ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE LAS ETAPAS DE PROCESAMIENTO DIGITAL E INTERFAZ GRÁFICA DE UN ANALIZADOR DE ESPECTROS DE 10 MHZ A 3 GHZ EN BASE AL ALGORITMO SPLIT-RADIX ASIMÉTRICO DE 2048 PUNTOS.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

GUARQUILA GUARQUILA LEONARDO RICARDO

DIRECTOR: Ph.D. PABLO ANIBAL LUPERA MORILLO

Quito, marzo 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Leonardo Ricardo Guarquila Guarquila, bajo mi supervisión.

_____

Ph. D. PABLO ANIBAL LUPERA MORILLO
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Leonardo Ricardo Guarquila Guarquila, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

_____

Leonardo Ricardo Guarquila Guarquila

DEDICATORIA

A mi padre, madre y hermana que me han motivado cada día a ser mejor, ya que nunca me han dejado solo. Gracias por ayudarme a cumplir una meta más.

AGRADECIMIENTO

A mi padre Marcelo y a mi madre Nieves por haberme dado la vida y haberme educado, ya que con su trabajo, esfuerzo y dedicación lograron convertirme en el hombre que ahora soy.

A mi hermana Erika que siempre a estado a mi lado compartiendo buenos momentos.

A la Escuela Politécnica Nacional y profesores de la Facultad por haberme brindado su conocimiento que fueron esenciales para mi formación académica.
Al Ph.D. Pablo Lupera por su ayuda y apoyo para el desarrollo de este trabajo de titulación.

A mis primos, especialmente a Gabriel y Danny que siempre han estado junto a mí en los buenos y malos momentos.

A mis amigos, por haber compartido buenos momentos a lo largo de la carrera.

ÍNDICE DE CONTENIDO

AVAL I
DECLARACIÓN DE AUTORÍA II
DEDICATORIA III
AGRADECIMIENTO IV
ÍNDICE DE CONTENIDO V
RESUMEN VII
ABSTRACT VIII
1. INTRODUCCIÓN 1
1.1 PREGUNTA DE INVESTIGACIÓN 1
1.2 OBJETIVO GENERAL 1
1.3 OBJETIVOS ESPECÍFICOS 1
1.4 ALCANCE 2
1.5 MARCO TEÓRICO 2
1.5.1 FPGA [2] 2
1.5.2. TARJETA DE ENTRENAMIENTO XC7K325T-2FFG900C [3] 4
1.5.3. DIAGRAMA DE BLOQUES DEL RECEPTOR [4] 4
1.5.4. TRANSFORMADA DE FOURIER [8] 9
2. METODOLOGÍA 32
2.1. LENGUAJE VHDL [16] 32
2.1.1. LIBRERÍA [16] 32
2.1.2 ENTIDAD [16] 34
2.1.3. ARQUITECTURA [16] 34
2.1.4. TIPOS DE OBJETOS 37
2.1.5. TIPOS DE DATOS [17] 38
2.2. VIVADO DESING SUITE 39
2.2.1. DESARROLLO EN VIVADO DEL BLOQUE FFT 39
2.2.2. COMPONENTES Y PROCESOS QUE CONFORMAN EL BLOQUE FFT 39
2.2.3. BLOQUE FFT EN VIVADO 47
2.3. LENGUAJE PYHTON [21]. 54

2.3.1. MÓDULOS [22] 55
2.3.2. FUNCIONES EN PYTHON [23] 56
2.3.3. TIPOS DE DATOS [23] 57
2.4. SPYDER [24] 57
2.4.1. DESARROLLO DE LA INTERFAZ PARA EL ANALIZADOR DE ESPECTRO EN PYTHON 58
2.5 PROGRAMACIÓN DE LA FPGA EN VIVADO 60
2.5.1. INTERFAZ GRÁFICA EN MATLAB PARA VERIFICACIÓN DEL BLOQUE FFT DE LA FPGA 63
3. RESULTADOS Y DISCUSIÓN 65
3.1. INTERFAZ GRÁFICA EN PYHTON PARA EL ANALIZADOR DE ESPECTRO. 65
3.2. PRUEBAS 66
3.3. ANÁLISIS DE LOS RESULTADOS 73
4. CONCLUSIONES Y RECOMENDACIONES 76
4.1. CONCLUSIONES 76
4.2 RECOMENDACIONES 77
5. REFERENCIAS BIBLIOGRÁFICAS: 78
ANEXOS 80

RESUMEN

El presente trabajo tiene como objetivo implementar un algoritmo en el dominio de la frecuencia en una FPGA (Field Programmable Gate Arrays o Arreglo de Compuestas Programables en el Campo) para la determinación del espectro de señales periódicas, utilizando el algoritmo Split-Radix asimétrico y una interfaz gráfica en lenguaje Python.

Para esto, se generan muestras en Matlab en formato de texto las cuales se envían por comunicación serial a la FPGA y esta procesa los datos, enviando así, los resultados de la FFT (Fast Fourier Transform) obtenida mediante el algoritmo Split-Radix y presentando en la interfaz gráfica.

Los resultados de este trabajo muestran que el algoritmo Split-Radix tiene mayor eficiencia debido a que combina el algoritmo Radix-2 y Radix-4, realizando un menor número de operaciones complejas para obtener la FFT.

PALABRAS CLAVE: FFT, Split-Radix, analizador de espectros.

ABSTRACT

This research aims to implement an algorithm in the frequency domain in an FPGA (Field Programmable Gate Arrays) in order to determine the periodic signals' spectrum, using the asymmetric Split-Radix algorithm and the graphical interface in Python language.

For this, samples are generated in Matlab in text format. This samples are sent by serial communication to the FPGA and the data is processed. Then, the results of the FFT (Fast Fourier Transform), obtained through the Split-Radix algorithm, are presented in the graphical interface.

The results of this work show that the Split-Radix algorithm has higher efficiency because it combines the Radix-2 and Radix-4 algorithm, performing fewer complex operations to obtain the FFT.

KEYWORDS: FFT, Split-Radix, spectrum analyzer.

1. INTRODUCCIÓN

La reducción del consumo de energía requerida para la transmisión efectiva de las señales desde la estación base hacia los terminales de usuario en las redes de comunicación móvil, provocará una reducción de las emisiones de CO_2 a la atmósfera [1] y un uso eficiente de la energía utilizada.

La determinación de la transformada de Fourier permite implementar posteriormente un algoritmo que calcule la DoA (Direction of Arrival), para mejorar la eficiencia de la energía utilizada en la transmisión de señales inalámbricas.

1.1 PREGUNTA DE INVESTIGACIÓN

Una baja carga computacional para el procesamiento de señales y que sea implementado en hardware para la determinación del espectro y posteriormente el cálculo del DoA permite tener una mejor eficiencia energética, esto hace que el tema sea de interés científico. Con este antecedente se propone como pregunta de investigación lo siguiente: “¿Es posible implementar un algoritmo en el dominio de la frecuencia para determinar la FFT de señales periódicas?”.

1.2 OBJETIVO GENERAL

Implementar las etapas de procesamiento digital de un analizador de espectro con el algoritmo Split-Radix de 2048 puntos de una señal en el rango de 10 MHz a 3 GHz.

1.3 OBJETIVOS ESPECÍFICOS

- Estudiar el algoritmo Split-Radix para la determinación del espectro de una señal.
- Proponer el diagrama de bloques de los circuitos necesarios para el analizador de espectros.
- Implementar en lenguaje VHDL el algoritmo Split-Radix de 2048 puntos.
- Implementar la interfaz gráfica en Phyton para mostrar el espectro de la señal.
- Realizar pruebas del analizador de espectros con diferentes tipos de señales y en presencia de ruido.

1.4 ALCANCE

Diagrama de bloques de los circuitos necesarios para la implementación de un analizador de espectros, las etapas que se implementarán corresponden a las siguientes: procesamiento digital de las muestras para la obtención del espectro mediante el algoritmo Split-Radix y herramienta de presentación del espectro en una interfaz gráfica desarrollada en Phyton.

La determinación del espectro se realizará mediante la implementación del algoritmo Split-Radix asimétrico de 2048 puntos. El algoritmo se programará en lenguaje VHDL para que sea implementado posteriormente en una tarjeta FPGA, y tendrá como entrada las 2048 muestras de señales conocidas.

Una vez obtenido el espectro de la señal se tiene previsto simular el envío del resultado desde la tarjeta FPGA a una computadora por comunicación serial [9]; en el computador se mostrará el espectro de la señal recibida por medio de una interfaz gráfica desarrollada en Phyton. Por medio de la interfaz se tiene la posibilidad de variar los siguientes parámetros: frecuencia central, span, frecuencia de muestreo y resolución.

1.5 MARCO TEÓRICO

1.5.1 FPGA [2]

El FPGA es un dispositivo en el que un diseñador de sistemas digitales puede programar de manera que realice determinadas instrucciones, es decir, que su estructura interna puede ser reprogramada. La cantidad de circuitos que se incorpore en una FPGA depende del número de recursos utilizables del dispositivo.

Una FPGA cuenta con un arreglo de elementos y conexiones programables. Entre los elementos que componen un FPGA tenemos:

- Bloques de entrada y salida
- Bloques lógicos configurables
- Red de interconexión

En la figura 1.1 se muestra la arquitectura de una FPGA, mientras que en la figura 1.2 se detalla de mejor manera cada componente que conforma una FPGA.

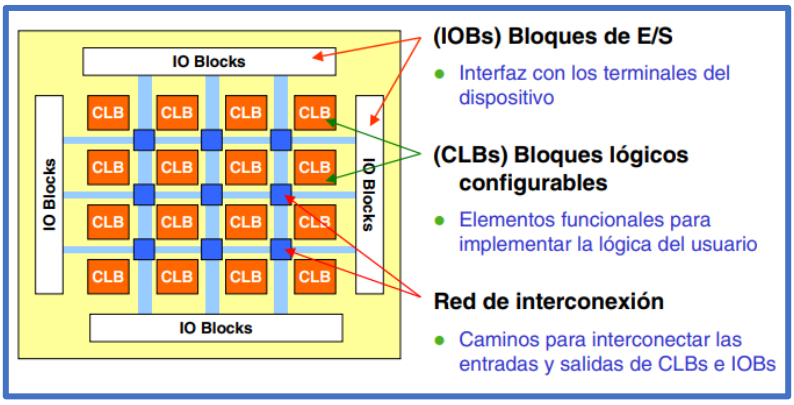


Figura 1.1. Arquitectura de una FPGA [2]

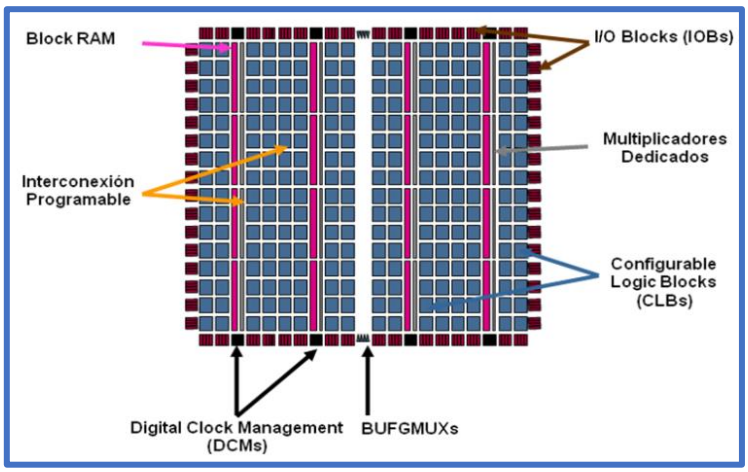


Figura 1.2. Estructura interna de una FPGA [2]

1.5.1.1. Bloques de entrada y salida

El bloque de entrada y salida (IOB) del FPGA permite la interconexión del arreglo interno con el mundo exterior. Estos bloques son configurables, es decir, el usuario decide si el bloque trabaja como entradas y/o salidas según se requiera [2].

1.5.1.2. Bloques lógicos configurables [2]

Una FPGA cuenta con un arreglo bidireccional de elementos los cuales corresponden al CLB (Bloque Lógico Configurable). Un CLB contiene varios lotes *(slices)*, los cuales cuentan con LUTs (tablas de consultas), flip-flops tipo D y multiplexores.

La LUT es una memoria que se puede usar para almacenar tablas de verdad dependiendo de la lógica digital que se implemente en un CLB, la unión de varios LUTs se emplea para crear RAMs distribuidas [2].

1.5.1.3. Red de interconexión

Son conexiones programables que definen internamente la ruta por la cual debe pasar la señal dentro de la FPGA. Realiza la conexión entre CLBs e IOBs, o entre CLBs.

1.5.2. TARJETA DE ENTRENAMIENTO XC7K325T-2FFG900C [3]

Para implementar el bloque FFT se utilizó la tarjeta Kintex-7, la cual se muestra en la figura 1.3.

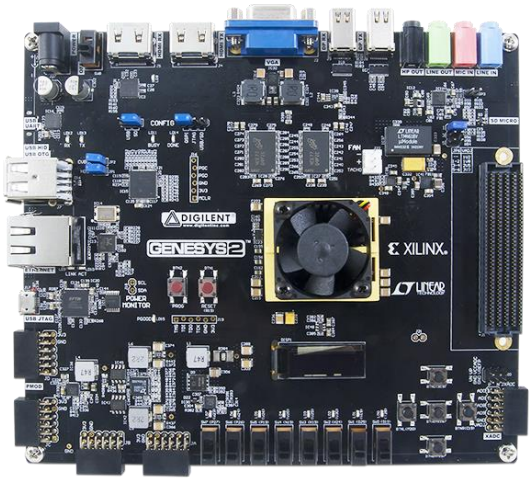


Figura 1.3. Tarjeta de entrenamiento XC7K325T-2FFG900C [5]

1.5.3. DIAGRAMA DE BLOQUES DEL RECEPTOR [4]

La señal de entrada al analizador de espectros necesita ser procesada previamente, para esto, se debe cambiar la frecuencia de la señal recibida trasladando a una frecuencia más baja mediante un mezclador de frecuencias. Es necesario conocer la frecuencia del oscilador local para establecer la frecuencia de la señal recibida y mostrarla correctamente en la interfaz del analizador. La intención de la traslación de frecuencia es realizar el proceso de muestreo de la señal a una frecuencia fija y no tener que estar configurando los parámetros de muestreo para cada una de las señales recibidas. Las etapas de radiofrecuencia que se deben llevar a cabo en el analizador de espectros se muestran en el diagrama de bloques de la figura 1.4.

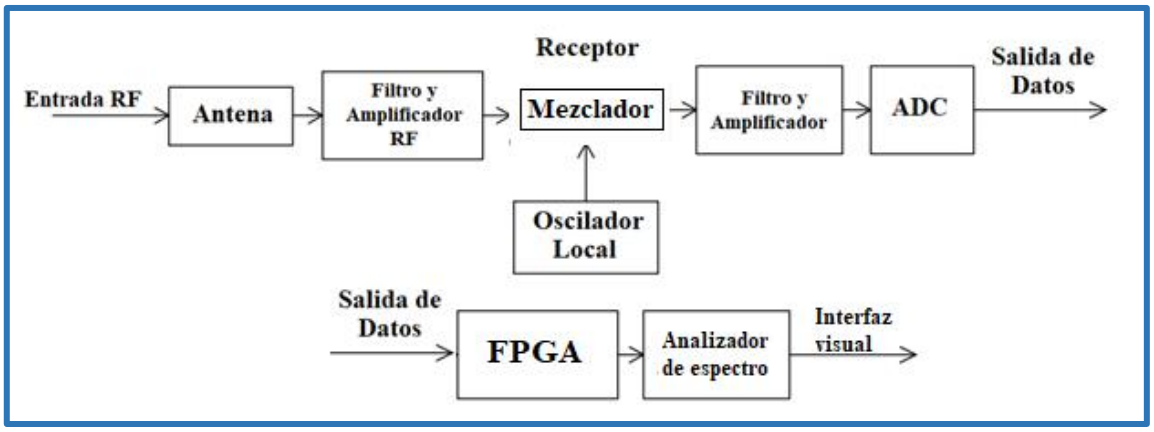


Figura 1.4. Diagrama de bloques del analizador de espectro

- **ANTENA:** Convierte las señales electromagnéticas en señales eléctricas que dependiendo de las dimensiones y forma de la misma opera en diferentes rangos de frecuencia.
- **FILTRO Y AMPLIFICADOR RF:** Aumenta el nivel de señal en voltaje para que pueda ser procesada posteriormente, además la señal es filtrada para eliminar parte del ruido que pueda haber en la señal recibida.
- **MEZCLADOR DE FRECUENCIA:** traslada en el dominio de la frecuencia la señal recibida dependiendo de la señal del Oscilador Local.
- **CONVERSOR ANÁLOGO A DIGITAL (ADC):** Muestrea la señal recibida analógica y convierte las muestras en señal digital.
- **FPGA:** Ejecuta el algoritmo Split-Radix para obtener la FFT de las muestras.
- **ANALIZADOR DE ESPECTRO:** Recibe los datos obtenidos de la FPGA y muestra los valores visualmente convertidos a niveles de potencia.

1.5.3.1 Mezclador de frecuencia [3], [5]

El mezclador es utilizado para pasar señales de banda base a banda de paso o de banda de paso a banda base. El mezclador de frecuencia en un dispositivo no lineal que modifica el espectro de las señales que se aplican a la entrada y entrega una señal en otra frecuencia dependiendo de las señales de entrada. Por lo general entrega dos frecuencias, una igual a la suma de las dos frecuencias de entrada y una igual a la diferencia de estas. Dependiendo del tipo de mezclador utilizado es posible que entregue más de dos frecuencias llamadas armónicos, esto es posible eliminar utilizando filtros [5], [4].

El mezclador realiza una multiplicación en el dominio del tiempo para trasladar el espectro de las señales de entrada. El mezclador dispone de tres puertos: 2 de entrada y 1 de salida. En un puerto de entrada se debe conectar la señal que se desea trasladar en frecuencia, mientras que, en el otro puerto de entrada se debe aplicar la señal del oscilador local (OL). En la figura 1.5 se muestra el símbolo de un mezclador de frecuencia.

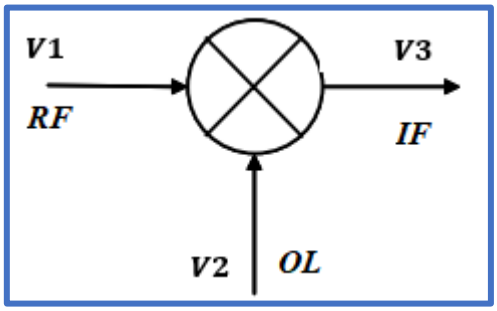


Figura 1.5. Símbolo del mezclador de frecuencia

Matemáticamente la operación del mezclador se presenta a continuación:

$$\boldsymbol{V1 = A_1 cos(w_1 t)} \tag{1.1}$$

$$\boldsymbol{V2 = A_2 cos(w_2 t)} \tag{1.2}$$

$$\boldsymbol{V3 = A_1 cos(w_1 t).A_2 cos(w_2 t) = \frac{cos((w_1 + w_2)t) + cos((w_1 - w_2)t)}{2} * A_1 A_2} \tag{1.3}$$

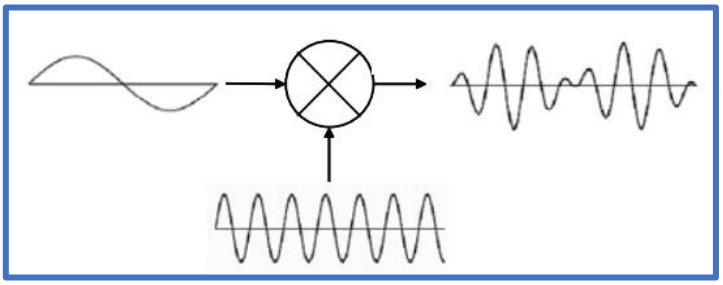


Figura 1.6. Mezclador en el dominio del tiempo

Existen tres tipos de mezcladores de frecuencia [5]:

- **Mezcladores pasivos:** para mezclar las frecuencias utilizan diodos.
- **Mezcladores activos:** utilizan transistores bipolares o de efecto de campo.
- **Mezcladores conmutados:** donde la señal del oscilador local es una señal de pulsos o una señal con amplitud muy grande para realizar estados de conducción y no conducción.

En el presente trabajo se realiza el estudio de un mezclador pasivo ya que proporciona un buen aislamiento de las señales que ingresan al mismo.

1.5.3.2 Mezclador pasivo [5]

Un mezclador pasivo doblemente balanceado utiliza un puente de diodos que proporciona un buen aislamiento entre las señales del oscilador local (OL), radio frecuencia (RF) y frecuencia intermedia (FI). El circuito de un mezclador doblemente balanceado se muestra en la figura 1.7 diseñado por mini-circuit [6].

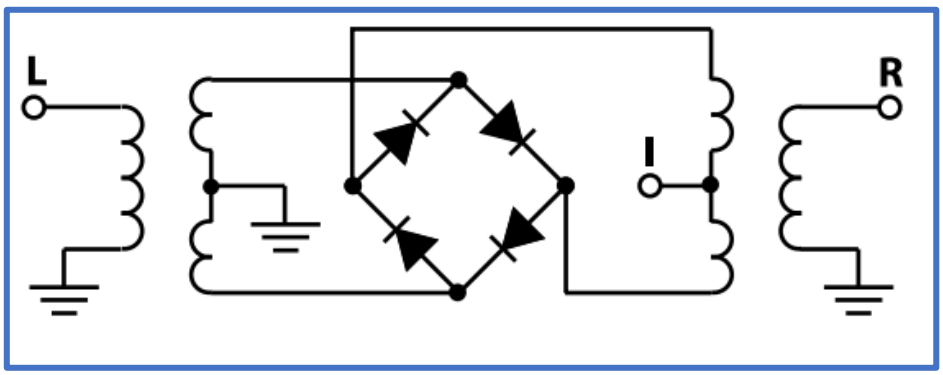


Figura 1.7. Circuito mezclador [8]

El mezclador de mostrado en la figura 1.7 tiene baja pérdida de conversión[1] igual a 6.65dB y un ancho de banda de 300MHz a 4300MHz.

