciclo de Repetición, o GRR por sus siglas en inglés. Para determinar el número de ciclos necesarios para un valor de paso dado se usa la siguiente ecuación:

$$GRR = \frac{2^A}{gcd(T, 2^A)} \tag{5}$$

Usando los datos de ejemplo de antes para un A=7 y un T=6 (en base 10), el GRR es de 64 ciclos. Si la salida del acumulador se trunca, entonces no todos los bits serán usados para generar las direcciones de memoria, tan solo los P bits más altos. Esto quiere decir que los B bits más bajos del acumulador, siendo B=A-P, son truncados, por lo que se genera un error, como se ha explicado antes. Este truncamiento puede servirnos para visualizar la señal de salida como una señal usando los A bits junto a una señal de error debido a los B bits truncados. Como la fuente de error es debido a la palabra truncada de B bits, entonces para analizar la distribución de los espurios generados se realizará un análisis de la parte truncada del acumulador.
Al igual que se ha hecho con la salida del acumulador entera, podemos calcular el GRR de la parte truncada. Para ello se define el ETW, que no es más que el resto del paso original una vez truncado.

$$ETW = T \ mod \ 2^B \tag{6}$$

Por ejemplo, usando T=5 (base 10) y B=2, entonces el ETW será 1. Ahora podemos calcular usando 5 el GRR de la parte truncada, teniendo como A=B=2 y T = ETW=1, esto nos da un periodo de repetición de cuatro ciclos (algo que se podía deducir de la figura 13 cuando viendo que el ciclo de error duraba 4 ciclos). Que el ciclo sea de cuatro nos indica que la señal de error es periódica y con un intervalo de cuatro. La señal de error proviene de un contador que tiene una capacidad máxima de 2^B, esto significa que se asemeja a una señal diente de sierra que se repite cada GRR ciclos. En el ejemplo, la señal de diente de sierra tendría como valor máximo 4 y un periodo de 4 ciclos. La frecuencia fundamental de la señal de error se puede calcular como:

$$f_{phase-error} = \frac{f_{clock} \cdot ETW}{2^B} \tag{7}$$

Que usando los datos de ejemplo, $f_{phase-error} = f_{clock} \cdot 1$. Los armónicos de la señal de error están distribuidos por el espectro en múltiplos de su fundamental y existen GRR/2 frecuencias asociadas a la señal de error, entonces el intervalo donde se propagan los armónicos en el espectro de salida será hasta $2 \cdot f_{phase-error}$, por lo que en este ejemplo, tanto la fundamental como los armónicos se distribuirán en los múltiplos de la frecuencia de muestreo f_{clock} hasta $2 \cdot f_{clock}$. Debido al *aliasing*, los espurios que sean múltiplos impares de $f_{clock}/2$ se mapearan directamente en la región de Nyquist de f_{clock}/2, mientras que los espurios múltiplos pares de f_{clock} /2 se mapearan como imágenes en la región de f_{clock}/2.
La segunda fuente de ruido es por la cuantificación de la amplitud en la memoria. Cuando se reproduce una señal cuantificada con un tamaño W la señal de salida aparece escalonada, con

saltos finitos entre dos estados contiguos. Estos escalones entre dos muestras son componentes de alta frecuencia superpuestos a la señal ideal y aparecen como espurios en el espectro de salida. Si el ruido de fase visto anteriormente afectaba al SFDR, la cuantificación de la amplitud de la señal es un factor decisivo en el SNR de la salida y el fondo de ruido. Dado que el efecto de la cuantificación de la amplitud recae en el tamaño de bits usados para codificar la señal, el SNR de salida puede calcularse como:

$$SNR = -6,02 \cdot W - 1,76 - 20 \cdot \log(\frac{bits\ used}{full\ scale\ bits}) - 10 \cdot \log(\frac{f_s}{BW \cdot 2})\ (dB) \quad (8)$$

El tercer término de la ecuación es una corrección del fondo de escala, en caso de no usar todos los bits disponibles. Por ejemplo, si usamos los 12 bits disponibles para cuantificar la señal, entonces obtendremos un SNR aproximado de 74 dBc, pero si de los 12 bits, solo usamos 11 entonces veremos reducido el SNR a 73,24 SNR, lo que supone una pérdida de 0,75 dB. El último término de la ecuación es la ganancia del proceso, donde f_s es la frecuencia de muestreo y BW es el ancho de banda de la señal. Este factor se ha de tener en cuenta si se sobremuestrea la señal, es decir, si el ancho de banda de la señal es menor a la mitad de la frecuencia de muestreo. Cuando se realiza una FFT, esta operación actúa como un analizador de espectro con un ancho de banda de f_s/M, donde M es el tamaño de la FFT. Si substituimos este valor en la ecuación 8, entonces queda como:

$$SNR = -6,02 \cdot W - 1,76 - 20 \cdot \log(\frac{bits\ used}{full\ scale\ bits}) - 10 \cdot \log(\frac{M}{2})\ (dB) \quad (9)$$

siguiendo el ejemplo anterior con -73.24 dB de SNR, si la FFT es de 1024 puntos, entonces el SNR serán -100.33 dB debido a este efecto.

La ecuación 9 limita el desempeño del DDS en cuanto el límite teórico para el fondo del ruido.

2.4.4. Mejoras del SFDR

Como se ha comprobado en la sección anterior, el SFDR venia dado por los parámetros del tamaño del acumulador, el truncamiento de este para acoplarlo a la memoria y el valor del paso y dependiendo del valor del paso, se podían generar espurios que empeoran el SFDR. Existen distintas técnicas que posibilitan una mejora de dichos espurios como se explica en [6] y en [7] usando aproximaciones de las señales a generar, normalmente senoiduales. El primer método es usar la simetría de la señal a reproducir. En muchos casos la señal suele ser periódica y puede presentar una simetría como es el caso de las señales senoidales o cosenoidales. En este caso es posible guardar solo una porción de la señal y usar lógica extra para generar la señal completa, de esta forma el tamaño aparente de la memoria es cuatro veces más grande que si solo se almacenase la señal completa. Aumentar el tamaño de la memoria implica en una disminución de la parte truncada, y por lo tanto una mejora en el SFDR en el espectro de salida. Si se quiere generar formas senoidales, es posible utilizar aproximaciones para mejorar la calidad de la señales de salida. Un ejemplo es el uso de aproximaciones como la técnica Sunderland, la cual utiliza las relaciones trigonométricas para reducir el tamaño necesario de bits para direccionar, a cambio de necesitar dos memorias en vez de una más un sumador. Otra opción es usar las aproximaciones de las series de Taylor (primera aproximación, segunda, etc). Existen más métodos de aproximaciones para reducir

el SFDR de la salida (uso del CORDIC, método de diferencia de seno-fase, interpolación, etc) pero todos ellos siempre comprometen algún aspecto del diseño, como más memoria, sumadores y/o multiplicadores adicionales, aumento de la latencia por ejemplo. Es trabajo del diseñador elegir el método que más le convenga para la aplicación donde vaya a ser usado el DDS.

Si las señales a generar no son estrictamente senoidales o no tienen la simetría para poder realizar los métodos de aproximaciones vistos, entonces se puede mejorar el SFDR con otros métodos. De la ecuación 5 se desprende que el periodo de repetición máximo del acumulador ocurrirá cuando el paso tenga un valor impar, dando como resultado un GCD de 1 y por consiguiente un GRR de 2^A. De esta forma, aún cuando el valor del paso no sea como el descrito en 4, los espurios se repartirán por todo el espectro reduciendo su amplitud y mejorando el SFDR. Otra consecuencia de 5 reside en la periodicidad del ruido. Dado que este ruido es periódico se puede disminuir su aportación añadiendo un ruido de pequeña magnitud en la salida del acumulador, de esta forma se rompe la periodicidad del ruido, reduciendo la amplitud de los espurios y por lo tanto mejorando el SFDR. La contrapartida de usar dithering como método de mejorar el SFDR es un aumento en el nivel de ruido. Para conseguir el ruido de dithering correcto se genera una serie de números pseudoaleatorios. Estos valores se suman a la salida del acumulador sin truncar, insertando el bit de mayor peso del ruido de dithering al menos una posición por debajo del bit de menor peso de la palabra a truncar. Por ejemplo si el tamaño del acumulador es de A=7 y el tamaño truncado es de P=5, entonces la señal a sumar como mucho puede ser de dos bits.
En la figura 14 se sumaría el ruido en las posiciones verdes. Por lo general, para no añadir un ruido suficientemente grande, el tamaño de dithering suele estar de un bit a cuatro bits, más tamaño de dithering empeora la calidad de la señal de salida.

2.4.5. LFSR

El método más simple de generar un ruido de dithering es mediante un tipo de algoritmo llamado Linear Feedback Shift Registers (LFSR) del cual existen dos tipos de arquitectura, los LFSR de Galois y los LFSR de Fibonacci.

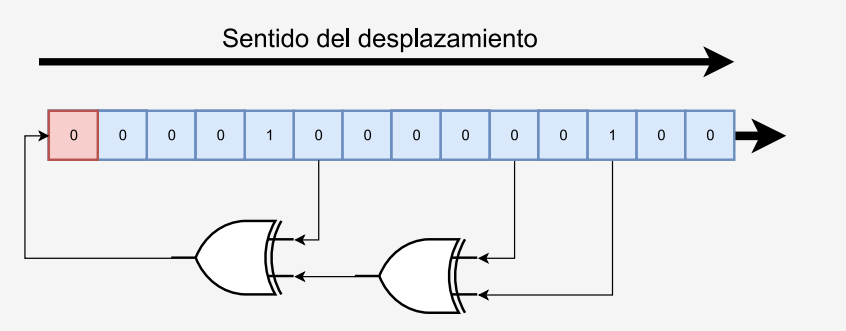


Figura 16: Estructura de un LSFR de Fibonacci

En la figura 16 se muestra un LFSR de Fibonacci. En general los LFSR son una función matemática determinista que usa la combinación de operaciones xor como recurso para crear series pseudoaleatorias, el valor de entrada del registro suele ser una combinación lineal de la salida o bits intermedios. Si se piensa en el registro de desplazamiento como un polinomio, se puede definir el polinomio de retroalimentación del LFSR para un registro de tamaño N bits como:

$$C_{N+1} \cdot x^{N+1} + C_N \cdot x^N + C_{N-1} \cdot x^{N-1} \ + \ ... \ + C_1 \cdot x^1 + 1 \tag{10}$$

Donde los coeficientes $[C_n, C_{N-1}, ...C_1]$ son las conexiones para cada posición del registro que están conectadas a la red de operaciones xor. Por ejemplo usando la figura 16 el polinomio resultante sería:

$$x^6 + x^1 0 + x^1 2 + 1 \tag{11}$$

El funcionamiento de la arquitectura de Galois es similar (figura 17), tan solo que ahora el valor de la entrada del registro de desplazamiento es el valor anterior de la salida, mientras que las operaciones xor se realizan con el valor del bit anterior y el valor de salida. Así por ejemplo si el valor de salida es 0, entonces el LFSR simplemente actúa desplazando hacia la derecha el valor de todos los bits, mientras que si el valor de salida es de 1, entonces los bits en las posiciones con xor tienen como valor el bit anterior invertido. Se puede convertir una arquitectura de Fibonacci en Galois y viceversa obteniendo el mismo resultado. La ventaja de usar una arquitectura de Galois en hardware reside en los tiempos de propagación de las puertas xor, en una arquitectura de Fibonacci las operaciones para determinar el valor siguiente están condicionadas por la cantidad de operaciones xor en cascada, por ejemplo en el caso de la figura 16 existen tres operaciones xor, por lo que si el tiempo de propagación es de t_s, entonces se necesitará como mínimo un tiempo de $3 \cdot t_s$ para evaluar el siguiente estado. Mientras que en una arquitectura de Galois solo es necesario un tiempo de t_S independientemente del número de operaciones xor que existan en el LFSR

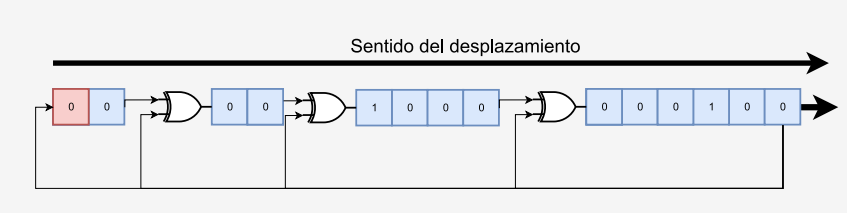


Figura 17: Estructura de un LSFR de Galois

Para que el periodo de repetición del LFSR sea máximo, se deben de cumplir las siguientes condiciones:

- El número de coeficientes del polinomio de retroalimentación, ecuación 10, han de ser pares.
- Los valores de los exponentes del polinomio de retroalimentación deben de ser coprimos entre si.

Capítulo 3

Desarrollo del proyecto

3.1. Diseño

En el apartado 2 se han presentado varias arquitecturas de generadores arbitrarios (true-arb, dds based,interpolate y trueform) presentadas por Keysight. Como se explicó, los AWG del tipo DDS o True Arb son los más comunes, dado que son más simples y baratos a nivel de recursos hardware que los basados en interpolación o Trueform. Por ello se va a elegir entre una de estas dos arquitecturas para realizar el diseño del presente proyecto.

3.1.1. Diferencias entre arquitectura DDS y True Arb

La principal diferencia entre los AWG basados en DDS y los True Arb reside en la frecuencia de muestreo, siendo fija para los basados en arquitecturas con DDS o variable para los True Arb. Esta diferencia en el reloj de muestreo implica unas ventajas e inconvenientes a tener en cuenta a la hora de decidir el tipo de arquitectura para el AWG.
Los basados en DDS son más simples y gastan menos recursos hardware en comparación con los True Arb, dado que tan solo necesitan un reloj fijo. La salida del generador depende de un ratio entre la frecuencia de muestreo, el valor del paso del acumulador de fase y el número de muestras guardadas en la memoria. Esto simplifica mucho la lógica del AWG, pero conlleva inconvenientes:

- Para una memoria dada y una frecuencia de muestreo fija, la salida depende directamente del paso aplicado al acumulador de fase, por lo es posible que no todas las muestras guardadas en la memoria sean utilizadas para construir la forma de onda de salida. Esto implica que puede no representarse la señal de forma fiel. Una solución a este problema sería aumentar el tamaño de las muestras almacenadas, aumentando la memoria, pero esta elección tiene un límite de diseño.
- Adicionalmente puede darse el caso de *jitter* en la salida, producido por variaciones en las direcciones de memoria a leer en cada iteración de un periodo de la señal.

Una arquitectura basada en True Arb no cuenta con un acumulador de fase, se leen todas las muestras almacenadas en la memoria pero variando la velocidad de muestreo de estas, por lo que la frecuencia de la señal de salida depende del reloj aplicado a la lectura de la memoria donde se

guardan los datos de la señal. Como se usan todas las muestras, los dos principales inconvenientes de la arquitectura en DDS no aparecen aquí, pero a cambio la lógica de cambiar la frecuencia de reloj aplicada al muestreo de las muestras es más complicada, y además no se puede cambiar entre dos frecuencias de salida distintas al instante, dado que un cambio en la frecuencia de muestreo también suele venir acompañado de un cambio en las muestras de la memoria para ser leídas, por lo que el proceso de modificar la frecuencia de salida es cuantitativamente más lento en este caso.

3.1.2. Elección de la arquitectura

Visto las principales ventajas y desventajas de las arquitecturas del AWG, la elección entre una de ellas viene dada directamente por la naturaleza de las aplicaciones para las que se va a usar el generador de señales. El ámbito de uso principal del AWG reside en simular señales de radar, con cambios en frecuencia, ratio de señales, modulaciones etc. Por esta razón la opción elegida será una arquitectura de AWG basada en DDS gracias a su capacidad de poder realizar cambios de frecuencia con más facilidad que una arquitectura True Arb. En la figura 18 se representa la arquitectura que desarrollaremos en las siguientes secciones. En ella se puede observar los distintos módulos que se implementarán, los cuales se pueden agrupar en tres grandes bloques:

- Módulos de datos: estos son los módulos que generan la señal de salida propiamente dicha, entre ellos están el acumulador de fase, el generador de ruido de dithering, el generador de direcciones, las memorias de muestras y el módulo de salida que contiene un generador de ruido gaussiano.

- Módulos de control: estos módulos se encargan de monitorizar el estado actual del AWG y de producir las señales de control necesarias para el siguiente estado. Entre ellos están la máquina de estados maestro-esclavo, la máquina de estados del AWG, el detector de flancos, el generador de máscara de salida.

- Módulos de interfaz AXI4 Lite: estos módulos tienen relación con la interfaz entre el AWG y el bus AXI4 Lite. Entre ellos están la interfaz AXI4 Lite, los registros de lectura/escritura del AWG, los cambios de dominio de reloj y la máquina de estados para escribir en las memorias.

El AWG tendrá dos dominios de reloj, uno referente al reloj del AXI4 Lite y otro al reloj del sistema, por lo que en la figura 18 se puede ver dos módulos de reset, cada uno asociado a uno de los dos relojes.Es común en los sistemas digitales emplear un reloj con una frecuencia menos restrictiva para los buses de comunicación que no requieran de un gran *throughput*, por ese motivo se ha añadido un cambio de dominio de reloj entre el bus de comunicación y el sistema, evitando así problemas de meta-estabilidad o la falta de sincronización entre bits.

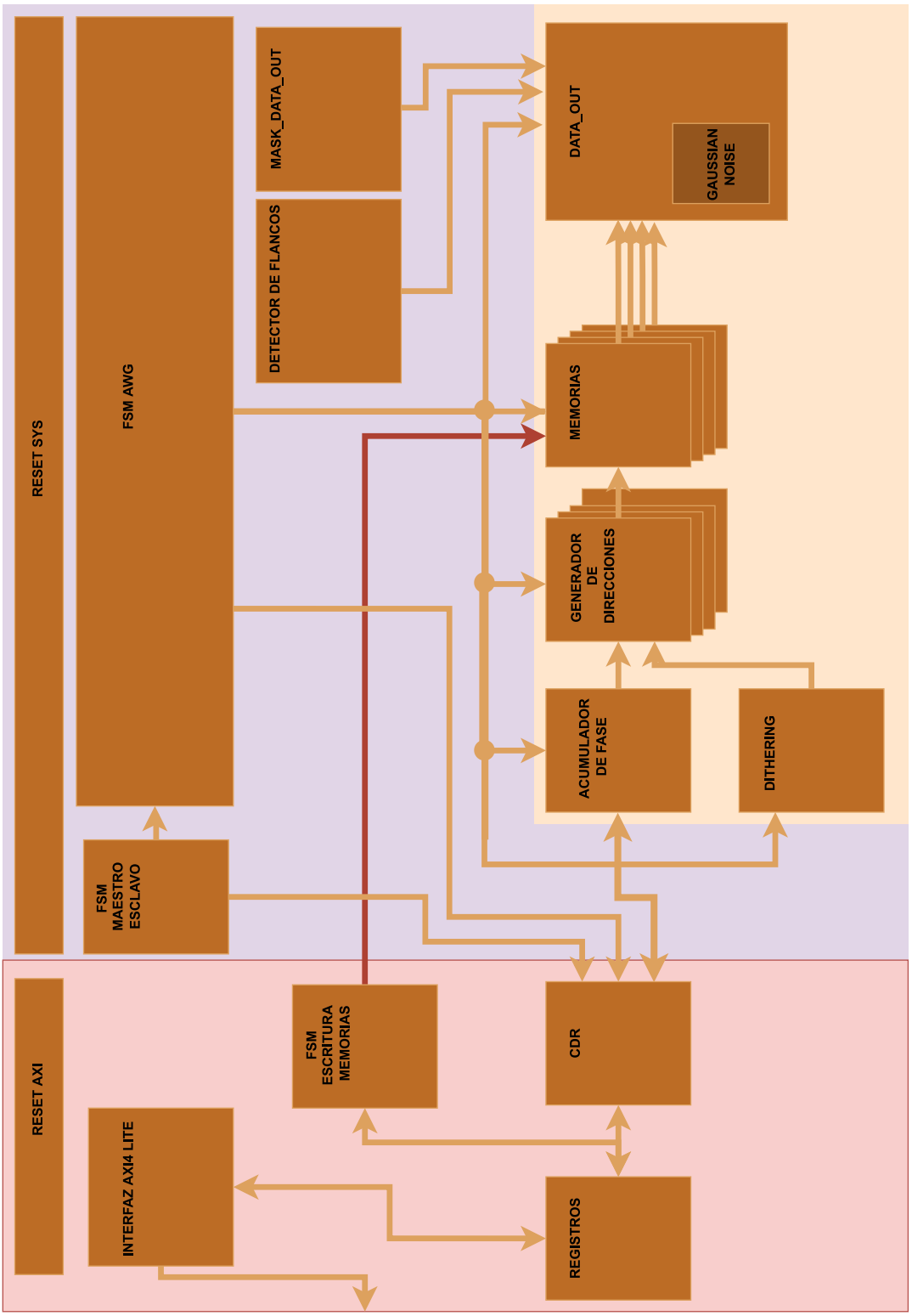


Figura 18: Estructura general del AWG

3.2. Estructura del DDS

El núcleo del AWG para la generación de las formas de onda será un DDS, el cual está formado por un acumulador de fase, un generador de direcciones para generar las distintas paralelizaciones y una serie de memorias donde se almacenaran los datos de la señal a reproducir.
La ecuación 1 describe la frecuencia de la señal de salida en función de ciertos parámetros (frecuencia de muestreo, tamaño de la memoria y el valor de paso del acumulador). Como se puede obtener de esta ecuación, la frecuencia máxima permitida a la salida si se respeta el teorema de Nyquist es de $f_{clk}/2$ es decir, cuando el paso del acumulador de fase es la mitad del tamaño de la memoria. El problema al que nos enfrentamos reside en poder reproducir señales cuyo ancho de banda están por encima del reloj de muestro, que se nos especifica en 312.5 MHz. Es por está razón que el diseño de nuestro DDS se hará paralelizado. Tal y como se propone en [8] se puede aumentar la frecuencia de muestreo del DDS paralelizando las muestras. Paralelizar las muestras implica paralelizar también la memoria y toda la cadena de datos siguiente hasta la salida, por lo que aumenta los recursos consumidos por el AWG en la FPGA. Al añadir paralelización en el diseño, la ecuación 1 se ve modificada incorporando el número de paralelizaciones, como se describe en la ecuación 12.

$$f_{output} = \frac{N \cdot K \cdot f_{clk}}{2^N} \tag{12}$$

Donde N son el número de paralelizaciones.
Ahora la frecuencia de salida máxima vendrá dada por $N \cdot f_{clk}/2$, pudiendo así generar señales con más frecuencia.
De la ecuación 12 se desprende que la elección del tamaño de la memoria afectará de manera directa a dos parámetros, la resolución vertical de la señal y el tamaño del acumulador de fase. Cuanta mayor sea la palabra de la memoria, mayor resolución vertical tendrá la señal, es decir, más valores de amplitud pueden ser representados. La profundidad de la memoria establecerá la cantidad de muestras almacenadas de la señal a reproducir, y como es lógico, a mayor profundidad, más muestras almacenadas de un periodo de la señal. Pero en la FPGA los recursos son finitos y se tiene que decidir un valor de compromiso para elegir el tamaño de la memoria, tanto de palabra como de profundidad. En el caso de este proyecto, se usarán unas FPGA de la familia de Kintex Ultrascale+ de Xilinx, cuyas memorias BRAM tienen 36 Kb (RAMB36E2) de memoria que pueden configurarse como 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18 o 1K x 36. La señal de salida para el AWG es de como mínimo de 12 bits, por lo que las opciones que quedan son la de 2K x 18 o la de 1K x 36. Ambas configuraciones ocuparán la misma cantidad de recursos, pero la opción de 2k x 18 nos habilita la posibilidad de dividir la memoria en dos regiones para almacenar dos formas de onda distintas de 1k x 18, por lo que la opción que se elegirá será la de usar una memoria para una forma de onda de 1k x 18 doble.

3.2.1. Acumulador de fase

La primera etapa del DDS es el acumulador de fase. Este módulo se encarga de generar un contador de paso variable, cuya salida se usará para crear las direcciones de memoria con las que leer las muestras de la señal en las distintas memorias. Como entradas, el módulo recibirá una señal de reloj, siendo esta la frecuencia de muestreo, una señal de reset, el valor de paso del acumulador,

una señal de activación, tal y como se muestra en la figura 19 . Como salida tendrá el valor del contador y su señal de dato válido.

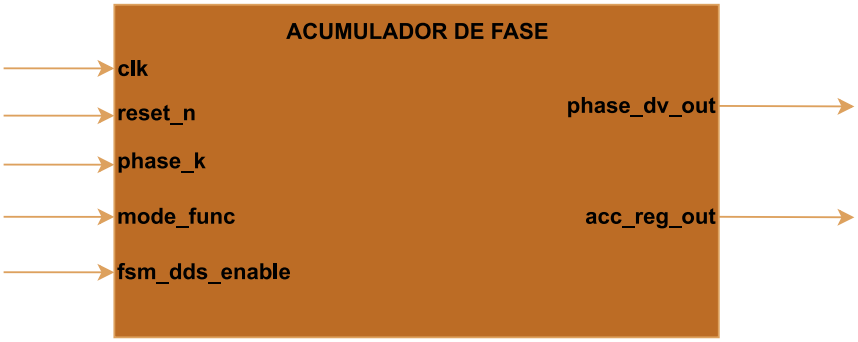


Figura 19: Esquema de entradas y salidas del acumulador de fase

Usando la ecuación 12 como base, el acumulador de fase debe generar un contador cuyo paso sea la señal phase_k a una frecuencia impuesta por el reloj clk cuando la señal fsm_dds_enable esté activa. Esto da como resultado que el acumulador de fase deba realizar dos operaciones distintas, una multiplicación entre el valor de K y el número de paralelizaciones y la suma de este resultado con el valor anterior de salida del acumulador de fase. Las dos operaciones quedarían como:

$$operacion_1 = K \cdot N$$
$$operacion_2 = operacion_2 + operacion_1$$

Que implementado quedaría tal y como se muestra en la figura 20.