1.5.3.3 Conversor analógico a digital [7].

El proceso para convertir una señal analógica a digital parte de muestrear una señal continua y representar en una señal discreta equivalente, de esta manera si aplicamos el proceso inverso se pueda recuperar la señal original. Para convertir en una señal digital se llevan a cabo tres procesos: muestreo, cuantificación y codificación [7].

- **MUESTREO:** Proceso mediante el cual se obtienen valores definidos finitos de una señal continua con valores de amplitud infinitos. Para obtener dichos valores se necesita de una frecuencia de muestreo, tal que sea mayor a dos veces la frecuencia máxima que se desea muestrear, esto en base al teorema de Nyquist[2], tal como se representa en la figura 1.8.

[1] Pérdida de conversión: nivel en que la señal de salida se amplifica o atenúa.
[2] Teorema de Nyquist: para discretizar una señal es necesario muestrear con una frecuencia mayor a 2 veces la frecuencia máxima de la señal continua en el tiempo.

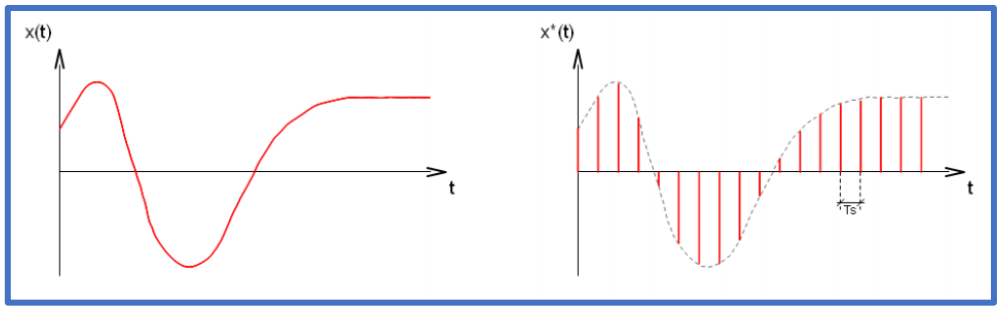


Figura 1.8. Muestreo de una señal [9]

- **CUANTIFICACIÓN:** Esta fase clasifica las muestras en intervalos de cuantificación, ya sea entre negativos y positivos. Los intervalos de cuantificación dependen de la cantidad de bits con que puede muestrear el convertidor analógico a digital. La figura 1.9 muestra la manera en que se cuantifican las muestras.

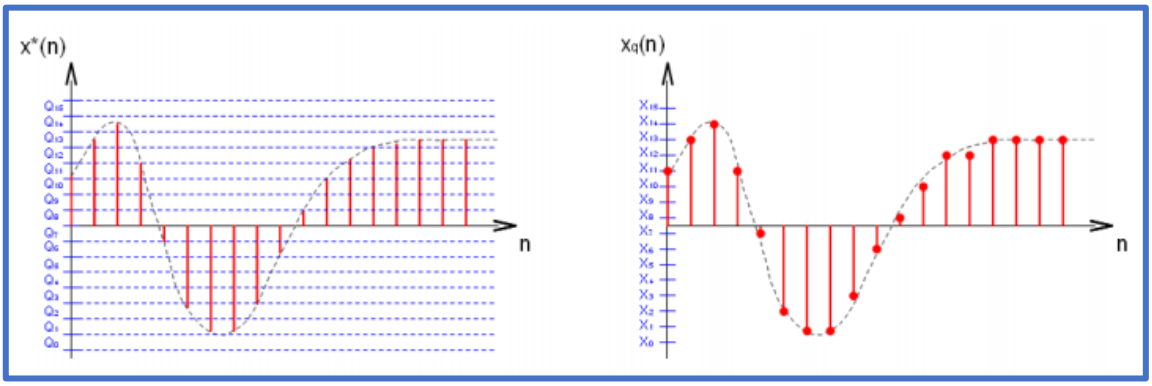


Figura 1.9. Cuantificación de las muestras [9]

- **CODIFICACIÓN:** Este proceso consiste en asignar un código binario dependiendo de la cuantificación de las muestras, la figura 1.10 muestra cómo se realiza la codificación dependiendo del intervalo de cuantificación.

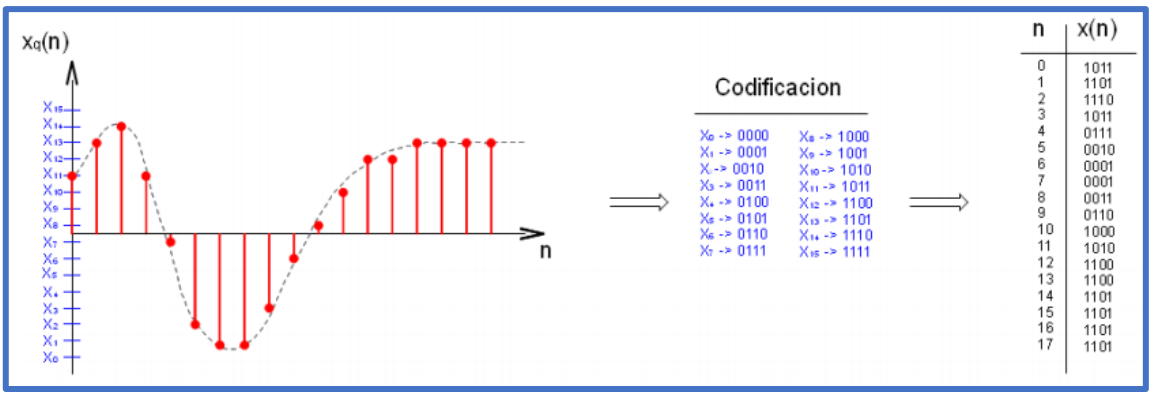


Figura 1.10. Codificación de las muestras [9]

Una vez codificadas las muestras son enviadas y recibidas por la FPGA para realizar el cálculo de la FFT.

1.5.4. TRANSFORMADA DE FOURIER [8]

La transformada de Fourier realiza procesos matemáticos que permiten cambiar la señal del dominio del tiempo al dominio de la frecuencia. La señal a la que se realiza esta operación puede ser continua o discreta y periódica o aperiódica, la combinación de estas da como resultado cuatro variantes de la transformada de Fourier, las cuales se presentan en la siguiente figura 1.11.

La transformada discreta de Fourier (DFT) puede ser implementada en tarjetas de procesamiento, debido a que las tarjetas almacenan un número finito de muestras para ser procesadas.

Dominio de tiempo	Periódica	**No periódica**	
Continua	Series de Fourier $X[k]=\frac{1}{T}\int_{<T>}x(t)e^{-jk\Omega_0 t}dt$ $x(t)=\sum_{k=-\infty}^{\infty}X[k]e^{jk\Omega_0 t}$	Transformada de Fourier $x(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty}X(j\Omega)e^{j\Omega t}d\Omega$ $X(j\Omega)=\int_{-\infty}^{\infty}x(t)e^{-j\Omega t}dt$	No periódica
Discreta	**Transformada Discreta de Fourier (DFT)** $x[n]=\sum_{k=<N>}X[k]e^{jk\omega_0 n}$ $X[k]=\frac{1}{N}\sum_{n=<N>}x[n]e^{-jk\omega_0 n}$	**Transformada de Fourier de Tiempo discreto (DTFT)** $x[n]=\frac{1}{2\pi}\int_{-\pi}^{\pi}X(e^{j\omega})e^{j\omega n}d\omega$ $X(e^{j\omega})=\sum_{n=-\infty}^{\infty}x[n]e^{-j\omega n}$	Periódica
	Discreta	Continua	**Dominio de la frecuencia**

Figura 1.11. Variaciones de la transformada de Fourier

1.5.4.1. Transformada discreta de Fourier [8]

La ecuación para el cálculo de la transformada discreta de Fourier se presenta en la ecuación (1.4).

$$X(k)=\sum_{n=0}^{N-1}x(n).e^{-j2\pi kn/N}, k=0,1,2,\dots,N-1 \quad (1.4)$$

Donde $x(n)$ representa la secuencia de N muestras de la señal en el dominio del tiempo discreto y $X(k)$ representa la secuencia de N frecuencias discretas en el dominio de la frecuencia obtenida mediante el cálculo de la DFT.

1.5.4.2. TRANSFORMADA RÁPIDA DE FOURIER (FFT) [9]

La transformada rápida de Fourier se basa en la descomposición de una DFT de tamaño N en varias DFTs de menor tamaño de manera recursiva. Esta descomposición permite tener dos algoritmos para el cálculo de la DFT: decimación en tiempo (DIT) [10], cuando se divide la secuencia $x(n)$ y decimación en frecuencia (DIF) [10], cuando se divide la secuencia $X(k)$.

La forma en la cual se descomponen las muestras resulta eficaz cuando N es factorizable, de esta manera se puede expresar como el producto de p números donde: $N = n_1.n_2.n_3 \ldots n_p$. si $n_1 = n_2 = n_3 = \ldots = n_p = n$, entonces $N = n^p$. Donde n es conocido como la base o la raíz del algoritmo. Así, por ejemplo, si $n = 2$ se conoce como Radix-2.

Se puede rellenar de ceros las muestras en el caso que *N* no sea factorizable, o a su vez utilizar un algoritmo FFT que no tenga restricción de que *N* sea factorizable, estos algoritmos sin restricciones pueden ser más complejos de implementar [11].

En este capítulo se detallarán los algoritmos Radix-2 y Radix-4, que son los algoritmos más conocidos a ser implementados computacionalmente, además de, comprender la importancia de estos algoritmos que combinados dan como resultado el algoritmo Split-Radix.

1.5.4.2.1. Algoritmo Radix-2 con decimación en tiempo [12]

Este algoritmo se basa en dividir en dos subsecuencias de longitud N/2 a las N muestras de entrada x(n); la una subsecuencia contiene las muestras pares, mientras que la otra subsecuencia las muestras impares, obteniendo así la ecuación (1.5) [12]:

$$X(k) = \sum_{n\ par} x(n) * W_N^{kn} + \sum_{n\ impar} x(n) * W_N^{kn} \qquad (1.5)$$

Donde el término $W_N^{kn} = e^{-j2*\pi kn/N}$ representa el factor de giro, dicho factor por su naturaleza nunca cambia en magnitud únicamente en fase.

El factor de giro se usa en la mayoría de los algoritmos FFT, debido que por sus propiedades de simetría y periodicidad, resulta bastante útil, ya que aplicándolo se logra disminuir el número de operaciones para el cálculo de la DFT [12].

Las propiedades del factor de giro son las siguientes:

- Simetría compleja conjugada

$$W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^* \tag{1.6}$$

- Periodicidad en n y k

$$W_N^{k(N+n)} = W_N^{kn} = W_N^{(k+N)n} \tag{1.7}$$

Si se realizan los siguientes cambios de variables en la ecuación (2) $n = 2r$ en la sumatoria de las muestras pares y $n = 2r$+1 en la sumatoria de muestras impares, obtenemos la ecuación (1.9)[12]:

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) * W_N^{2kn} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) * W_N^{k(2r+1)} \tag{1.8}$$

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) * W_N^{2kn} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) * W_N^{k2r} * W_N^{k} \tag{1.9}$$

En la ecuación (1.9) se observa que el término W_N^k de la segunda sumatoria no se encuentra en función de r, por lo que es factible sacarlo del sumatorio.

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) * W_N^{2kn} + W_N^k * \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) * W_N^{k2r} \tag{1.10}$$

De la ecuación (1.10) cada sumatoria representa a una DFT de N/2, en la primera sumatoria se utilizan las muestras pares, mientras que con la segunda sumatoria se utilizan las muestras impares. Dicha explicación puede ser más visible si a la variable k se iguala a cero, teniendo así:

$$G(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) * W_N^{2kn}; x(0) + x(2) + x(4) + \cdots (muestras\ pares) \tag{1.11}$$

$$H(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) * W_N^{k2r}; x(1) + x(3) + x(5) + \cdots (muestras\ impares) \qquad (1.12)$$

Es así como el algoritmo Radix-2 ha dividido en dos DFTs de longitud N/2, partiendo a la mitad las muestras de la secuencia original.

La figura 1.12 representa la descomposición de las muestras de entrada con el algoritmo Radix-2 para una longitud x(n) igual a 8 muestras.

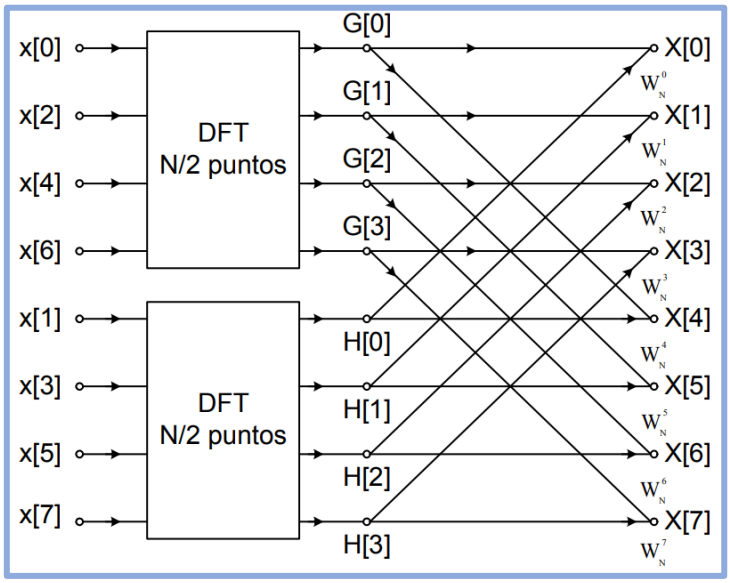


Figura 1.12. Descomposición en dos DFTs de (N/2) puntos a partir de una DFT de N puntos [11]

En la figura 1.12 se observa que las primeras 4 muestras corresponden a las muestras pares, las cuales se agrupan y se procesan en la primera DFT correspondiente a una DFT de 4 puntos, obteniendo así la secuencia $G(k)$. De igual manera se ejecutan las muestras impares con las cuales se obtiene la secuencia $H(k)$.

Los factores de giro que se observan en la figura 1.12 corresponden al W_N^k de la ecuación (4) y que junto a las secuencias $G(k)$ y $H(k)$ dan como resultado $X(k)$. Por periodicidad de los factores de giro tanto $G(k)$ y $H(k)$ son periódicas en k cada *N* puntos.

El primer diezmado realizado sobre la DFT de N puntos se puede repetir sobre las nuevas DFT de N/2 puntos obtenidas, de esta manera el algoritmo se repite recursivamente y debido a que N es potencia de base 2 este proceso terminará en el momento que se obtengan varias DFT de 2 puntos. Este proceso de recursividad se muestra en la figura 1.13, en la cual se observa como continúa dicho proceso para N=8.

La figura 1.13 ilustra 4 DFTs de dos puntos; las dos primeras se derivan de la primera DFT de cuatro puntos que agrupa las muestras pares, mientras que las dos DFTs restantes de 2 puntos se obtienen a partir de la segunda DFT de 4 puntos, la cual agrupa las muestras impares [9].

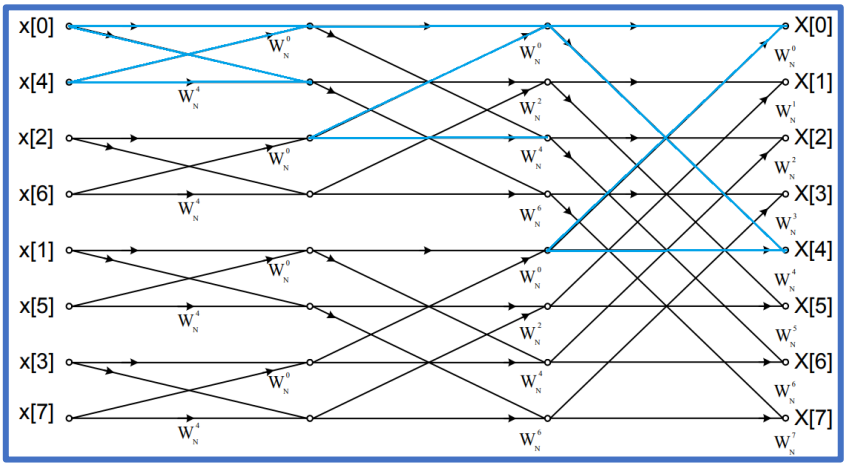


Figura 1.13. Descomposición de una DFT de 8 puntos en 4 DFTs de 2 puntos [11]

Observando la figura 1.13 se puede concluir que el número de etapas del algoritmo Radix-2 es igual a $\log_2 N$; por ejemplo, para el caso de N=8 el número de etapas presentes en este algoritmo es igual a 3. De igual manera se observa que en cada etapa existen N sumas y multiplicaciones complejas, las cuales se representan en una estructura llamada “mariposa” (marcada con líneas celestes). En la figura 1.14 se observa la estructura de una mariposa.

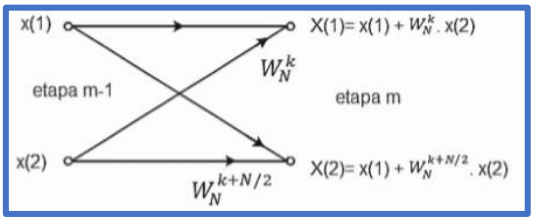


Figura 1.14. Estructura genérica de una mariposa del algoritmo Radix-2 [12]

Cada mariposa incluye dos sumas y dos productos complejos. Aprovechando la simetría y la periodicidad del factor de giro es posible reducir el número de operaciones complejas aplicando lo siguiente:

$$W_N^{N/2} = e^{-j\left(\frac{2\pi}{N}\right)\left(\frac{N}{2}\right)} = e^{-j\pi} = -1 \qquad (1.13)$$

Por lo tanto, el factor de giro $W_N^{k+N/2}$ se puede reducir a $-W_N^k$.

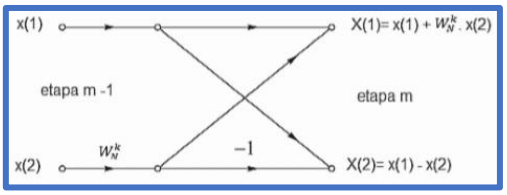


Figura 1.15. Estructura de una mariposa del algoritmo Radix-2 simplificada [12]

En la figura 1.15 se observa dicha simplificación en la estructura de la mariposa. Con ello la estructura de la mariposa solo abarcaría una multiplicación compleja en vez de dos. Con ello se reduce el número total de productos complejos que se realiza con el algoritmo Radix-2 de $\mathbf{N} * \mathbf{log_2}\, \boldsymbol{N}$ a $\frac{\mathbf{N}}{\mathbf{2}} * \mathbf{log_2}\, \boldsymbol{N}$ multiplicaciones complejas, pero el número de sumas complejas sigue siendo $\mathbf{N} * \mathbf{log_2}\, \boldsymbol{N}$.

Aplicando la estructura simplificada de la mariposa se obtiene la figura 1.16, la cual ilustra el algoritmo completo de Radix-2 para un número de muestras igual a 8 puntos, en el cual se observa claramente la reducción del número de operaciones.

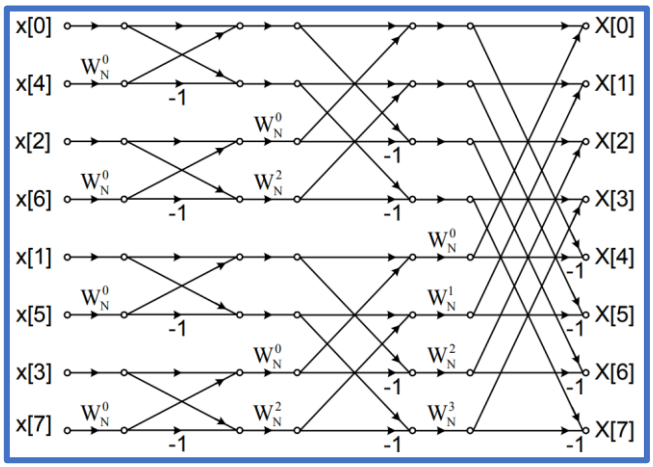


Figura 1.16. Grafo completo del algoritmo Radix-2 DIT con una estructura de mariposa simplificada para N=8 [12]

La figura 1.16 se observa que el orden de las muestras de entrada, es decir la secuencia de x(n) se encuentran en un orden diferente a las muestras de salida X(k) que se encuentran en orden natural. Esta característica se debe a la naturaleza recursiva del algoritmo, ya que separa de manera repetitiva las muestras pares de las impares de los datos de entrada. Es por ello que antes de empezar el algoritmo es necesario que la secuencia de entrada se encuentre ordenada tal y como se muestra en la figura 1.16, por ello es necesario realizar un proceso de reordenamiento de x(n), ya que por defecto los datos de entrada llegarán en orden natural. El orden en el que se deben presentar las muestras de entrada se conoce como orden de “bit inverso” [10], ya que es el resultado

de invertir los bits de la dirección de memoria en la que se encuentran los datos. La tabla 1.1 muestra este proceso para una secuencia de longitud 8.

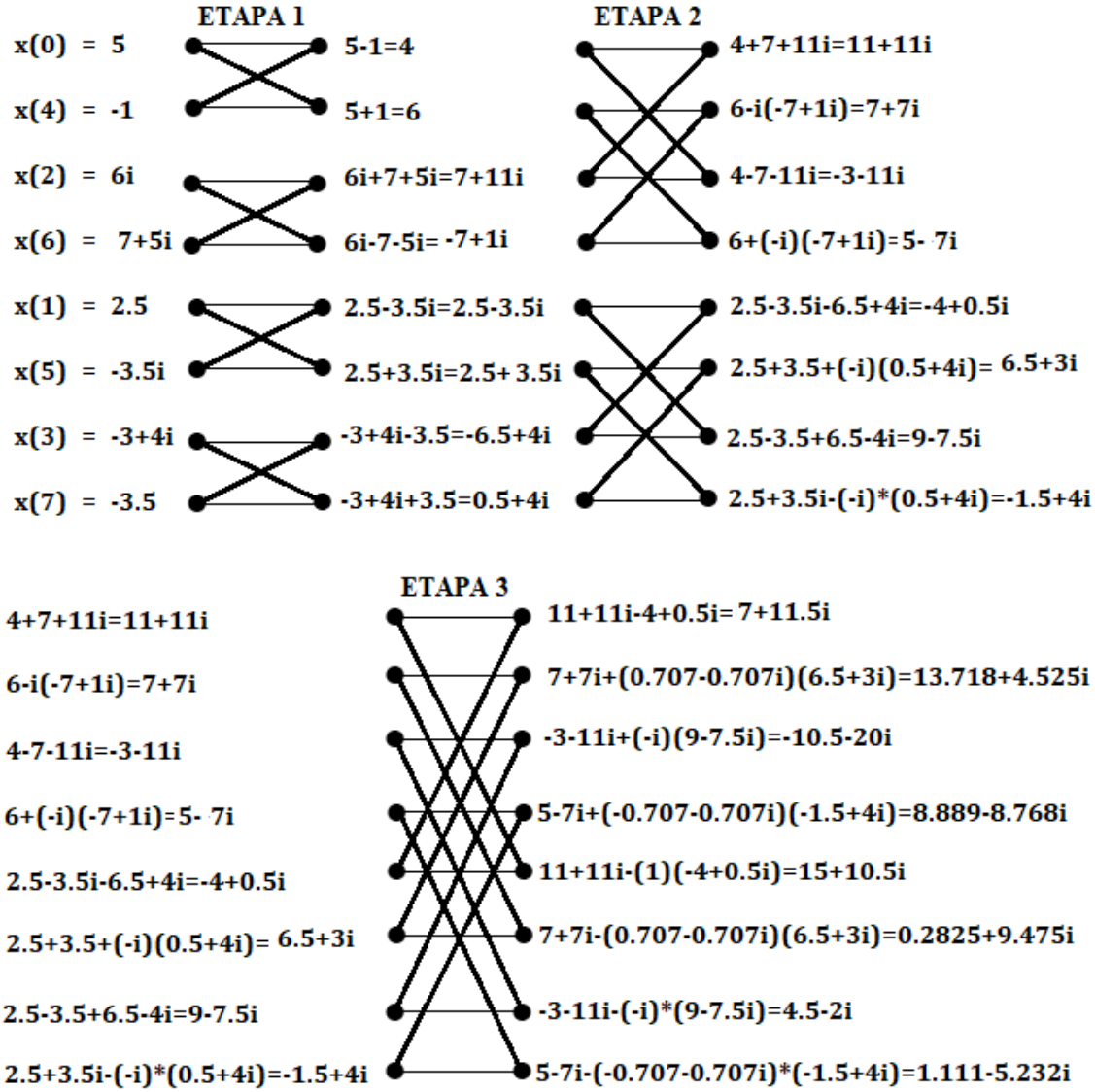
Tabla 1.1. Reordenamiento de bit inverso [13]

x(n) en orden natural	Dirección de memoria	Inversión de los bits de memoria	x(n) en orden de bit inverso
x(0)	000	000	x(0)
x(1)	001	100	x(4)
x(2)	010	010	x(2)
x(3)	011	110	x(6)
x(4)	100	001	x(1)
x(5)	101	101	x(5)
x(6)	110	011	x(3)
x(7)	111	111	x(7)

EJEMPLO RADIX-2 DIT DE OCHO PUNTOS

Ordenamiento en bit inverso

x(0) = 5	**x(0) = 5**
x(1) = 2.5	**x(4) = -1**
x(2) = 6i	**x(2) = 6i**
x(3) = -3+4i	**x(6) = 7+5i**
x(4) = -1	**x(1) = 2.5**
x(5) = -3.5i	**x(5) = -3.5i**
x(6) = 7+5i	**x(3) = -3+4i**
x(7) = -3.5	**x(7) = -3.5**



Respuesta

$X[0] = 7 + 11.5i$

$X[1] = 13.718 + 4.525i$

$X[2] = -10.5 - 20i$

$X[3] = 8.889 - 8.768i$

$X[4] = 15 + 10.5i$

$X[5] = 0.2825 + 9.475i$

$X[6] = 4.5 - 2i$

$X[7] = 1.111 - 5.232i$

1.5.4.2.2. Algoritmo Radix-2 con decimación en frecuencia [12].

Si en lugar de dividir la secuencia de entrada dividimos la secuencia de salida, la fórmula de la ecuación (1.14), la cual representa la fórmula general de la DFT, se la puede expresar en dos ecuaciones una para el cálculo de los puntos pares ecuación (1.15) y otro para el cálculo de los puntos impares ecuación (1.16) [12].

$$X(k) = \sum_{n=0}^{N-1} x(n) * e^{-j2\pi kn/N}, \quad k = 0,1,2,3,\dots,N-1 \tag{1.14}$$

$$X(2k) = \sum_{n=0}^{N/2-1} \left(x(n) + x\left(n + \frac{N}{2}\right) \right) * W_{N/2}^{kn}, \quad k = 0,1,2,\dots,\frac{N}{2} - 1 \tag{1.15}$$

$$X(2k+1) = \sum_{n=0}^{N/2-1} ([x(n) - x\left(n + \frac{N}{2}\right)] * W_N^n) * W_{N/2}^{kn}, \quad k = 0,1,2,\dots,\frac{N}{2} - 1 \tag{1.16}$$

Si llamamos $g(n) = x(n) + x(n + \frac{N}{2})$ y $h(n) = \left[x(n) - x\left(n + \frac{N}{2}\right)\right] * W_N^n$; para realizar el cómputo de la DFT, primero se deben generar dichas secuencias y finalmente aplicar una DFT de N/2 puntos, tal como se observa en la figura 1.17.

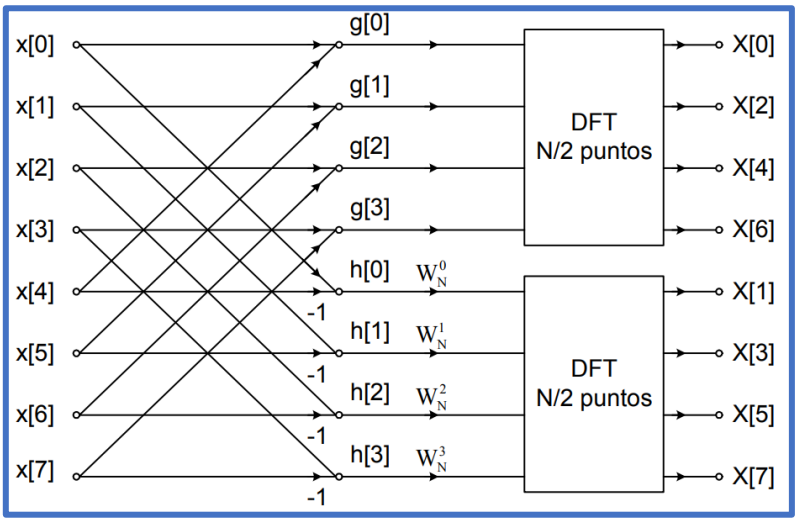


Figura 1.17. Descomposición inicial de una DFT de N=8 en dos DFTs de longitud N/2 [12]

De igual forma que en la decimación en tiempo este proceso es recursivo, es decir, que se repite sucesivamente hasta llegar a tener varias DFTs de 2 puntos, una vez alcanzadas dichas DFTs el algoritmo con decimación en frecuencia termina. Este proceso repetitivo se puede observar en la figura 1.18, en la cual se tiene que en la última etapa únicamente se presentan las DFTs de 2 puntos.

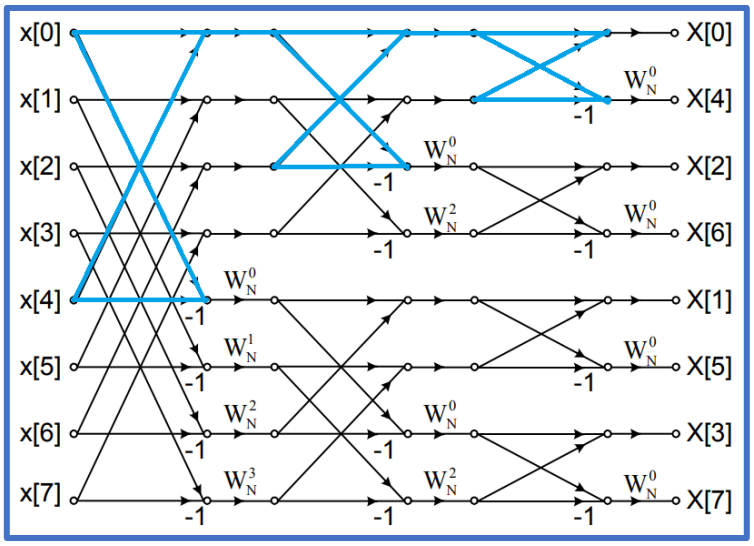
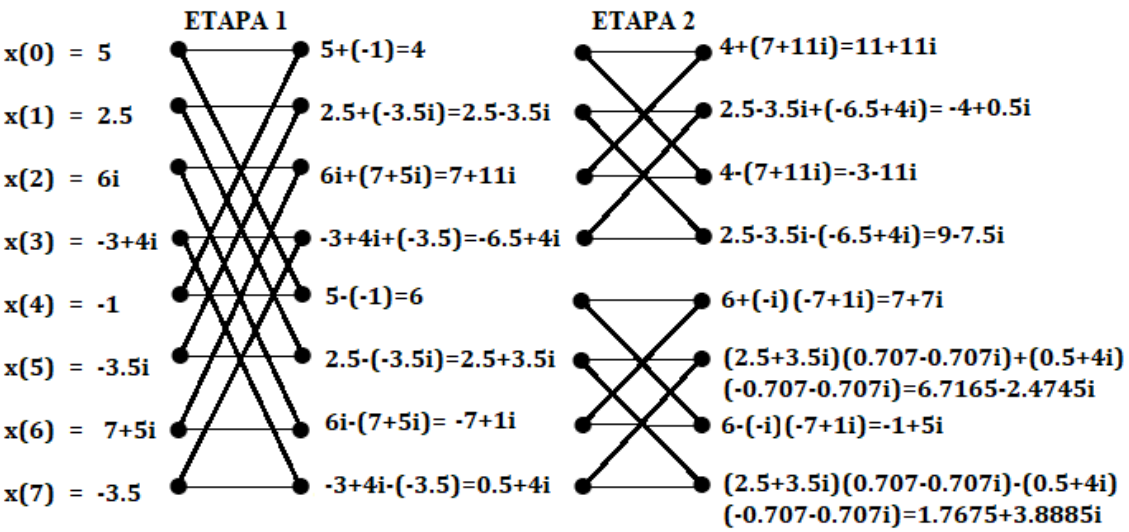
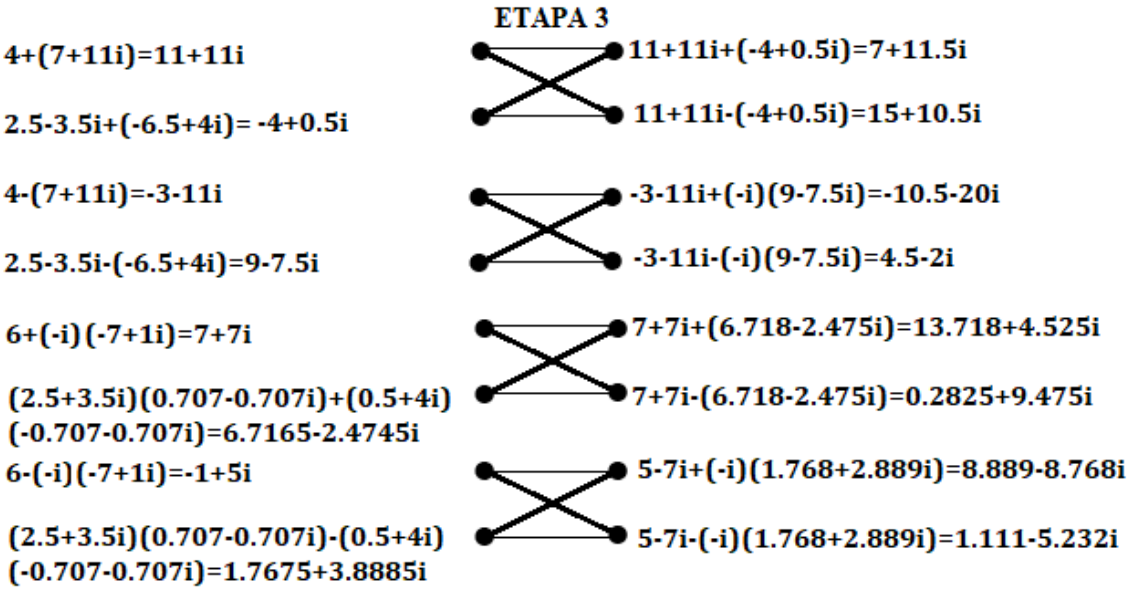


Figura 1.18. Grafo completo del algoritmo Radix-2 DIF con una estructura de mariposa simplificada para N=8 [12]

De la figura 1.18 se aprecia que existe una estructura que se repite en cada etapa, la misma que corresponde a una mariposa Radix-2 DIF, que se encuentra remarcada con líneas celestes. De igual manera, si se invierte el sentido de las flechas y se cambia las entradas por las salidas se llegaría a tener la estructura de una mariposa Radix-2 DIT. Además, se observa que las entradas están en orden natural, mientras que las salidas se encuentran en orden de bit inverso, contrario a lo que se tenía en el algoritmo Radix-2 DIT [12].

EJEMPLO RADIX-2 DIF DE OCHO PUNTOS





Respuesta

Ordenamiento en bit inverso

$$\boldsymbol{X[0] = 7 + 11.5i}$$

$$\boldsymbol{X[1] = 13.718 + 4.525i}$$
$$\boldsymbol{X[2] = -10.5 - 20i}$$
$$\boldsymbol{X[3] = 8.889 - 8.768i}$$
$$\boldsymbol{X[4] = 15 + 10.5i}$$
$$\boldsymbol{X[5] = 0.2825 + 9.475i}$$
$$\boldsymbol{X[6] = 4.5 - 2i}$$
$$\boldsymbol{X[7] = 1.111 - 5.232i}$$

1.5.4.2.3. Algoritmo Radix-4 con decimación en tiempo (dit) [13]

Para ejecutar el algoritmo Radix-4, la longitud de $x(n)$ debe ser potencia de 4, es así, $N = 4^p$, donde p corresponde al número de etapas que contiene el algoritmo Radix-4, de esta manera se obtienen las siguientes fórmulas si reemplazamos N en la ecuación general de la DFT dada por (1.4).

$$X(k) = \sum_{n=0}^{\frac{N}{4}-1} x(4n).W_{n/4}^{nk} + e^{-j2\pi k/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+1).W_{n/4}^{nk} + e^{-j2\pi(2k)/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+2).W_{n/4}^{nk} + e^{-j2\pi k/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n3).W_{n/4}^{nk} \quad (1.17)$$

$$X(k+N/4) = \sum_{n=0}^{\frac{N}{4}-1} x(4n).W_{n/4}^{nk} - je^{-j2\pi k/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+1).W_{n/4}^{nk} - e^{-j2\pi(2k)/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+2).W_{n/4}^{nk} + je^{-j2\pi k/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+3).W_{n/4}^{nk} \quad (1.18)$$

$$X(k+N/2) = \sum_{n=0}^{\frac{N}{4}-1} x(4n).W_{n/4}^{nk} - e^{-j2\pi k/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+1).W_{n/4}^{nk} + e^{-j2\pi(2k)/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+2).W_{n/4}^{nk} - e^{-j2\pi k/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+3).W_{n/4}^{nk} \quad (1.19)$$

$$X(k+3N/4) = \sum_{n=0}^{\frac{N}{4}-1} x(4n).W_{n/4}^{nk} + je^{-j2\pi k/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+1).W_{n/4}^{nk} - e^{-j2\pi(2k)/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+2).W_{n/4}^{nk} - je^{-j2\pi(3k)/N}.\sum_{n=0}^{\frac{N}{4}-1} x(4n+3).W_{n/4}^{nk} \quad (1.20)$$

$$k = 0,1,2,\dots,\frac{N}{4}-1$$

Las cuatro sumatorias que se encuentran dentro de cada ecuación representan las cuatro DFTs de longitud de *N/4* que se obtienen al dividir las muestras *x(n)* y así sucesivamente hasta llegar a tener varias DFT de longitud 4, en este momento se han utilizado todas las muestras y termina la decimación en tiempo. En la figura 1.19 se presenta el grafo del algoritmo Radix-4 DIT con muestras de longitud 16 en la que se remarca de color naranja la primera mariposa de la segunda etapa.

Los signos de los sumatorios de la ecuación (1.20) son *(1, j, -1, -j)*, los cuales se encuentran remarcados de color verde en la figura 1.20, esto significa que cada ecuación

tiene como signos de los sumatorios el recuadro remarcado de color naranja. Así mismo, los vectores de giro de cada sumatorio se encuentran remarcados de color púrpura.

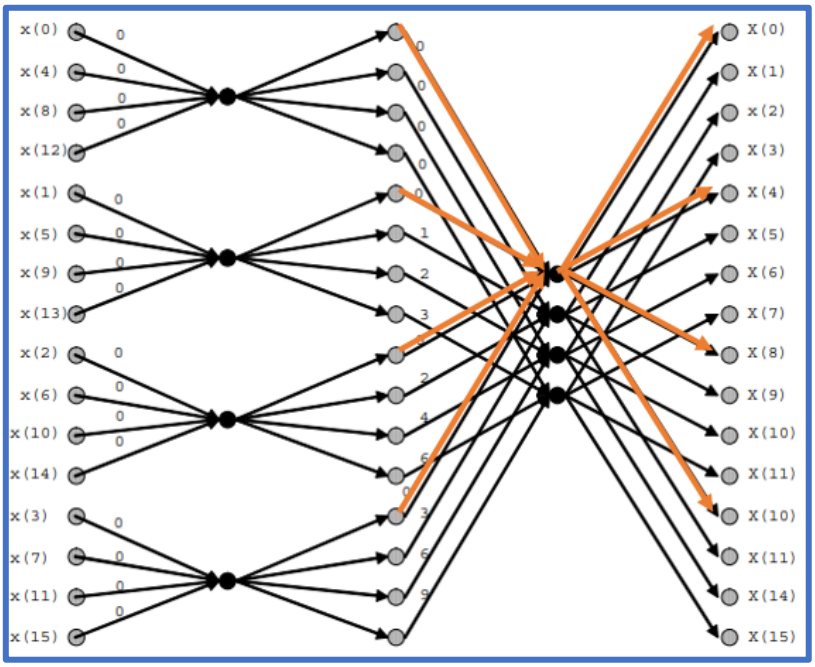


Figura 1.19. Grafo completo del algoritmo Radix-4 DIT para N=16 [14]

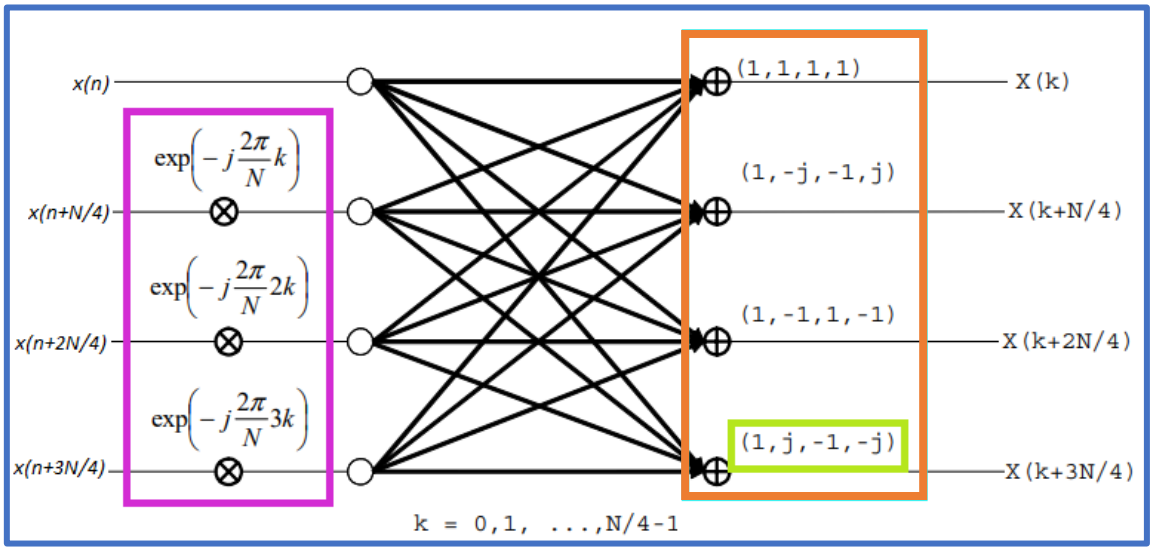


Figura 1.20. Grafo de la Mariposa Radix-4 DIT [13]

Si se reducen las expresiones de los vectores de giro de la figura 1.20, se obtiene un grafo de la mariposa Radix-4 más comprimida, como se observa en la figura 1.21, teniendo así una mejor apreciación del grafo del algoritmo.