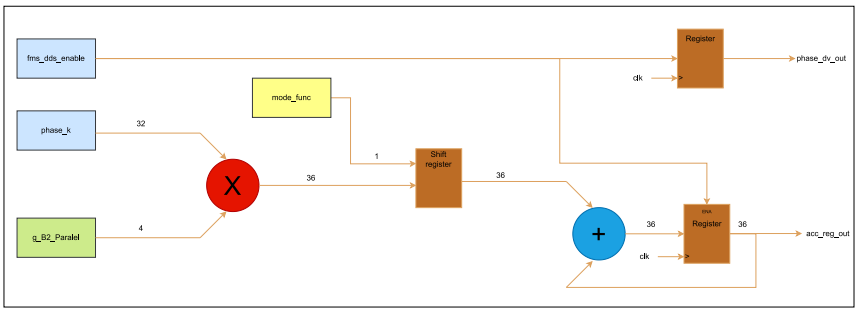


Figura 20: Esquema interno del acumulador de fase

Adicionalmente se incluye una señal de control más, $mode_{func}$, la cual controlará el modo de funcionamiento de interleaving, pudiendo emparejar dos módulos de AWG para funcionar con paralelización N pero uno de ellos solo sacará las muestras pares y el otro las muestras impares, pudiendo así generar el doble de frecuencia de la señal de salida.

3.2.1.1. Elección de los parámetros del acumulador

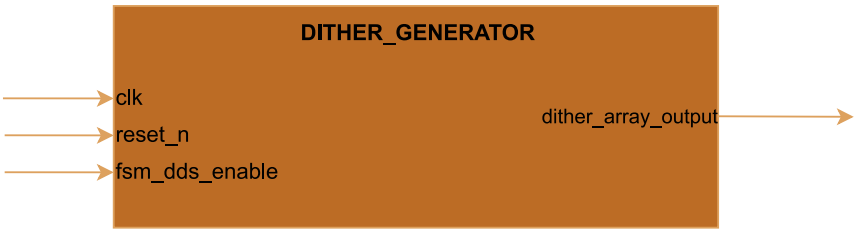
El parámetro principal del acumulador es el tamaño del contador, el cual debe, ser como mínimo, igual al tamaño de la memoria donde se almacenan las muestras de la señal a generar. Como

se ha explicado anteriormente, el tamaño elegido de la memoria es de 1k x 18, por lo que como mínimo el contador del acumulador debe tener modulo 1024. Como se ha explicado en 2.4.3 el tamaño del acumulador se suele escoger mayor al tamaño de la memoria, para así tener un GRR más grande y poder esparcir los espurios por el espectro. El tamaño elegido para el contador del acumulador es de 36 bits principalmente por dos razones:

- El valor del paso del acumulador será cargado a través de un registro mapeado en el bus AXI4 lite, el cual tiene un tamaño de palabra de 32 bits, por lo que la carga del valor sería directo, los 4 bits restantes son el resultado de multiplicar por el valor de las paralelizaciones máximas que pueden ser codificadas en un número de 4 bits.

- Un tamaño de 32 bits cumple con la regla de A-P >4, siendo A=32 y P=11.

3.2.2. Generador de dithering



Como se ha expuesto en 2.4.3, para mejorar el SFDR es aconsejable añadir un ruido de dithering a la salida del acumulador. Esto se conseguirá añadiendo un módulo que genere una serie pseudoaleatoria y sumando el bit más significativo del ruido al bit A-P-1 del acumulador, consiguiendo sumar 0 o 0,5 bits al valor truncado. El tamaño del vector de dithering será igual al tamaño del acumulador menos el tamaño del truncado, si tenemos un valor de A=32 y de P=11, entonces el valor D=21. El polinomio elegido para la serie será:

$$x^{21} + x^{10} + x^{18} + x^{16} + 1 \tag{13}$$

Dando un periodo de repetición de $2^{21} - 1$. La elección de los coeficientes se ha elegido de forma que sean coprimos entre si, y un número par, obteniendo de este modo el periodo máximo para el tamaño del vector LFSR, tal y como se indica en 2.4.5. En el caso de tener solamente dos coeficientes, la diferencia entre una arquitectura de Galois o de Fibonacci no es significativa en cuanto a tiempo de propagación por las operaciones xor, pero en la arquitectura de Galois se usan dos comparaciones xor, mientras que en la arquitectura de Fibonacci solamente se evalúa con dos operaciones xor, por lo que a la hora de modelar el módulo de dithering se empleará una arquitectura de Fibonacci por comodidad.

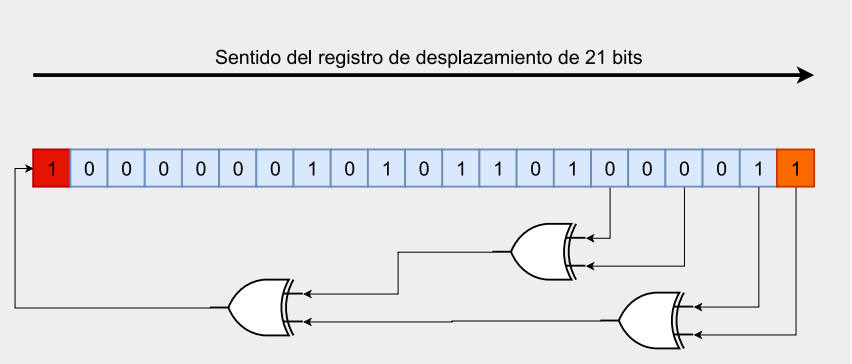


Figura 21: Estructura del LFSR empleada.

En la figura 21 se observa la arquitectura implementada para la generación del ruido de dithering, la semilla del vector es 22147, valor que se carga cuando existe un reset del módulo.

3.2.3. Generador de direcciones

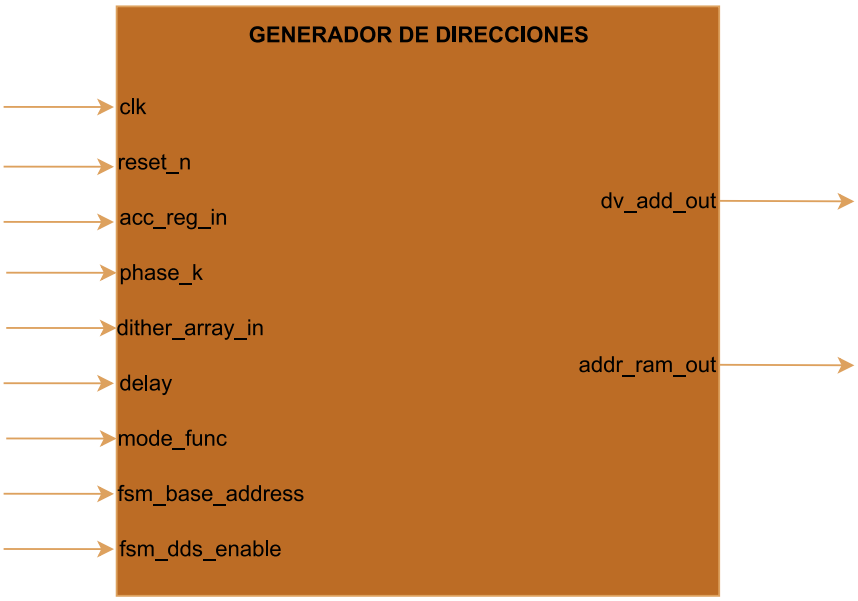


Figura 22: Esquema de entradas y salidas del generador de direcciones

Como se ha explicado en 3.2 se van a paralelizar las muestras para conseguir que cada ciclo el AWG genere N muestras aumentando la frecuencia de salida en un factor N. Para ello es necesario un módulo intermedio que genere las N direcciones de memoria necesarias a partir de la rampa de salida del acumulador. Un esquema de las entradas y las salidas del generador de direcciones se expone en la figura 22. El módulo tiene las siguientes entradas:

- clk: el reloj del sistema.
- reset_n: el reset de lógica negada asociado al reloj.

- acc_reg_in: la salida del acumulador de fase.
- phase_k: el valor del paso del acumulador.
- dither_array_in: el valor de salida del generador de dithering.
- delay: el valor del registro de desfase de la señal.
- mode_func: la señal de control del modo interleaving.
- fsm_base_address: la señal de control que determina si se lee de la parte baja o la parte alta de la memoria para elegir entre las dos señales almacenadas.
- fsm_dds_enable: el data valid de salida del acumulador de fase.

El generador de direcciones debe implementar la siguiente operación:

$$acc\_reg\_in \ + \ (i \ + 1) \cdot K + dither + delay \tag{14}$$

Donde i es el factor de la paralelización de la instancia del módulo, que puede tomar los valores entre [0,N-1]. Por ejemplo, si tenemos una paralelización N=4, entonces i valdrá [0,1,2,3], y cada una de estas paralelizaciones generará la correspondiente dirección para la memoria N de salida. De la ecuación 14 se desprende que son necesarias las operaciones siguientes:

- La multiplicación del paso del acumulador por el factor de la paralelización. Esta operación se lleva a cabo entre una constante (el factor de la paralelización) y una señal que varia poco durante el uso del AWG. El tamaño del paso es de 32 bits y el tamaño de la paralelización está acotado entre 0 y 15, por lo que con 4 bits es suficiente para codificar dicho valor. Por lo tanto el tamaño resultante de la operación será de 36 bits.
- Las sumas de acc_reg_in, dither, delay y el resultado de la multiplicación anterior. Esta serie de operaciones de suma intervienen cuatro operandos, por lo que se realizará sumando dos a dos y el resultado se sumará de nuevo, necesitando tres operaciones de suma en total. Las dos primeras operaciones serán la suma de acc_reg_in (36 bits) con el resultado de la multiplicación (36 bits). Dado que el valor del bit de mayor peso del paso del acumulador nunca puede ser 1, el resultado de la suma se quedará en 36 bits también. La segunda suma será entre el valor del delay de 32 bits y el valor del dithering de 22 bits. El valor de delay está codificado en los 10 bits de mayor peso, por lo que al sumar el valor de dithering de 22 bits, nunca llevará a una situación de acarreo, y el resultado de la suma también sera de 32 bits. Por último se suman el resultado de las dos operaciones anteriores, dando como resultado un valor de 36 bits.

La arquitectura interna del generador de direcciones se puede ver en la figura 23. Se puede observar que se usará un multiplicador, tres sumadores y tiene un total de 3 ciclos de latencia.

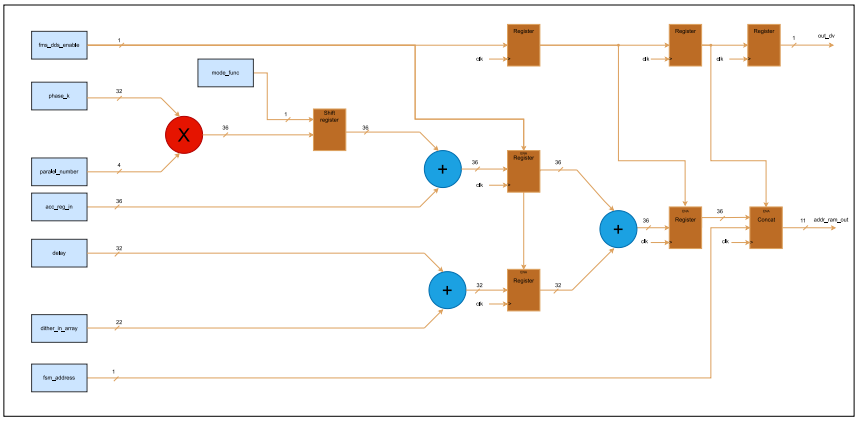


Figura 23: Esquema del generador de direcciones

3.2.4. Memoria de datos

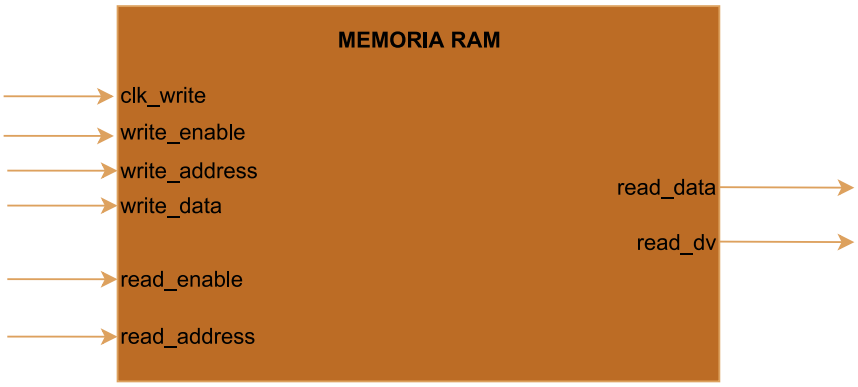


Figura 24: Esquema de las memorias de datos

La figura 24 presenta el esquema de entradas y salidas de las memorias donde se almacenaran las muestras de las señales. Se puede observar que tienen dos puertos separados, uno de lectura y otro de escritura, con dominios de reloj distintos. Esto se ha diseñado así para facilitar la escritura de las muestras, ya que los datos de las muestras se recibirán a través del bus AXI4 Lite, el cual tiene un reloj más lento que el reloj del sistema. Para no realizar un cambio de dominio de reloj de las muestras y las direcciones, se implementa una memoria de doble puerto, de esta forma la escritura se realizará a través del domino del reloj de escritura (el reloj del AXI4 Lite) y la lectura de datos en el dominio del reloj del sistema. Como se ha indicado en 3.2, para obtener la máxima eficiencia de los recursos disponibles en las FPGA, se van a implementar las memorias de 2k x 18 bits para cada paralelización. Así, por ejemplo, si configuramos el AWG con paralelización 4, entonces se usarán 4 BRAMs completamente.

3.2.5. Detector de periodos

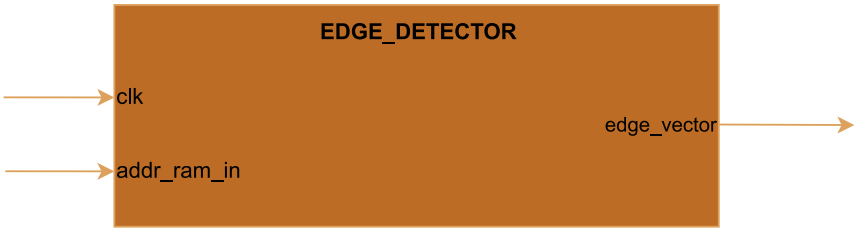


Figura 25: Esquema del detector de periodos

Cuando se configura el AWG como generador de pulsos, es necesario conocer cuando acaba un periodo de señal completo, dado que se puede dar el escenario de configurar un PW con un valor de paso que al acabar el tren de pulsos se quede la última sección de la señal sin acabar el periodo completo. Este fenómeno crea espurios indeseados en el espectro de salida y para ello es necesario diseñar un módulo que detecte cuando ha acabado un periodo de señal completo para un valor de paso.

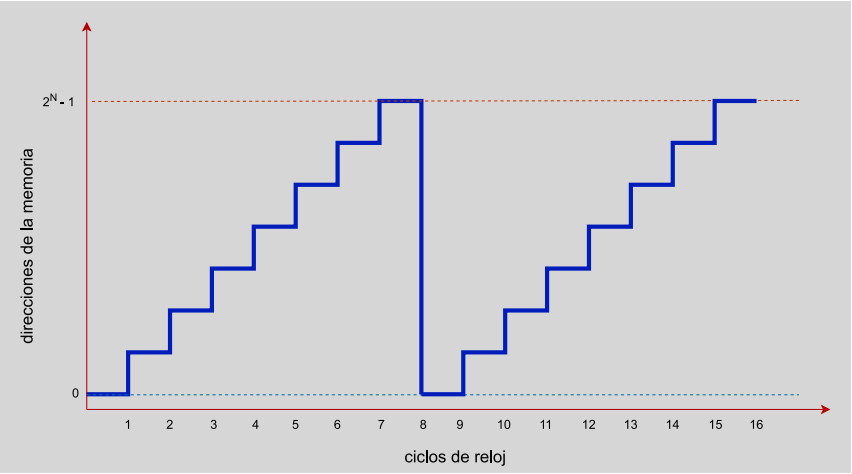


Figura 26: Salida del acumulador con paso 1 y N=3 y paralelización 1

En la figura 26 se puede observar la salida de un acumulador de fase de N=3 y paso igual a 1. En este caso el acumulador recorre las 8 posibles posiciones que puede generar. Con este valor de paso cada periodo de señal nuevo empieza siempre en el mismo punto inicial. Pero si variamos el valor del paso, puede no suceder que cada nuevo periodo empiece en el mismo valor inicial. Por ejemplo en la figura 27 el para la misma N=3 el paso es de 3. El segundo periodo de señal no empieza en 0, sino 1, y el segundo periodo empieza en 2. Si usamos la ecuación del GRR 5 el periodo de repetición es de 8 ciclos, tal y como se puede comprobar en la figura 27.

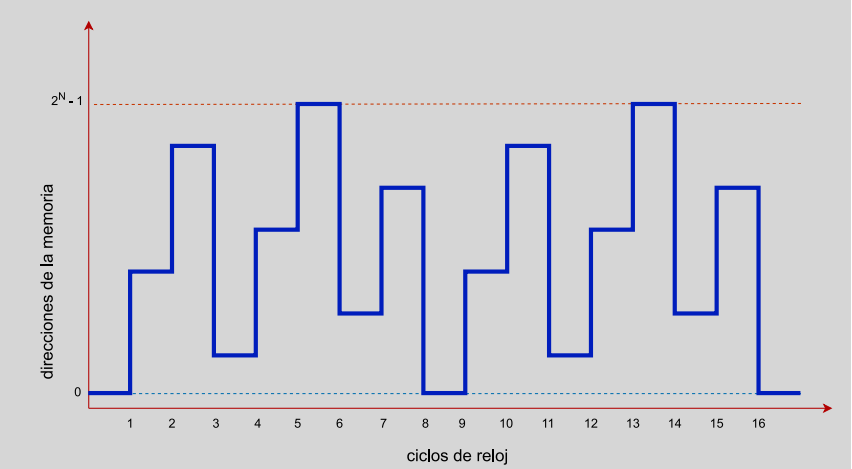


Figura 27: Salida del acumulador con paso 3 y N=3 y paralelización 1

El ejemplo visto es con una paralelización de 1, cada ciclo de reloj se genera una sola dirección de memoria. Pero para una paralelización P, cada ciclo se generan P muestras. Esto implica la posibilidad de que el final del periodo acabe en una de las muestras de una paralelización.

Ciclo	Paralelización 0	Paralelización 1	Paralelización 2	Paralelización 3
1	0	3	6	1
2	4	7	2	5
3	0	3	6	1

Tabla 1: Ejemplo de acumulador con N=3, paso=3 y paralelización=4

Por ejemplo, si paralelizamos el ejemplo de la figura 27 entonces obtenemos la secuencia descrita en la tabla 1. Como se puede ver, en el primer ciclo el periodo completo de la señal ha acabado en la paralelización 2, mientras que en el ciclo 2 el periodo ha acabado en la paralelización 1. Para detectar si se ha completado un periodo completo se va a comparar cada ciclo los valores de cada muestra con la anterior. Si denotamos a $A[s] = A_0[s], ..., A_p[s]$, donde p es el número de paralelizaciones, como las muestras del ciclo actual y a $A[p-1] = A_0[s-1], ..., A_p[s]-1$ a las muestras del ciclo anterior, entonces se compararán las muestras como:

$$
\begin{aligned}
&A_n[s-1] < A_0[s],\\
&A_0[s] < A_1[s],\\
&...,\\
&A_{n-1}[s] < A_n[s]
\end{aligned}
$$

Siguiendo el ejemplo anterior, para el ciclo 2 se compararán los valores como:

$$1 > 4 = false,$$
$$4 > 7 = false,$$
$$7 > 2 = true,$$
$$2 > 5 = false,$$

El resultado de estas comparaciones se almacena en un vector que tendrá la información de las finalizaciones de periodos en cada ciclo de reloj. En el ejemplo anterior como resultado de las operaciones se guardará la información en el vector [0,0,1,0], informando que existe una finalización de periodo en el ciclo actual, concretamente entre la paralelización 1 y 2.

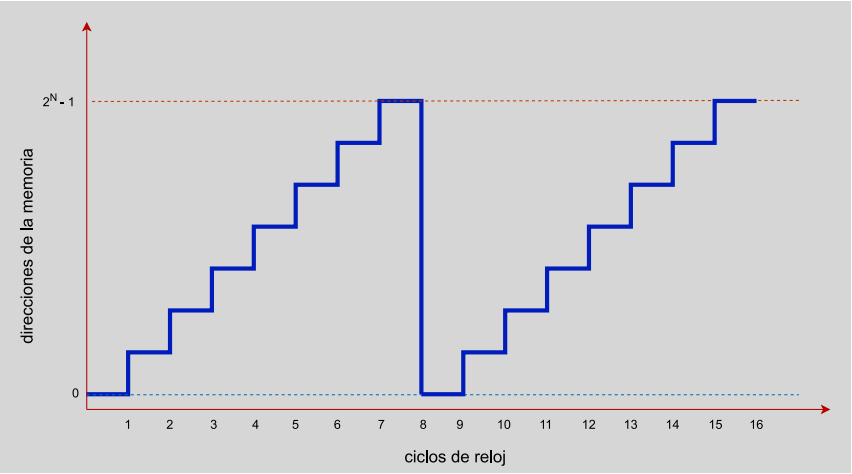


Figura 28: Esquema del funcionamiento del detector de periodos

En la figura 28 se presenta la arquitectura del detector de periodos, el cual toma como entrada las P muestras de salida del acumulador truncado y genera en la salida un vector con la información de la finalización de los periodos en ese ciclo. La operación tarda un ciclo de reloj en comparar y generar el vector de salida.

3.2.6. Máscara de datos

El módulo anterior recogía la información acerca de cuantos periodos de señal completa existía en un ciclo. En el ejemplo propuesto se podía advertir que existía una finalización de periodo completo por cada ciclo para un paso de 3. Pero en el caso de que exista más de una finalización de periodo por ciclo, solo interesa el primer valor de finalización, por ejemplo si el vector de salida del detector de periodos es [0,1,0,1] solo nos interesaría el primer cambio de ellos, es decir, que necesitaríamos modificar el vector a [0,1,1,1].

El módulo tendrá como dato de entrada la salida del detector de periodos y generará como salida la máscara a aplicar en la salida, proceso que se explicará en la sección del módulo de datos de salida 3.2.8. En la figura 29 se puede ver el esquema de las entradas y salidas del módulo.
El funcionamiento es sencillo: llamamos M al vector se salida de la máscara y al vector del detector

Figura 29: Esquema de la máscara de datos

de periodos D. Entonces para cada bit del vector $M[i]$ se compara si el valor entero del subvector formado por los valores $D[i], ..., D[0]$ es mayor que 0. Si el valor del subvector es mayor que 0, entonces indica que al menos ha habido una finalización de periodo, y por lo tanto el valor de la posición M[i] valdrá 0, en caso contrario será 1. Si cogemos el ejemplo anterior con D = [0,1,0,1] en la figura 30 se ilustra como sería el resultado. Al igual que el módulo del detector de periodos, la generación de la máscara se computa en un solo ciclo de reloj.

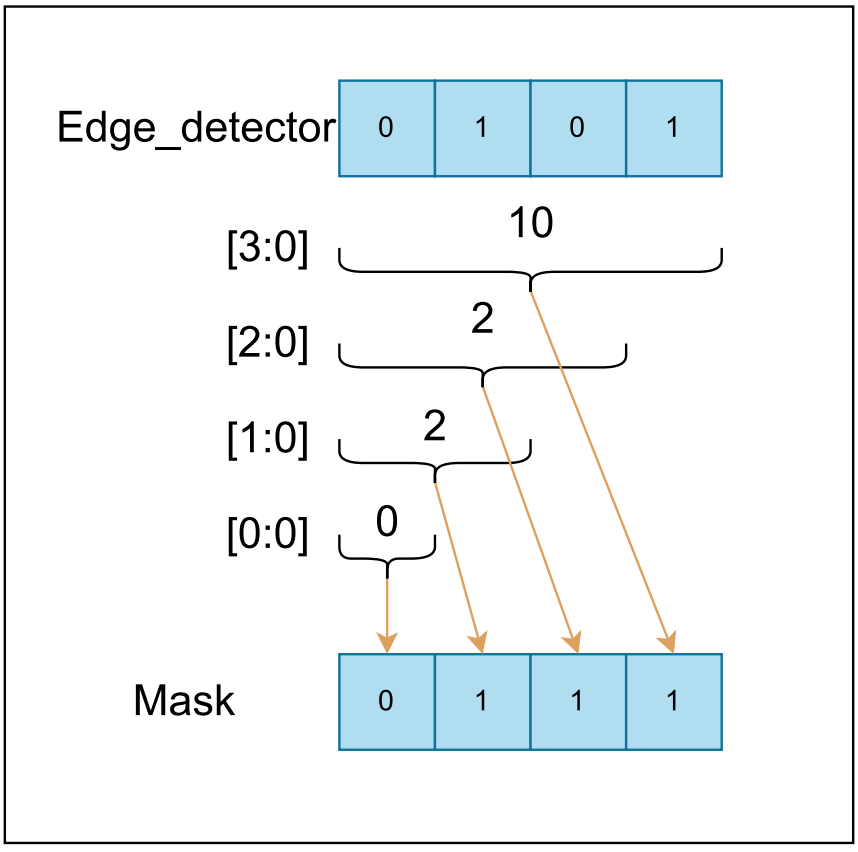


Figura 30: Ejemplo del funcionamiento de la máscara de datos

3.2.7. FSM principal

El AWG tiene la posibilidad de poder trabajar con tres modos distintos:

- Modo continuo: en este modo el AWG genera a la salida una señal de frecuencia f de forma continua. Se puede detener con la señal de paro, momento en el que el AWG se queda en un estado de espera a la elección del siguiente modo.
- Modo pulso único: en este modo se especifica al AWG un valor de PW y de frecuencia y se espera a una señal de disparo, momento en el que el AWG genera un tren de pulsos de la frecuencia elegida durante un tiempo PW. Al acabar el módulo se queda a la espera de otra señal de disparo para otra secuencia o la señal de paro para cambiar de modo.
- Modo pulsos repetidos: en este modo además de especificar la frecuencia y el PW como en el modo de pulso único, también se especifica el PRI. Se espera a una señal de disparo, momento en el que el AWG empieza a generar señales pulsadas de frecuencia f, PW a intervalos PRI. La señal se repite hasta que el operador habilita la señal de paro, momento en el que el módulo se queda en un estado de espera.