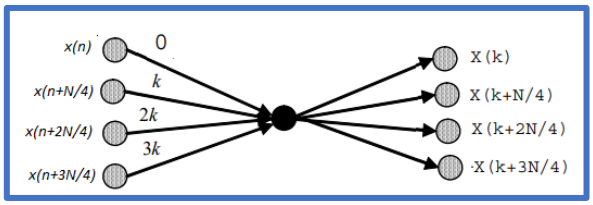


Figura 1.21. Grafo comprimido de la Mariposa Radix-4 DIT [13]

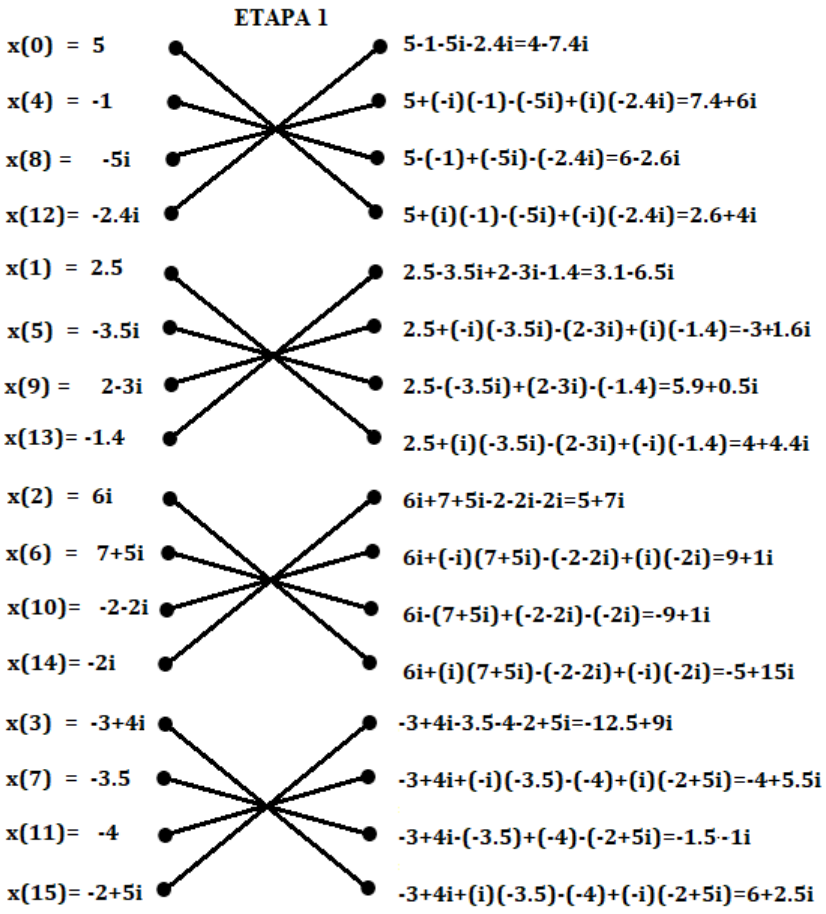
En la figura 1.19 se observa que las entradas *x(n)* para el algoritmo Radix-4 DIT de las mariposas de la primera etapa se encuentran en orden de bit inverso y la salida se encuentra en orden de bit natural. En la tabla 1.2 se puede observar el ordenamiento para 16 puntos de una secuencia *x(n).* Para representar la posición de las muestras *x(n)* se necesitan 2 dígitos debido a que tenemos 16 muestras y su base es 4. De esta manera si tomamos la posición 4 en base decimal su equivalente en base 4 es 10, para obtener el bit inverso se invierten estos dígitos de manera que obtenemos 01 en base 4 y transformando su equivalente a base decimal es el número 1. Si realizamos este proceso con todas las muestras *x(n)* se logra el ordenamiento de la posición de cada muestra tomada.

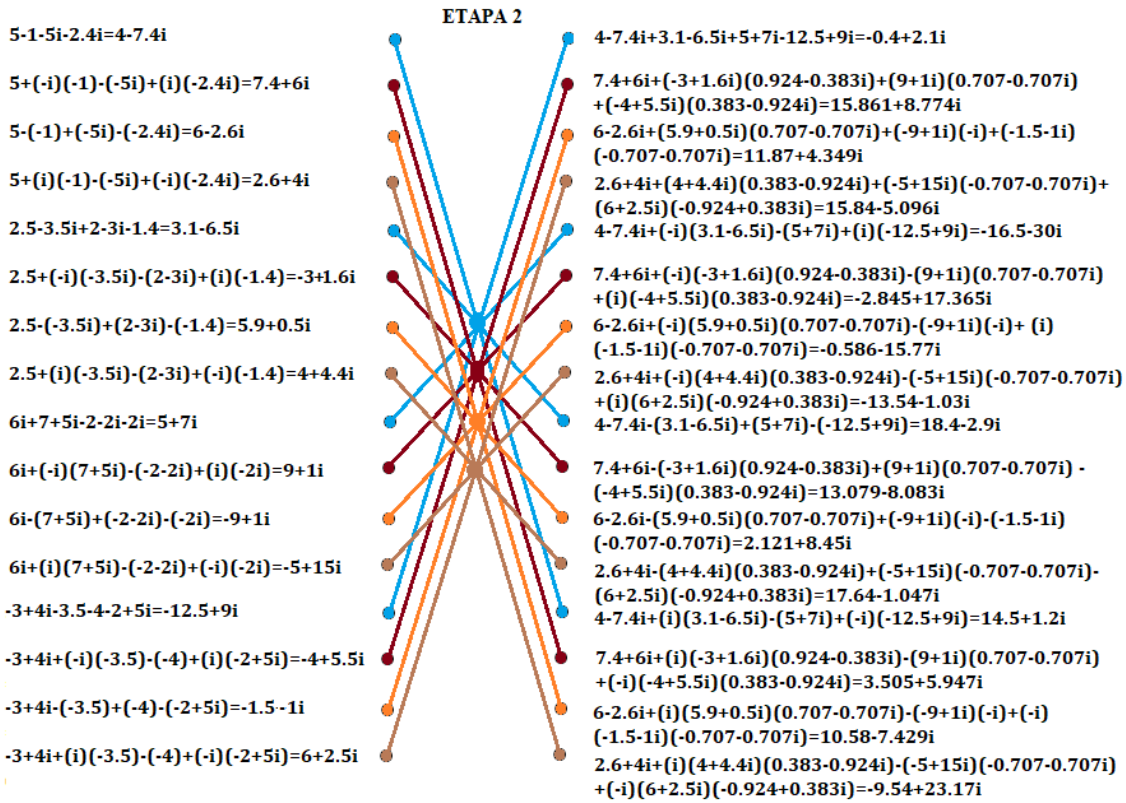
Tabla 1.2. Reordenamiento de bit inverso para mariposa Radix-4 DIT [13]

Orden en bit natural, base 10	Orden en bit natural, base 4	Orden en bit inverso, base 4	Orden en bit inverso, base 10
0	00	00	0
1	01	10	4
2	02	20	8
3	03	30	12
4	10	01	1
5	11	11	5
6	12	21	9
7	13	32	13

Para resolver una mariposa es necesario realizar 8 adiciones y 3 multiplicaciones complejas; en cada etapa existe N/4 mariposas y el número de etapas totales es igual a $\log_4 N$. Es así, que el número de adiciones complejas es igual a $2\mathrm{N}\log_4 N$ y el número de multiplicaciones complejas es igual a $3(\mathrm{N}/4)\mathrm{N}\log_4 N$. Las operaciones que se realizan en este algoritmo son menores que las de Radix-2 para la determinación de la DFT.

EJEMPLO RADIX-4 DIT DE OCHO PUNTOS





Respuesta

$\boldsymbol{X[0] = -0.4 + 2.1i}$

$\boldsymbol{X[1] = 15.861 + 8.774i}$

$\boldsymbol{X[2] = 11.87 + 4.349i}$

$\boldsymbol{X[3] = 15.84 - 5.096i}$

$\boldsymbol{X[4] = -16.5 - 30i}$

$\boldsymbol{X[5] = -2.845 + 17.365i}$

$\boldsymbol{X[6] = -0.586 - 15.77i}$

$\boldsymbol{X[7] = -13.54 - 1.03i}$

$\boldsymbol{X[8] = 18.4 \ - 2.9i}$

$\boldsymbol{X[9] = 13.079 - 8.083i}$

$\boldsymbol{X[10] = 2.121 + 8.45i}$

$\boldsymbol{X[11] = 17.64 - 1.047i}$

$\boldsymbol{X[12] = 14.5 + 1.2i}$

$\boldsymbol{X[13] = 3.505 + 5.947i}$

$\boldsymbol{X[14] = 10.58 - 7.429i}$

$\boldsymbol{X[15] = -9.54 + 23.17i}$

1.5.*4.2.4.* *Algoritmo Radix-4 con decimación en frecuencia* [13].

Para ejecutar el algoritmo se necesita que la longitud de la secuencia de entrada $x(n)$ sea potencia de base 4; en donde N es el número de puntos y al aplicar $\log_4 N$ se obtiene el número de etapas presentes en el algoritmo Radix-4. Al ser decimación en frecuencia se debe dividir a la secuencia de salida $X(k)$ en cuatro subsecuencias, las cuales tienen la misma longitud correspondiente a N/4, al realizar dicha división basándose en la ecuación (1.4), se realiza el siguiente análisis [13].

$$X(4k) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) + x\left(n + \frac{N}{4}\right) + x\left(n + \frac{N}{2}\right) + x(n + \frac{3N}{4})] * W_{N/4}^{kn} \tag{1.21}$$

$$X(4k+1) = \sum_{n=0}^{\frac{N}{4}-1} \{[x(n) - jx\left(n + \frac{N}{4}\right) - x\left(n + \frac{N}{2}\right) + jx(n + \frac{3N}{4})] * W_N^n\} * W_{N/4}^{kn} \tag{1.22}$$

$$X(4k+2) = \sum_{n=0}^{\frac{N}{4}-1} \{[x(n) - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{N}{2}\right) - x(n+\frac{3N}{4})] * W_N^{2n}\} * W_{N/4}^{kn} \quad (1.23)$$

$$X(4k+3) = \sum_{n=0}^{\frac{N}{4}-1} \{[x(n) + jx\left(n+\frac{N}{4}\right) - x\left(n+\frac{N}{2}\right) - jx(n+\frac{3N}{4})] * W_N^{3n}\} * W_{\frac{N}{4}}^{kn} \quad (1.24)$$

$$k = 0,1,2,\ldots,\frac{N}{4}-1$$

Al igual que los algoritmos anteriores el Radix-4 DIF es recursivo, es decir se realizan diferentes divisiones de forma repetitiva hasta conseguir varias DFTs de 4 puntos, es por ello que la secuencia de salida tiene un orden de bit inverso, tal como se puede apreciar en la figura 1.22 que nos muestra la forma de este algoritmo para un número de N=16.

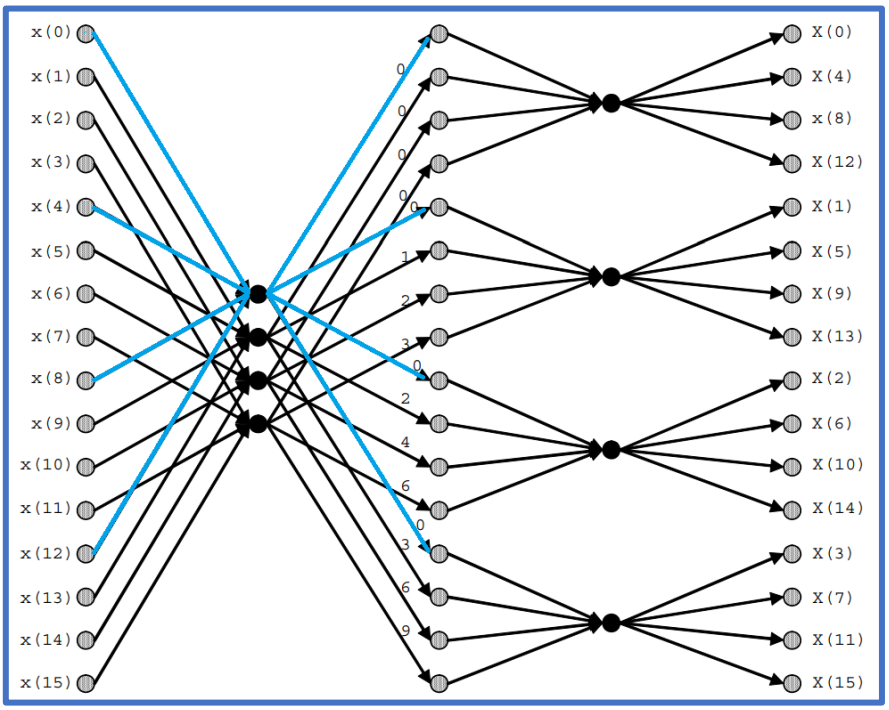


Figura 1.22. Radix-4 DIF para una DFT de N=16 [13]

En la figura 1.22 las líneas remarcadas de color celeste representan la mariposa Radix-4 DIF que es la estructura base de este algoritmo. Esta estructura engloba un total de tres multiplicaciones y ocho sumas complejas, con ello para todas las etapas se tiene un total de $\frac{3N}{4} * \log_4 N$ productos complejos y $2N * \log_4 N$ adiciones complejas [13]. Esto se puede apreciar de mejor manera en la figura 1.23 que representa la estructura simplificada de la mariposa Radix-4.

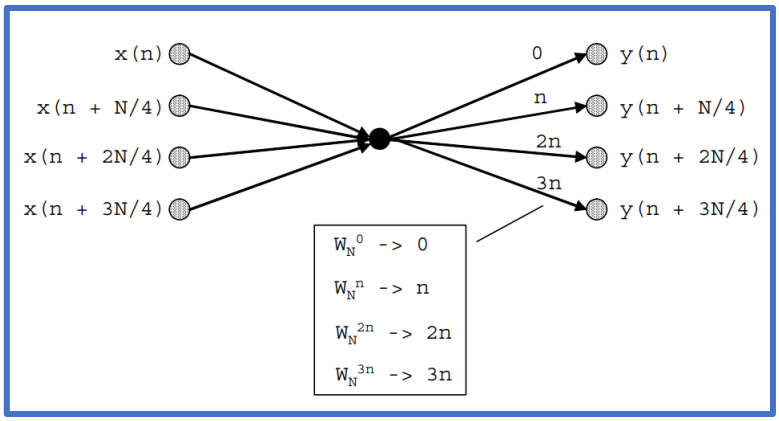
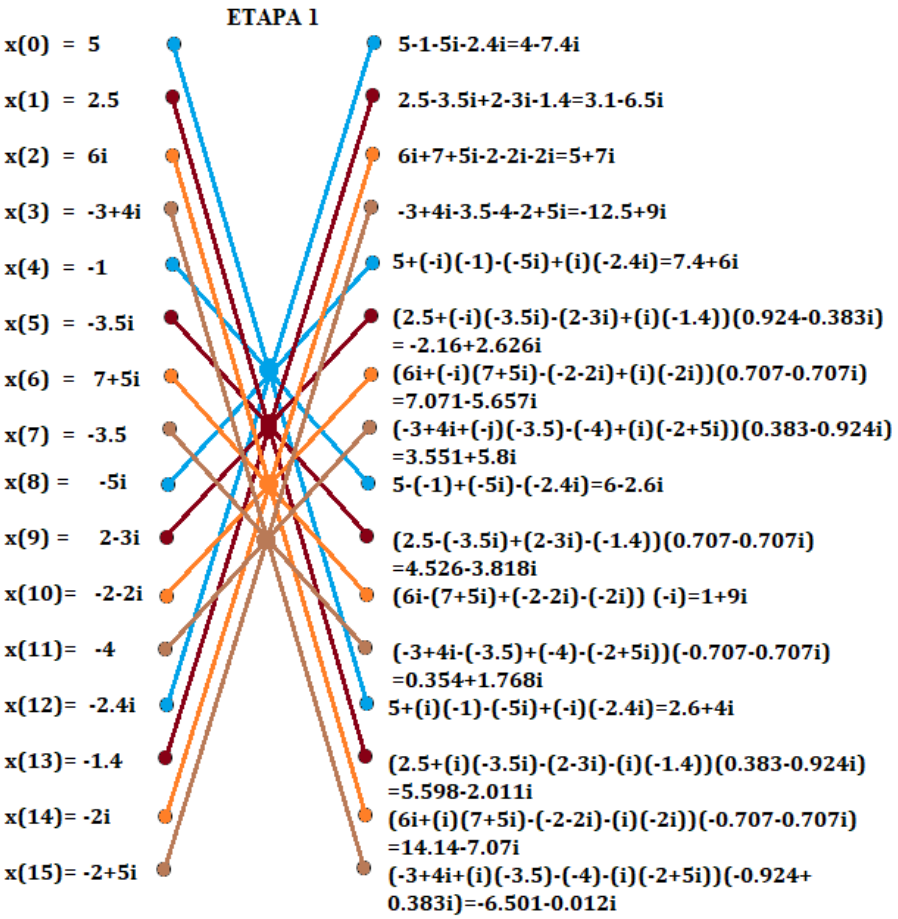
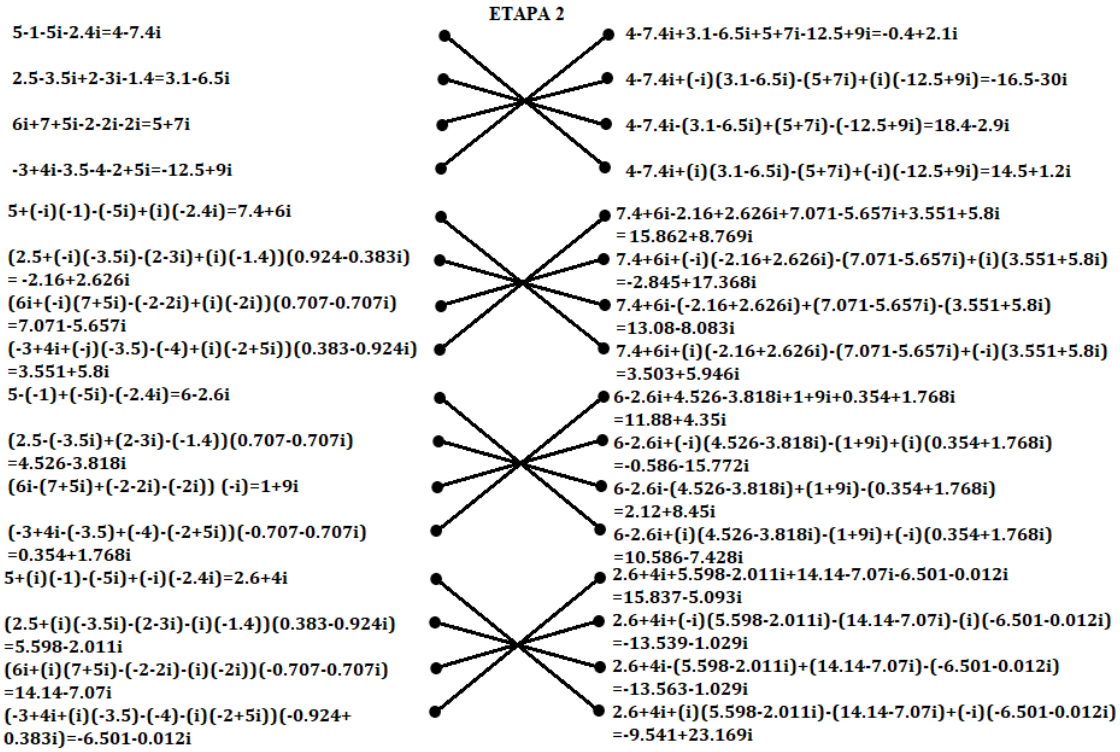


Figura 1.23. Estructura simplificada de una mariposa Radix-4 DIF [13]

Los términos que se encuentran dentro del rectángulo corresponden a los factores de giro que se detallan en las ecuaciones (1.22), (1.23) y (1.24) respectivamente.

EJEMPLO RADIX-4 DIF DE OCHO PUNTOS





Respuesta

$X[0] = -0.4 + 2.1i$

$X[1] = 15.861 + 8.774i$

$X[2] = 11.87 + 4.349i$

$X[3] = 15.84 - 5.096i$

$X[4] = -16.5 - 30i$

$X[5] = -2.845 + 17.365i$

$X[6] = -0.586 - 15.77i$

$X[7] = -13.54 - 1.03i$

$X[8] = 18.4 \ - 2.9i$

$X[9] = 13.079 - 8.083i$

$X[10] = 2.121 + 8.45i$

$X[11] = 17.64 - 1.047i$

$X[12] = 14.5 + 1.2i$

$X[13] = 3.505 + 5.947i$

$X[14] = 10.58 - 7.429i$

$X[15] = -9.54 + 23.17i$

1.5.4.2.5. Algoritmo Split-Radix con decimación en frecuencia [14]

Para efectuar este algoritmo es necesario que la longitud de la secuencia de entrada $x(n)$ sea potencia de base dos, el cálculo de la DFT con este algoritmo separa en tres DFTs una de longitud N/2 y dos de longitud N/4, esto requiere la implementación de los algoritmos Radix-2 y Radix-4 descritos anteriormente. Con la primera transformada se obtienen las muestras pares de la secuencia resultante $X(k)$, gracias al algoritmo Radix-2, estos resultados pares se obtienen aplicando la ecuación (1.15).

Por otra parte, las dos DFTs restantes se ejecutan mediante el algoritmo Radix-4 y con ello se obtienen las muestras impares de la secuencia $X(k)$, estos resultados se obtienen al aplicar las ecuaciones (1.23) y (1.24). Con la ecuación (1.23) se obtienen las muestras $X(1), X(5),\ X(9)\ldots$, mientras que con la ecuación (1.24) se obtienen las muestras $X(3), X(7), X(11)\ldots.$

Al igual que los algoritmos Radix-4 y Radix-2 el Split-Radix tiene una naturaleza recursiva sobre las diferentes DFTs de menor orden que se generan en las etapas del algoritmo. Para poder observar con mayor claridad y así poder entender de mejor manera, se presenta la figura 1.24 que muestra la forma en que opera el algoritmo Split-Radix DIF para una DFT de N=32 puntos.

En la figura 1.24 se aprecia la estructura base del algoritmo, que son las mariposas. Las mariposas están resaltadas de color morado; en cada etapa se observa que existe un conjunto de mariposas, las cuales forman bloques, dichos bloques están resaltados con líneas azules, por ejemplo, en la etapa 2 se pueden observar 3 bloques de 8 muestras de entrada, cada una de las cuales se utilizan para las operaciones de las mariposas teniendo un total de cuatro mariposas en cada bloque.

En la figura 1.25 se puede observar la mariposa mencionada anteriormente con mayor detalle, mostrando las operaciones que se deben realizar para el cálculo de los cuatro puntos. Mientras que en la figura 1.26 se puede observar la mariposa Radix-2 sin factores de giro que va a calcularse en la etapa final del Split-Radix [15].

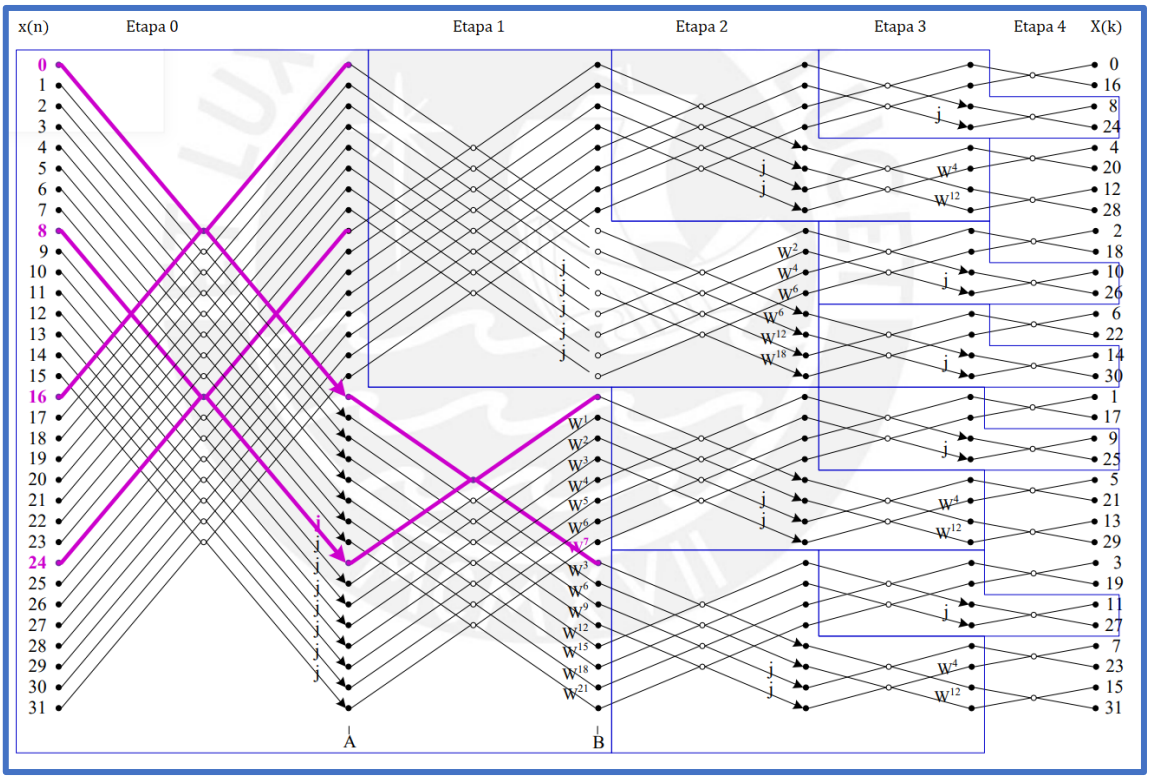


Figura 1.24. Grafo completo del algoritmo Split-Radix DIF para N=32 [15]

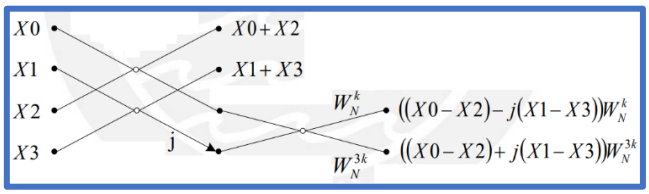


Figura 1.25. Estructura de la mariposa Split-Radix DIF [15]

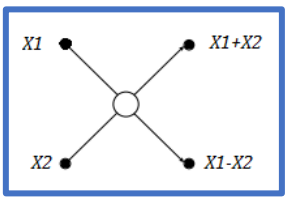


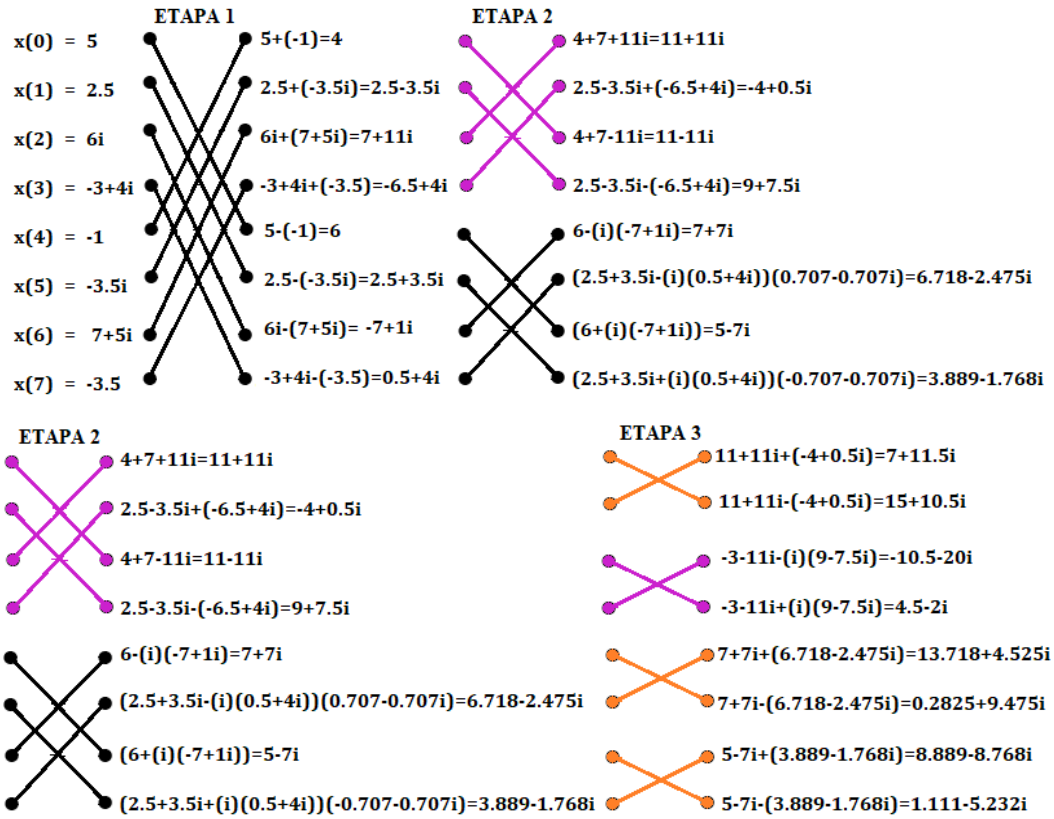
Figura 1.26. Estructura de la mariposa Radix-2 DIF [15]

El algoritmo Split-Radix al combinar el algoritmo Radix-2 y Radix-4 realiza un menor número de operaciones complejas, ejecutando únicamente dos sumas y cuatro productos complejos, el número de operaciones que se realiza en cada algoritmo se detalla en la tabla 1.3, en la que se puede evidenciar que el número de operaciones son considerablemente menores. Además, el algoritmo Split-Radix tiene como ventaja que puede realizar la DFT con un número de muestras que sea potencia de 2.

Tabla 1.3. Número de multiplicaciones y adiciones complejas en cada algoritmo [14]

	Multiplicaciones Reales				Adiciones Reales			
N	**Radix-2**	**Radix-4**	**Radix-8**	**Split-Radix**	**Radix-2**	**Radix-4**	**Radix-8**	**Split-Radix**
16	24	20	N/A	20	152	148	N/A	148
32	88	N/A	N/A	68	408	N/A	N/A	388
64	264	208	204	196	1032	976	972	964
128	712	N/A	N/A	512	2054	N/A	N/A	2308
256	1800	1392	N/A	1284	5896	5488	N/A	5380
512	4360	N/A	3204	3076	13566	N/A	12420	12292
1024	10248	7856	N/A	7172	30728	28336	N/A	27652

EJEMPLO SPLIT-RADIX DIF DE OCHO PUNTOS



Respuesta

Ordenamiento en bit inverso

$$\boldsymbol{X[0] = 7 + 11.5i}$$
$$\boldsymbol{X[1] = 13.718 + 4.525i}$$
$$\boldsymbol{X[2] = -10.5 - 20i}$$
$$\boldsymbol{X[3] = 8.889 - 8.768i}$$
$$\boldsymbol{X[4] = 15 + 10.5i}$$
$$\boldsymbol{X[5] = 0.2825 + 9.475i}$$
$$\boldsymbol{X[6] = 4.5 - 2i}$$
$$\boldsymbol{X[7] = 1.111 - 5.232i}$$

2. METODOLOGÍA

2.1. LENGUAJE VHDL [16]

El significado de VHDL proviene de VHSIC Hardware Description Languaje *(Very High Speed Integrated Circuits)*. Este tipo de lenguaje permite acelerar el proceso de diseño para el procesamiento digital de las señales. VHDL es un lenguaje de descripción de hardware que permite diseñar circuitos sincrónicos[3] y asincrónicos[4]. Conocer la sintaxis de VHDL no implica necesariamente diseñar en este lenguaje, ya que no todas las descripciones pueden ser traducidas a circuitos digitales [16].

Para diseñar en VHDL se deben describir básicamente tres secciones: librería, entidad y arquitectura, las mismas que se describen a continuación.

2.1.1. LIBRERÍA [16]

Son archivos de código que se describen al inicio del proyecto que se va a realizar para facilitar su utilización en el diseño de un proyecto. Usualmente estos se encuentran en paquetes para luego ser incluidos en una o varias librerías. Para acceder a un paquete primero debemos importar la librería y seguidamente su paquete como se indica en la código 2.1.

```
library <nombre de la librería>;
use <nombre de la librería>.<nombre del paquete>.frag;
```

Código 2.1. Sintaxis para importar un paquete de una librería

Un diseñador puede crear librerías e incluir los paquetes que necesite para su proyecto, pero cabe mencionar que ya se encuentran incluidas dos librerías con paquetes: IEEE *(Institute of Electrical and Electronics Enginners)* y STD *(Standard).*

Los paquetes que contiene la librería STD son:

- **Paquete Standard:** ya se encuentra incluido por defecto; contiene los tipos de datos que se pueden utilizar en el lenguaje VHDL como: integer, boolean, bit, real.
- **Paquete texto:** se usa para declaraciones de archivos de texto.

[3] Circuito sincrónico: circuito que cambia de estados por medio de una señal de reloj que comanda todo el sistema.
[4] Circuito asincrónico: circuito que puede cambiar de estados sin necesidad de una señal de reloj.

Los paquetes que contiene la librería IEEE son:

- **Paquetes numeric_std:** este paquete permite la conversión de datos de un tipo a otro. Además, posee operaciones matemáticas definidas para datos signed, unsigned e integer, pero no para std_logic y std_logic_vector. Esto podemos apreciar en la figura 2.1.

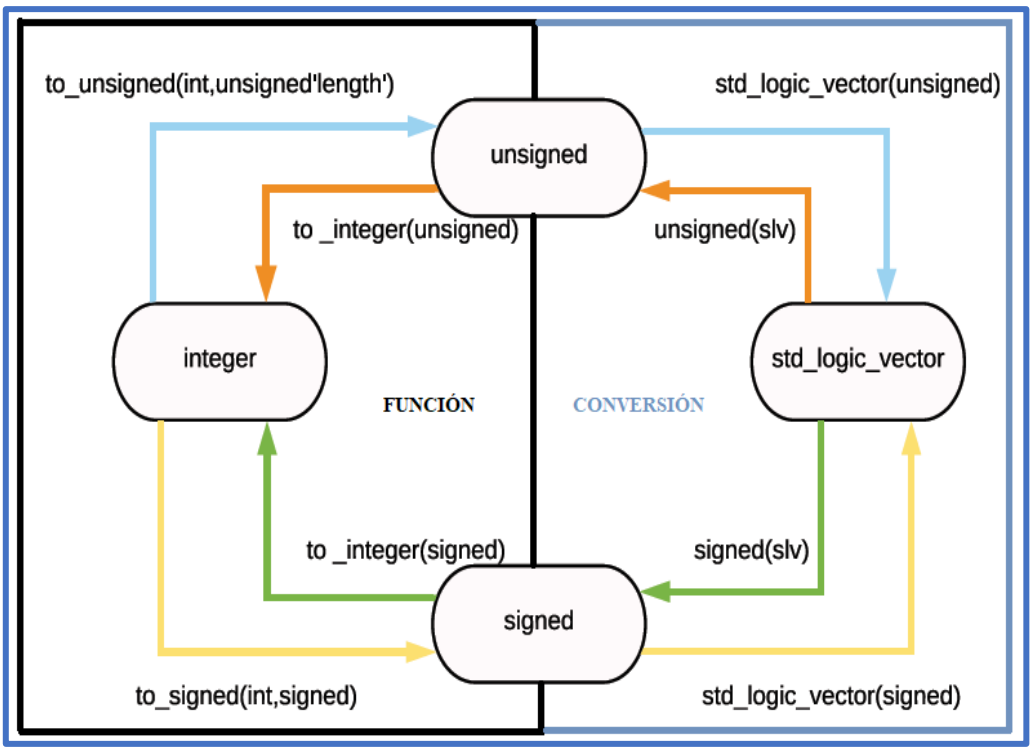


Figura 2.1. Conversiones y funciones dentro del paquete numeric_std [17]

- **Paquete std_logic_1164:** dentro de este paquete se encuentra el estándar lógico (*std_logic)* que permite trabajar hasta con 9 niveles lógicos incluidos el 1L y 0L; y el estándar logic vector (*std_logic_vector)*, que representa la unión de varios paquetes std_logic.
- **Paquete std_logic_arith:** dentro de este paquete se permite utilizar funciones aritméticas como: suma, resta y multiplicación. Además, de declarar datos signed y unsigned.
- **Paquetes std_logic_unsigned y std_logic_signed:** permite declarar datos con signo y sin signo para ser utilizados en operaciones aritméticas. Estos paquetes no se encuentran estandarizados por la IEEE.

2.1.2 ENTIDAD [16]

En la entidad se define únicamente la parte externa del circuito, en la que se indican las entradas y salidas del diseño. La forma en la cual se define una entidad se presenta en el código 2.2.

```
entity <nombre de la entidad> is
    port (<nombre puerto 1>: <modo> <tipo>;
          <nombre puerto 2>: <modo> <tipo>;
          .
          .
          .
          <nombre puerto n>: <modo> <tipo>);
end <nombre de la entidad>;
```

Código 2.2. Sintaxis para declarar una entidad en VHDL

En cada puerto se debe declarar: nombre, modo *(in/out/inout)* y tipo. Si declaramos un puerto como Input *(in)* se define como un puerto únicamente de lectura, mientras que si declaramos Output *(out)* se define como un puerto únicamente de escritura. Si se define un puerto como in/out se puede realizar lectura y escritura en el mismo puerto, pero definirlos de esta manera no es aconsejable cuando el proyecto se realiza en manera de síntesis[5].

Lo más recomendable al momento de declarar un puerto es definirlos como std_logic y std_logic_vector. En la entidad también se pueden declarar valores genéricos como constantes del circuito y ciertas propiedades[6].

2.1.3. ARQUITECTURA [16]

En la arquitectura se define el comportamiento de la entidad utilizando sentencias y expresiones. Si realizamos una analogía, la entidad es todo lo que se puede observar, mientras que la arquitectura comprende las conexiones y circuitos internos.

Su sintaxis se presenta en el código 2.3.

[5] Proyecto en manera de síntesis: que es implementable en hardware y no únicamente para modelado o simulación.
[6] Valores genéricos: como ejemplo la constate π que puede ser utilizada en uno o varios procesos de la entidad.

```
architecture <nombre> of <nombre de la entidad>
    (declaraciones)
    (señales)
    (componentes)
    begin
    (código)
end <nombre>;
```

Código 2.3. Sintaxis para declarar una arquitectura en VHDL

En la arquitectura antes del begin se pueden definir señales globales de la entidad, declaraciones de máquinas de estados y componentes, estas son entidades previamente definidas únicamente integrándolas al proyecto. La manera de declarar un componente se observa en el código 2.4.

```
component <nombre del componente> is
    port (<nombre puerto 1>: <modo> <tipo>;
          <nombre puerto 2>: <modo> <tipo>;
          .
          .
          .
          <nombre puerto n>: <modo> <tipo>);
end component;
```

Código 2.4. Sintaxis para declarar un componente en VHDL

Cuando se declara un componente en la arquitectura es necesario instanciarlo[7] después del begin de la arquitectura su sintaxis se indica en el código 2.5.

```
<etiqueta>:<nombre del componente> port map (conexiones)
```

Código 2.5. Sintaxis instanciar un componente en VHDL

La etiqueta de cada componente es indispensable, ya que permite diferenciar a cada componente. A continuación, PORT MAP y entre paréntesis las conexiones que tiene la componente con el proyecto principal. Es decir, aquí se realiza la conexión con la entidad principal, con un proceso[8] de la arquitectura o con otra componente.

En el begin se definen partes de código que se ejecutan paralelamente, también se pueden definir diferentes procesos, a esto se lo conoce como código concurrente.

[7] Instanciar: llamar a la componente y hacer una conexión con señales internas o externas al sistema.
[8] Proceso en VHDL: parte de código donde las instrucciones son ejecutadas en secuencia. Si existen dos o más procesos estos se ejecutan en paralelo.

Dentro de cada proceso se pueden crear señales locales y variables utilizadas únicamente en dicho proceso. Después del begin se define el código que se ejecutará de manera secuencial, a esto se le conoce como código secuencial.

En un proceso podemos utilizar variables que se crearon globalmente asignando valores a estas señales únicamente dentro de un único proceso y siendo leídas en el mismo proceso o en otro proceso. La sintaxis para definir un proceso se muestra en código 2.6.

```
[nombre]:process(lista de sensibilidad)
    (constantes y variables)
    begin
        (código secuencial)
end process;
```

Código 2.6. Sintaxis para declarar un proceso en VHDL

Colocar el nombre no es estrictamente necesario, pero si es recomendable. En la lista de sensibilidad se coloca una o varias señales (de la arquitectura en la que se encuentra ubicado) que al cambiar de valor provocan la ejecución del proceso.

Si existen partes de código secuencial que se necesitan ejecutar en diferentes procesos, se pueden crear las funciones [16] con el fin de simplificar el código y siendo únicamente llamadas desde cualquier proceso. Una función se define antes del BEGIN de la arquitectura, la forma de sintaxis para una función se muestra en el código 2.7.

Cuando se declara una función esta puede tener o no argumentos de entrada; estos argumentos, no pueden ser modificados, únicamente su valor puede ser leído. Si la función tiene más de un argumento de entrada, entre cada argumento de entrada se debe separar por punto y coma.

Hay que tomar en cuenta que el valor que devuelve la función debe coincidir con el tipo de señal o variable que va a recibir dicho valor.

```
function <nombre de la función>(argumentos)return<tipo>is
    (Declaración de variables y constantes)
begin
    (Sentencias secuenciales)
return Valor
end function;
```

Código 2.7. Sintaxis para declarar una función en VHDL

2.1.4. TIPOS DE OBJETOS

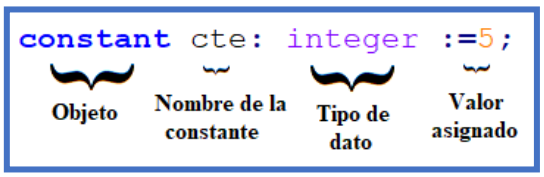
Los objetos que pueden declararse en VHDL son: constantes, variables y señales. Los objetos deben ser declarados como se observa en el código 2.8.

```
<Tipo de objeto>:<nombre> <tipo de dato>
```

Código 2.8. Sintaxis para declarar un objeto en VHDL

Constantes

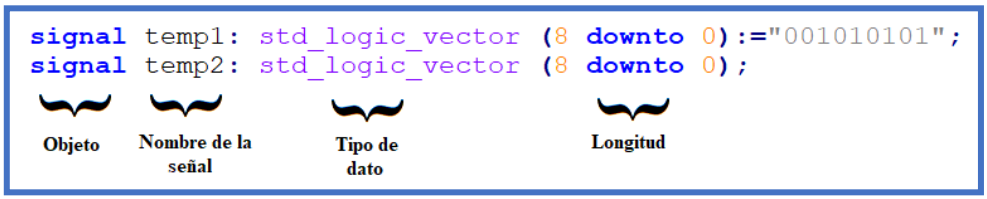
Cuando se define una constante el valor asignado no puede ser modificado a lo largo del código. En el código 2.9 se muestra la sintaxis para declarar una variable llamada *cte,* con valor 5.