Para poder gestionar los distintos modos se diseña una maquina de estados principal, la cual generará las distintas señales de control en base al estado actual y las señales de estado de distintos módulos.

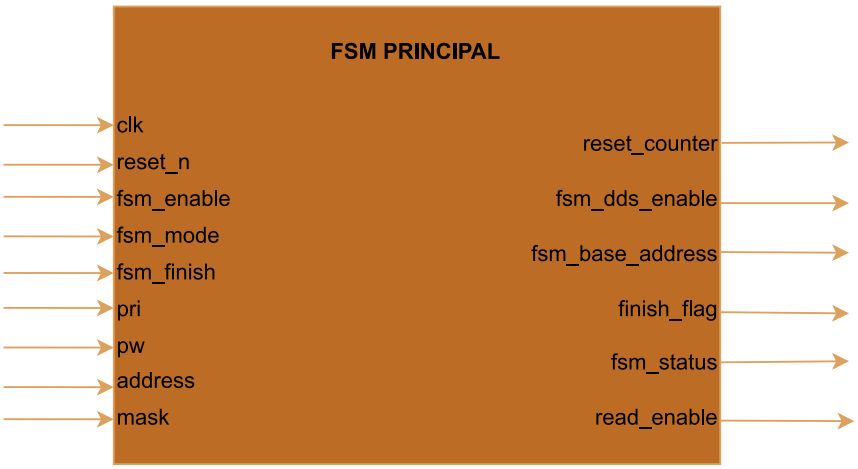


Figura 31: Esquema de entradas y salidas de la fsm principal

La máquina de estados principal recibe dos tipos de entradas:

- Entradas de configuración. Estas corresponden a las entradas que el operador configura en los registros del AWG. Son el fsm_enable, fsm_mode, fsm_finish, burst_trigger, pri, pw y address.
- Entradas de estado. Son dos señales de estado del AWG explicadas en módulos anteriores. Corresponden al contador de número de periodos y a la máscara de datos.

En cuanto a las salidas se pueden agrupar en dos tipos también:

- Salidas de control. Corresponden a aquellas salidas que modifican el flujo de otros módulos. Estas son el reset_counter, fsm_dds_enable, fsm_base_address, finish_flag y read_enable.

- Salidas de estado. La señal fsm_status da la información del estado actual de la maquina de estados principal que se guarda en un registro de estado mapeado en el AXI4 Lite.

La arquitectura de la fsm principal se basará en una máquina de tipo Mealy, que contará con 9 estados para generar los tres tipos diferentes de modos de funcionamiento. Los estados son:

- **IDLE**: Este es el estado por defecto. Se trata de un estado de espera a que se configure la transición a alguno de los modos. También es el estado al que se deriva la fsm si el siguiente estado no es ninguno de codificados.

- **CONTINUO**: Este estado es el que genera las señales de control para una generación continua de la señal de salida.

- **BURST_WAIT**: Este estado es la antesala para generar un único tren de pulsos. Cuando se configura el modo de pulso único se entra en este estado a la espera de el disparo.

- **BURST_EXTEND**: Este estado sirve para finalizar el tren de pulsos del modo pulso único si fuera necesario

- **FIRST_TRIGGER**: este es el primer estado del modo pulsos continuos. Cuando se configura el modo pulsos se espera que el primer disparo sea enviado por el operador.

- **PULSE**: estado que genera los pulsos dependiendo de su frecuencia y el PW.

- **PULSE_WAIT**: este estado evalua si es necesario ampliar el pulso para que el último periodo de la señal se complete o si supera el valor del PRI.

- **TRIGGER_WAIT**: este estado espera a que el contador de PRI llegue al final de la cuenta, significando que un nuevo tren de pulsos debe de ser generado en el estado PULSE.

- **FINISH**: a este estado llegan todos los estados anteriores salvo el IDLE cuando la señal de entrada fsm_finish se configura a 1, momento que se vuelve al estado IDLE.

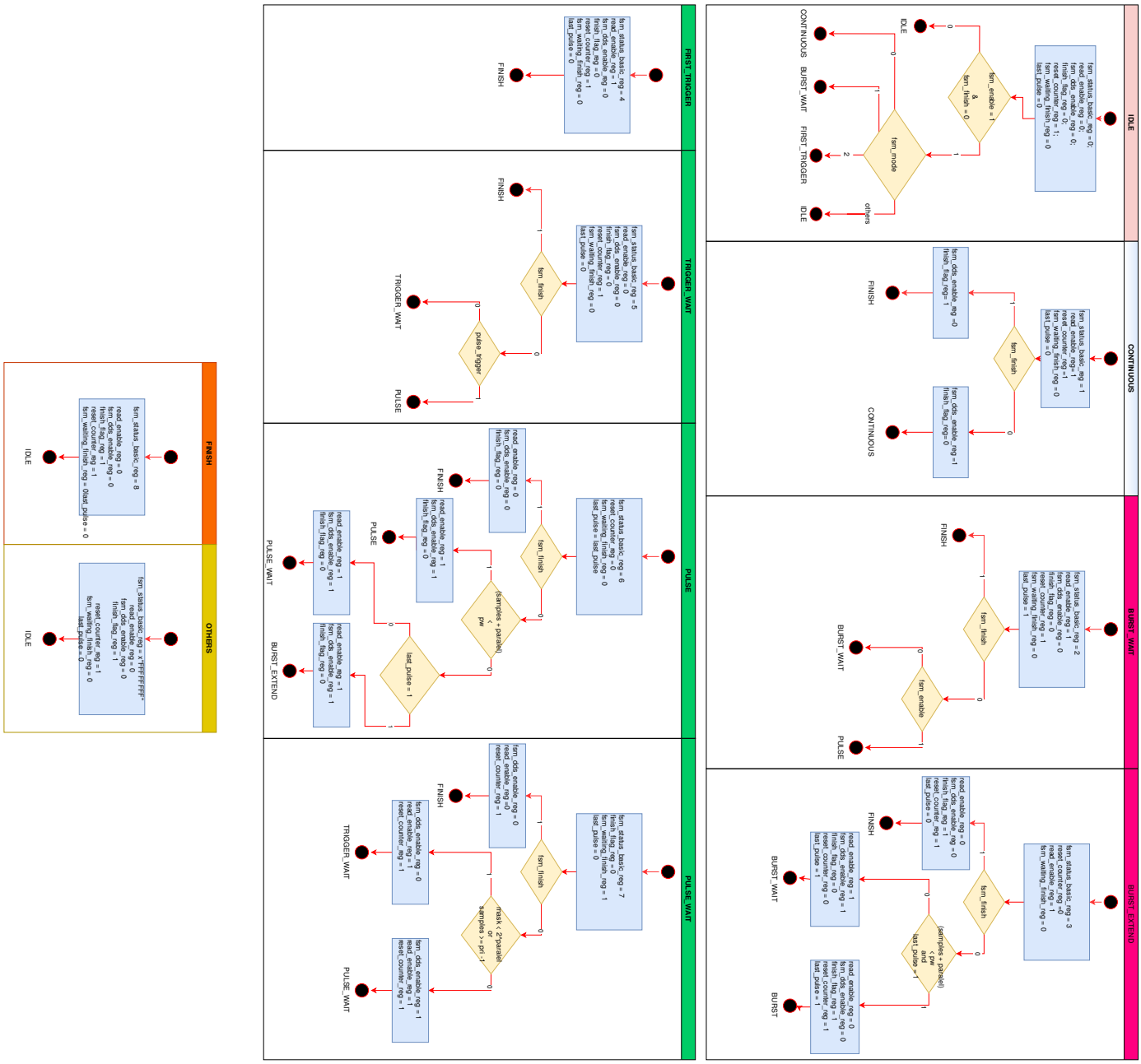


Figura 32: Diagrama de flujo de la fsm principal

3.2.8. Módulo de salida

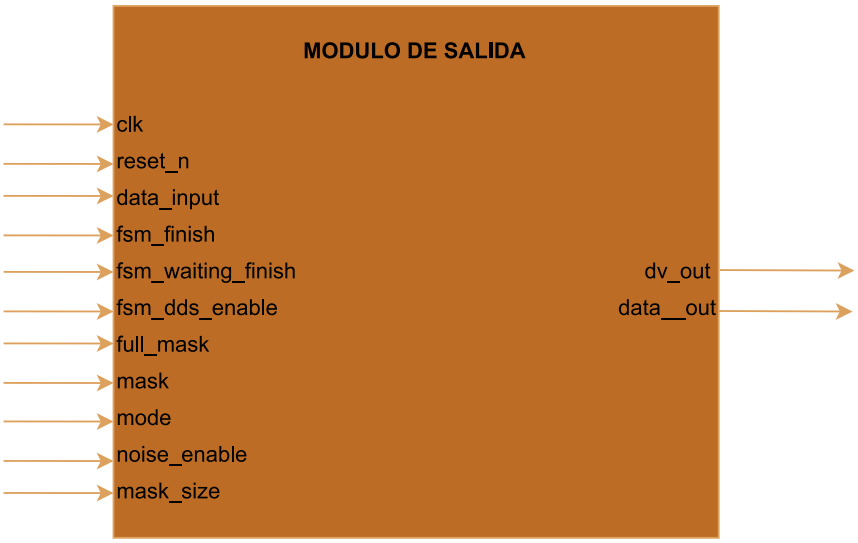


Figura 33: Esquema del módulo de datos

El esquema de entradas y salidas del módulo de salida de datos se puede ver en la figura 33. De los módulos de la cadena de datos es el que más señales de control tiene, esto es debido a que realiza dos tareas:

- La primera tarea reside en el modo pulsos. Cuando se especifica en el modo pulsos los parámetros de PW y de PRI para una frecuencia en concreto, es posible que el tren de pulsos acabe en un valor no múltiplo de la paralelización, produciendo artefactos que conllevan la aparición de espurios, como se ha comentado en la sección 3.2.5

- La segunda tarea es para implementar un generador de ruido gaussiano, el cual se usará para añadir ruido de amplitud a la salida, pudiéndose modificar la cantidad de bits añadidos mediante un registro de control.

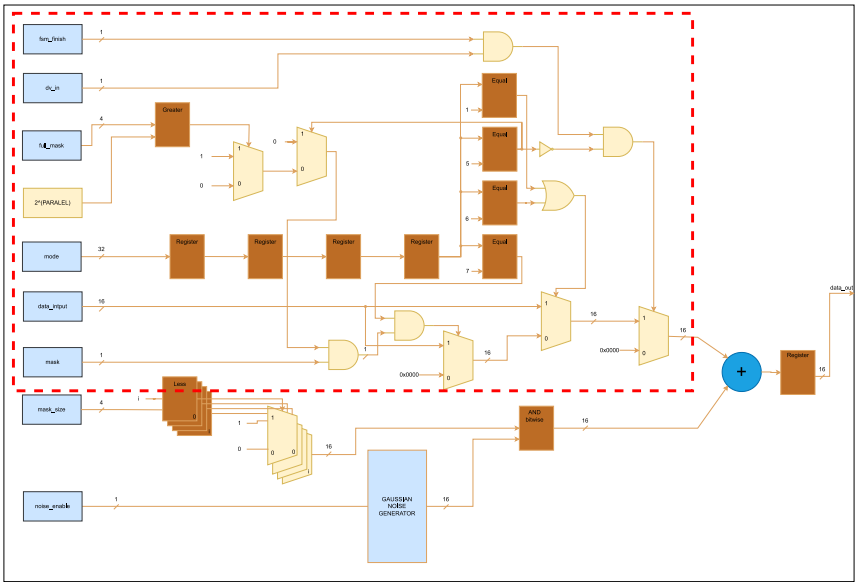


Figura 34: Esquema del módulo de datos de salida

3.2.8.1. Descarte de las últimas muestras de pulso

Mediante los módulos de la FSM principal y la máscara de datos se puede generar la lógica para conocer el instante en el que se debe dejar de introducir datos, eliminando parte del pulso ya que este ha finalizado, en la figura 34 se corresponde con el área delimitada por el recuadro rojo. La secuencia es la siguiente:

Existen dos procesos en paralelo:

- El primer proceso determina ya ha acabado el periodo del último pulso entre dos pulsos. Para ello se sirve de la máscara generada por el módulo descrito en 3.2.6. Si el modo de la FSM principal es el de esperando disparo, entonces la señal stop_pulse se establece en 0, indicando que aún no se ha de parar el pulso. Si no se está esperando al disparo entonces se comprueba el valor de la máscara. Si el valor es inferior al límite máximo en decimal del array, es decir $2^{Paralelizaciones} - 1$, entonces existe al menos una finalización de periodo dentro de ese ciclo, y por lo tanto de establece el valor de stop_pulse a 1. Se no es el caso, entonces el valor de stop_pulse mantiene el valor anterior.

- El segundo proceso establece si la muestra es válida o no dependiendo de cinco señales de control: fsm_finish, data_valid, estado de la FSM principal, el valor del bit de la máscara que corresponda a la paralelización que se esta evaluando y la señal descrita en el anterior punto, stop_pulse. En la figura 35 se presenta el flujo de elecciones para el recorte de las muestras. El estado de la FSM principal se retrasa 5 ciclos debido a la latencia conjunta del acumulador de fase, el generador de direcciones y la memoria de muestras.

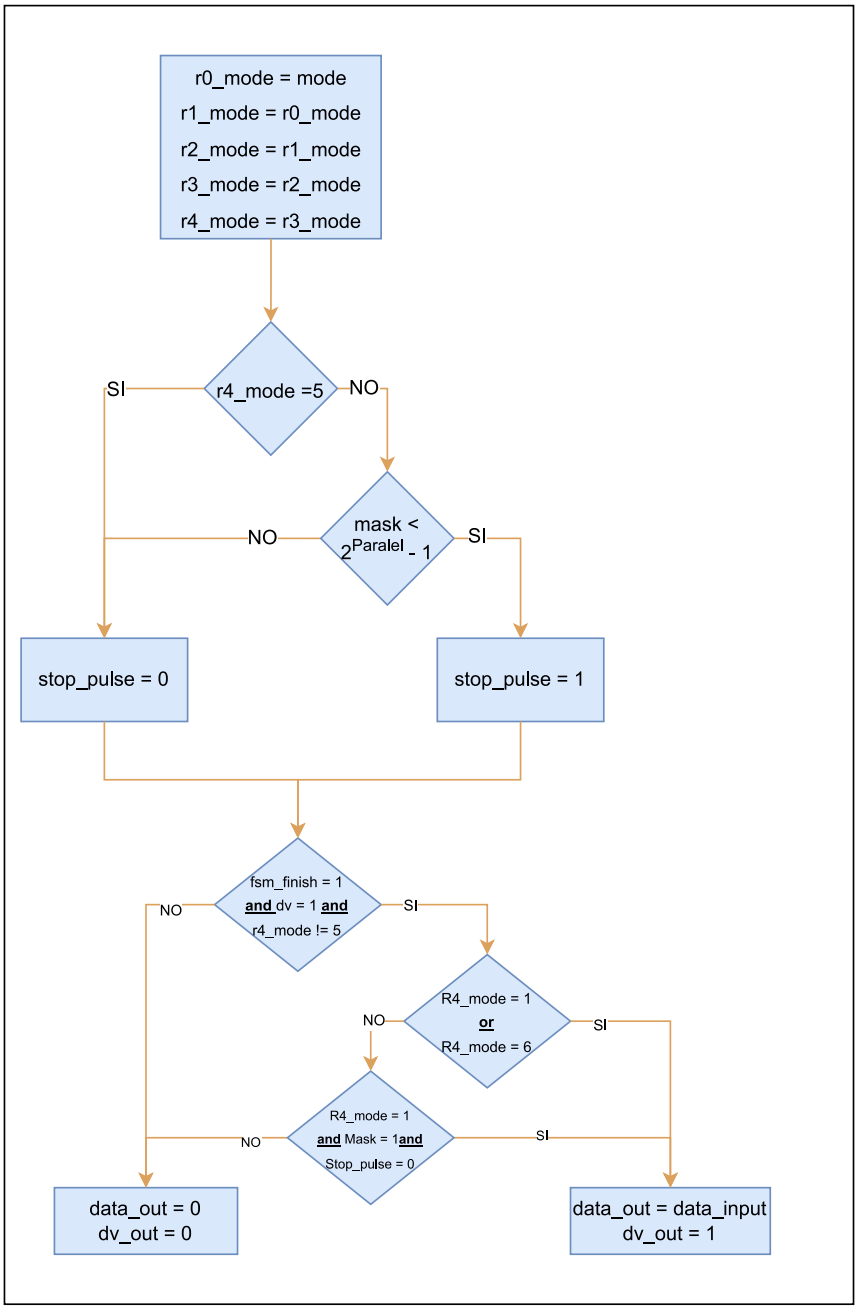


Figura 35: Diagrama de flujo para el recorte de muestras

3.2.8.2. Generador de ruido gaussiano

La segunda parte del módulo de datos de salida es la adición de ruido gaussiano a la información de amplitud de la salida. Adicionalmente se puede elegir la cantidad de bits de ruido que se añade a la amplitud desde un registro interno del AWG. El módulo de ruido gaussiano no se ha diseñado para este trabajo, está recogido de un trabajo de máster anterior hecho por Alberto Martínez Cuesta titulado DISEÑO E IMPLEMENTACIÓN DE UN JAMMER CONFIGURABLE EN FGPA. El módulo del ruido gaussiano implementado se basa en los LFSR vistos en 2.4.5 y en el

teorema del límite central. Este teorema establece que si se tiene una serie de variables aleatorias e independientes entre si, su media seguirá una distribución normal. Haciendo uso de este teorema, se puede diseñar un generador de ruido de distribución normal o gaussiana usando elementos pseudoaleatorios generados por LFSR. Si cada LFSR se inicializa con una semilla distinta, aún cuando el polinomio de retroalimentación sea el mismo para todos, estos generaran series distintas, independientes entre si. De cada LFSR se obtiene el bit de salida para formar palabras cuyos elementos serán independientes entre si, generando una distribución uniforme. Si se generan varias de estás palabras y se realiza una media de ellas, entonces se obtendrá una señal de salida cuya serie en el tiempo tendrá una distribución gaussiana.
Como se explica en [9], se usan 64 LFSR para generar 4 palabras de 16 bits cada una de distribución uniforme, las cuales se promedian para obtener una palabra de 16 bits de distribución gaussiana. La salida del módulo de ruido gaussiano es enmascarado con el valor del registro de tamaño de ruido para determinar la cantidad de ruido a añadir a la salida del AWG. Una vez enmascarado el ruido, se decide si añadir el ruido a la señal de salida mediante un bit de control en el registro general del AWG.

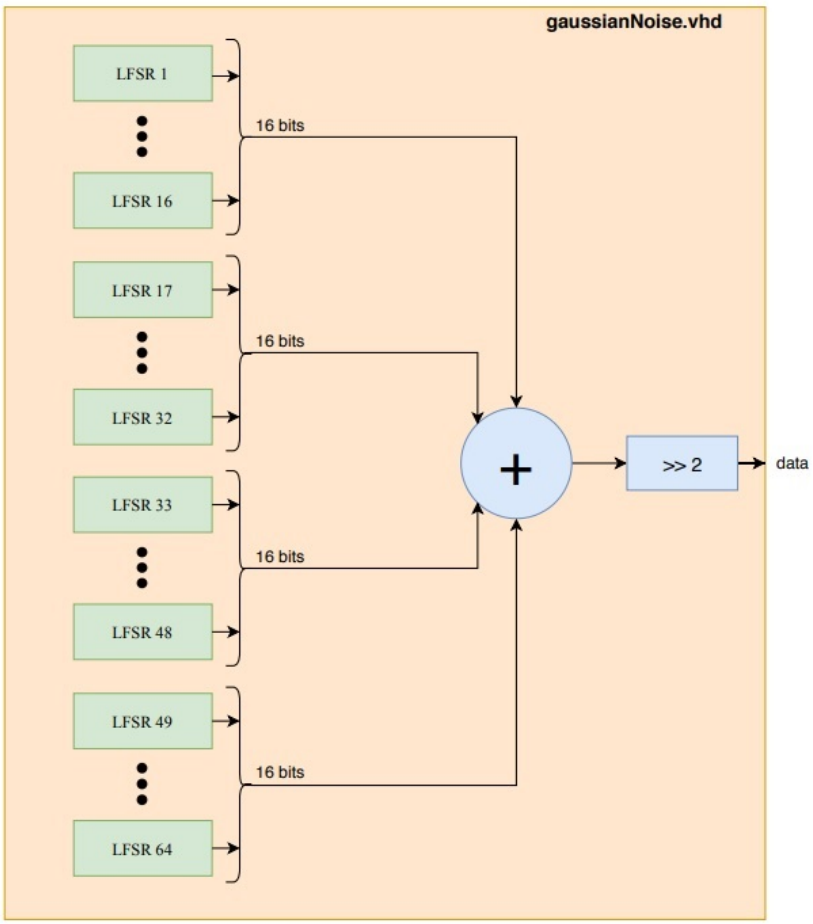


Figura 36: Diagrama del generador de ruido gaussiano [9].

3.2.9. FSM de escritura a memoria

Las muestras almacenadas en las memorias deben de ser independientes por cada canal del AWG que se instancie. Es por eso que se ha diseñado un módulo con un protocolo que permite escribir en cada memoria de cada canal a través de un solo registro del AXI4 Lite.

Figura 37: Esquema de entradas y salidas de la máquina de estados de escritura

Esta máquina de estados cuenta con 5 estados diferentes, los cuales sirven para determinar cuando y a que memoria escribir los datos. Los estados son los siguientes:

- **IDLE**: estado inicial de espera a que se escriba en el registro el bit de write_enable.
- **WAIT_STROBE**: este estado espera a que un dato válido se escriba en el registro del AXI4 Lite.
- **CHECK_CHANNEL**: este estado realiza la comprobación de la finalización de la escritura cuando el bit de finalización del registro se pone a 1.
- **SEND_DATA**: desentrelaza la información guardada en el registro del AXI4 Lite en dirección de memoria y dato de escritura y canal para su envío a la memoria correspondiente.
- **FINISH**: cuando el bit de finalización se pone a 1 el estado de **CHECK_CHANNEL** deriva a este estado para finalizar la secuencia de escritura.

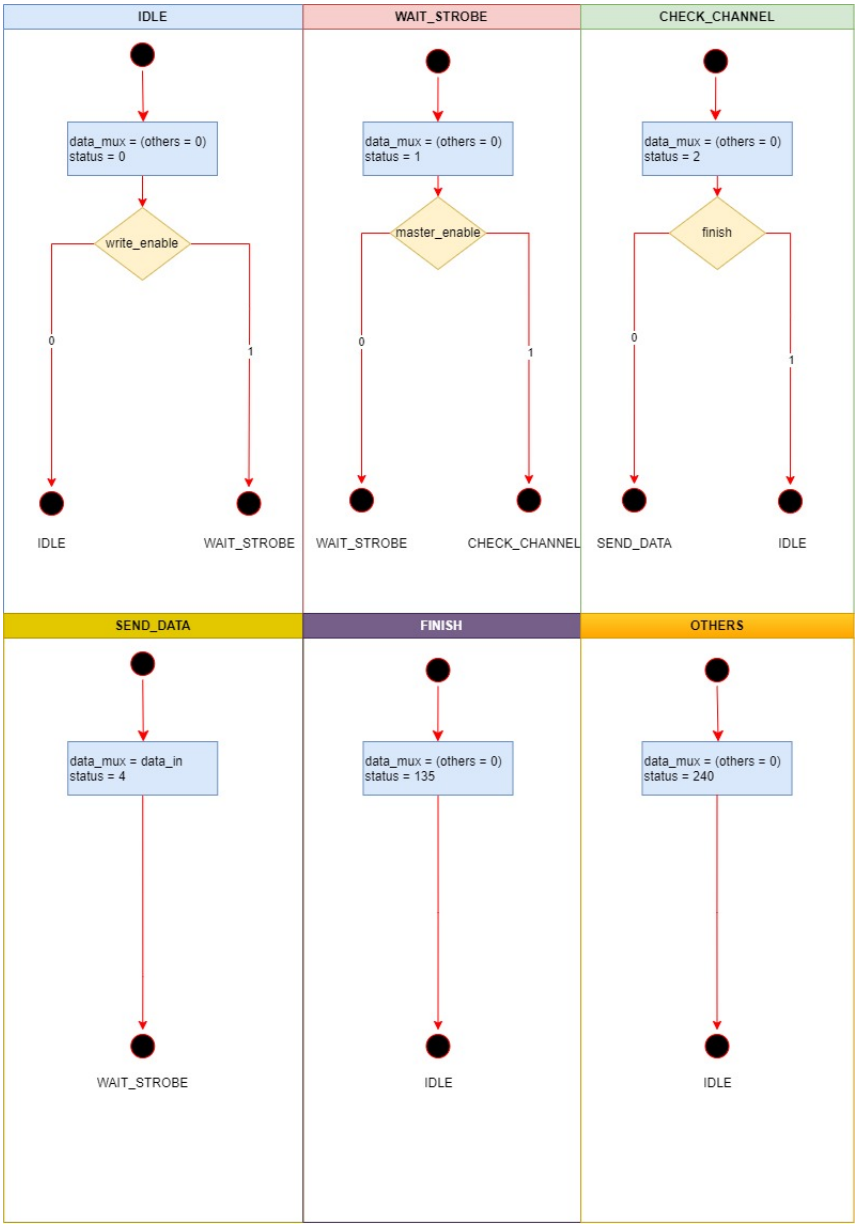


Figura 38: Diagrama de flujo de la máquina de estados de escritura

La información de dirección, dato y canal del estado **SEND_DATA** es enviada a un multiplexor para elegir que salida de las ocho disponibles debe de enviar la información. El protocolo a seguir para usar está máquina de estados es el siguiente:

- En el registro general de control con dirección 0xF8 del AXI4 Lite (ver anexo a) se configura el bit de escritura de la memoria a 1.
- En el registro de escritura con dirección 0xFC se escribe la información de dirección de memoria, dato a escribir y canal del AWG al que pertenece tal y como muestra el registro.
- Se realiza la operación anterior para escribir posición a posición en cada canal.