Código 2.9. Sintaxis para declarar una constante en VHDL

Señales

Cuando se define una señal, ya sea en la arquitectura o en un proceso, esta puede cambiar su valor a lo largo del proceso en el que se encuentra. La sintaxis para definir una señal se muestra en el código 2.10, la señal creada puede tener o no un valor inicial. El código 2.10 muestra una señal std_logic_vector con una longitud de 9 bits donde la parte izquierda es la más significativa. Para cambiar el valor de una señal se utiliza el símbolo **<=**, tal como se muestra en el código 2.11.



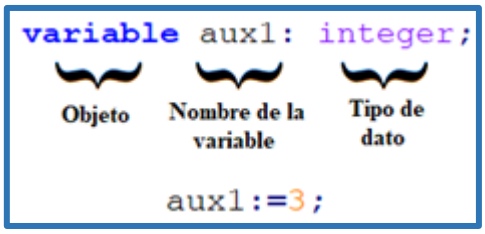
Código 2.10. Sintaxis para declarar una señal en VHDL

```
temp1<="11010111";
```

Código 2.11. Sintaxis para cambiar de valor una señal en VHDL

Variables

Una variable únicamente puede ser definida dentro de un proceso y tiene significado solamente dentro de este. Debido a esto, se pueden crear variables con el mismo nombre en diferentes procesos. Para cambiar el valor de una variable se utiliza el símbolo **:=**. En el código 2.12 se muestra la sintaxis de cómo crear una variable de tipo integer.

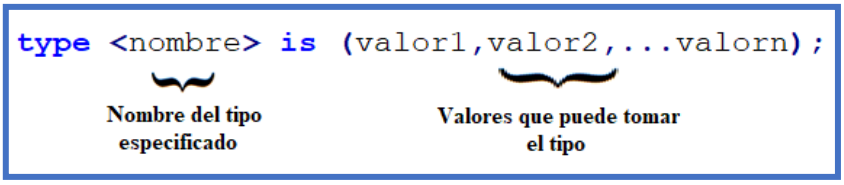


Código 2.12. Sintaxis para declarar una variable en VHDL

2.1.5. TIPOS DE DATOS [17]

Los tipos de datos se encuentran definidos dentro de las librerías y paquetes de VHDL. Los tipos de datos más utilizados son: boolean, natural (enteros no negativos), integer, enumerados, std_logic_vector y std_logic.

Un tipo de dato enumerado puede tomar n valores posibles de un conjunto especificado; estos valores se especifican en paréntesis y separados por comas, la sintaxis para crear un tipo de dato enumerado se muestra en el código 2.13. Estos datos son principalmente utilizados cuando se quieren utilizar máquinas de estado, las mismas que serán detalladas más adelante. Los tipos de datos enumerados también pueden ser definidos como arreglos. En el código 2.14 se muestra la sintaxis para crear tipos de datos compuestos en una dimensión.



Código 2.13. Sintaxis para declarar un tipo de dato enumerado en VHDL

```
type <nombre del arreglo> is array (rango) of <tipo de datos>;
```

Código 2.14. Sintaxis para declarar un tipo de dato compuesto en VHDL

2.2. VIVADO DESING SUITE

Vivado es una herramienta para el desarrollo de System on Chip (SoC), centrada en propiedad intelectual[9] (IP) y centrado en el sistema[10], que se ha creado desde cero para abordar los cuellos de botella de productividad en la integración e implementación a nivel de sistema. Vivado es compatible con los siguientes dispositivos: Virtex-7, familias UltraScale, Artix-7, Zynq-700 y Kintex-7. Se dispone de un simulador integrado que admite lenguaje Verilog, SystemVerilog y VHDL, con la que es posible comprobar el funcionamiento del diseño creado [18].

Para comprender el funcionamiento de Vivado Desing Suite, en el anexo A se describe una compuerta AND, la cual equivale al “hola mundo” en lenguaje VHDL.

2.2.1. DESARROLLO EN VIVADO DEL BLOQUE FFT

El principal objetivo del proyecto es implementar una FFT en la FPGA para analizar el espectro de señales periódicas y comprobar su funcionamiento.

2.2.2. COMPONENTES Y PROCESOS QUE CONFORMAN EL BLOQUE FFT

La entidad transformada está compuesta por los puertos de entrada y salida descritos en la tabla 2.1. y la representación de este bloque se muestra en la figura 2.2.

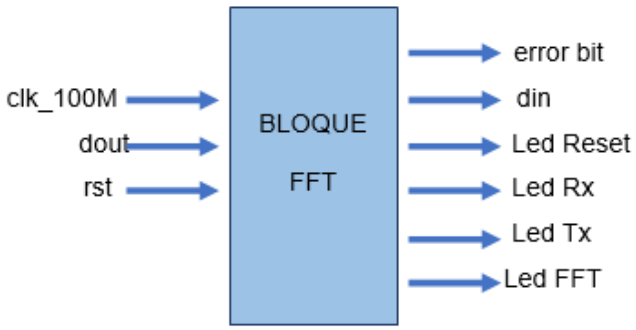


Figura 2.2. Bloque FFT de la entidad transformada

[9] Centrada en IP: Proyectos previamente diseñados que pueden ser incluidos en el proyecto.
[10] Centrado en sistema: Proyectos creados por el usuario.

Tabla 2.1. Descripción de pines del bloque FFT

Puerto	Descripción
din	Se usa para recibir los bits muestreados en forma serial
clk_100M	Reloj para la ejecución del bloque FFT
rst	Se usa para reiniciar los valores del bloque FFT a sus valores originales. Se activa con 1L
dout	Se usa para enviar los bits de la FFT calculada en forma serial
error_bit	Se pone en 1L cuando una trama recibida tuvo error y detiene el proceso de recepción hasta reiniciarlo.
Led Rx	Se pone en 1L cuando la FPGA está recibiendo datos Se pone en 0L cuando la FPGA no recibe datos
Led FFT	Se pone en 1L cuando la FPGA está calculando la FFT Se pone en 0L cuando la FPGA termina de realizar la FFT
Led Tx	Se pone en 1L cuando se esta enviando datos desde la FPGA Se pone en 0L cuando se termina de enviar los datos desde la FPGA
Led Reset	Se pone en 1L cuando el botón rst se encuentra en 1L

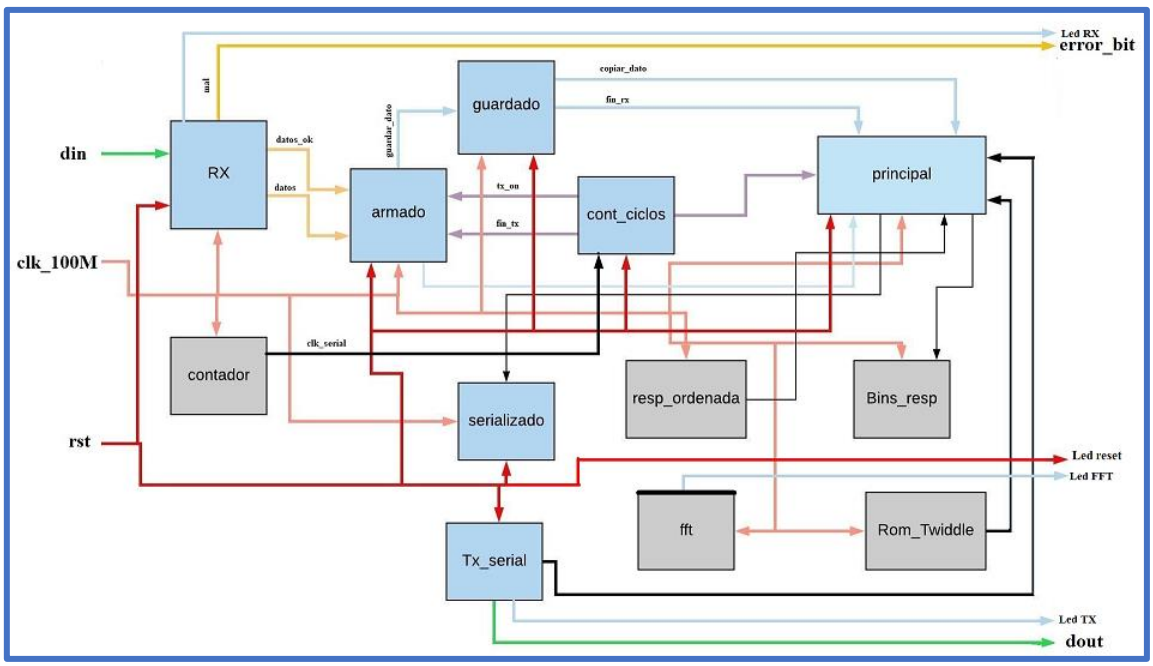


Figura 2.3. Diagrama interno del bloque FFT

Los procesos creados en la entidad transformada son controlados mediante máquina de estados finitos (FSM) [19], esto permite modelar circuitos secuenciales; estas máquinas de estado tienen un número limitado de estados, en los que se lleva acabo ciertas líneas de comandos dependiendo de lo que se desea hacer, así mismo en la FSM se especifican los saltos que se deben ejecutar a los otros estados. Todos los procesos creados tienen sincronía con la señal de reloj de 100MHz (clk_100M). La sintaxis para crear máquinas de estados se muestra el código 2.15, y la forma de llamar a las FSM en un proceso se muestra en el código 2.16.

```
type enumerated is <nombre> (FSM1, FSM2);
signal state: <nombre>;
```

Código 2.15. Sintaxis para declarar una máquina de estados en VHDL

```
fsm: process(rst, clk)
begin
    if (rst='1') then

        estado <= FSM1;
    elsif (clk'event and clk='1') then

        case estado is
            when FSM1 =>
                estado <= FSM2;

            when FSM2 =>
                estado <= FSM1;
        end case;
    end if;
end process;
```

Código 2.16. Sintaxis para llamar a una máquina de estados en un proceso en VHDL

Para realizar operaciones matemáticas de punto fijo[11] es necesario utilizar los paquetes “sixed_float_types” y “fixed_pkg_c” que deben ser compilados e importados hacia el proyecto mediante la librería ieee_proposed, además de ejecutar las líneas de comandos en el command de vivado mostradas en el código 2.17. Estos paquetes no están integrados en VHDL, sino que fueron creados por David Bishop [20]; allí se definen los tipos ***sfixed*** (formato de punto fijo con signo) y ***ufixed*** (formato de punto fijo sin signo).

En el proyecto únicamente se usará el tipo *sfixed* donde su representación se muestra en el código 2.18 con representación de la parte decimal con un signo menos.

[11] Punto fijo: El punto decimal se mantiene fijo sin importar el número que se va a representar.

```
add_files -norecurse <path to package file>/fixed_pkg_2008.vhd
set_property library ieee [get_files <path to package file>/fixed_pkg_2008.vhd]
read_vhdl -vhdl2008 ./my_design.vhd
launch_runs synth_1 -jobs 4
wait_on_run synth_1
open_run synth_1 -name synth_1
```

Código 2.17. Sintaxis para llamar a una máquina de estados en un proceso en VHDL

```
variable aux: sfixed (1 downto -10);
                      1 bit parte entera   10 bit parte fraccionaria
```

Código 2.18. Sintaxis para declarar una variable de tipo sfixed

La longitud de la variable *aux* es de 12 bits, donde un bit es la parte entera, diez bits de parte fraccionaria y un bit de signo tal como se muestra en la figura 2.4.

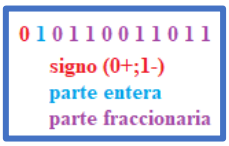


Figura 2.4. Representación de parte entera, signo y fracción de una variable

En este paquete también se incluyen operaciones aritméticas básicas (suma, resta, multiplicación y división) que pueden ser empleados con datos sin signo o datos con signo.

Las librerías utilizadas en el proyecto son: numeric_std para realizar operaciones aritméticas con variables y señales de tipo integer, ieee_proposed que es necesaria para variables de punto fijo y std_logic_1164 que es integrada automáticamente después de crear el proyecto, tal y como se muestra en el código 2.19.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library ieee_proposed;
use ieee_proposed.fixed_float_types.ALL;
```

Código 2.19. Sintaxis en Vivado para declarar librerías utilizadas en el proyecto

2.2.2.1. COMPONENTE CONTADOR

El componente contador recibe únicamente como argumento de entrada una señal de reloj y en su arquitectura implementa un contador para obtener una salida de reloj a 4800 Hz. Esto es necesario ya que la velocidad de transmisión y recepción de la interfaz serial es de 4800 baudios. El contador implementado es de módulo 10416 ascendente. Si realizamos los cálculos para obtener la frecuencia de salida debemos dividir el contador

al reloj de 100MHz (reloj de entrada) obteniendo así los 4800 Hz. El esquema de este componente se muestra en la figura 2.5.

Figura 2.5. Esquema del componente contador

Cada vez que el contador llega a 10415 se reinicia el contador y el estado de la señal reloj de salida cambia, teniendo así cambios de estado cada 104.167 us que sumarían un período de 208.33 us.

2.2.2.2. COMPONENTE DE MEMORIA RAM PARA ALMACENAR DATOS RECIBIDOS Y RESULTADOS DE LA FFT

Los datos recibidos por la interfaz serial tiene una longitud de 32 bits teniendo únicamente parte real. Estos bits son almacenados en la componente RAM que es una Memoria de Acceso Aleatorio (RAM), la cual fue creada con 2048 localidades de 64 bits cada una, ya que a la salida del bloque FFT se tiene parte real e imaginaria con longitud de 32 bits cada una.

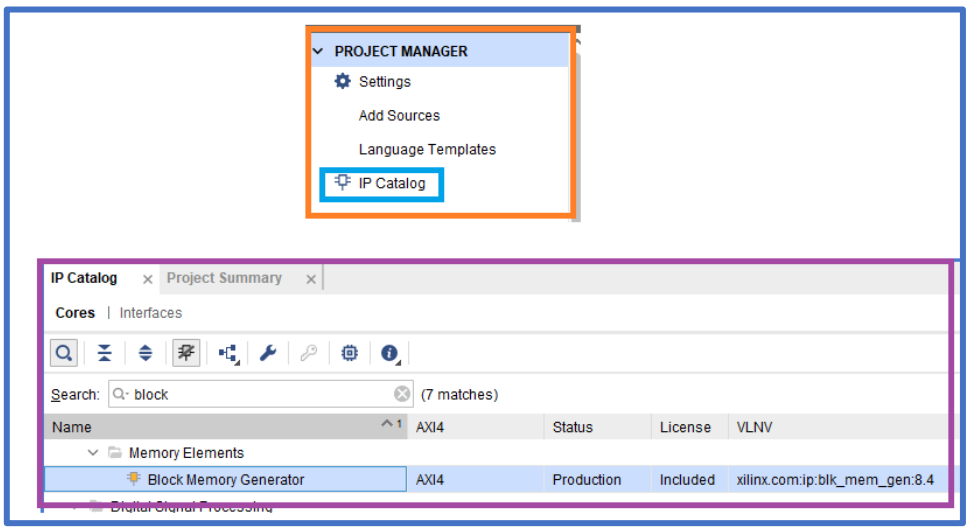


Figura 2.6. Creación de una componente de memoria RAM

Para crear la componente RAM en IP CATALOG en el PROJECT MANAGER de Vivado se debe buscar el elemento *Block Memory Generator,* como se muestra en la figura 2.6. Al seleccionar esta opción se despliega una interfaz para la configuración de la misma, como se muestra en la figura 2.7.

En esta interfaz se elige la opción Single Port RAM, que corresponde a una memoria RAM con un único puerto por el cual se permite lectura y escritura, se debe mencionar que no es posible realizar las dos acciones al mismo tiempo. De la misma manera en esta interfaz se establece el número de localidades que se requieren utilizar (direcciones de memoria) y el ancho (bits de datos).

Una vez creada la componente se debe instanciar inmediatamente después del begin de la arquitectura. En el código 2.20 se muestra la sintaxis de como llamar a la componente creada y como instanciarlo.

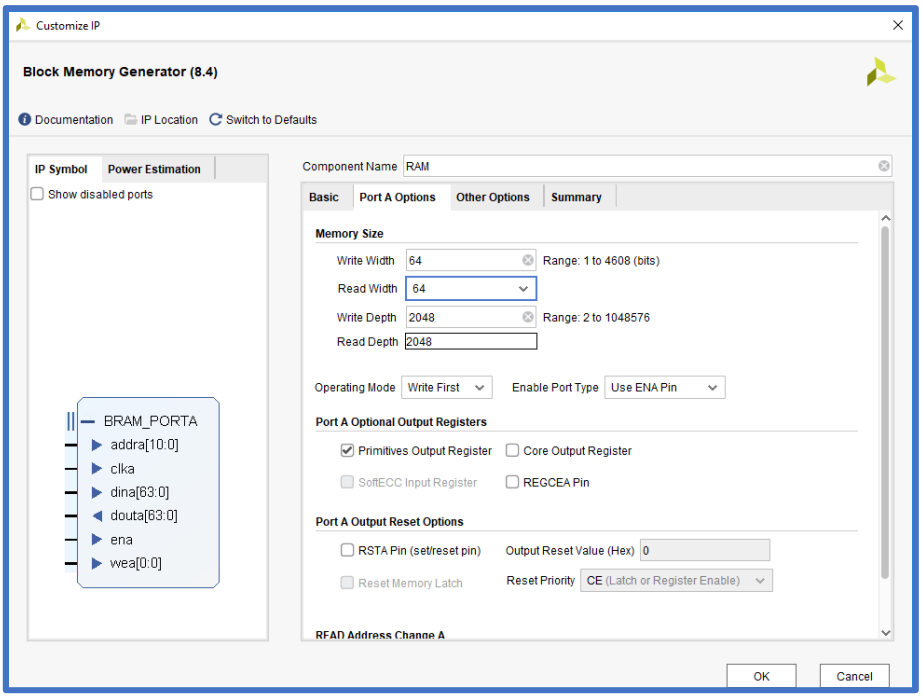


Figura 2.7. Interfaz para definir la componente de memoria RAM

En este proyecto es necesario crear tres memorias RAM, necesario para la recursividad de este, es decir, lo que se recibe se guarda en una memoria RAM *(RAM)* y así mismo sus datos se copian en otra memoria RAM *(RAM1)* para ser utilizada en el proceso de cálculo de la FFT. De la misma manera, los resultados obtenidos en este bloque se copian en otra memoria RAM *(RAM2)* para disponer de dichos resultados en el proceso de transmisión. Esto se realiza debido a que se necesita enviar y recibir datos simultáneamente. Como ya se mencionó, esta componente únicamente puede leer o

escribir, entonces al momento de la recepción se escribe en una memoria, mientras que si se envía al mismo tiempo se deben leer los datos de otra memoria.