- Una vez finalizado la escritura del canal se configura el bit de paro (bit 4) del registro de escritura para dejar de cargar las muestras en las memorias del canal.

- Si es necesario se vuelve a empezar el proceso para un canal distinto.

3.2.10. FSM maestro/esclavo

El módulo puede contar con hasta 8 canales con sus paralelizaciones, y a menudo se querrá que dichos canales estén sincronizados entre si, para poder generar desfases con el ángulo lo más preciso posible. Para ello se pretende diseñar un módulo que sea capaz de gestionar la sincronización entre canales. Cada canal contará con una FSM de maestro esclavo, de tal manera que un canal actuará como maestro, mientras que los demás canales actuarán como esclavos. Independientemente si se es maestro o esclavo, todos tendrán las mismas señales de entrada: una señal de activación, una señal de identificación, y las dos señales de control que se usarán para habilitar: el disparo y la finalización de operación. En la figura 39 está representado el esquema de entradas y salidas del módulo. La fsm cuenta con 3 posibles estados:

- **IDLE**: por defecto se vuelve siempre a este estado a la espera que la señal master_enable se configure a 1. Esta señal está ubicada en el registro de control general (0xF8) bit 1 (ver en anexo a). Cuando se establece este bit, se consulta el bit de identificación de maestro esclavo de cada registro de control de canal, el bit 3 Master/Slave, con el que se decide que canal actuará como maestro y cuales como esclavo.

- **MASTER**: si la configuración del canal está dispuesta como master, entonces este recibirá directamente las señales de control de fsm_trigger y fsm_finish de los registros pertinentes y las derivará a los canales esclavos a través de sus correspondientes FSM.

- **SLAVE**: si por el contrario el canal está configurado como esclavo, entonces esperará las señales remitidas por el maestro para activar sus modos de operación.

La señal de fsm_trigger actúa como disparo para la máquina de estados principal, permitiendo lanzar de forma síncrona los tres modos de operación solo cuando un pulso de como mínimo un ciclo de reloj del sistema es usado en la señal fsm_trigger. La señal puede ser configurada como interna o como externa al AWG. Si es configurada como interna (bit 5 del registro de control general 0xF8) , entonces se escribe en el bit 0 del registro de control general. Este bit debe de ser borrado vía software para crear el pulso, y debe de tener al menos 12 ciclos del reloj del AXI4 Lite para poder funcionar correctamente. Si se configura en modo externo, entonces la señal de disparo viene del puerto external_sync, por el cual se espera un pulso de al menos un ciclo de reloj del sistema.

Figura 39: Esquema de entradas y salidas de la fsm maestro/esclavo

El diagrama de flujo de los estados de la FSM es el más sencillo de todas las demás FSM, tres estados. En la figura 40 se muestra su diagrama de flujo.

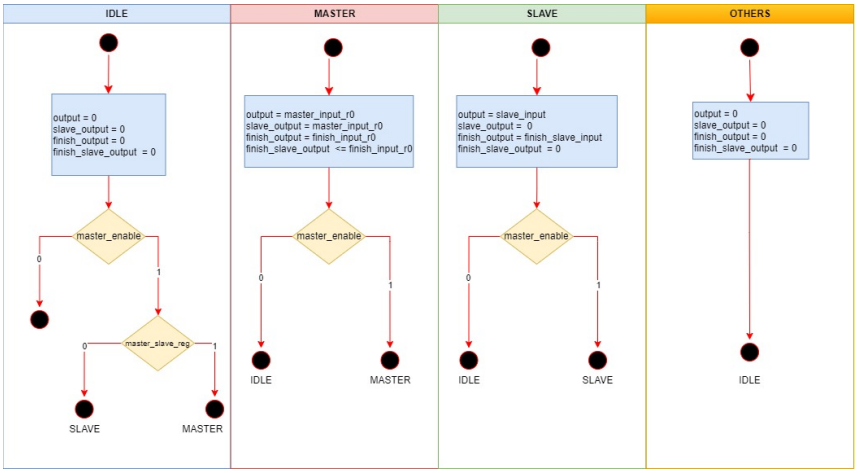


Figura 40: Diagrama de flujo de la fsm maestro/esclavo

3.2.11. Bus de configuración y control: AXI4 Lite

Para poder realizar una comunicación entre el microprocesador y el AWG se implementará un interfaz compatible AMBA AXI4, una serie de buses de comunicación y sus protocolos para la comunicación de IPcores con los microcontroladores de ARM. Existen tres tipos de buses AXI 4:

- AXI4: bus de comunicación para memorias mapeadas con alto rendimiento.
- AXI4 Lite: una versión del AXI4 más liviana en recursos, útil para mapear registros de control o de estado.
- AXIstream: bus especializado en transmisión de datos en modo stream a altas velocidades.

En el entorno de este proyecto, solamente será necesario el uso de registro de estado y de

control, por lo que el uso del bus AXI4 Lite será más que suficiente para la comunicación con el ARM.

La figura 41a muestra un esquema de como se realiza una lectura de datos desde el maestro al esclavo. Existe un canal de lectura donde se escribe la dirección de memoria a la que acceder, y después de unos ciclos se recibe el dato de esa posición. El protocolo para realizar una lectura en AXI4 Lite es el siguiente:

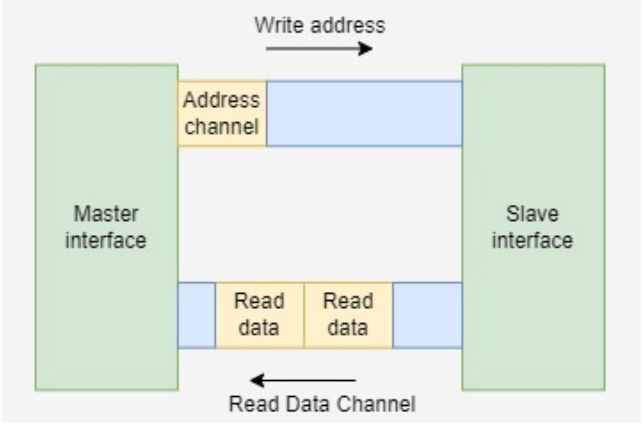
- El interfaz maestro escribe en el bus Read Address (normalmente llamado S_AXI_ARADDR) la dirección que se desea leer a la vez que levanta la señal de S_AXI_ARVALID indicando que hay una dirección válida en el bus y la señal S_AXI_RREADY indicando que el maestro está disponible para aceptar una respuesta.

- El interfaz esclavo levanta la señal S_AXI_ARREADY para indicar que puede admitir la dirección escrita.

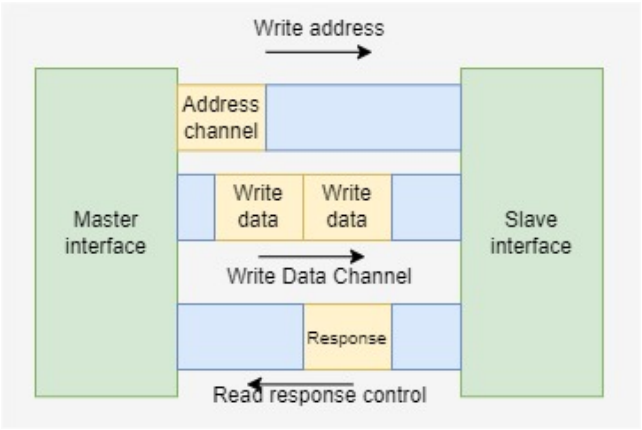
- Cuando las señales de S_AXI_ARVALID y S_AXI_ARREADY están levantadas, entonces en el ciclo de reloj siguiente se produce el handshake entre maestro y esclavo, momento en que tienen que bajar cada uno las señales S_AXI_ARVALID y S_AXI_ARREADY.

- Una vez que se han bajado, el esclavo coloca el dato leído de la dirección pedida en el bus READ DATA (S_AXI_RDATA) y levanta la señal de S_AXI_RVALID indicando que hay un dato válido en el bus. También envía la información de error en caso de que hubiera en el bus S_AXI_RRESP.

- Cuando la señal del maestro S_AXI_RREADY y la señal del esclavo S_AXI_RVALID están levantadas, el maestro lee del bus S_AXI_RDATA en el siguiente ciclo de reloj y se bajan las señales S_AXI_RREADY y S_AXI_RVALID para otra transacción.



(a) Esquema de lectura del bus AXI4 Lite



(b) Esquema de escritura del bus AXI4 Lite

Figura 41: Esquema de lectura (41a) y de escritura (41b del bus AXI4 Lite

Para la escritura de datos, la figura 41b nos muestra tres canales, uno para la dirección, otro para el dato a escribir y un tercero con la respuesta del esclavo. El protocolo a seguir para escribir un dato sería:

- El maestro escribe la dirección de escritura en el bus S_AXI_AWADDR y el dato en S_AXI_WDATA a la vez que levanta las señales de S_AXI_AWVALID indicando que hay una dirección válida y S_AXI_WVALID indicando que hay un dato válido. Adicionalmente se levanta la señal S_AXI_BREADY indicando que el maestro está a la espera de una respuesta del esclavo.
- El esclavo levanta las señales de S_AXI_AWREADY y S_AXI_WREADY indicando que está libre para admitir la dirección y el dato respectivamente.
- Cuando las dos señales de VALID y las dos señales de READY están levantadas se produce el handshake entre el maestro y el esclavo, procediendo a bajar las cuatro señales.
- El esclavo levanta la señal de S_AXI_BVALID indicando el estado de la escritura, normalmente un 2b00 para el ok y un 2b11 para indicar un error.

En total el AWG contará con 64 registros entre control y estado de cada canal. En el anexo a está una tabla con todos los registros y a continuación el detalle de cada tipo de registro. Se van a implementar cuatro tipos de bloques:

- **Registro de identificación**: son dos registros donde está la información de la versión del módulo y el identificador del IPCore asignado por DAS Photonics para la integración en los proyectos.

- **Registros de control de canal**: cada canal dispondrá de un control independiente que consistirá en 5 registros:

  - Un registro de control de canal donde se especificará el modo de operación o la señal a leer.
  - Un registro para el valor del paso del acumulador.
  - Un registro con el valor del desfase de la señal.
  - Un registro con el valor del PRI.
  - Un registro con el valor del PW.

- **Registros de estado de las fsm**: cada canal contará con un registro para el estado de su fsm principal, y un registro adicional para el estado de la fms de escritura.

- **Registros de control general**: son dos registros para el control las configuraciones que afectan a todos los canales:

  - Registro de control general, donde se configura el tamaño del ruido, se habilita la fuente de la señal de disparo, se para el AWG o se genera el disparo por software.
  - Registro de carga de memoria, donde se colocan los datos para escribir en las memorias del AWG como se mencionó en el apartado 3.2.9.

El módulo que realiza la interfaz entre el AXI4 Lite y el AWG trabaja con un reloj más lento que el los módulos integrantes del AWG, usa el dominio del AXI4 Lite a 125 MHz. Es por ello que es necesario realizar cambios de dominio de reloj entre la interfaz y el AWG como tal. Existen dos tipos de cambio de dominio, el primero concierne a las señales de control, que deben de cambiar entre el dominio de 125 MHz y los 312,5MHz, es decir de un reloj lento a otro más rápido. El segundo corresponde a las señales de estado que provienen del AWG hacia los registros mapeados del AXI4 Lite, realizando el cambio inverso, es decir, del reloj rápido al lento. Para realizar ambos cambios de dominio, se usará un IPCore desarrollado por DAS Photonics para tal propósito. Se trata del cross clock domain, un módulo que se basa en el uso de la macro de Xilinx xpm_cdc_handshake, primitiva que nos permite realizar cambios de dominio de señales de más de un bit. El módulo del cross clock domain añade lógica adicional para instanciar la cantidad de xpm_cdc_handshake necesarios tanto para los cambios de registros de estado como de registros de control.

3.3. Reset

Algunos módulos del sistema necesitan un mecanismo de reset para poder asegurar que en el momento de inicialización las cadenas de control y la cadena de datos empiece en un estado conocido y controlado. Por ello se implementará un módulo de reset que actue de forma que se lleve a dichos módulos a una posición conocida mientras dure el evento de reset.
Existen dos dominios de reloj en el sistema como ya se ha explicado, el dominio del reloj del sistema que actua sobre la cadena de datos y la cadena de control, y el dominio del bus AXI4 Lite que actúa sobre los registros mapeados y la fsm de escritura a memoria. Cada dominio necesita de un reset emparejado con su reloj, por lo que el módulo de reset se instanciará de la siguiente forma:

- Un reset global en el dominio del AXI4 Lite, que proveerá de la señal al módulo de la fsm de escritura a memoria.
- Un reset en cada instanciación de canal del AWG para proveer de la señal de reset de forma individual a cada canal. De esta forma reducimos el fan out de la señal de reset.

El módulo de reset tiene como entrada el reloj del dominio donde trabaje, y como salida la señal de reseteo. Se activa al recibir el primer flanco ascendente de reloj, y mantiene la señal de reset durante una cantidad de ciclos especificada mediante un generic MAX_CYCLES. La señal de reset es activa a nivel bajo, todos los módulos que admiten un reset en el AWG son activos a nivel bajo dado que la FPGA donde se usará el sistema recomienda que los resets sean activos a nivel bajo. Una vez han pasado los ciclos configurados, el módulo cambia la salida a estado alto. Básicamente el módulo de reset es un contador que cuando llega al valor de MAX_CYCLES deja de contar. En la figura 42 se muestra el esquema de entrada y salida.

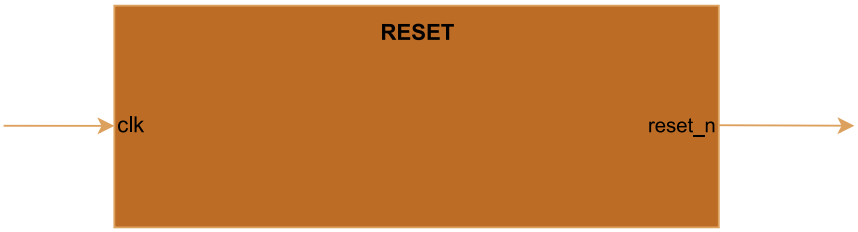


Figura 42: Esquema de entrada y salida del reset

3.4. Control mediante Python

Los proyectos para los que actualmente se usa el AWG se apoyan en tarjetas de procesado, las cuales incluyen las FPGA y un microprocesador ARM. Se usa un sistema operativo basado en Linux desde el que se puede acceder a los registros de los módulos de FPGA a través del microprocesador usando el bus AXI4 Lite. El proyecto del AWG se concibió para ser independiente del sistema completo, pudiendo actuar con el hardware básico de las FPGA, por lo que en un principio no es necesario crear un driver en linux para controlarlo. Para poder llevar a cabo las tareas de configuración y control del AWG se desarrolló una serie de archivos en Python. Constan de cuatro archivos, awg_class.py, colors.py, control_awg.py y launch_awg.py, archivos que se encuentran

en el anexo b. El primer archivo, awg_class.py, contiene las funciones básicas del AWG, la generación de la señal a guardar en las memorias, la carga de dicha señal, la función de configuración de un canal y la configuración general. El segundo de los archivos, bcolors.py solo contiene los códigos de colores para una mejor visualización de los mensajes en la terminal de linux. El tercer archivo, control_awg.py, crea los archivos .bash que se enviarán a las tarjetas que contendrán las instrucciones para la configuración y control del AWG. Y por último launch_awg.py crea una interfaz entre el usuario y el AWG basada en la ejecución del archivo con el uso de argumentos, como la frecuencia, los desfases, las direcciones IP etc.

Capítulo 4

Simulaciones

Como se ha descrito en secciones anteriores, el AWG cuenta con tres modos distintos de funcionamiento: modo continuo, modo pulso único y modo pulsos continuos. Cada modo cuenta con una configuración distinta, usando distintas señales para controlar el sistema, por lo que las simulaciones de los módulos se van a dividir en tres grandes bloques, uno para cada modo. Las simulaciones individuales de los módulos se realizará usando un banco de pruebas en vhdl con el entorno de simulación predeterminado del IDE Vivado, el Isim. Los resultados de la integración de todo el sistema y su prueba en un ambiente real se presentará en el capítulo de 5 donde se explicará la metodología. El banco de pruebas usado en VHDL será configurable para poder realizar los cambios necesarios para cada escenario, como por ejemplo cambio de modo de trabajo, cambio de frecuencia de salida, etc. Adicionalmente se capturará la salida del AWG para hacer un estudio rápido de las propiedades de las señales de salida, como el SNR, el SFDR o la frecuencia de salida mediante un programa en python que recoja las muestras y las procese. Cada escenario de modo de trabajo se aplicará con tres valores distintos de paso del acumulador, el valor mínimo, el máximo y uno intermedio para verificar los casos extremos y uno intermedio. La funcionalidad de la adicción de ruido de amplitud se estudiará usando el modo continuo con diferentes valores de nivel de ruido en un apartado final. El módulo se configurará del siguiente modo:

- Se instanciarán 2 módulos de AWG.
- Cada módulo contará con una paralelización de 4.
- La señal de salida estará truncada a 12 bits.
- Se cargará una señal senoidal en la memoria de cada canal del AWG.
- La frecuencia del sistema se fijará a 312,5 MHz mientras que la frecuencia del bus AXI4 Lite se configurará en 125 MHz.

4.1. Simulaciones en modo continuo

La simulación del modo continuo es la más sencilla de todas, dado que implica la menor cantidad de módulos trabajando, puesto que no necesita la información del detector de periodos.

4.1.1. Simulación en modo continuo del acumulador de fase

El valor de paso mínimo para el AWG que permite una salida con el menor ruido posible según se ha explicado en el apartado 2.4.3 equivale al valor en hexadecimal 0x00400000 produciendo una frecuencia de salida de 1.2207 MHz teóricos. El valor máximo sería entonces 0x80000000, produciendo un valor de 625 MHz teóricos, cualquier valor intermedio producirá un valor de frecuencia de salida dentro del intervalo (1.2207 MHz ; 625 MHz). La figura 43 corresponde con los primeros ciclos de funcionamiento del acumulador con paso mínimo. Se puede observar como el valor de la variable phase_k, correspondiente al paso del acumulador, tiene el valor de 0x00400000. Este valor es multiplicado por el número de paralelizaciones, en este caso 4, obteniendo el valor de mult_k, el cual se suma cada ciclo al valor del acumulador, cuya salida es acc_reg_out.

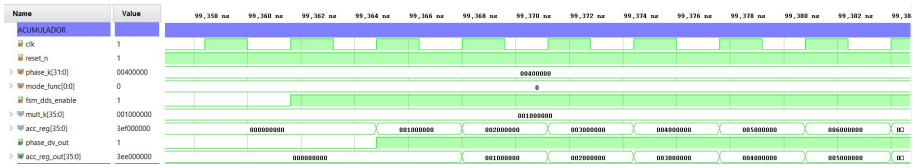


Figura 43: Simulación del acumulador de fase en modo continuo y paso mínimo

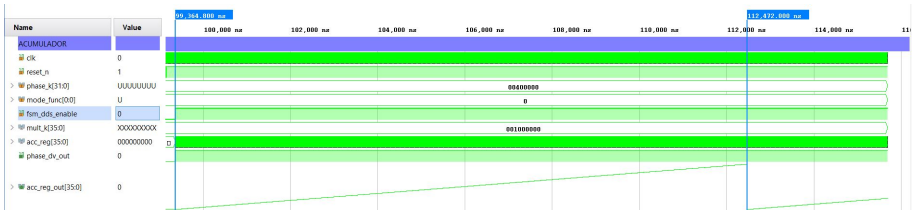


Figura 44: Vista completa de un periodo del acumulador con paso mínimo

En la figura 44 se nos muestra un periodo completo del acumulador de fase con el paso mínimo. En ella podemos ver que tarda un total de 13,1072 ns en completar un periodo, lo que equivale a 4096 ciclos de reloj. Este resultado se puede comprobar haciendo uso de la ecuación 5 con un valor de A=36 y de T=16.777.216 (el valor del paso por la cantidad de paralelizaciones).
Con el caso opuesto, es decir con el paso en su valor máximo, obtenemos la simulación presentada en la figura 45. Como con el paso anterior, en la figura 46 se ha capturado un periodo completo, siendo este de 8 ciclos. Igualmente se puede comprobar como en el caso anterior usando ahora T=8.589.934.592.

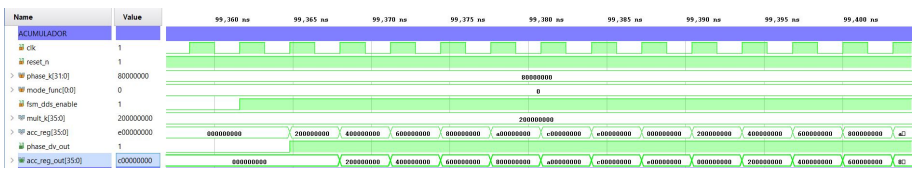


Figura 45: Simulación del acumulador de fase en modo continuo y paso máximo

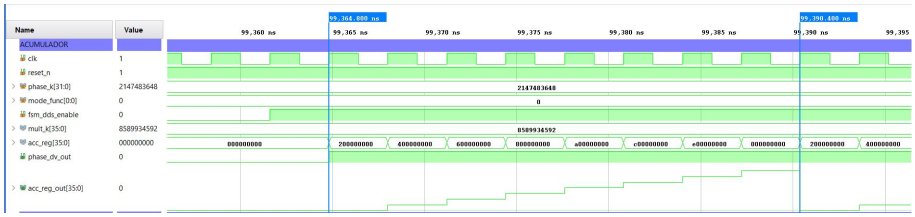


Figura 46: Vista completa de un periodo del acumulador con paso máximo

4.1.2. Simulación en modo continuo del generador de dithering

El módulo de dithering es independiente del modo y del valor de cualquier variable que el operador pueda gestionar una vez el módulo esté implementado.

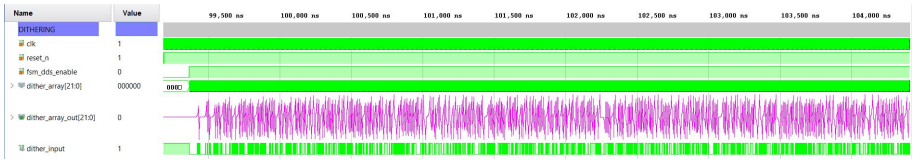


Figura 47: Ruido del módulo de dithering

En la figura 47 se observa la salida del módulo de dithering, donde la señal morada es la salida de 21 bits del ruido generado, el cual se sumará al valor de la salida del acumulador antes de ser truncado.

4.1.3. Simulación en modo continuo del generador de direcciones

La simulación del generador de direcciones, figura 48 verifica que la latencia del módulo es de tres ciclos de reloj del sistema y el funcionamiento de las operaciones descritas en la ecuación 14, las cuales están divididas en las siguientes:

- phase_val : multiplicación del valor de la paralelización por el valor de paso del acumulador.
- operation1: suma del valor de phase_val con la salida del acumulador (acc_reg_in).
- operation2: suma del valor de desfase y del ruido de dithering.
- addr_aux: suma de las dos operaciones anteriores.
- addr_ram: truncado de addr_aux para adaptarlo a las direcciones de memoria (de 36 a 10 bits).

Las cuatro señales en color morado de la figura 48 son las cuatro salidas de cada uno de los generadores de direcciones instanciados. Para el valor de paso de 1 se puede comprobar que cada ciclo se generan cuatro direcciones con salto de 1 entre cada salida.

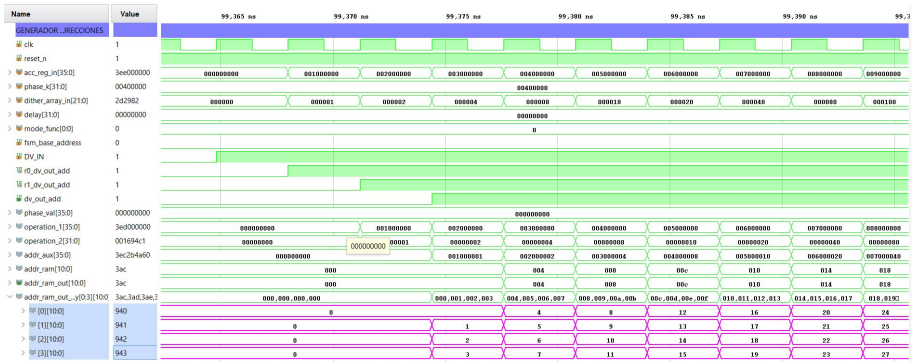


Figura 48: Simulación del generador de direcciones en modo continuo

4.1.4. Simulación en modo continuo del generador de las memorias

Al igual que ocurre con el módulo de dithering, el funcionamiento de las memorias no dependen del modo de operación del AWG, solo tienen dos operaciones, la escritura de las muestras y su lectura, ambas operaciones en dominios de reloj distintos. La escritura de memoria se realiza con el reloj más lento, el reloj usado para el bus AXI4 Lite. Este proceso es el más lento y ocurre en ocasiones contadas, al principio del arranque del sistema para guardar las muestras iniciales, o cuando el operador decide cambiar el escenario modificando la señal a generar. En la figura 49 se ha simulado una escritura de 1024 muestras en una de las memorias. Con el reloj del AXI4 Lite a 125 MHz, se tarda 24,576 μs en completar una carga completa de las 1024 muestras.