```
architecture Behavioral of transformada is
    component ram
          PORT (
          clka : IN STD_LOGIC;
          wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
          addra : IN STD_LOGIC_VECTOR(10 DOWNTO 0);
          dina : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
          douta : OUT STD_LOGIC_VECTOR(63 DOWNTO 0));
     end component;
    signal habilita : STD_LOGIC_VECTOR(0 DOWNTO 0):= (others => '0');
    signal direccionRAM : STD_LOGIC_VECTOR(10 DOWNTO 0);
    signal datosRAM : STD_LOGIC_VECTOR(63 DOWNTO 0);
    signal datosoutRAM : STD_LOGIC_VECTOR(63 DOWNTO 0);
begin

    U1: ram PORT MAP (clk_100M, habilita, direccionRAM, datosRAM, datosoutRAM);


end Behavioral;
```

Código 2.20. Sintaxis en Vivado para instanciar la memoria RAM

Para definir la componente memoria RAM (figura 2.8) se debe definir los puertos descritos en la tabla 2.2. Creando señales internas en la arquitectura y realizando su conexión a la memoria.

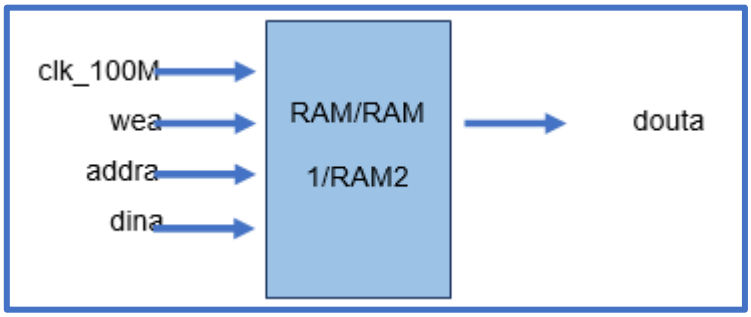


Figura 2.8. Esquema del componente Memoria RAM

Tabla 2.2. Descripción de pines del bloque memoria RAM

Puerto	Descripción
clk_100M	Reloj para activación de procesos dentro de la arquitectura RAM
wea	Señal para habilitar lectura o escritura (1L escritura y 0L lectura).
addra	Señal para direccionamiento
dina	Dato a guardar en la dirección apuntada
douta	Dato a leer en la dirección apuntada

2.2.2.3. COMPONENTE DE MEMORIA ROM PARA ALMACENAR FACTORES DE GIRO

De la misma manera que se creó la componente RAM se crea la componente ROM, a diferencia que en la interfaz de configuración se debe escoger Single Port ROM. De igual forma, se debe configurar el número de localidades y la longitud de cada localidad. Esta componente únicamente sirve para lectura donde sus datos son cargados en la interfaz de configuración como se muestra en la figura 2.9.

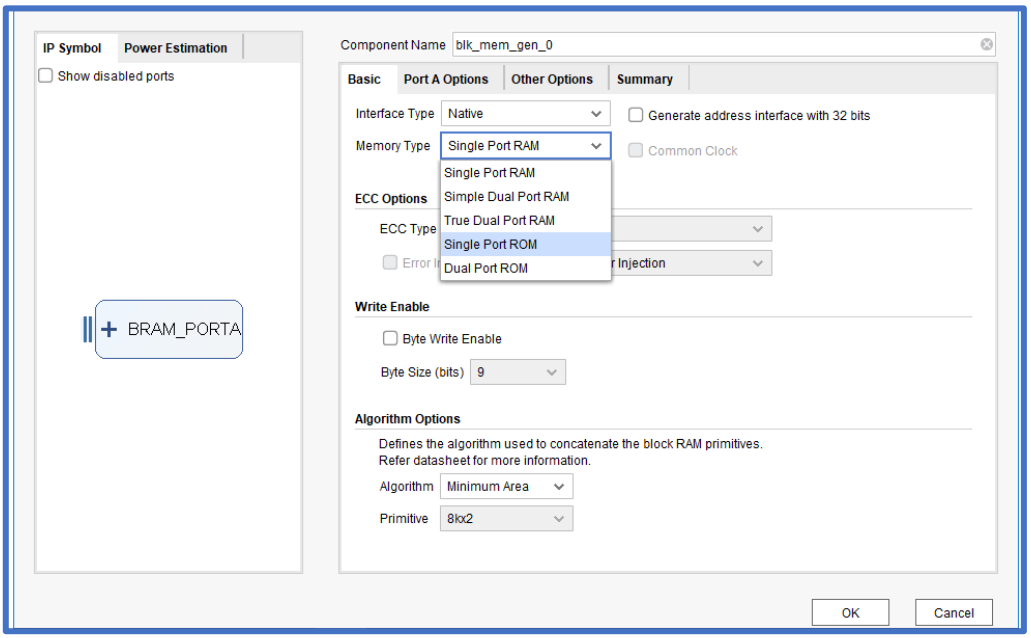


Figura 2.9. Interfaz para definir la componente memoria ROM

Los valores cargados a esta componente son creados en Matlab en un archivo con extensión.*coe,* en el anexo B se encuentra el programa para crear los vectores de giro.

Para este proyecto se necesitan 512 localidades de memoria ROM, debido a que los factores de giro se reutilizan cada N/4 muestras *x(n),* ya mencionadas en el capítulo 1:

- Para hallar el valor de un exponente que es mayor o igual a N/4, pero menor a N/2, se resta N/4 al valor de dicho exponente y se extrae el valor de la componente ROM; se intercambia la parte real con la parte imaginaria y luego se conjuga.
- Para hallar el valor de un exponente que es mayor o igual a N/2, se resta N/4 al valor de dicho exponente y se extrae el valor de la componente ROM; para cambiar de signo a la parte real e imaginaria.

La componente ROM tiene una longitud de 32 bits por cada localidad, de los cuales, los 16 bits más significativos corresponden a la parte real y los 16 bits menos significativos a la parte imaginaria. Cuando un dato de memoria ROM es leído, esta se coloca en una variable de punto fijo con longitud de 16 bits (1 signo, 1 parte entera y 14 parte decimal).

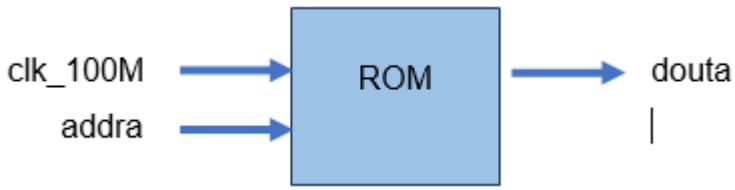


Figura 2.10. Esquema del componente de Memoria ROM

2.2.3. BLOQUE FFT EN VIVADO

2.2.3.1. ENTRADA DE DATOS

El proyecto cuenta con un proceso llamado RX, este recibe los datos por la interfaz serial mediante el puerto din. La máquina de estados que controla el funcionamiento de este proceso es la *fsm_RX* con su señal *est_RX* y su representación se muestra en la figura 2.11.

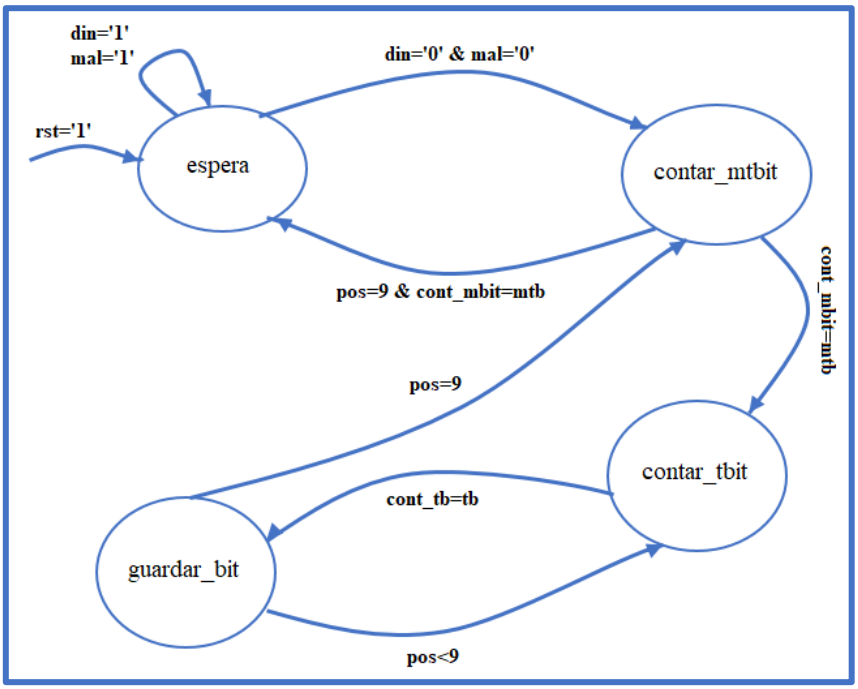


Figura 2.11. Máquina de estados del proceso RX

Una vez que el rst sea seteado en 1L, la máquina de estados de este proceso comienza y su descripción se presenta a continuación.

- **Estado espera:** Estado que monitorea el puerto din con cada flanco de subida del reloj clk_100M, espera recibir un 0L que indica el inicio de una trama.
- **Estado contar_mtbit:** Con cada flanco de subida del reloj clk_100M acumula un contador hasta llegar a la mitad del tiempo de bit para empezar a muestrear los datos recibidos.
- **Estado contar_tbit:** Con cada flanco de subida del reloj clk_100M acumula un contador hasta llegar a un tiempo de bit y obtener el valor del bit recibido mediante el puerto din.
- **Estado guardar_bit:** Estado que recibe el bit muestreado y almacena su valor en registro, que cuenta con una longitud de 9 bits (8 bits datos y 1 parada). Cuando los bits recibidos son menores a 9, se hace un salto al estado *contar_tbit* para seguir recibiendo más datos. Una vez que se recibieron todos los bits estos son verificados para saber si existe o no errores. Si los datos recibidos no tienen errores, estos son almacenados en una señal global llamada *datos*.

2.2.3.2 ARMADO DE LA MUESTRA

Este proceso recibe los 8 bits de datos y los almacena en una señal hasta recibir en total cuatro tramas que comprenden los 32 bits de una muestra, siendo 15 bits parte fraccionaria, 16 de parte entera y 1 bit de signo. Una vez recibidas las cuatro tramas se completan con 32 ceros en la parte menos significativa que indica que la parte imaginaria tiene el valor de cero. La máquina de estados de este proceso se muestra en la figura 2.12.

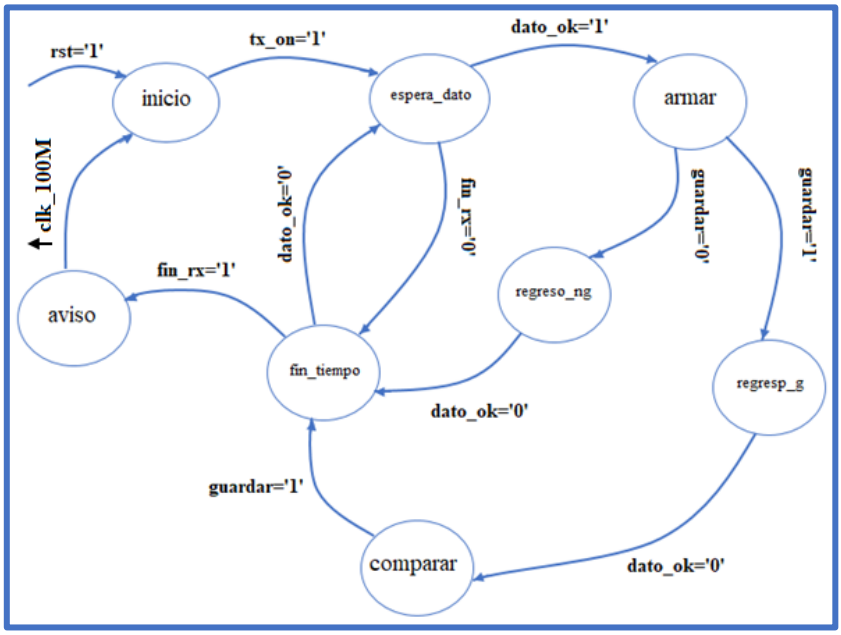


Figura 2.12. Máquina de estados del proceso Armado

- **Estado espera_dato:** Este estado espera hasta que el proceso RX indique que recibió una trama sin errores por medio de la señal *dato_ok*.
- **Estado armar:** Estado encargado de formar los 32 bits de datos de una muestra enviada por la interfaz serial. Este debe ir recibiendo de 8 en 8 bits de datos que se obtienen del proceso RX para unirlos en un vector. Dentro de este estado se encuentra otra máquina de estados, que cuenta con cuatro etapas con diferentes vectores para almacenar lo que va recibiendo del proceso RX.
    - **Estado 0:** Contiene un vector llamado *temp1* que almacena los 8 primeros bits más significativos de una muestra recibida e incrementa un contador para saltar al estado 1.
    - **Estado 1:** Contiene un vector llamado *temp2* que almacena los 16 primeros bits de una muestra, es decir, toma los datos del vector *temp1* y une a los 8 bits siguientes que se recibió en el proceso RX e incrementa un contador para saltar al estado 2.
    - **Estado 2:** Contiene un vector llamado *temp3* que almacena los terceros 8 bits de una muestra recibida e incrementa un contador para saltar al estado 3.
    - **Estado 3:** Contiene un vector llamado *temp4* que almacena los 32 primeros bits de una muestra, es decir, toma los datos del vector *temp3, temp1* y une a los 8 bits siguientes que se recibió en el proceso RX y configura el contador en cero para volver a realizar el mismo proceso desde el estado 0.
- **Estado regreso_ng:** Estado que espera hasta que la señal *dato_ok* cambie a 0L.
- **Estado regreso_g:** Estado que espera hasta que la señal *dato_ok* cambie a 0L.
- **Estado compara:** Estado que revisa si la cantidad de datos recibidos es igual a 2048 muestras mediante la variable *count.* Para indicar al proceso *SR_asim* que puede iniciar mediante la señal *bins_ok*.
- **Estado fin_tiempo:** Espera un flanco de subida del reloj *clk_100M* para saltar al estado espera_dato.
- **Estado aviso:** Reinicia el contador *count* que es el encargado de revisar que se hayan recibido las 2048 muestras**.**

2.2.3.3. GUARDAR DATOS

Este estado se encarga de direccionar a la localidad donde se van a guardar los 32 bits recibidos y armados en el proceso anterior. Cuando se hayan recibido las 2048 muestras

este proceso indica al proceso SR_asim que obtenga la FFT de las muestras recibidas. La máquina de estados de este proceso se muestra en la figura 2.13.

- **Estado reposo:** Espera que los parámetros modo de transmisión y transmisión se encuentren activados.
- **Estado espera_bin:** Estado que espera que el proceso anterior haya recibido los 32 bits de datos.
- **Estado direccionar_RAM:** Estado que direcciona donde almacenar los datos recibidos. Una vez direccionado se aumenta en uno la posición.
- **Estado guardar_RAM:** Una vez direccionado este estado guarda los 32 bits en la posición apuntada en el estado direccionar_RAM.
- **Estado retardo:** Espera un ciclo de reloj para volver al estado inicio.

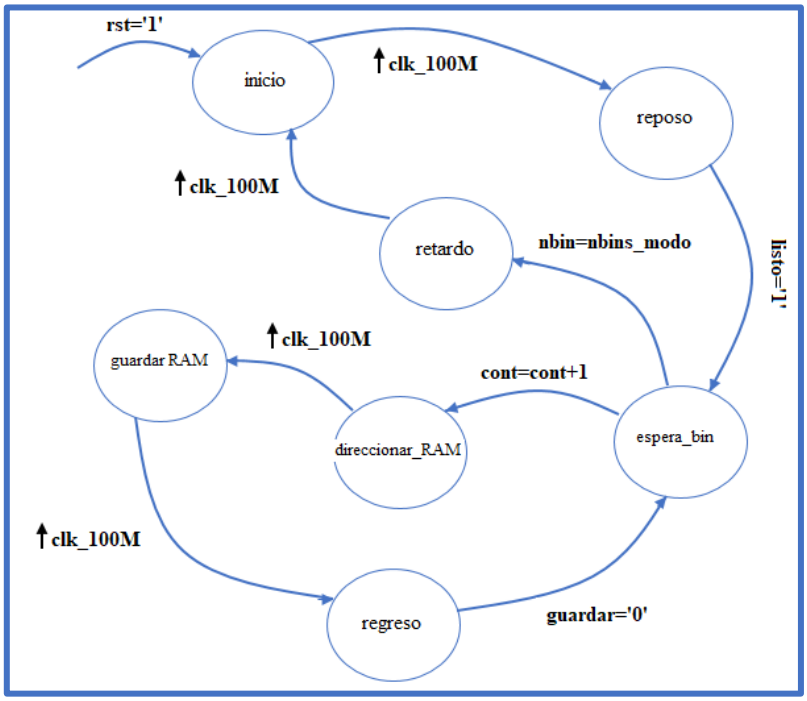


Figura 2.13. Máquina de estados del proceso Guardado

2.2.3.4. CÁLCULO DE LA FFT

Proceso que realiza el cálculo de la FFT mediante el algoritmo Split-Radix asimétrico, luego de recibir 2048 muestras a través del enlace serial. La máquina de estados de este proceso se muestra en la figura 2.14.

- **Estado ver_bins_ok:** Espera hasta que las 2048 muestras hayan sido recibidas mediante comunicación serial y restablece los valores de las variables para volver a calcular una nueva FFT.
- **Estado espera_RAM_r:** Estado que activa lectura de memoria RAM.
- **Estado leer_Xn:** Dentro de este estado se extraen las cuatro muestras para poder obtener los cálculos de una mariposa Radix-4. La forma en que se extraen los datos se detalla a continuación:
    - **Estado 0:** Extrae las n muestras correspondientes al primer punto de la mariposa y apunta a la siguiente dirección del primer punto de la mariposa siguiente.
    - **Estado 1:** Extrae las muestras n+num/4 correspondientes al segundo punto de la mariposa y apunta a la siguiente dirección del segundo punto de la mariposa siguiente.
    - **Estado 2:** Extrae las muestras n+num/2 correspondientes al tercer punto de la mariposa y apunta a la siguiente dirección del tercer punto de la mariposa siguiente.
    - **Estado 3:** Extrae las muestras n+3num/4 correspondientes al cuarto punto de la mariposa y apunta a la siguiente dirección del cuarto punto de la mariposa siguiente.
- **Estado mari_asim:** Realiza las operaciones necesarias para obtener los resultados de la mariposa Radix-4 y almacena la respuesta del primer punto.
- **Estado guardar_nuevo_Xn:** Guarda los tres resultados restantes en las direcciones de memoria correspondientes.
- **Estado ver_pos_Xn:** Dentro de este proceso se revisa si todavía existen bloques por calcular, si todavía quedan bloques, se apunta al siguiente bloque. Todo esto se realiza con ayuda de los vectores pilotos p1, p2 y p3.
- **Estado ver_etapa:** Proceso que verifica si existen etapas por calcular, si todavía hay etapas, se suma en uno la etapa y si se ha llegado a la penúltima etapa se termina de calcular las mariposas Radix-4 e indica que se debe realizar el cálculo de las mariposas Radix-2. Además, indica que se terminó de realizar la FFT.
- **Estado espera_dato_r2:** espera un ciclo de reloj.

- **Estado leer_Dato_r2:** Extrae las dos muestras necesarias para obtener la mariposa Radix-2.
- **Estado mari_r2:** Realiza las operaciones necesarias para el cálculo de la mariposa Radix-2. Además, de almacenar el primer resultado obtenido.
- **Estado guardar_res_r2:** Almacena el siguiente resultado de la mariposa Radix-2 obtenida en el estado mari_r2.
- **Estado sig_mari_r2:** Revisa si existen bloques por calcular dentro de la última etapa mediante los vectores pilotos p1, p2 y p3.
- **Estado fin:** Indica al proceso extraer_respuestas que terminó con el cálculo de la FFT y que empiece el envío de los resultados.

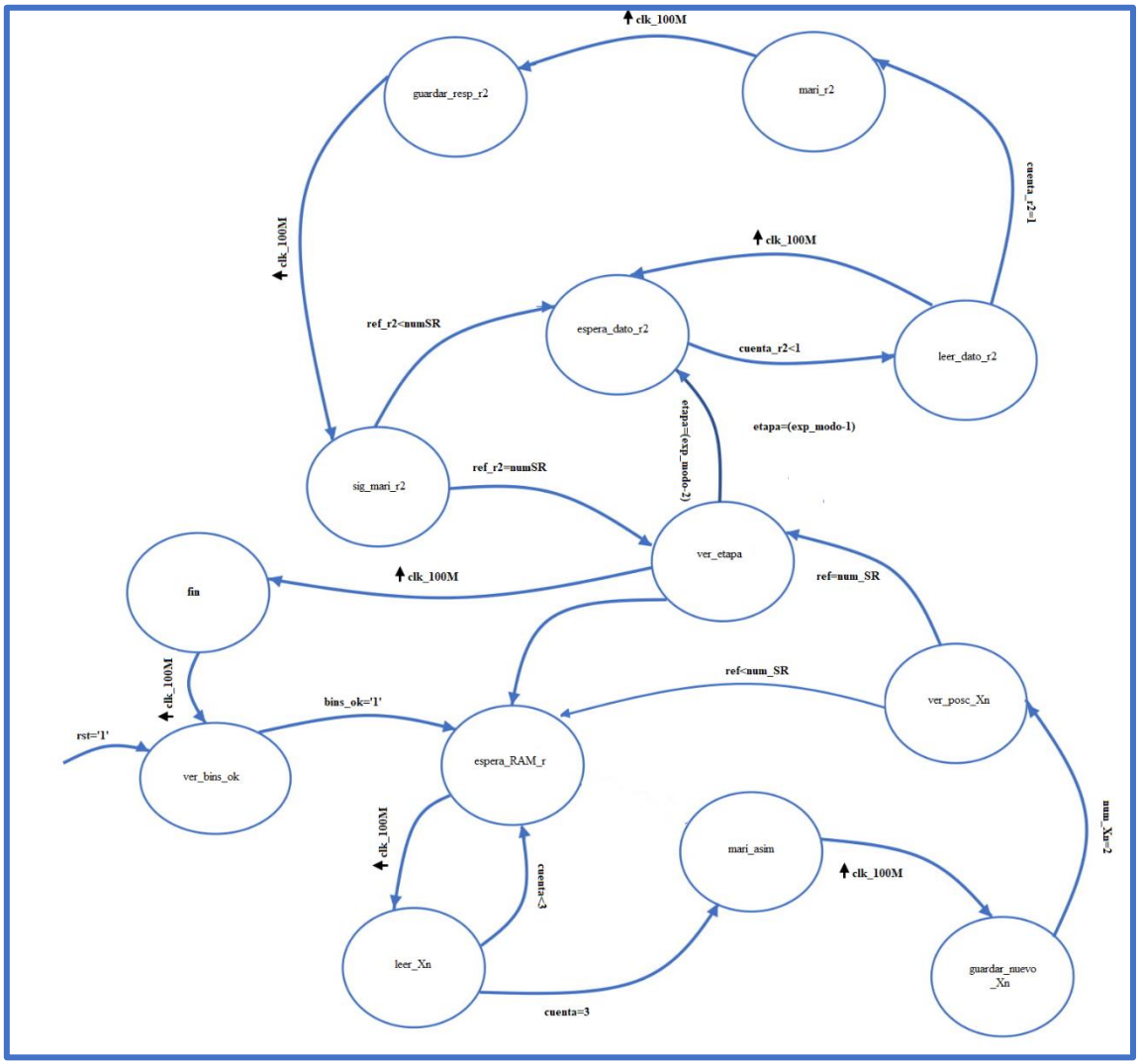


Figura 2.14. Máquina de estados del proceso SR_asim

2.2.3.5. EXTRAER RESULTADOS

Este proceso extrae los resultados de la memoria RAM una vez que haya terminado el proceso de la FFT de acuerdo al proceso anterior. La máquina de estados de este proceso se muestra en la figura 2.15.

- **Estado inicio:** Espera hasta que el proceso SR_asim termine de obtener la FFT mediante la señal fin_FFT.
- **Estado extraer_resp:** Llama a la función *reverse* para obtener la posición en bit inverso y ordenar las muestras para el envío.
- **Estado espera_envio:** Espera hasta que la señal *hecho* sea igual a uno para extraer la siguiente muestra de la memoria RAM.
- **Estado ultima_muestra:** Estado que espera hasta que se envíe la última muestra.
- **Estado fin:** Espera un ciclo de reloj para volver al estado inicio.

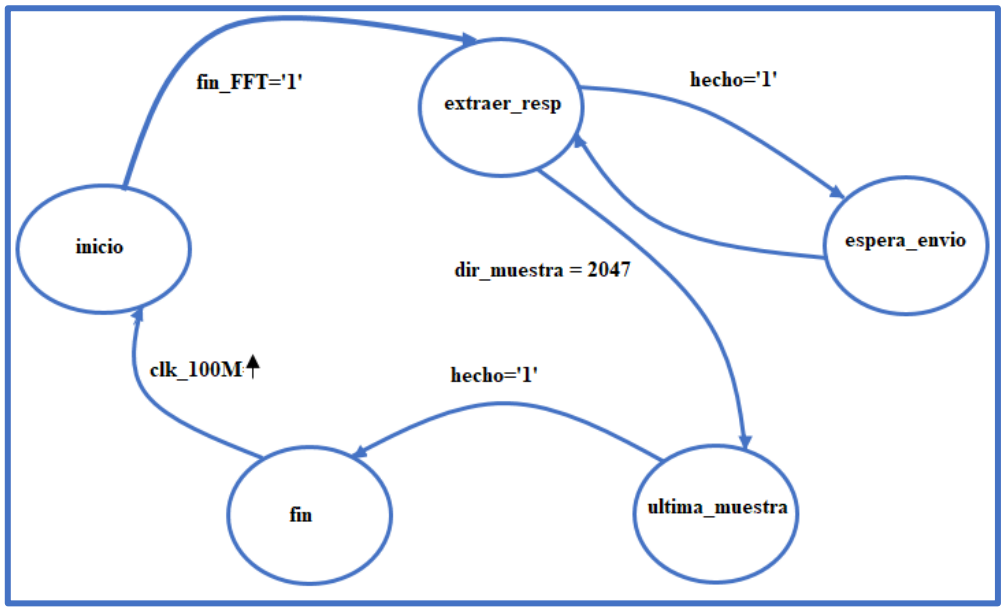


Figura 2.15. Máquina de estados del proceso extraer_respuestas

2.2.3.6. SERIALIZADO PARA ENVÍO

Este proceso se encarga de recibir la muestra obtenida en el proceso anterior y agrega el bit de inicio y de parada para poder enviar por comunicación serial. La máquina de estados de este proceso se muestra en la figura 2.16.

- **Estado inicio:** Espera hasta que el proceso SR_asim termine de obtener la FFT mediante la señal fin_FFT.

- **Estado espera_orden:** Espera que la señal *serializar* sea igual a uno, esto indica que se extrajo el dato correctamente de la memoria RAM.
- **Estado tren_serial:** Divide los datos en ocho tramas para poder ser enviados. Además, añade un bit de inicio y parada por cada trama conformando una señal de 80 bits.
- **Estado retardo:** Espera un ciclo de reloj.
- **Estado regreso:** Estado que indica que se terminó de conformar las tramas para ser enviadas.

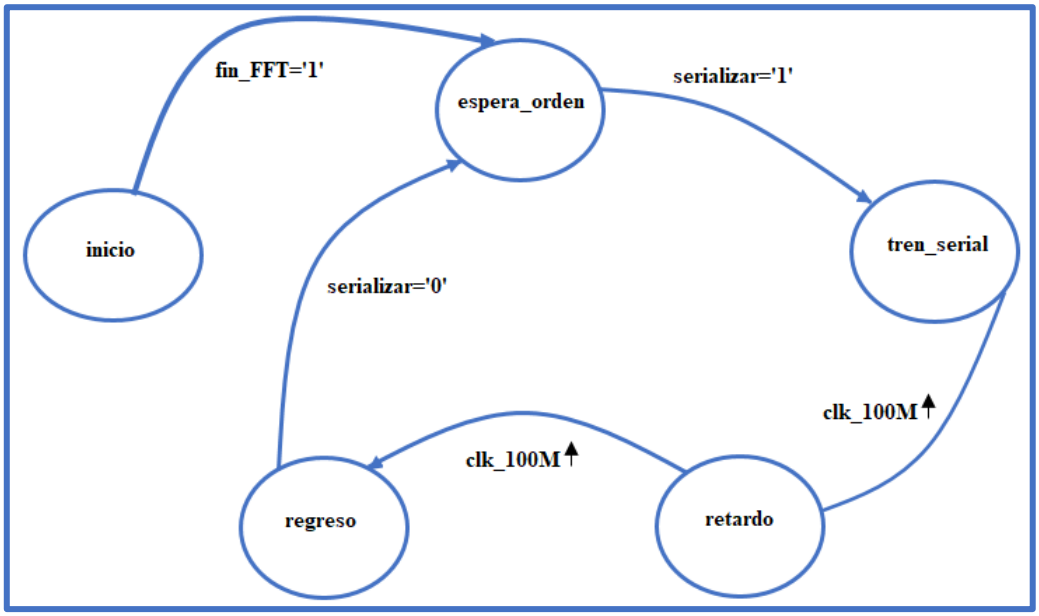


Figura 2.16. Máquina de estados del proceso serializado

2.2.3.7. TRANSMISIÓN SERIAL

Estado que es controlado por el reloj de 4800 Hz para poder enviar bit a bit la señal conformada en el proceso anterior. Espera hasta que la señal *enviar* sea igual a uno para comenzar el envío e indica que se terminaron de enviar los 80 bits mediante la señal *hecho* para poder recibir el siguiente dato.

2.3. LENGUAJE PYHTON [21].

Lenguaje creado por Guido van Rossum a inicios de los años 90, su nombre fue inspirado por el grupo de cómicos ingleses Monty Python. Python es un lenguaje de script o interpretado, no es necesario definir el tipo de dato que va a contener una variable, para

cambiar el tipo de variable es necesario convertir de forma explícita dicha variable, orientado a objetos y multiplataforma [21].

- **Lenguaje de script o interpretado**: Es aquel que necesita de un programa intermedio llamado intérprete que ejecute el código dentro de un script.
- **Orientado a objetos:** Este tipo de programación se traslada a clases y objetos dentro del programa creado.
- **Multiplataforma:** Está disponible para varias plataformas o softwares.

2.3.1. MÓDULOS [22]

Un módulo contiene código PYTHON y su extensión es .py, en la cual se almacenan variables e implementan funciones. Estos módulos pueden ser ejecutados desde consola o también pueden ser llamados mediante el código 2.21 que extrae todo el módulo, además es posible utilizar el código 2.22 para extraer una parte del módulo [22].