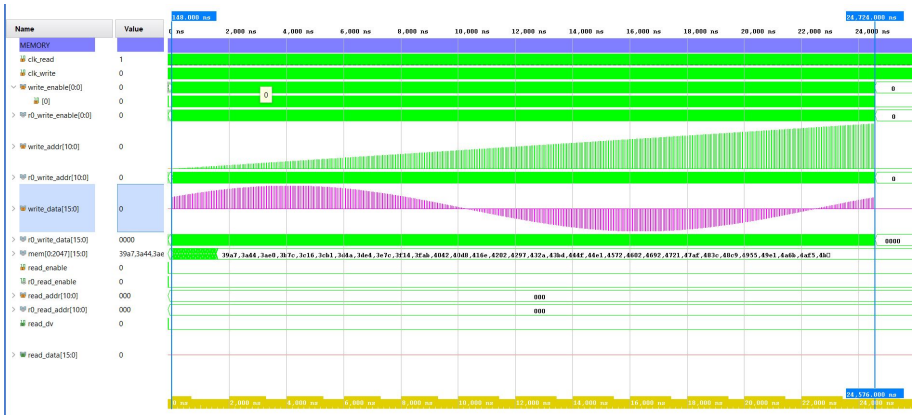


Figura 49: Simulación de la escritura de la memoria

La figura 50 muestra un detalle de la escritura de la memoria. Se puede comprobar como se tardan 3 ciclos de reloj entre dato y dato de escritura, debido a la latencia del protocolo AXI4 Lite implementado. Se puede comprobar entonces que si se escriben 1024 muestras cada 3 ciclos de reloj de 125 MHz, entonces el tiempo total de la operación de escritura es de $1024\ muestras * 3\ ciclos * 8ns = 24,576\mu s$.

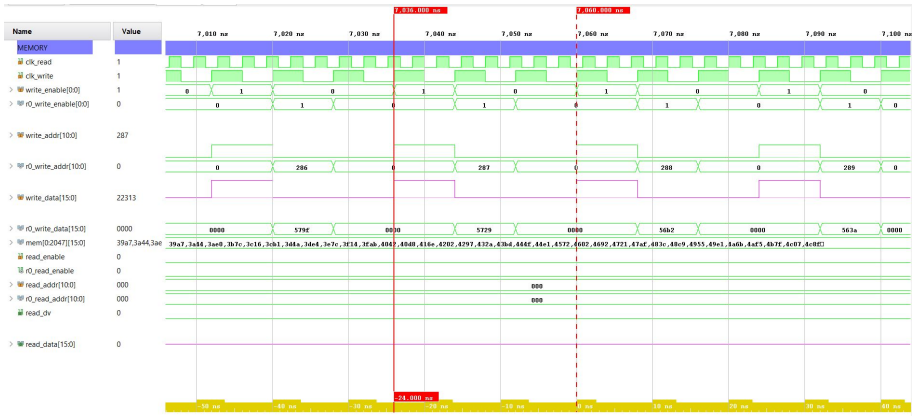


Figura 50: Muestra de escritura de la memoria

En la figura 49 se ve como la señal almacenada es una senoidal con la particularidad de que la fase inicial no es 0, tiene un ligero desfase de 30º, esto se a la simetria de la señal. Si se empezará con un desfase de 0º, entonces existirián unos valores del paso del acumulador que generarían siempre una señal con dos valores nulos, por ejemplo si el paso es 512, entonces el primer valor de salida seria de 0 y para el paso en 512 volvería a ser 0, empezando de nuevo la serie. Si desplazamos la señal senoidal para que la fase inicial no sea 0º, entonces nos aseguramos que la señal no es simétrica con respecto a la mitad de las muestras, produciendo así una asimetría que evita la aparición de muestras con el mismo valor para algún parámetro de paso del acumulador.
La lectura se realiza cada vez que una dirección válida es aplicada a la memoria, junto a la señal de read_enable. Como se muestra en la figura 51, se genera una señal senoidal, de la que podemos extraer la información de frecuencia, la cual viene dada por la diferencia entre los dos puntos marcados en azul. La diferencia en tiempo es de 0.8192 μs, que pasado a frecuencia es 1.2207MHz, la frecuencia de salida que se obtiene de 12, usando los parámetros de N=4, K=1, f_{clock}=312.5MHz y N=10.

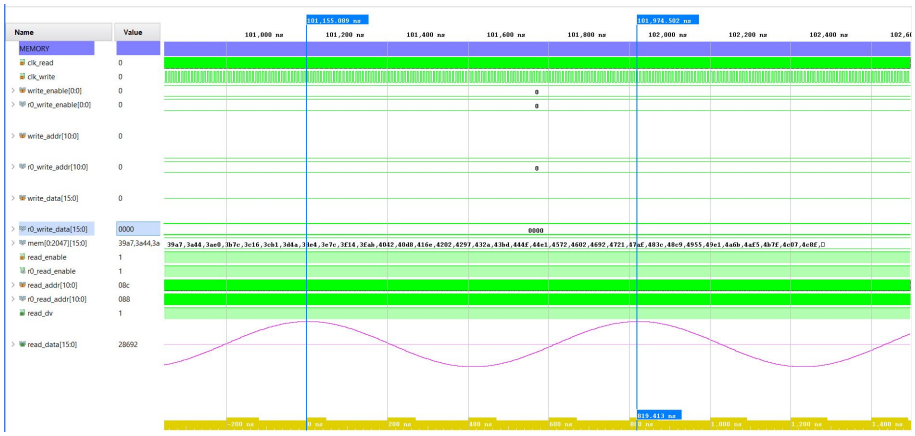


Figura 51: Lectura de la memoria

Por último, la figura 52 nos muestra la operación de la memoria completa, tanto la escritura como la lectura.

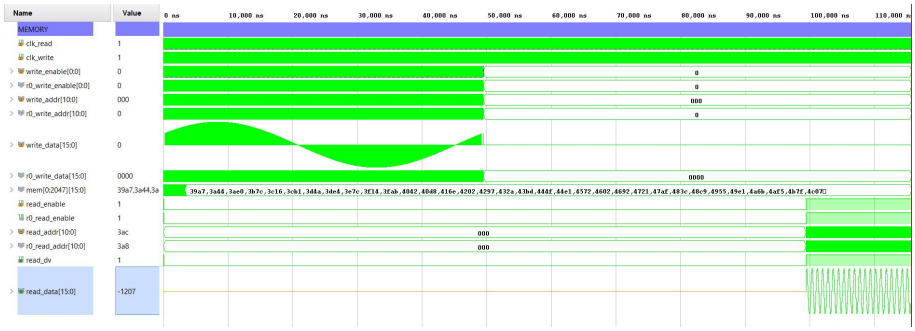


Figura 52: Operación completa de lectura y escritura de la memoria

4.1.5. Simulación en modo continuo de la FSM principal

La máquina de estado principal en modo continuo no es muy compleja, estando en el estado inicial de IDLE, espera a que se active la señal de habilitación de la fsm principal y el modo de operación, momento en el que empieza a generar la señal de salida con las características configuradas en cada canal. Esta secuencia se puede ver en la figura 53 junto a las señales de control que genera para los distintos módulos.

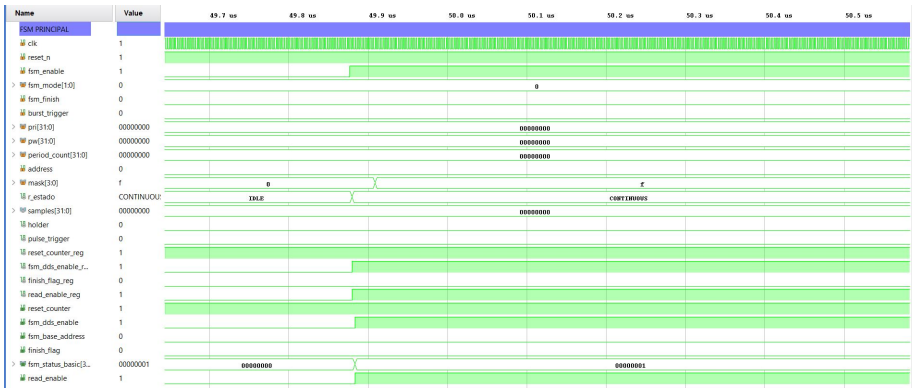


Figura 53: FSM principal en modo continuo

4.1.6. Simulación en modo continuo del módulo de salida

El módulo de salida no hace uso de todo su potencial en el modo continuo, dado que ignora la información de las señales de control que atañen a los dos modos de pulsos.

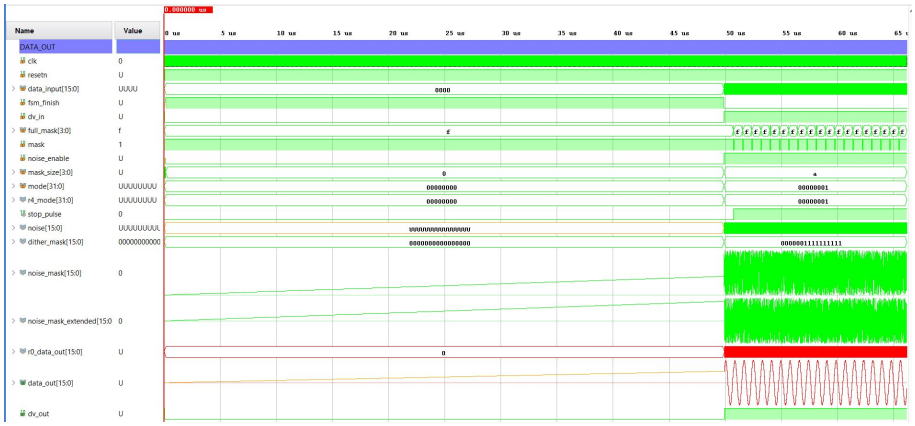


Figura 54: Simulación del módulo de salida para el modo continuo

En la figura 54 se muestra la simulación del módulo de salida. En ella podemos ver como la señal de salida empieza en el momento en el que el modo de la fsm principal vale 1, es decir el modo continuo. Además esta activado el generador de ruido gaussiano, con una máscara de 10 bits, es decir que se le añadirán los 10 bits de menor peso del ruido a la señal de salida de 16 bits (antes de truncar a 12 bits). Como detalle, aún cuando no se use, la información del módulo de máscara de datos si es visible en la simulación, cambiando el valor cuando se produce un periodo completo.

4.1.7. Simulación en modo continuo de la FSM de escritura de memoria

La máquina de estados de escritura se usa para cargar las muestras en todas las memorias de cada canal desde la información que se recibe del registro de control de escritura (0xFC). En la figura 55 se puede ver una visión completa del funcionamiento del módulo. Se puede observar como existen 8 salidas, una por cada posible canal a instanciar, pero solo se usan dos de ellas, la 0 y la 1 correspondientes a la configuración de solo 2 canales. La figura 56 nos muestra una transacción de escritura, donde se parte del estado WAIT STROBE a la espera de un dato válido en el registro de escritura. Cuando se produce un flanco ascendente de la señal, entonces se pasa al estado CHECK CHANNEL, el cual comprueba si existe orden de paro, y como no existe se procede a enviar la información de canal, dirección y dato en el estado SEND DATA, momento que se vuelve a esperar la señal de strobe en WAIT STROBE.

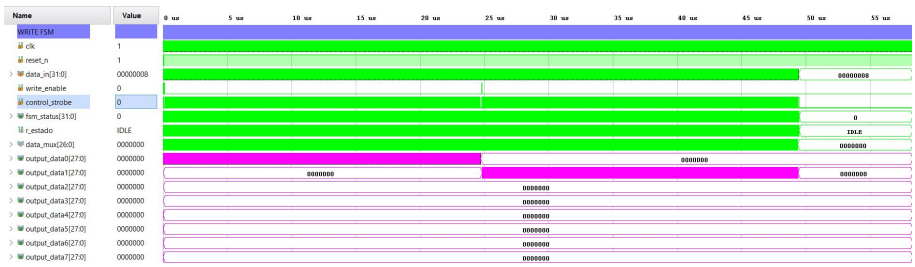


Figura 55: Simulación de la máquina de estados de escritura

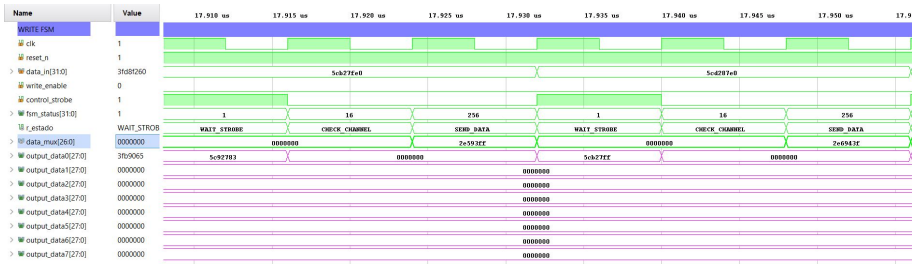
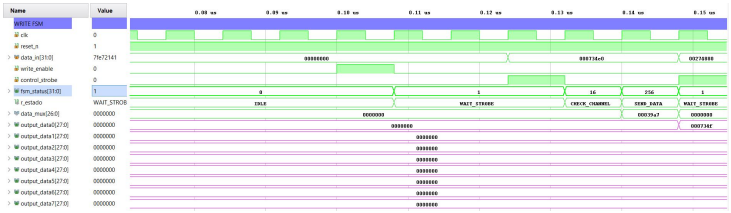
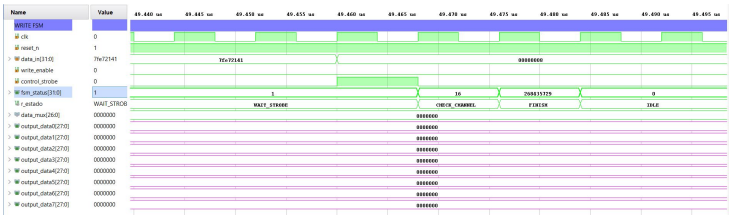


Figura 56: Vista de los estados de la fsm de escritura

La figura 57 nos muestra el estado inicial de la fsm de escritura, que parte de IDLE a la espera de que se configure el registro de write enable del registro general de control (0xF8), momento que empieza la secuencia de estados WAIT STROBE-CHECK CHANNEL-SEND DATA hasta que se escribe la señal de paro, tal y como se ve en la figura 57b.



(a) Primeros estados de la fsm de escritura



(b) Finalización de la escritura

Figura 57: Vista del inicio de la escritura 57a y finalización 57b

4.1.8. Simulación en modo continuo de la FSM maestro/esclavo

Este módulo también es independiente de cualquier configuración de modo del AWG.

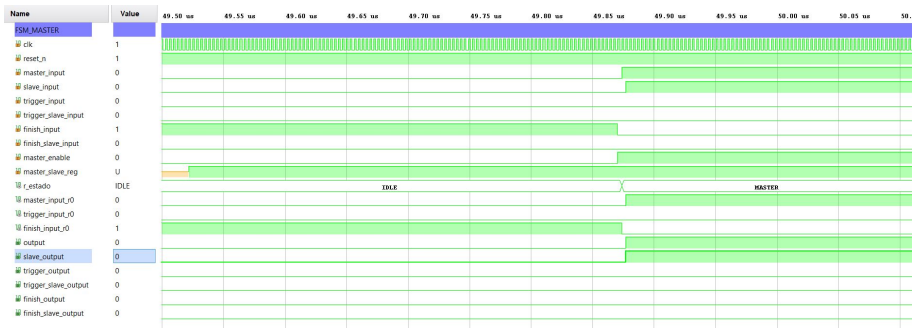


Figura 58: Simulación de la fsm maestro

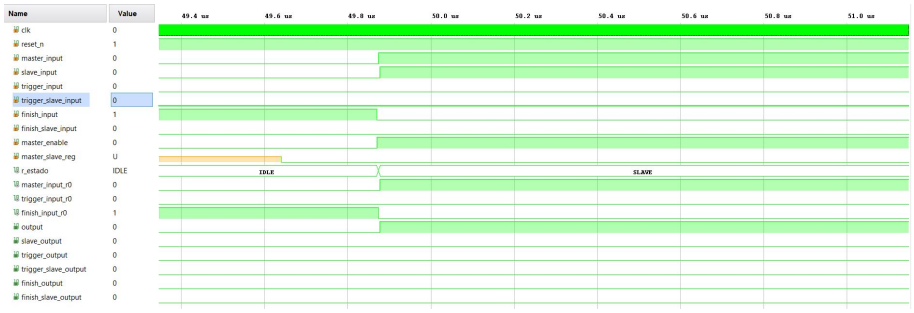


Figura 59: Simulación de la fsm esclava

4.1.9. Simulación en modo continuo del interfaz AXI4 Lite

Tal y como se explica en el apartado 3.2.11, el módulo del AXI4 Lite sirve como interfaz entre el bus de comunicaciones y el AWG.

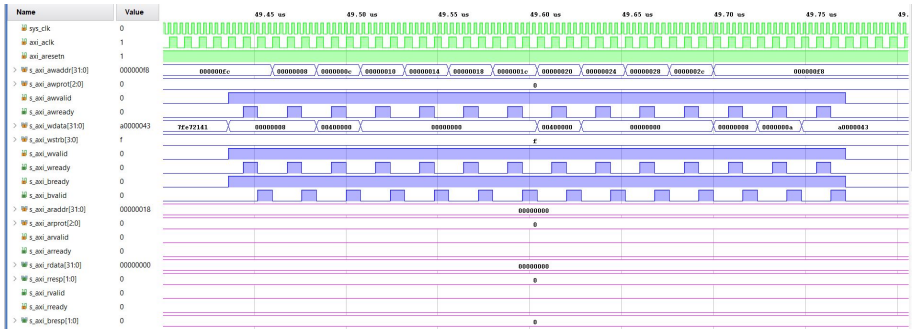


Figura 60: Simulación de escritura en AXI4 Lite

En la figura 60 se muestra la simulación de las operaciones de escritura con el interfaz para el AXI4 Lite. En azul están representadas las señales para la escritura, en este caso escribiendo la configuración de los 5 registros por canal implementado. La figura 61 muestra en color rosado las señales de lectura, realizando la operación para leer el estado de los registros de control del canal 0. Se puede comprobar que las secuencias de lectura y escritura siguen el protocolo establecido en el apartado 3.2.11. En la simulación se puede observar que la latencia tanto de escritura como lectura es de tres ciclos del reloj AXI, necesitando un total de 120 μs para configurar un canal completo.

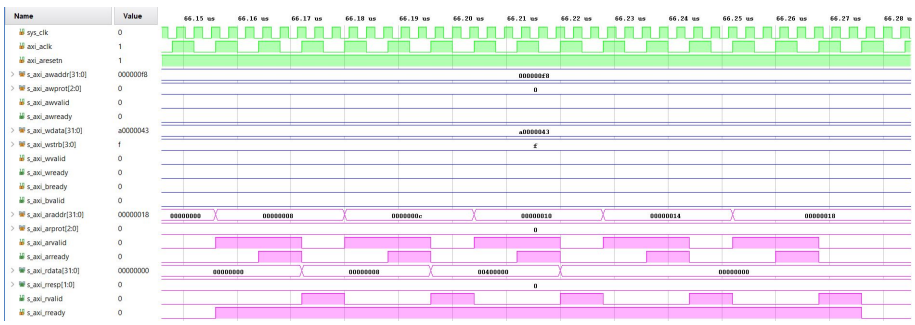


Figura 61: Simulación de escritura en AXI4 Lite

4.1.10. Simulación del módulo de reset

La señal de reset se aplica en dos dominios diferentes, el del sistema y el del AXI4 Lite. El único parámetro configurable es la cantidad de ciclos de reloj que la señal de reset está activa, y solo cuenta como entrada la señal del reloj. La figura 62 muestra la simulación del módulo con un ciclo de estado activo, mientras que la figura 63 muestra el reset configurado con 15 ciclos de estado activo. Por defecto la salida del módulo se establece como activa, aún antes de que llegue el primer flanco de reloj.

Figura 62: Simulación del módulo de reset con 1 ciclo

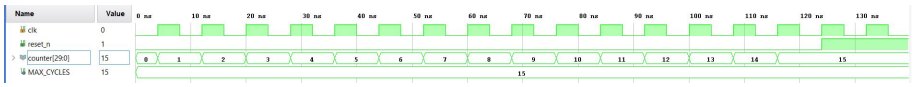


Figura 63: Simulación del módulo de reset con 15 ciclos

4.2. Simulaciones en modo pulso único

El modo de pulso único genera un solo tren de pulsos basado en el número de periodos de la frecuencia deseada. La cantidad de periodos completos se escribe en el registro de configuración del PW de cada canal, permitiendo un valor distinto de burst en cada canal. La mayoría de módulos se comportan similar que en las simulaciones vistas en el modo continuo. Para simular el modo pulso único se va a proceder con una configuración de paso del acumulador de 1 y una cantidad de tiempo de PW de 4096 ns. El rango posible para configurar el PW es de $[1,2^{32}]$ ciclos de reloj. Si usamos la simulación con los parámetros descritos, se puede ver como hay módulos que no trabajan todo el tiempo como en el modo continuo. Por ejemplo, el acumulador de fase ahora solo trabajará el tiempo que el tren de pulsos esté activo, como se ve en la figura 64. El generador de direcciones y la memoria se comportan igual, solo durante el tiempo del tren de pulsos estarán con el data valid activo, como se puede ver en la figura 65. Se puede apreciar como el tamaño del intervalo de trabajo es ligeramente superior, de 4121.6 ns (es decir 25.6 ns, que traducido a ciclos significa que hay un exceso de 8 ciclos de reloj). Esto se debe a la latencia del datapath más la latencia propia de las señales de control de la máquina de estados principal, 5 ciclos provienen del datapath, 2 ciclos de registrar las entradas y salidas de la propia máquina de estados y un ciclo extra del cambio entre estados de la fsm principal.

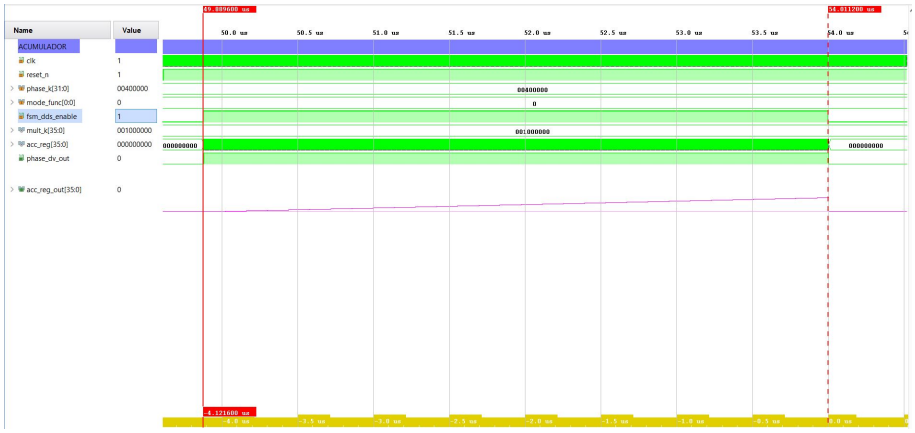


Figura 64: Simulación del acumulador en modo pulso único

Este efecto de alargamiento del data valid debido a las latencias se soluciona en el módulo de salida mediante el uso de señales de control de la máquina de estados principal. En la figura 66 se puede comprobar que el intervalo donde el data valid de salida, y por consiguiente la señal de salida, están activos es de 4096 ns, el tiempo esperado. También se aprecian los 5 ciclos de la señal de salida y como cuando acaba el tren de pulsos, la salida solo tiene ruido del generador gaussiano. También se puede observar como la señal de full_mask marca los finales de cada periodo.

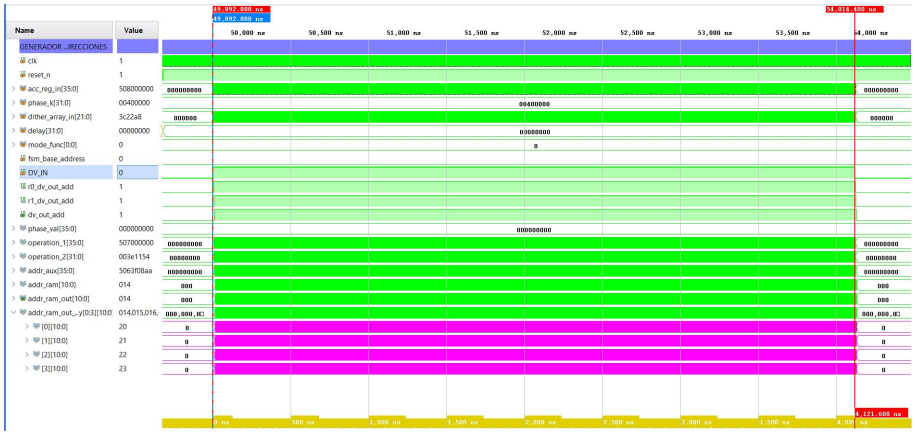


Figura 65: Simulación del generador de direcciones en modo pulso único

Figura 66: Simulación del módulo de salida en modo pulso único

La máquina de estados principal parte del estado inicial de IDLE hasta que el pulso de la señal de disparo es lanzado, produciendo el cambio al estado de BURST_WAIT para pasar al estado de PULSE y empezar a generar la señal (figura 67. En el momento en el que el contador de muestras, la máquina de estados vuelve al estado de BURST_WAIT a la espera del siguiente disparo.

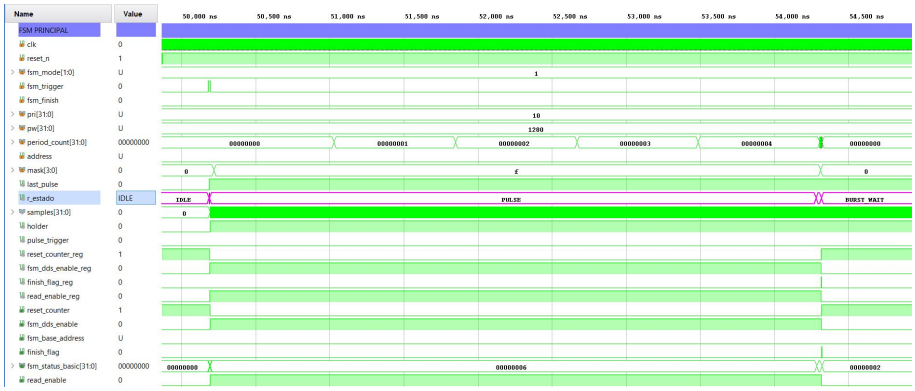


Figura 67: Simulación de la fsm principal en modo pulso único

Como ya se ha visto, en este modo de operación es necesario el uso del detector de periodos. La figura 68 muestra una visión de los dos módulos, donde se usa directamente la matriz de direcciones de memoria para calcular un periodo completo.

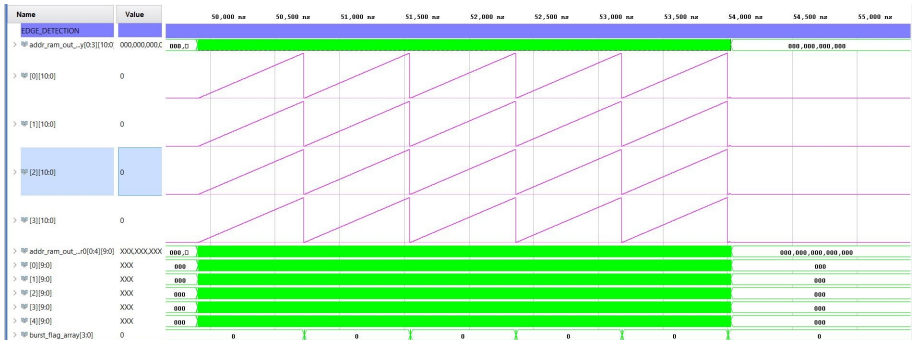


Figura 68: Simulación del detector de periodos en modo pulso único

Con un valor de paso de 1, ocurre un solo periodo cada 256 ciclos, en la figura 69 se puede observar la finalización del periodo en la matriz de direcciones, cuando en este caso se pasa por 0.

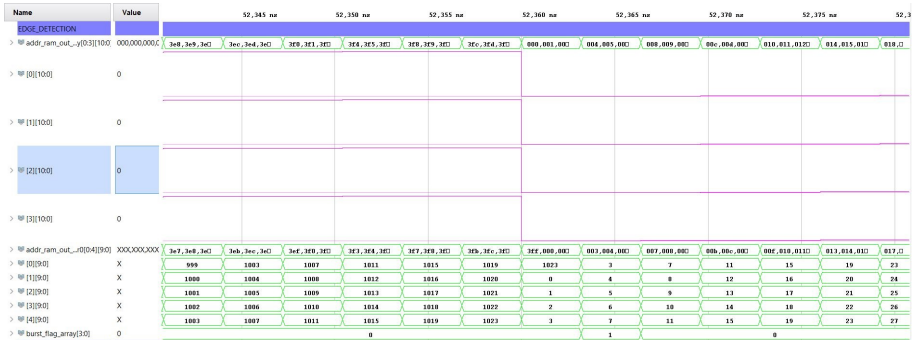


Figura 69: Detalle del detector de periodos en modo pulso único

Si cambiáramos el valor del paso para obtener una frecuencia distinta, por ejemplo un paso de 401, la finalización de los periodos no siempre ocurrirán como en el ejemplo anterior. Por ejemplo en la figura 70 se puede ver la simulación del detector de periodos para el paso de 401. El vector de burst_array_flag varia su valor, indicando que la finalización del periodo cambia cada ciclo, información que usará el módulo de datos de salida para recortar la señal de salida si fuera necesario.

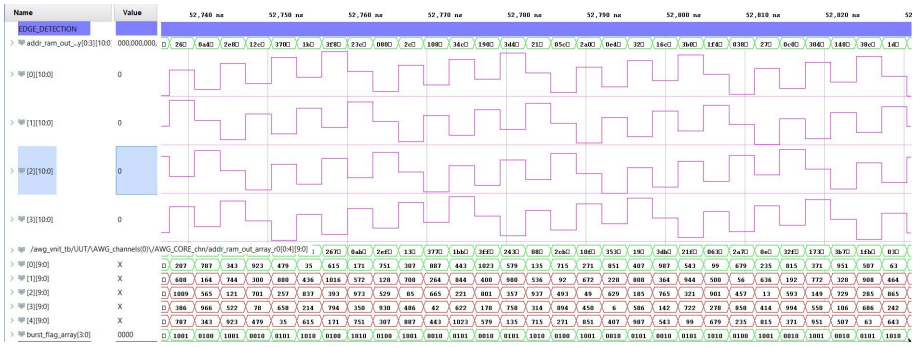


Figura 70: Detalle del detector de periodos en modo pulso único para un paso de 401

4.3. Simulaciones en modo pulsos continuos

Este modo de operación es similar al modo de pulsos únicos, con la siguiente excepción, en el modo de pulso único solo hacía falta especificar el valor de PW dado que solo hay un pulso, en el modo de pulso continuo hay que especificar además el valor de PRI. Para ver en detalle el modo de funcionamiento, se va a configurar el paso con un valor de 510, el PW = 5 y el PRI = 10. En la figura 71 se puede ver la simulación de la máquina de estados principal operando en modo pulsos continuos. Se aprecia que el estado de la fsm empieza en IDLE para pasar al estado de FIRST_TRIGGER en cuanto la señal de disparo es levantada. Después pasa al estado de PULSE generando el tren de pulsos hasta que pasan los 5 ciclos impuestos por el PW, para pasar al estado de TRIGGER_WAIT a la espera de la finalización del contador de PRI, momento en que empieza de nuevo el ciclo desde el estado PULSE.

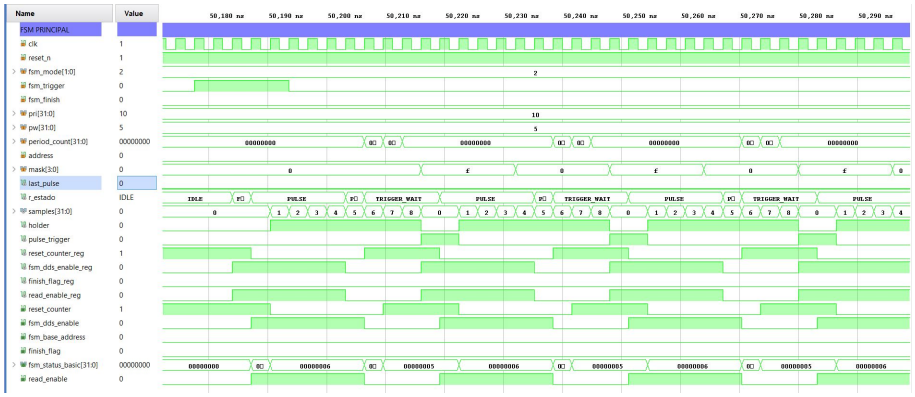


Figura 71: Simulación de la fsm principal en modo pulsos

La salida del módulo se puede observar en la figura 72, donde se aprecia el valor de PW, 16 ns que corresponde a 5 ciclos de reloj, y el valor de PRI de 32 ns, correspondiente a los 10 configurados.

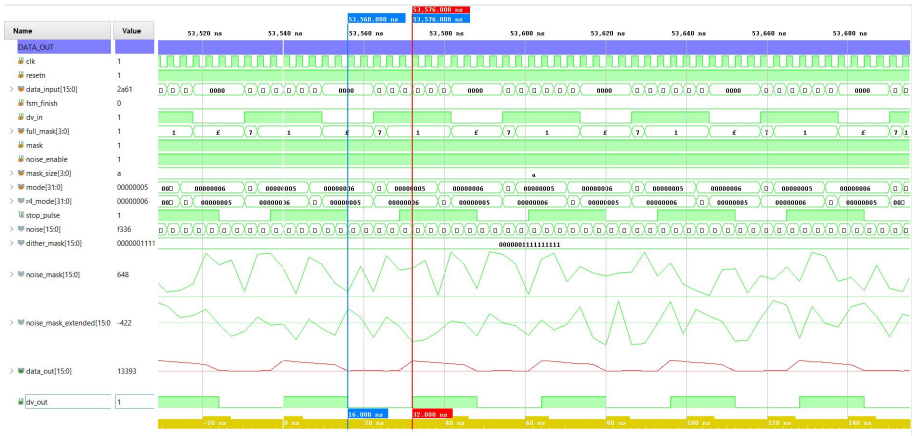


Figura 72: Simulación del módulo de salida en modo pulsos

4.4. Adición de ruido gaussiano

En secciones anteriores se ha visto que el módulo de salida de datos dispone de la opción de añadir ruido gaussiano mediante la configuración de los registros noise size y noise enable del registro de control general.
Para la configuración de 12 bits de salida, se han realizado las medidas con 4,5,6,7,8 y 9 bits de ruido.

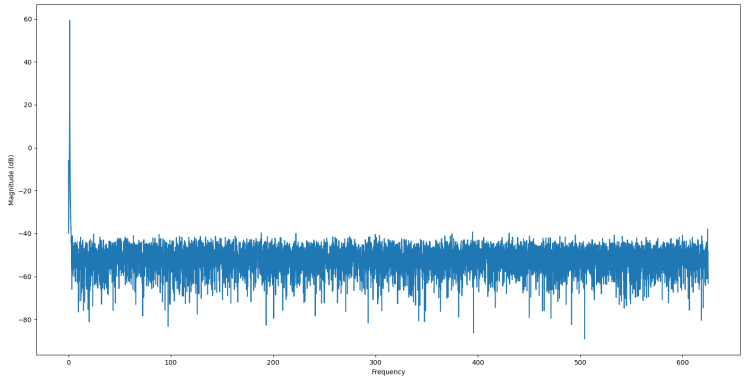


Figura 73: Salida del awg con 4 bits de ruido añadido

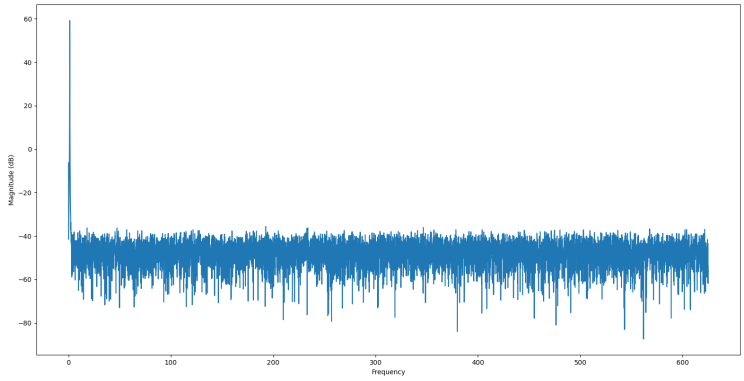


Figura 74: Salida del awg con 5 bits de ruido añadido

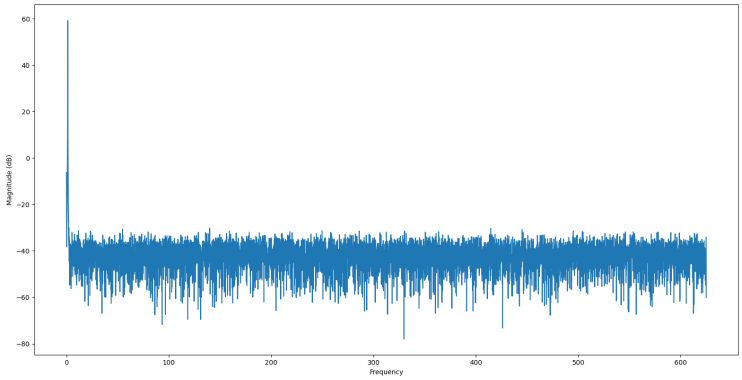


Figura 75: Salida del awg con 6 bits de ruido añadido

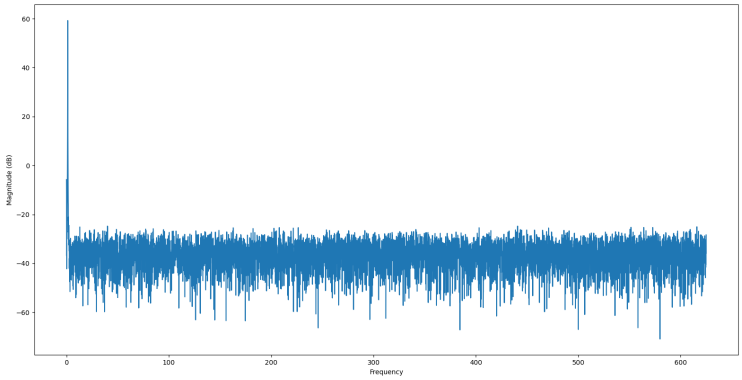


Figura 76: Salida del awg con 7 bits de ruido añadido

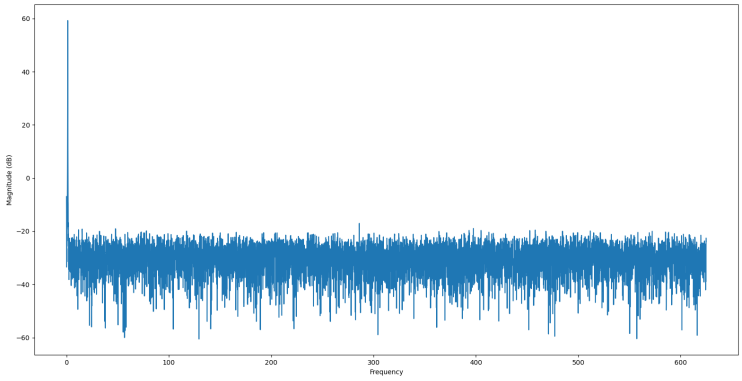


Figura 77: Salida del awg con 8 bits de ruido añadido

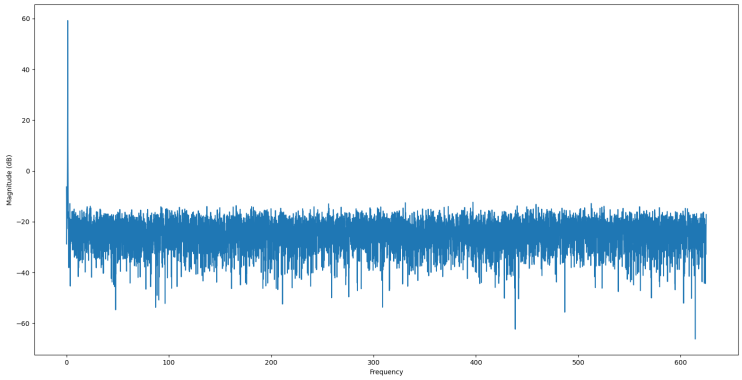


Figura 78: Salida del awg con 9 bits de ruido añadido

La salida de la memoria es de 16 bits, a la que se le añaden los bits de ruidos en los bits más bajos. Al truncar la salida del módulo con los 12 bits más altos, el ruido empieza a hacerse patente a partir de añadir 5 bits. En la figura 73 se observa como el fondo de ruido está en torno a los -40 dB, pero mientras se incrementa el nivel de bits, el valor medio del ruido aumenta, hasta llegar a un valor aproximado de -22 dB para 9 bits. En las imágenes se puede comprobar como la ecuación 9 se cumple para los valores configurados de 12 bits de salida, usando el 90 % de los bits y procesando los datos con una FFT de 1024 puntos nos queda un valor de SNR de:

$$SNR = -6,02 \cdot 12 - 1,76 - 20 \cdot \log(1{,}11) - 10 \cdot \log(\frac{1024}{2})\ dB$$
$$SNR = -101{,}99\ dBc$$

Capítulo 5

Integración

El módulo del AWG se ha incorporado en dos proyectos de DAS Photonics para su uso al realizar pruebas simulando señales de radar, pudiendo así realizar pruebas en niveles tempranos del diseño sin necesidad de usar un equipo completo del producto (sin necesidad de usar antenas de recepción con el hardware asociado). Uno de los objetivos era realizar un diseño que consumiera lo mínimo posible en cuanto a recursos disponibles en las FPGA usadas por DAS Photonics. La versión de prueba se realizó en una placa de evaluación ZCU106 de Xilinx, la cual posee la arquitectura Zynq ultrascale+. Al realizar la síntesis y la implementación del sistema con una configuración de 4 canales, se obtiene que se usa un total de 6724 LUT's, que equivale a un 3 % del total de la placa, 585 LUTRAM's (1 %), 26914 flip flops (6 %), 16 BRAM's (5 %) 2 BUFG (<1 %) y 0 DSPs. La imagen 80 se puede ver el resumen de utilización que Vivado ofrece después de la implementación donde se detallan los recursos usados en general. El IDE Vivado nos permite realizar un informe detallado por jerarquía (figura 80, donde queda desglosado los recursos por módulos.

Name	Constraints	Status	Progress	Incremental	WNS	TNS	WHS	THS	WBSS	TPWS	Total Pow...	Failed Ro...	LUT	FF	BRAMs	URAM	DSP	LUTRAM	IO	GT	BUFG	MMCM	PLL	PCIe
synth_1	constrs_1	synth_design Complete!	100%	Off									6858	30996	16.00	0	0	585	330	0	2	0	0	0
impl_1	constrs_1	route_design Complete!	100%	Off	0.771	0.000	0.003	0.000	11.761	0.000	1.103	0	6724	26914	16.00	0	0	585	330	0	3	0	0	0

Figura 79: Resumen de recursos usados en la placa ZCU106

En cuanto a la capacidad del AWG de poder trabajar a las frecuencias requeridas, se ha usado el reporte de análisis temporal para obtener la frecuencia máxima que el proyecto en la placa ZCU106 podría admitir. Para ello hemos registrado las entradas y las salidas del módulo en un archivo VHDL jerárquicamente superior y usado la ecuación que Xilinx nos proporciona para obtener la frecuencia máxima de operación:

$$max\_frecuency = \frac{1}{Clock\_period - Worst\_negative\_slack} \tag{15}$$

Donde el periodo del reloj es de 3.2 ns y según la figura 79 el *Worst_negative_slack* (WNS) es de 0.771, obteniendo una frecuencia de operación máxima de 411.69 MHz, una frecuencia superior a la requerida por los objetivos del proyecto.

Name	CLB LUTs (230400)	Block RAM Tile (312)	Bonded IOB (360)	HPIOB_M (144)	HPIOB_S (144)	HDIOB_M (24)	HDIOB_S (24)	HPIOB_SNGL (24)	GLOBAL CLOCK BUFFERs (544)	CLB Registers (460800)	CARRY8 (28800)	F7 Muxes (115200)	F8 Muxes (57600)	CLB (28800)	LUT as Logic (230400)	LUT as Memory (101760)
awg_wrapper	6724	16	330	136	136	20	18	20	3	26945	414	121	27	2882	6139	585
UUT (awg_top)	6724	16	0	0	0	0	0	0	0		414	121	27	2877	6139	585
AWG_channels[0].AWG_CORE_chn (awg_core)	1283	4	0	0	0	0	0	0	0		96	0	0	570	1137	146
add_gen_generate[0].addr_gen (address_generator_574)	5	0	0	0	0	0	0	0	0		4	0	0	16	5	0
add_gen_generate[1].addr_gen (address_generator__par...	5	0	0	0	0	0	0	0	0		8	0	0	11	5	0
add_gen_generate[2].addr_gen (address_generator__par...	36	0	0	0	0	0	0	0	0		8	0	0	21	36	0
add_gen_generate[3].addr_gen (address_generator__par...	6	0	0	0	0	0	0	0	0		8	0	0	15	6	0
dither_gen (dither_generator_578)	0	0	0	0	0	0	0	0	0		0	0	0	2	0	0
fsm_basic_module (fsm_basic_579)	169	0	0	0	0	0	0	0	0		22	0	0	42	169	0
fsm_master_module (fsm_master_580)	6	0	0	0	0	0	0	0	0		0	0	0	5	6	0
mask_data_out_inst (mask_data_out_581)	4	0	0	0	0	0	0	0	0		0	0	0	3	4	0
out_generate[0].out_data_inst (out_data_582)	192	0	0	0	0	0	0	0	0		5	0	0	130	147	45
out_generate[1].out_data_inst (out_data__parameterized...	175	0	0	0	0	0	0	0	0		5	0	0	133	138	37
out_generate[2].out_data_inst (out_data__parameterized...	167	0	0	0	0	0	0	0	0		5	0	0	141	138	29
out_generate[3].out_data_inst (out_data__parameterized...	173	0	0	0	0	0	0	0	0		5	0	0	130	138	35
phase_acc_mod (phase_acc_586)	0	0	0	0	0	0	0	0	0		4	0	0	12	0	0
ram_array[0].ram_1 (ram_mem_587)	1	1	0	0	0	0	0	0	0		0	0	0	6	1	0
ram_array[1].ram_1 (ram_mem_588)	1	1	0	0	0	0	0	0	0		0	0	0	4	1	0
ram_array[2].ram_1 (ram_mem_589)	1	1	0	0	0	0	0	0	0		0	0	0	4	1	0
ram_array[3].ram_1 (ram_mem_590)	1	1	0	0	0	0	0	0	0		0	0	0	15	1	0
reset_module (reset_591)	16	0	0	0	0	0	0	0	0		6	0	0	6	16	0
AWG_channels[1].AWG_CORE_chn (awg_core_0)	1276	4	0	0	0	0	0	0	0		96	0	0	601	1130	146
AWG_channels[2].AWG_CORE_chn (awg_core_1)	1276	4	0	0	0	0	0	0	0		96	0	0	559	1130	146
AWG_channels[3].AWG_CORE_chn (awg_core_2)	1279	4	0	0	0	0	0	0	0		96	0	0	625	1132	147
axi_interface (awg_reg)	1282	0	0	0	0	0	0	0	0		24	121	27	909	1282	0
reset_module (reset)	16	0	0	0	0	0	0	0	0		6	0	0	6	16	0
write_bram (fsm_write)	16	0	0	0	0	0	0	0	0		0	0	0	27	16	0

Figura 80: Resumen de recursos usados en la placa ZCU106 por jerarquía

La pruebas realizadas con los sistemas de DAS Photonics han producido resultados muy positivos a la hora de usar el AWG, generando estímulos dentro del procesado de señal, funcionando como substituto de los ADC reales del sistema. Se ha usado para verificar el funcionamiento de módulos internos del diseño sobre FPGA e incluso para funcionalidades del software embebido. Las siguientes pruebas se han realizado usando el sistema de RESM como plataforma donde integrar el AWG para capturar los datos a través de los módulos internos, en concreto de un procesado básico de señal consistente en un enventanado, un procesado de FFT, y conversión a polar en escala logarítmica. Para capturar los datos se usa una herramienta diseñada en DAS Photonics para sus proyectos que proporciona un entorno visual de los datos de los módulos de procesado, pudiendo acceder a información espectral o de pulsos. La imagen 81 muestra el espectro de salida de un sistema RESM donde se ha inyectado una señal usando el AWG con una frecuencia teórica de salida de 10 MHz. El sistema cuenta con tres canales, cada uno con su propio canal de AWG. El espectro que se observa se ha calculado con una FFT de 1024 puntos. Se observa que la frecuencia de salida no es exactamente de 10 MHz, sino 9.77 MHz pero este valor entra en el rango de error del generador, que es del paso mínimo, que si usamos la ecuación 12, nos da un valor de ± 1.22 MHz. Se puede ver como existen desfases entre los tres canales, siendo de 10º y 50º teóricos, y de 9.8º y 49.9º reales. La resolución de fase del AWG configurado es de $360/1024 = 0{,}3515$. En la imagen 82 se configura con 278 MHz de salida y la diferencia de fase entre canales se ha configurado como nula, por lo que al ser todas las muestras en fase entre canales, el desfase es constante a 0º en todo el espectro.

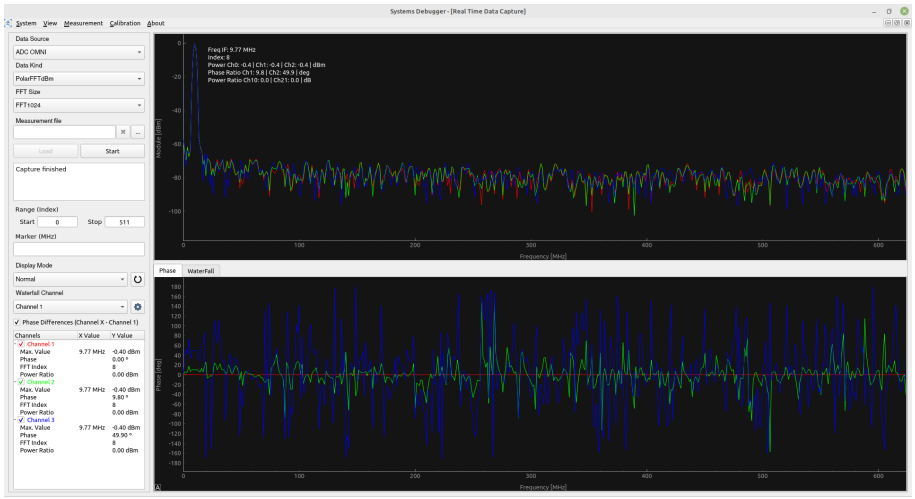


Figura 81: Salida del awg a 10 MHz

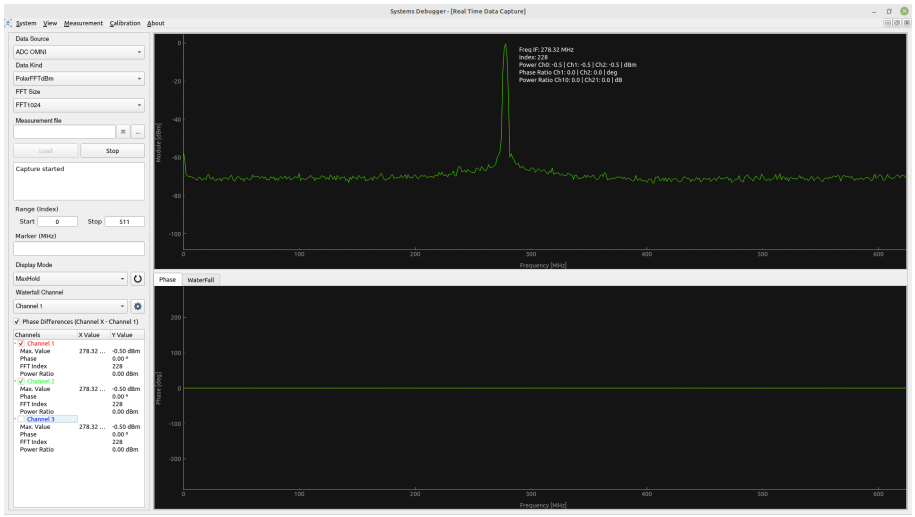


Figura 82: Salida del awg a 278 MHz

La figura 83 muestra el generador configurado con tres señales, una a 11 MHz, a 220 MHz y a 330 MHz con desfases entre canales de 10º y 50º. Se puede observar como la potencia de las señales cae con respecto a la dos configuraciones anteriores, desde -0.3/-0.5 dBm hasta los -9.3 dBm. Este efecto se explica teniendo en cuenta la figura 12, donde se ve como la potencia de la señal se reduce en la banda de Nyquist conforme la frecuencia aumenta desde 1 hasta $\frac{F_S}{2}$.

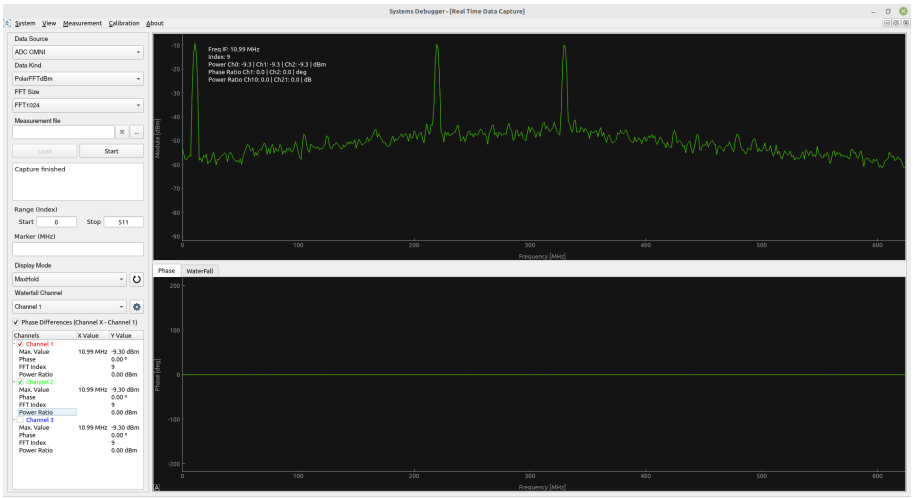


Figura 83: Salida del awg con más de una frecuencia y fases distintas

La misma señal compuesta pero añadiendo desfases entre canales se puede ver en la figura 84 donde se han añadido desfases de 75º y de -103º teóricos, aportando unos desfases reales de 72.5º y de -103.7º entre canales. Las frecuencias se mantienen en sus valores de 11 MHz, 220 MHz y 330 MHz y la potencia de salida de la señal igual que en la figura 83

Figura 84: Salida del awg a 278 MHz añadiendo desfases

En cuanto a la generación de pulsos, para el sistema de captura que se usa de pruebas es muy complicado realizar una captura de pulso único, no lo detecta con suficiente rapidez, por lo que las siguientes figuras corresponden a la captura del modo de pulsos continuos. En la figura 85 se puede observar el espectro de salida de un pulso de 6.4 μs de PW y 16 μs de PRI para una frecuencia de 600 MHz. Se puede observar como la potencia de la señal ha descendido drásticamente hasta los -84 dBm.

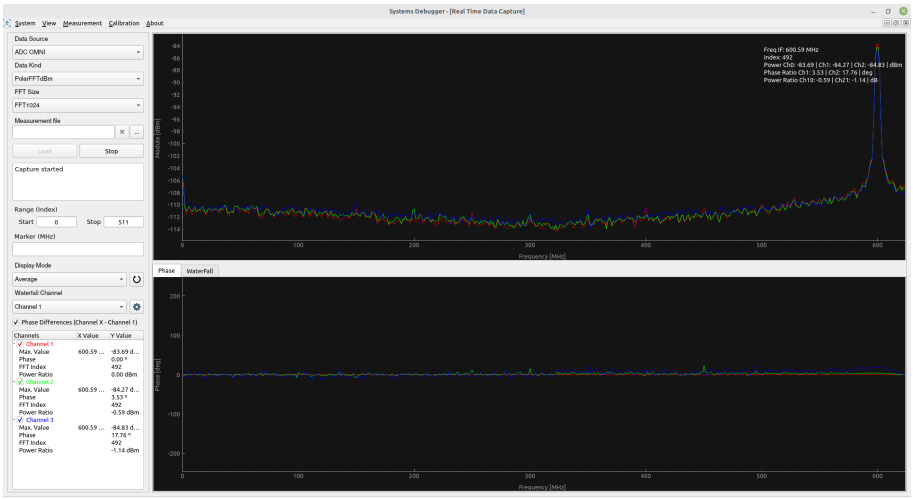


Figura 85: Pulso de 6.4 μs de PW y 16 μs de PRI a 600 MHz

La herramienta de captura nos permite obtener información adicional de los pulsos, como por ejemplo su PW, como se ve en la figura 86, donde en la columna de Pulsewidth se observa un valor de 6513 ns de PW, es decir un error de 513 ns que se puede deber a la extensión que realiza el AWG en el modo pulsos explicado en el apartado 28.

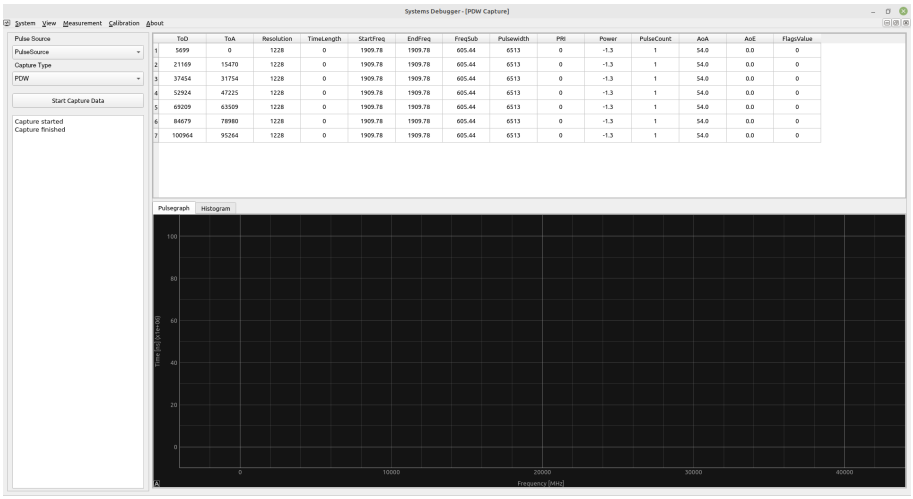


Figura 86: Detalle de la información de pulso de 6.4 μs de PW y 16 μs de PRI

Al igual que en el modo de señal continua, el modo pulsos también admite la generación de pulsos con más de una frecuencia a la vez. En la figura 87 se puede ver el espectro de un pulso con cuatro frecuencias distintas, 11 MHz, 100 MHz, 200 MHz y 312.5 MHz, con un PW de 1.6 μs. Si observamos la información de pulsos de la herramienta, podemos ver como la estadística de PW se mantiene igual, pero ahora aparecen las distintas frecuencias asociadas a la señal, como se ve en la figura 88, donde aparecen en la columna de Freqsub las frecuencias mencionadas junto con la potencia asociada a cada una de ellas.

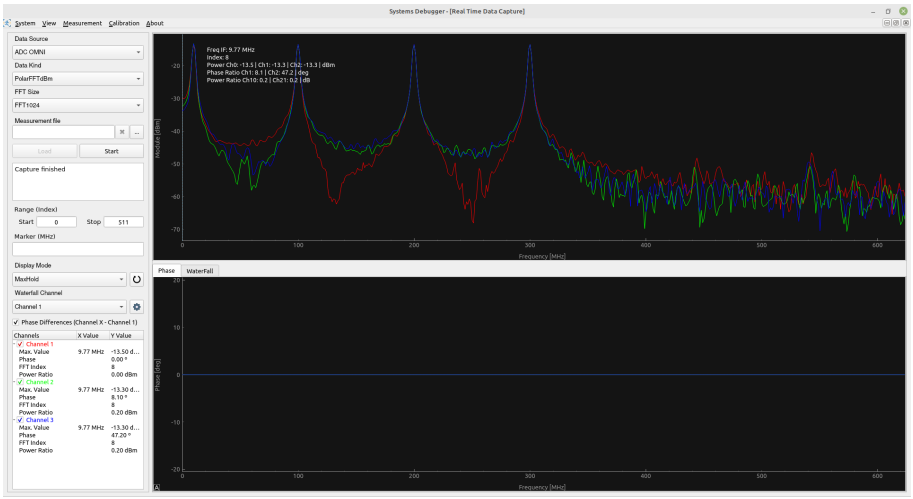


Figura 87: Pulso de 1.6 μs con múltiples frecuencias

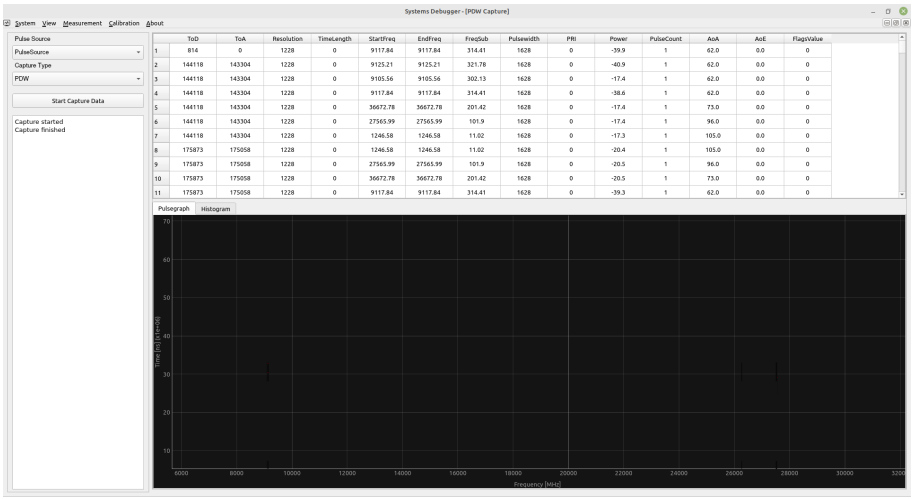


Figura 88: Detalle del pulso de 1.6 μs con múltiples frecuencias

Llevando al límite de generación de pulsos, si se configura el AWG con un PW muy pequeño, la señal saldrá muy distorsionada. Por ejemplo en la figura 89 se puede ver el espectro de salida de un pulso de 76 ns de PW y 16 μs de PRI a 600 MHz, mientras que la información del pulso se aprecia en la figura 90, donde el PW es de exactamente 76 ns.

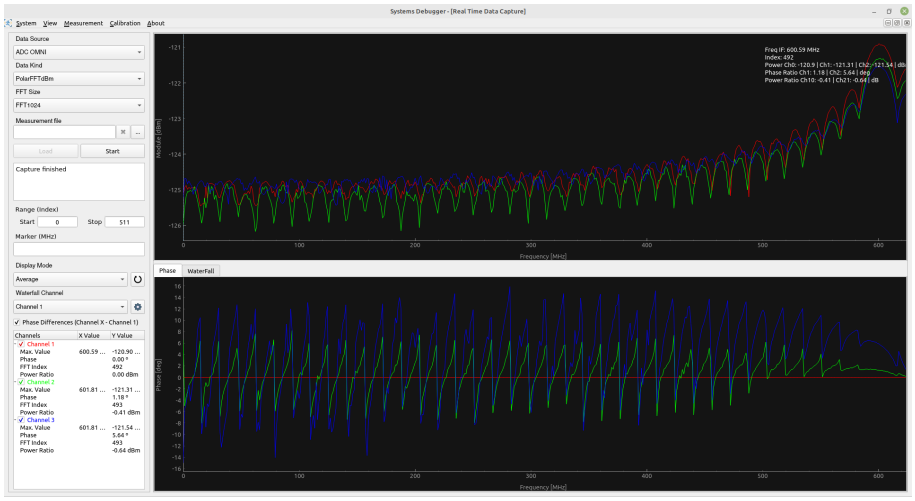


Figura 89: Pulso de PW de 76 ns y PRI de 16 μs

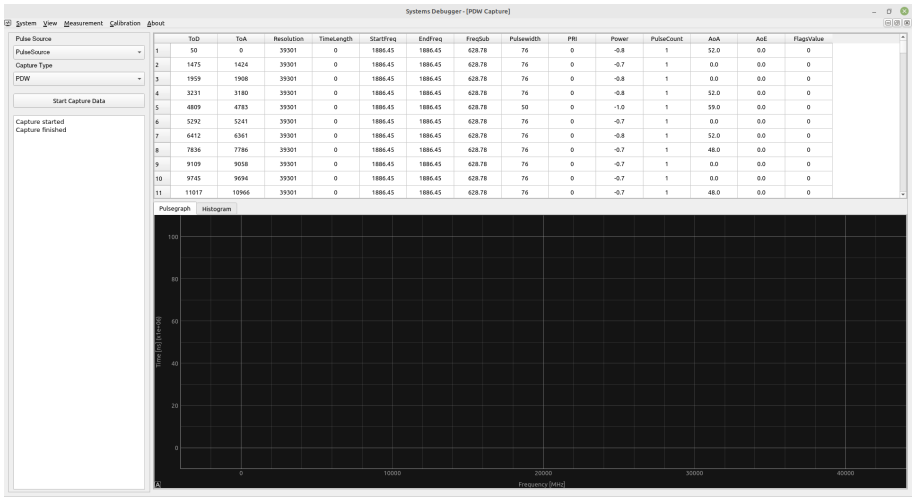


Figura 90: Detalle del pulso de PW de 76 ns y PRI de 16 μs

Si aumentamos en un factor 25 el PW, es decir que ahora tenemos un PW de 1.6 μs, el espectro se distorsiona menos, como se puede ver en la figura 91, el espectro tiene menos distorsión. Aunque de forma teórica se puede alcanzar PW de hasta 12.8 ns, utilizar por debajo de 1 μs de PW distorsiona mucho la señal.

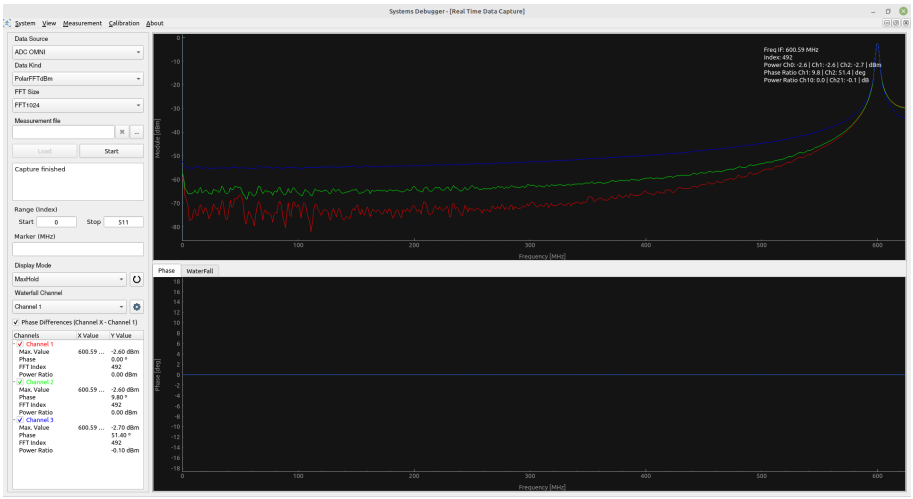


Figura 91: Detalle del pulso de PW de 1.6 μs y PRI de 16 μs

Capítulo 6

Conclusiones

Al concluir el desarrollo del AWG se han alcanzado la lista de objetivos expuestos en 1.1 y en 1.2. Se ha conseguido un sistema AWG con la capacidad de trabajar en 3 modos distintos (continuo, pulso único y pulsos continuos), con la capacidad de configurar cada canal instanciado por separado, pudiendo aplicar desfases, frecuencias, PW o PRI distintos.
El AWG es capaz de instanciar hasta un máximo de 8 canales con paralelización máxima de 16. Las señales se pueden almacenar de forma dinámica en cada memoria de cada canal por separado. El proyecto se ha realizado usando el lenguaje de programación VHDL y se han dejado los parámetros de diseño importantes como generics, como el tamaño del acumulador, el tamaño de la memoria, el tamaño de los datos de salida, el número de canales a instanciar o el número de paralelizaciones por canal. Gracias a esto el diseño es fácilmente adaptable a distintos proyectos que requieran distintas configuraciones. Cada canal instanciado está sincronizado con un canal maestro, consiguiendo desfases entre canales como máximo de un ciclo de reloj. Las señales se crean con un programa de python, el cual en base a los parámetros de número de frecuencias, frecuencias, desfases y amplitudes, es capaz de generar las muestras para ser almacenadas en el AWG. El proyecto actual se ha puesto a prueba en varios sistemas reales, con hardware real, consiguiendo resultados muy positivos para llevar a cabo sus funciones de generar señales con parámetros conocidos y así poder realizar validaciones de sistemas digitales complejos. Durante el desarrollo del proyecto del AWG se han integrado material de tfms anteriores, en concreto del TFM de Alberto Martínez Cuesta [9] y de macros de Xilinx como el xpm_cdc_handshake.

Capítulo 7

Trabajos futuros

Con todo lo expuesto en los capítulos anteriores, queda patente que la versión del AWG tiene potencial para ser mejorado en ciertos aspectos. Por ejemplo se podría trabajar en añadir modulaciones como FM, AM, QPSK, etc, las cuales abrirían aún más la versatilidad del módulo para validación de sistemas de radar. Otro campo abierto para trabajos posteriores podría ser la adición de un módulo de salida que incluyera las no linealidades de los ADC, dado que las salidas actuales del AWG son muy ideales, quitando el hecho de la adición del ruido gaussiano. Se podría realizar estudios de las funciones de transferencia de distintos ADC para caracterizarlos y así simular más fielmente el comportamiento de la parte digital usando el AWG. Otro aspecto en el que se puede trabajar en un futuro es la adición de un control de secuencias y escenarios, el AWG N8241A de Agilent tiene la opción de predefinir una serie de secuencias de formas de onda almacenadas en la memoria para crear salidas más complejas aún, y agrupar secuencias en escenarios para reproducir estímulos. La idea se basaría en la creación de dos máquinas de estado con dos memorias para almacenar las directrices de elección de formas de onda y duración para recrear la opción de escenarios y secuencias en el AWG.

Bibliografía

[1] *GITFLOW*. Accessed: 20/03/2022. 2020. URL: http://datasift.github.io/gitflow/IntroducingGitFlow.html.

[2] Xilinx. *ZCU106 Evaluation Board, user guide*. 2019, pág. 134. URL: https://www.xilinx.com/support/documentation/boards_and_kits/zcu106/ug1244-zcu106-eval-bd.pdf.

[3] *Vunit*. Accessed: 15/06/2022. 2020. URL: https://vunit.github.io/.

[4] Keysight Technologies. *Keysight Fundamentals of Arbitrary Waveform Generation Reference Guide*. 2015, pág. 203. URL: https://www.keysight.com/us/en/assets/9018-03815/reference-guides/9018-03815.pdf.

[5] Analog Devices. *A Technical Tutorial on Digital Signal Synthesis*. 1999, pág. 122. URL: https://www.ieee.li/pdf/essay/dds.pdf.

[6] Lionel Cordesses. "Direct Digital Synthesis:A Tool for Periodic Wave Generation (Part 1)". En: *IEEE Signal Processing Magazine* 21 (jul. de 2004), págs. 50-54. ISSN: 1053-5888. URL: https://ieeexplore.ieee.org/document/1311140.

[7] Lionel Cordesses. "Direct Digital Synthesis:A Tool for Periodic Wave Generation (Part 2)". En: *IEEE Signal Processing Magazine* 21 (sep. de 2004), págs. 100-112. ISSN: 1053-5888. URL: https://ieeexplore.ieee.org/document/1328096.

[8] Ke Liu y col. "Precisely synchronous and cascadable multi-channel arbitrary waveform generator". En: *Review of Scientific Instruments* 88 (mar. de 2017). URL: https://doi.org/10.1063/1.4978067.

[9] Alberto Martínez Cuesta. *Diseño e implementación de un jammer configurable en FPGA*. 2020. URL: https://m.riunet.upv.es/bitstream/handle/10251/136750/Mart%C3%ADnez%20-%20Dise%C3%B1o%20e%20implementaci%C3%B3n%20de%20un%20jammer%20configurable%20en%20FPGA.pdf?sequence=1&isAllowed=y.

Parte II

Anexos

Apéndice a

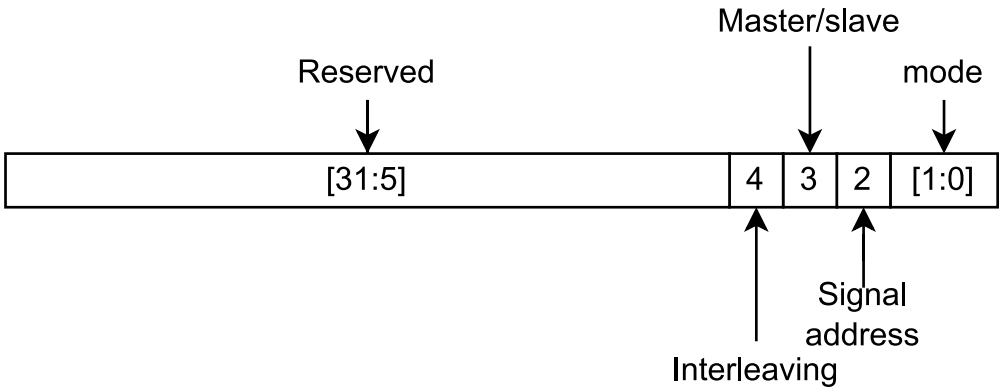
Registros del AWG

I. Tabla de registros

REGISTRO	DIRECCION	VALOR INICIAL
IPcore version	0x00	0x00000000
IPcore ID	0x04	0x00000000
Control canal 0	0x08	0x00000000
Frecuencia canal 0	0x0C	0x00000000
Fase canal 0	0x10	0x00000000
PRI canal 0	0x14	0x00000000
PW canal 0	0x18	0x00000000
Control canal 1	0x1C	0x00000000
Frecuencia canal 1	0x20	0x00000000
Fase canal 1	0x24	0x00000000
PRI canal 1	0x28	0x00000000
PW canal 1	0x2C	0x00000000
Control canal 2	0x30	0x00000000
Frecuencia canal 2	0x34	0x00000000
Fase canal 2	0x38	0x00000000
PRI canal 2	0x3C	0x00000000
PW canal 2	0x40	0x00000000
Control canal 3	0x44	0x00000000
Frecuencia canal 3	0x48	0x00000000
Fase canal 3	0x4C	0x00000000
PRI canal 3	0x50	0x00000000
PW canal 3	0x54	0x00000000
Control canal 4	0x58	0x00000000
Frecuencia canal 4	0x5C	0x00000000
Fase canal 4	0x60	0x00000000
PRI canal 4	0x64	0x00000000
PW canal 4	0x68	0x00000000
Control canal 0	0x6C	0x00000000
Frecuencia canal 5	0x70	0x00000000
Fase canal 5	0x74	0x00000000
PRI canal 5	0x78	0x00000000
PW canal 5	0x7C	0x00000000
Control canal 0	0x80	0x00000000
Frecuencia canal 6	0x84	0x00000000
Fase canal 6	0x88	0x00000000
PRI canal 6	0x8C	0x00000000
PW canal 6	0x90	0x00000000
Control canal 0	0x94	0x00000000
Frecuencia canal 7	0x98	0x00000000
Fase canal 7	0x9C	0x00000000
PRI canal 7	0xA0	0x00000000
PW canal 7	0xA4	0x00000000
Reserved	0xA8	0x00000000
.....	....	.....
Reserved	0xD0	0x00000000
FSM WRITE STATUS	0xD4	0x00000000
FSM AWG 0 STATUS	0xD8	0x00000000
FSM AWG 1 STATUS	0xDC	0x00000000
FSM AWG 2 STATUS	0xE0	0x00000000
FSM AWG 3 STATUS	0xE4	0x00000000
FSM AWG 4 STATUS	0xE8	0x00000000
FSM AWG 5 STATUS	0xEC	0x00000000
FSM AWG 6 STATUS	0xF0	0x00000000
FSM AWG 7 STATUS	0xF4	0x00000000
Registro control general	0xF8	0x00000000
Registro escritura memoria	0xFC	0x00000000

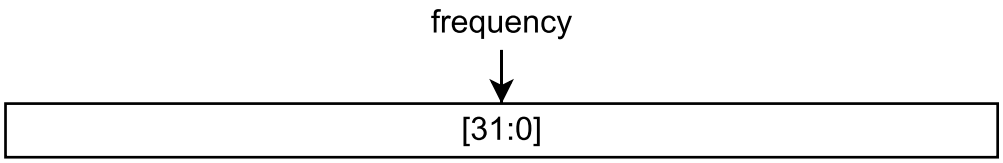
Tabla 2: Registros del AWG

Channel control register



Bits	Name	Default	Description
[31:5]	Reserved	N/A	Reserved
4	Interleaving	0x0	Set interleaving mode. 0 = no interleaving 1 = Interleaving mode activated
3	Master/slave	0x0	Set master or slave behavioral. 0 = Slave channel 1 = Master channel
2	Signal address	0x0	Set which memory stored signal is reproduced.
[1:0]	Mode	0x0	AWG operation mode. 0x0 = continuous mode 0x1 = one pulse 0x2 = pulse mode

frequency register



Bits	Name	Default	Description
[31:0]	frequency	0x00000000	frequency value

Only the 10 MSB of this register are used for frequency configuration,

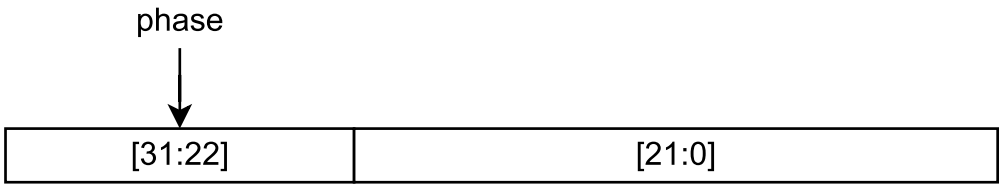
the remaining 22 bits are used for noise injection.
Frecuency is stored as a value from 0 to 512, generating an output

frequency using the following ecuation:

$$f_{output} = \frac{P \cdot frequency \cdot f_{clk}}{2^N}$$

Where:
*P is the paralelization
*frequency is the register value
*fclk is the sampling clock
*N is the awg memory deep
Example:
If the register value is 0x3E800000 (250) with a sampling clock of 312.5MHz, paralelization 4 and memory deep of 10 bits, the output frequency is 305.17MHz

phase register

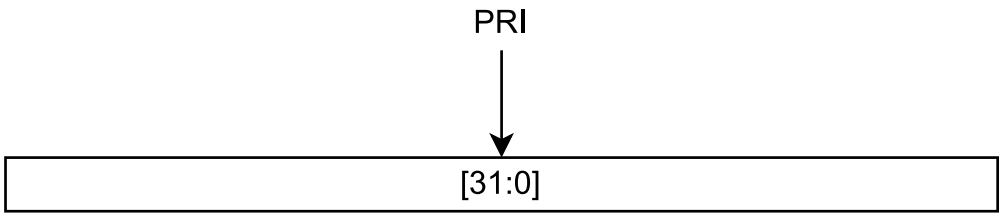


Bits	Name	Default	Description
[31:22]	Phase	0x00000000	Phase value
[21:0]	Reserved	N/A	Reserved

The 10 MSB of this regiter store the phase information of the channel. This value are a unsigned value and the output phase value can be calculated as folow:

$$Phase_{output}(^{o}) = \frac{Phase \cdot 2^{N}}{360}$$

PRI register

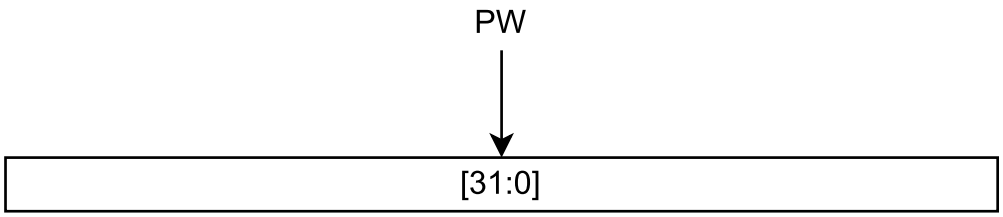


Bits	Name	Default	Description
[31:0]	PRI	0x00000000	PRI value

PRI value of the channel.
The output PRI value in time can be calculated as:

$$PRI_{output}(ns) = PRI_{register} \cdot \frac{1}{f_{clk}}$$

PW register

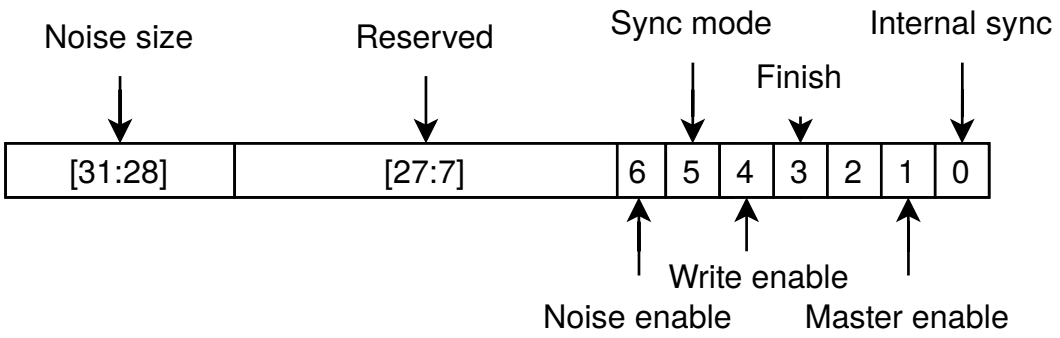


Bits	Name	Default	Description
[31:0]	PW	0x00000000	PW value

PW value of the channel.
The output PW value in time can be calculated as:

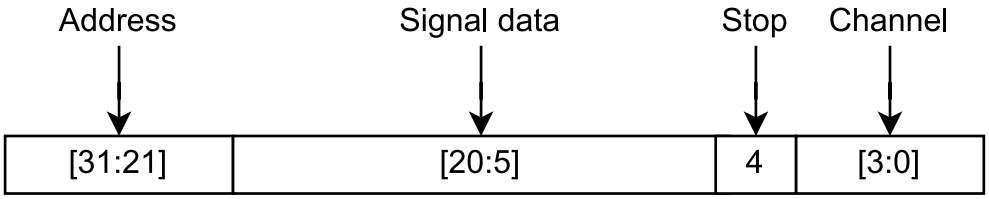
$$PW_{output}(ns) = PW_{register} \cdot \frac{1}{f_{clk}}$$

General control register



Bits	Name	Default	Description
[31:28]	Memory address	0x000	4 bits with b10 noise size
6	Noise enable	0	Bit to enable amplitud noise: 0: noise disabled 1: noise enabled
5	Sync mode	0	Bit for sync signal source: 0: internal source 1: external source
4	Write enable	0	Enable write data to memories. Self clear.
3	Finish	0	Finish signal control: 0: No stop awg 1: Stop awg
1	Master enable	0	Enable awg's fsm
0	Internal sync	0	When set to 1 and sync mode is 0, launch a sync signal to start modes. Need to be cleared

Load control register



Bits	Name	Default	Description
[31:21]	Memory address	0x000	11 bits address where data will be stored
[20:5]	Data	0x0000	16 bits of signal data to be stored
4	Stop	0x0	This bit stops the load FSM
[3:0]	Channel	0x0	4 bits of channel where data will be stored

Apéndice b

Control del AWG mediante Python

I. Clase AWG

```
import numpy as np
import os
import bcolors

class awg ():

    def load_samples(channel,ba,ip):
        file ="signal.csv"
        samples = np.loadtxt(file, dtype=int, delimiter=",")
        command ="/sbin/devmem " +  str(hex(0xF8 + ba)) + " 32 0x18; \n"
        for j in range(0,channel):
            for i in range(1024):
               command +="/sbin/devmem " +  str(hex(0xFC + ba)) + " 32 " \
                         + str(j + (samples[i]<<5) + (i<<21)) + ";\n"
        command += "/sbin/devmem " +  str(hex(0xFC + ba)) + " 32 0x08;\n"

        return command

    def general_config(operation,sect,ip,base_address):

        #OMNI
        if sect == 0:
            adc_address_bit = 0x0
            fpga_cross_base_address = 0x1
            fpga_cross = 0x2
        #SEC0
        elif sect == 1:
            adc_address_bit = 0x3
            fpga_cross_base_address = 0x4
            fpga_cross = 0x5
        #SEC1
        elif sect == 2:
            adc_address_bit = 0x5
            fpga_cross_base_address = 0x6
            fpga_cross = 0x7
        #SEC2
        else:
```

```
            adc_address_bit = 0x8
            fpga_cross_base_address = 0x9
            fpga_cross = 0xA

        #operation commands
        command = ""
        if operation == "config_param":
            command +=" /sbin/devmem " + str(hex(adc_address_bit)) \
                    + " 32 " + str(hex(0x0)) + " ;\n"
            command += " /sbin/devmem " +   str(hex(0xF8 + base_address)) \
                    + " 32 " + str(hex(0x2A)) + " ;\n"
            command += " /sbin/devmem " +   str(hex(0xF8 + base_address)) \
                    + " 32 " + str(hex(0x22)) + " ;\n"

        elif operation == "int_trigger":
            config_value = 1 + (1 << 1) + ( 0 << 3) + (0 << 5)
            command +=  " /sbin/devmem " +   str(hex(0xF8 + base_address)) \
                    + " 32 " + str(hex(config_value)) + " ;"
        elif operation == "int_finish":
            config_value = 0 + (1 << 1) + ( 1 << 3) + (0 << 5)
            command +=  " /sbin/devmem " +   str(hex(0xF8 + base_address)) \
                    + " 32 " + str(hex(config_value)) + " ;"
        elif operation == "ext_trigger":
            command += " /sbin/devmem " +   str(hex(fpga_cross_base_address)) \
                    + " 32 " + str("0x3F") + " ;\n"
        return command


    def config_channel( channel, mode, master_slave, interleaving, \
                        frequency, delay, pri, pw, ba,ip,fclk):
        #config channel
        config_value = mode + (master_slave << 3) + (interleaving << 4)
        command =" /sbin/devmem " +   str(hex(8  + channel*20 + ba)) \
                + " 32 " + str(hex(config_value)) + " ;\n"
        #frequency
        freq = int((frequency * (2**10)*2**22)/(4*fclk)) + 1365
        command += " /sbin/devmem " +   str(hex(12 + channel*20 + ba)) \
                + " 32 " + str(hex(freq)) + " ;\n"
        #delay
        delay_data = int((delay / 360) * 1024) << 22
        command +=" /sbin/devmem " +   str(hex(16 + channel*20 + ba)) \
                + " 32 " + str(hex(delay_data)) + " ;\n"
        #PRI
        command += " /sbin/devmem " +   str(hex(20 + channel*20 + ba)) \
                + " 32 " + str(hex(pri)) + " ;\n"
        #PW
        command +=" /sbin/devmem " +   str(hex(24 + channel*20 + ba)) \
                + " 32 " + str(hex(pw)) + " ;\n"
        return command

    def signal_generator(tones, freqs, delays, ampl):
        if os.path.isfile('./signal.csv'):
            os.remove("./signal.csv")

        M = 1024
        quant = (2**(16-1))-1
        t = np.linspace(0.0, M, M, endpoint = False)
        signal = []
```

```
96          muestras = []
97          sine_signal = [0]*M
98          for i in range(0,tones):
99              print(i,freqs[i],delays[i],ampl[i])
100             sine_signal += np.float(ampl[i])*(np.sin(2.0 * np.pi \
101             * np.float(freqs[i]) * t/M + np.radians(np.float(delays[i]))))
102
103         max_peak = max(sine_signal)
104         for i in range(len(sine_signal)):
105             if sine_signal[i] > 0:
106                 sine_signal[i] = np.int16(sine_signal[i] \
107                                  *quant/(1.11*max_peak))
108             elif sine_signal[i] < 0:
109                 sine_signal[i] = 2**(16) -\
110                     np.int16(abs(sine_signal[i])*quant/(1.11*max_peak))
111                 if sine_signal[i] > 2**16:
112                     sine_signal[i] = 2**(15)
113             else:
114                 sine_signal[i] = np.int16(0)
115         signal = np.concatenate((signal,sine_signal))
116
117         if (len(signal) < 2048):
118             difference = 2048- len(signal)
119             zeros = np.zeros(difference)
120             values = np.concatenate((signal,zeros))
121         else:
122             values = signal
123         for i in range(0, 2048):
124                 muestras.append(np.uint32(values[i]))
125         np.savetxt('signal.csv', muestras,delimiter=',',fmt='%s')
126         return (muestras, t)
```

II. Clase colores

```
class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKCYAN = '\033[96m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
    Black = "\033[30m"
    Red  = "\033[31m"
    Green = "\033[32m"
    Yellow       = "\033[33m"
    Blue         = "\033[34m"
    Magenta      = "\033[35m"
    Cyan         = "\033[36m"
    LightGray    = "\033[37m"
    DarkGray     = "\033[90m"
    LightRed     = "\033[91m"
    LightGreen   = "\033[92m"
    LightYellow  = "\033[93m"
    LightBlue    = "\033[94m"
    LightMagenta = "\033[95m"
    LightCyan    = "\033[96m"
    White        = "\033[97m"
    CLEANWINDOW  = "\033[2J"
```

III. Control del AWG

```
import awg_class as awg
import subprocess
import bcolors
import time

class awg_control():
    upper_ba = 0x0
    lower_ba = 0x1
    #=========================================================================
    #Configure channels of AWG with the parameters
    #=========================================================================
    def channels_config(abaco_1_ip,abaco_2_ip,freq,phases,pri,pw,mode,fclk):

        print("start config")
        start_procesing_time = time.time()
        #omni
        command1 = awg.awg.config_channel( \
                  0,mode,1,0,freq,phases[0],pri[0],pw[0],\
                    upper_ba,abaco_1_ip,fclk)
        command1 += awg.awg.config_channel(1,mode,0,0,freq, \
                    phases[1],pri[1],pw[1],upper_ba,abaco_1_ip,fclk)
        command1 += awg.awg.config_channel(2,mode,0,0,freq, \
                    phases[2],pri[2],pw[2],upper_ba,abaco_1_ip,fclk)
        command1 += awg.awg.general_config("config_param",0, \
                                           abaco_1_ip,upper_ba)
        print(f"{bcolors.bcolors.LightGreen} \
              omni channels configurated{bcolors.bcolors.ENDC}")
        #sec0
        command1 += awg.awg.config_channel(0,mode,1,0,freq, \
                    phases[3],pri[3],pw[3],lower_ba,abaco_1_ip,fclk)
        command1 += awg.awg.config_channel(1,mode,0,0,freq, \
                    phases[4],pri[4],pw[4],lower_ba,abaco_1_ip,fclk)
        command1 += awg.awg.config_channel(2,mode,0,0,freq, \
                    phases[5],pri[5],pw[5],lower_ba,abaco_1_ip,fclk)
        command1 += awg.awg.config_channel(3,mode,0,0,freq, \
                    phases[6],pri[6],pw[6],lower_ba,abaco_1_ip,fclk)
        command1 += awg.awg.general_config("config_param",1, \
                                           abaco_1_ip,lower_ba)
        print(f"{bcolors.bcolors.LightGreen} \
                    sec0 channels configurated{bcolors.bcolors.ENDC}")
        if abaco_2_ip != 0:
          #sec1
          command2 = awg.awg.config_channel(0,mode,1,0,freq, \
                     phases[7],pri[7],pw[7],upper_ba,abaco_2_ip,fclk)
          command2 += awg.awg.config_channel(1,mode,0,0,freq, \
                      phases[8],pri[8],pw[8],upper_ba,abaco_2_ip,fclk)
          command2 += awg.awg.config_channel(2,mode,0,0,freq, \
                      phases[9],pri[9],pw[9],upper_ba,abaco_2_ip,fclk)
          command2 += awg.awg.config_channel(3,mode,0,0,freq, \
                      phases[10],pri[10],pw[10],upper_ba,abaco_2_ip,fclk)
          command2 += awg.awg.general_config("config_param",2, \
                      abaco_2_ip,upper_ba)
          print(f"{bcolors.bcolors.LightGreen} \
                    sec1 channels configurated{bcolors.bcolors.ENDC}")
          #sec2
          command2 += awg.awg.config_channel(0,mode,1,0,freq, \
```

```
                    phases[11],pri[11],pw[11],lower_ba,abaco_2_ip,fclk)
      command2 += awg.awg.config_channel(1,mode,0,0,freq, \
                    phases[12],pri[12],pw[12],lower_ba,abaco_2_ip,fclk)
      command2 += awg.awg.config_channel(2,mode,0,0,freq, \
                    phases[13],pri[13],pw[13],lower_ba,abaco_2_ip,fclk)
      command2 += awg.awg.config_channel(3,mode,0,0,freq, \
                    phases[14],pri[14],pw[14],lower_ba,abaco_2_ip,fclk)
      command2 += awg.awg.general_config("config_param",3, \
                                         abaco_2_ip,lower_ba)

      print(f"{bcolors.bcolors.LightGreen} \
            sec2 channels configurated{bcolors.bcolors.ENDC}")
    #launch trigger
    command1 += awg.awg.general_config("ext_trigger",0,abaco_1_ip,upper_ba)
    with open('config1.sh', 'w') as bash_file:
        bash_file.write(command1)

    send = "sshpass -p user scp config1.sh user@" + \
            abaco_1_ip + ":/mnt/emmc;\n"
    send += "sshpass -p user ssh user@" + abaco_1_ip + \
            " chmod +x /mnt/emmc/config1.sh;\n"
    send += "sshpass -p user ssh user@" + abaco_1_ip + \
            " sh /mnt/emmc/config1.sh;\n"
    subprocess.run(send,shell = True, stderr=subprocess.DEVNULL)

    if abaco_2_ip != 0:
      with open('config2.sh', 'w') as bash_file:
          bash_file.write(command2)

      send = "sshpass -p user scp config2.sh user@" + \
              abaco_2_ip + ":/mnt/emmc;\n"
      send += "sshpass -p user ssh user@" + abaco_2_ip + \
              " chmod +x /mnt/emmc/config2.sh;\n"
      send += "sshpass -p user ssh user@" + abaco_2_ip + \
              " sh /mnt/emmc/config2.sh;\n"
      subprocess.run(send,shell = True, stderr=subprocess.DEVNULL)

    stop_procesing_time = time.time()
    print("Config in: ",(stop_procesing_time-start_procesing_time),"sec")
    real_freq = 10
    return real_freq

#=========================================================================
# Load samples into AWG using a bash file
#=========================================================================
def load_fpga(abaco_1_ip,abaco_2_ip,signal_input):

    start_procesing_time = time.time()
    print(f"{bcolors.bcolors.LightBlue} \
          Creating signal for AWG memory{bcolors.bcolors.ENDC}")
    signal,t = awg.awg.signal_generator(signal_input[0],\
               signal_input[1],signal_input[2],signal_input[3])

    command1 = awg.awg.general_config("config_param",0, \
                                      abaco_1_ip,upper_ba)
    command1 += awg.awg.general_config("config_param",1, \
                                       abaco_1_ip,lower_ba)
    if abaco_2_ip != 0:
```

```
115           command2 = awg.awg.general_config("config_param",2, \
116                                             abaco_2_ip,upper_ba)
117           command2 += awg.awg.general_config("config_param",3, \
118                                             abaco_2_ip,lower_ba)
119         print(f"{bcolors.bcolors.LightBlue} \
120               Configurign windows filters data valid bypass{bcolors.bcolors.
    ENDC}")
121         print(f"{bcolors.bcolors.LightBlue}\
122               Loading AWG memory{bcolors.bcolors.ENDC}")
123         with open('load1.sh', 'w') as bash_file:
124             bash_file.write(command1)
125 
126         send = "sshpass -p user scp load1.sh user@" + \
127                 abaco_1_ip + ":/mnt/emmc;\n"
128         send += "sshpass -p user ssh user@" + abaco_1_ip + \
129                 " chmod +x /mnt/emmc/load1.sh;\n"
130         send += "sshpass -p user ssh user@" + abaco_1_ip + \
131                 " sh /mnt/emmc/load1.sh;\n"
132         subprocess.run(send,shell = True, stderr=subprocess.DEVNULL)
133         if abaco_2_ip != 0:
134           with open('load2.sh', 'w') as bash_file:
135               bash_file.write(command2)
136 
137           send = "sshpass -p user scp load2.sh user@" + \
138                   abaco_2_ip + ":/mnt/emmc;\n"
139           send += "sshpass -p user ssh user@" + \
140                   abaco_2_ip + " chmod +x /mnt/emmc/load2.sh;\n"
141           send += "sshpass -p user ssh user@" + \
142                   abaco_2_ip + " sh /mnt/emmc/load2.sh;\n"
143           subprocess.run(send,shell = True, stderr=subprocess.DEVNULL)
144 
145 
146         send1 =  "sshpass -p user scp signal.csv user@" + \
147                   abaco_1_ip + ":/mnt/emmc/ ;\n"
148         send1 += "sshpass -p user ssh user@" + \
149                   abaco_1_ip + " sh /mnt/emmc/launch_load.sh ;\n"
150         if abaco_2_ip != 0:
151           send2 =  "sshpass -p user scp signal.csv user@" + \
152                     abaco_2_ip + ":/mnt/emmc/ ;\n"
153           send2 += "sshpass -p user ssh user@" + \
154                     abaco_2_ip + " sh /mnt/emmc/launch_load.sh ;\n"
155 
156         out =  subprocess.run(send1,shell = True,capture_output = True)
157         if out.stderr != None:
158           print(out.stderr)
159         if abaco_2_ip != 0:
160           out =  subprocess.run(send2,shell = True,capture_output = True)
161           if out.stderr != None:
162             print(out.stderr)
163         print(f"{bcolors.bcolors.LightGreen}loaded AWG")
164         stop_procesing_time = time.time()
165         print("Config done in: ",(stop_procesing_time-start_procesing_time),"
    sec")
166         return signal,t
```

IV. Intefaz del AWG

```
import control_awg as awg
import argparse
from argparse import RawTextHelpFormatter
import sys

#version
VERSION = "v1.0"

SCRIPT_DESCRIPTION = "Control of awg"

SCRIPT_COPYRIGHT = '''
Copyright 2021. DAS Photonics. All Rights Reserved.
No guarantee of usage is provided.

''' + VERSION


def main(argv):
    global IDCnt, config
    parser = argparse.ArgumentParser(
        description = SCRIPT_DESCRIPTION,
        epilog = SCRIPT_COPYRIGHT,
        formatter_class=RawTextHelpFormatter)
    parser.add_argument('-l',
                        '--LOAD',
                        # nargs='+',
                        dest = 'load',
                        default = 0,
                        choices = [0,1],
                        help ="operation mode: 0--> single load. 1--> freq sweep",
                        type=int)
    parser.add_argument('-pl',
                        '--PHASE_LIST',
                        nargs='+',
                        dest = 'phase_list',
                        default = [0],
                        help ="Phase input list",
                        type=int)
    parser.add_argument('-fl',
                        '--FREQ_LIST',
                        nargs='+',
                        dest = 'freq_list',
                        default = [1],
                        help ="Frequency list signals",
                        type=float)

    parser.add_argument('-f',
                        '--AWG_FREQ',
                        #nargs='+',
                        dest = 'awg_freq',
                        default = 1,
                        help ="awg operation frequency",
                        type=float)

    parser.add_argument('-c',
```

```
                        '--CLOCK',
                        #nargs='+',
                        dest = 'clock',
                        default = 312.5,
                        help ="awg operation system clock",
                        type=float)

    parser.add_argument('-mode',
                        '--MODE',
                        #nargs='+',
                        dest = 'mode',
                        default = 0,
                        choices = [0,1,2],
                        help ="awg operation mode: \n0--> single tone \n1--> single pulse \n2--> continuous pulses.\nDefault mode is 0, single tone",
                        type=int)

    parser.add_argument('-s',
                        '--STEP',
                        #nargs='+',
                        dest = 'step',
                        default = 1,
                        help ="awg frecuency sweep step",
                        type=int)

    parser.add_argument('-pw',
                        '--PW',
                        #nargs='+',
                        dest = 'pw',
                        default = 1000,
                        help ="PW values",
                        type=int)

    parser.add_argument('-pri',
                        '--PRI',
                        #nargs='+',
                        dest = 'pri',
                        default = 20000,
                        help ="PRI values",
                        type=int)

    parser.add_argument('-ip1',
                        '--IP_IC',
                        #nargs='+',
                        dest = 'ip_ic',
                        default = "192.168.1.1",
                        help ="IP IC board",
                        type=str)

    parser.add_argument('-ip2',
                        '--IP_ABACO_1',
                        #nargs='+',
                        dest = 'ip_abaco_1',
                        default = "192.168.1.1",
                        help ="IP ABACO board 1",
                        type=str)

    parser.add_argument('-ip3',
```

```
                        '--IP_ABACO_2',
                        #nargs='+',
                        dest = 'ip_abaco_2',
                        default = "192.168.1.1",
                        help ="IP ABACO board 2",
                        type=str)


    config = parser.parse_args(argv)

    freq_awg = min(config.freq_list)
    print("awg freg: ",freq_awg)
    if len(config.phase_list) < len(config.freq_list):
        for i in range( len(config.phase_list) ,len(config.freq_list),1):
            print("insert phase for frequency: " + str(config.freq_list[i]))
            config.phase_list.append(int(input()))

    elif len(config.phase_list) > len(config.freq_list):
        for i in range( len(config.freq_list) ,len(config.phase_list),1):
            print("insert freq value for phase: " + str(config.phase_list[i]))
            config.freq_list.append(int(input()))
    normalized_freqs = [x/freq_awg for x in config.freq_list]
    normalized_ampl = [1 for x in normalized_freqs]


    #IP assigment
    ip_ip = config.ip_ic
    ip_abaco_1 = config.ip_abaco_1
    ip_abaco_2 = config.ip_abaco_2

    #signal properties
    signal = [len(normalized_freqs),normalized_freqs,config.phase_list,
    normalized_ampl]
    print(signal)
    if config.awg_freq > min(config.freq_list):
        awg_freq = min(config.freq_list)
        print("awg frecuency is higher than minimum signal frecuency stored, \
                frecuency stalled to: " + str(awg_freq) + " MHz")
    else:
        awg_freq = config.awg_freq
    phases_list = [0,10,50,0,10,50,80,0,0,0,0,0,0,0,0]
    fclk = config.clock
    pri = config.pri
    pw = config.pw
    pri_list    = [pri,pri,pri,pri,pri,pri,pri,pri,pri,pri,pri,pri,pri,pri,pri]
    pw_list     = [pw,pw,pw,pw,pw,pw,pw,pw,pw,pw,pw,pw,pw,pw,pw]
    signal,t = awg.awg_control.load_fpga(ip_abaco_1,ip_abaco_2,signal)
    if config.load == 0:
        mode = config.mode
        real_freq = awg.awg_control.channels_config(ip_abaco_1,ip_abaco_2, \
                    config.awg_freq,phases_list,pri_list,pw_list,mode,fclk)
    else:
        mode = 0
        print("Starting sweep from",awg_freq," MHz with step of",config.step)
        for i in range(int(awg_freq),625,config.step):
            print("Actual frecuency:",i)
            real_freq = awg.awg_control.channels_config(ip_abaco_1,\
                        ip_abaco_2,i,phases_list,pri_list, \
```

```
170                             pw_list,mode,fclk)
171 #launch main
172 main(sys.argv[1:])
```