```
from nombre_modulo import *
```

Código 2.21. Sintaxis en Python para declarar módulos

```
from nombre_modulo import variable
from nombre_modulo import funcion
```

Código 2.22. Sintaxis en Python para declarar módulos y extraer una parte de este

Módulos necesarios para la implementación de la interfaz gráfica del analizador de espectros.

- **Serial:** Módulo que permite configurar el puerto por el cual se va a realizar la comunicación serial.
- **Time:** Módulo que permite establecer un tiempo para la configuración del puerto.
- **Re:** Módulo que permite hacer búsquedas dentro de una variable.
- **Numpy:** Módulo que permite realizar operaciones matemáticas y trigonométricas tales como: valor absoluto, seno, coseno, tangente, etc.
- **Tkinter:** Módulo que permite crear la interfaz gráfica y sus componentes tales como: botones, entradas de texto, salidas de texto, etc.
- **Scipy:** Módulo que permite obtener la FFT.
- **Matplotlib.pyplot:** Módulo que permite graficar funciones.

- **Matplotlib.backends.bacjend_tkaff:** Módulo que permite el ingreso del cursor a la gráfica y obtener la coordenada al dar un clic.

En el siguiente proyecto se utiliza el módulo *Time* para dar 3 segundos de espera hasta que se configure el puerto serial con los parámetros ingresados en el módulo *Serial* y empezar la comunicación. El módulo Re se utiliza para buscar datos en un array (arreglo o vector) y poder unir datos, mientras que el módulo *Numpy* se utiliza para crear vectores. El módulo *Tkinter* se utiliza para crear la interfaz gráfica añadiendo botones, entradas y salidas de texto y poder ubicarlas en el lugar que se desee.

El módulo *Scipy* se utiliza para obtener la FFT de las muestras que ingresen a dicho módulo, mientras que el módulo *Matplotlib.pyplot* permite graficar funciones y el módulo *Matplotlib.backends.bacjend_tkaff* permite obtener la coordenada del punto en que se da un clic para posteriormente graficar la coordenada obtenida.

2.3.2. FUNCIONES EN PYTHON [23]

Una función puede o no tener valores de entrada y en caso de requerirse puede realizar operaciones en base a estos. La función puede recibir uno o más parámetros, los cuales irán separados por una coma. La sintaxis para declarar una función se muestra en el código 2.23 [23]. Los valores de entrada pueden ser modificados dentro de la función y ser reutilizados en otra, para esto es necesario declarar que se trata de una variable global dentro de la función, su sintaxis se muestra en el código 2.24. La manera para llamar a una función creada dentro del script se muestra en el código 2.25.

```
def funcion():
    #algoritmo

def funcion(parametro1, parametro2):
    #algoritmo
```

Código 2.23. Sintaxis para declarar funciones en Python

```
def funcion(parametro1, parametro2):
    global parametro1
    parametro1=parametro1+parametro2

def funcion1(parametro1):
    global parametro1
    print(str(parametro1))
```

Código 2.24. Sintaxis para declarar variables globales en Python

```
aux=funcion(2,4)
```

Código 2.25. Sintaxis para llamar e ingresar datos a una función en Python

Existen funciones previamente definidas en Python, es decir, funciones implementadas con palabras reservadas que no pueden ser utilizadas como variables en el código. Ejemplo la función *"print"* que imprime en consola una cadena de caracteres.

Dentro del proyecto se utilizó varias funciones ya que son procedimientos recursivos, tal como el envío y recepción de datos. De la misma manera para la presentación de las gráficas y sus límites en el eje horizontal.

2.3.3. TIPOS DE DATOS [23]

Los tipos de datos más utilizados dentro de Python son: entero, flotante, cadena de caracteres y booleano. Es posible cambiar el tipo de dato si antecedemos las palabras reservadas *int, float o str.* La sintaxis para cambiar el tipo de datos se muestra en el código 2.26.

```
variable1= float(2.0)
variable1=int(variable1)
```

Código 2.26. Sintaxis para cambiar el tipo de dato

Las variables de tipo flotante se utilizan al momento de recibir las muestras y convertirlas en el valor equivalente, mientras que el dato de tipo entero es utilizado en la entrada de texto de la frecuencia central ya que si ingresamos un dato sin decimales la variable se definirá con entero.

2.4. SPYDER [24]

Spyder integra funciones avanzadas de edición, depuración, análisis con exploración de datos, inspección profunda, ejecución interactiva y capacidades de visualización de un paquete científico [24].

Las componentes de Spyder son:

- **Editor:** Espacio de trabajo donde se definen librerías, funciones y código de ejecución. Posee un navegador de funciones/clases, finalización automática, herramientas de análisis de código y definición de acceso.

- **Consola de IPython:** Espacio donde se ejecuta un código por línea, celda o archivo y se pueden apreciar gráficos.
- **Explorador de archivos:** Permite visualizar todos los archivos que se encuentran dentro de la carpeta en que se está trabajando.

2.4.1. DESARROLLO DE LA INTERFAZ PARA EL ANALIZADOR DE ESPECTRO EN PYTHON

La interfaz del analizador de espectro tiene como parámetros de entrada: frecuencia central ya sea en KHz o MHz, resolución, frecuencia de muestreo y span estos parámetros varían de acuerdo con la figura 2.17.

La frecuencia central y span elegidos muestran el ancho de banda de la gráfica mostrada, mientras que la frecuencia de muestreo y la resolución depende del tipo de señal que se desea observar.

La formación de la interfaz gráfica comprende de un archivo con extensión .py, el cual se encuentra en el anexo C de este proyecto.

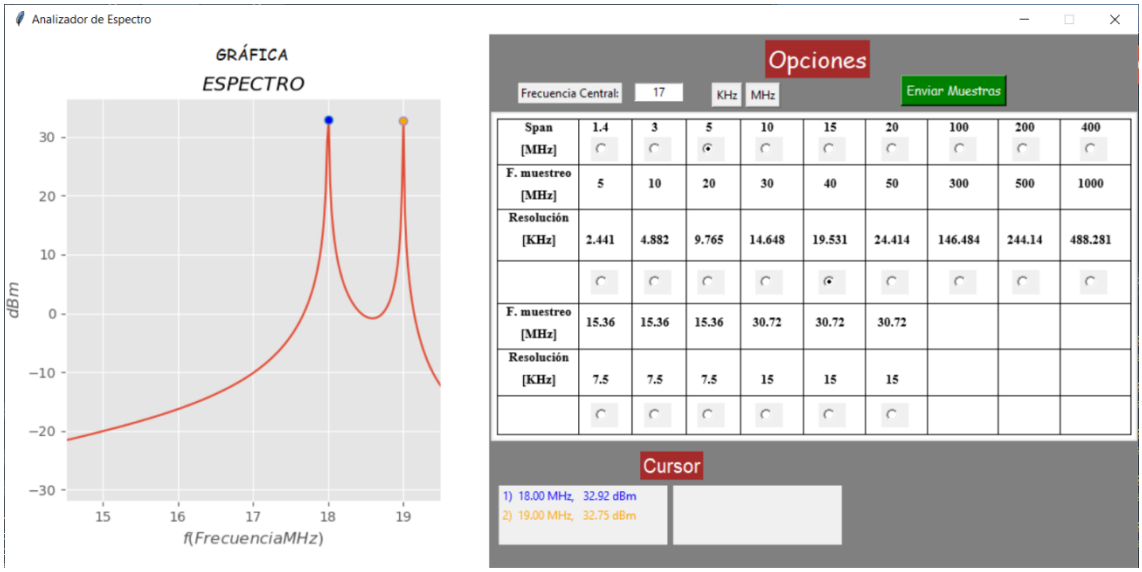


Figura 2.17. Interfaz gráfica para el analizador de espectros

La gráfica muestra en el eje horizontal la frecuencia, mientras que en el eje vertical se especifica la potencia en decibelios por miliwatio. Ya que las muestras son señales de voltaje es necesario la conversión a miliwatios, mediante la siguiente fórmula:

$$dBm = 10\log_{10}\left(\frac{V^2}{Z} * 1000\right) \quad (2.1)$$

Donde V es la magnitud del voltaje medido en voltios, mientras que Z es la impedancia característica de la antena del receptor.
La interfaz permite el ingreso del puntero del ratón a la gráfica y obtener las coordenadas del punto en que se da un clic, de manera que muestra el valor correspondiente en el área llamada cursor.

2.4.1.1. ENTRADA Y SALIDA DE DATOS

Para la transmisión de las muestras y recepción de los resultados obtenidos en la FPGA, es necesario abrir un puerto de comunicación serial en Python, como se muestra en el código 2.27.

```
nombre_comunicacion=serial.Serial("pueto de comunicación", velocidad de transmisión, ...
                                  ...timeout=7, stopbits=1, bytesize=8, parity='N')
```

Código 2.27. Sintaxis para la recepción y transmisión de datos en Phyton

El módulo *serial* permite la configuración de parámetros para la comunicación serial, tal como: puerto COM, velocidad de transmisión, tiempo de inactividad del puerto, bits de parada, bits de datos y paridad.

- **Puerto de comunicación:** La computadora debe reconocer a la FPGA mediante un puerto para la comunicación, mismo que puede verificarse en el administrador de dispositivos.
- **Velocidad de transmisión:** Velocidad en baudios en que los datos son enviados y recibidos mediante el puerto COM abierto.
- **Timeout:** Tiempo de inactividad del puerto COM, si no recibe o envía información en el tiempo configurado se cierra el puerto.
- **Stopbits:** Bits de parada que conforma la trama.
- **Bytesize:** Bits de datos que conforma la trama.
- **Paridad:** Paridad de la trama enviada o recibida: sin paridad ‘N’, paridad par ‘E’ y paridad impar ‘O’.

Para la configuración del puerto es necesario dar un tiempo de establecimiento mediante el comando mostrado en el código 2.28, en este caso se esperan 3 segundos hasta que la configuración del puerto haya sido exitosa.

```
time.sleep(3)
```

Código 2.28. Sintaxis para tiempo de espera en Phyton

La transmisión de las muestras se hace mediante el comando *write,* mientras que el comando *encode* codifica las muestras en código LATIN, sintaxis que se muestra en el código 2.29.

```
nombre_comunicacion.write(muestras.encode('Latin'))
```

Código 2.29. Sintaxis para transmisión de datos en Phyton

La recepción de los datos se hace mediante el comando *readall*, mientras que el comando *decode* decodifica las muestras en código LATIN, sintaxis que se muestra en el código 2.30.

```
datos_recibidos=nombre_comunicacion.readall()
datos_decodificados=datos_recibidos.decode('Latin')
```

Código 2.30. Sintaxis para recepción de datos en Phyton

2.5 PROGRAMACIÓN DE LA FPGA EN VIVADO

Para verificar errores de codificación es necesario hacer una síntesis del programa a implementar el cual se encuentra en el anexo D de este proyecto., para esto en el subpanel FLOW NAVIGATOR escoger la opción *run synthesis* (figura 2.18). Si no existen errores se desplegará una ventana para hacer la implementación del código en la opción *run implementation* (figura 2.19), caso contrario los errores se mostrarán en la pestaña *TCL Console* remarcada de color celeste (figura 2.18). A continuación, se deben asignar los pines de la entidad transformada mediante el archivo transformada.xdc (figura 2.20), que se encuentra en el anexo E de este proyecto.

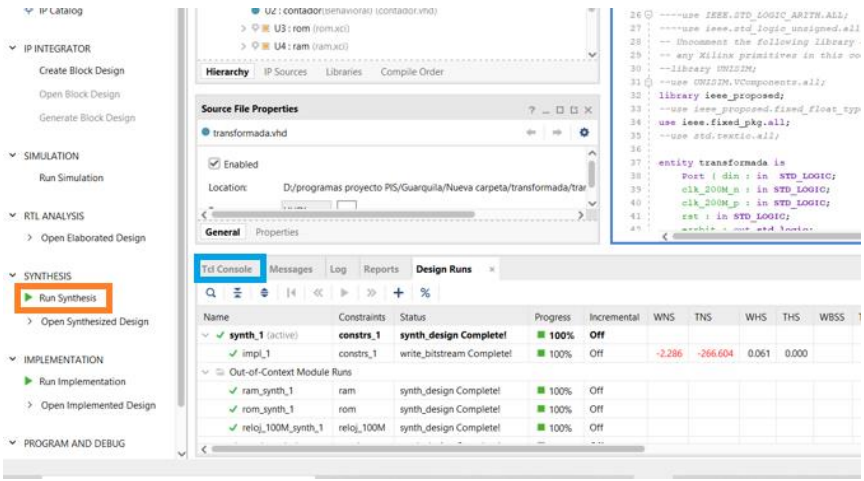


Figura 2.18. Síntesis del código a implementar

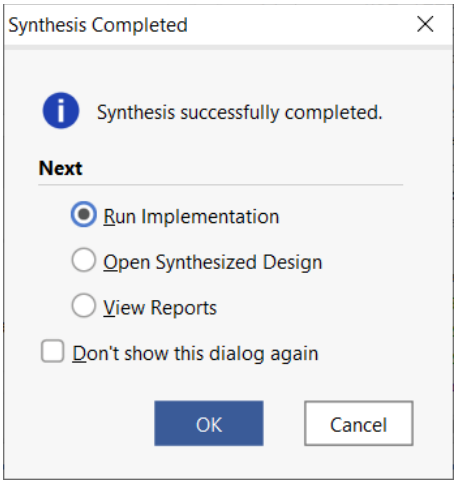


Figura 2.19. Síntesis exitosa

```
set_property PACKAGE_PIN AD12 [get_ports clk_200M_p]
set_property PACKAGE_PIN Y20 [get_ports din]
set_property PACKAGE_PIN Y23 [get_ports dout]
set_property PACKAGE_PIN W23 [get_ports errbit]
set_property PACKAGE_PIN W24 [get_ports led_reset]
set_property PACKAGE_PIN V26 [get_ports led_rx]
set_property PACKAGE_PIN V20 [get_ports led_fft]
set_property PACKAGE_PIN U29 [get_ports led_tx]
set_property PACKAGE_PIN P27 [get_ports rst]

set_property IOSTANDARD DIFF_HSTL_II_18 [get_ports clk_200M_p]
set_property IOSTANDARD LVCMOS18 [get_ports din]
set_property IOSTANDARD LVCMOS18 [get_ports dout]
set_property IOSTANDARD LVCMOS18 [get_ports errbit]
set_property IOSTANDARD LVCMOS18 [get_ports led_reset]
set_property IOSTANDARD LVCMOS18 [get_ports led_rx]
set_property IOSTANDARD LVCMOS18 [get_ports led_fft]
set_property IOSTANDARD LVCMOS18 [get_ports ledtx]
set_property IOSTANDARD LVCMOS18 [get_ports rst]
```

Figura 2.20. Asignación de pines

Si la implementación se realizó con éxito se debe generar el archivo .bit, el cual se debe cargar a la FPGA, para esto se escoge la opción *Generate Bitstream* (figura 2.21). A continuación, se debe verificar que la tarjeta se encuentre conectada a la computadora mediante el puerto USB y JTAG de la FPGA, para esto en el subpanel FLOW NAVIGATOR dentro de la opción *Open Hardware Manager* elegir *Open Tarjet* y escoger *Auto Connect* (figura 2.22). Seguidamente en el subpanel Hardware se muestra el tipo de tarjeta conectada remarcado de color naranja (figura 2.23) y se habilita la opción *Program Device* en el subpanel FLOW NAVIGATOR opción remarcada de color celeste. Inmediatamente se desplegará la ventana PROGRAM DEVICE donde se indica la ubicación y el archivo que se va a cargar a la FPGA (figura 2.24).

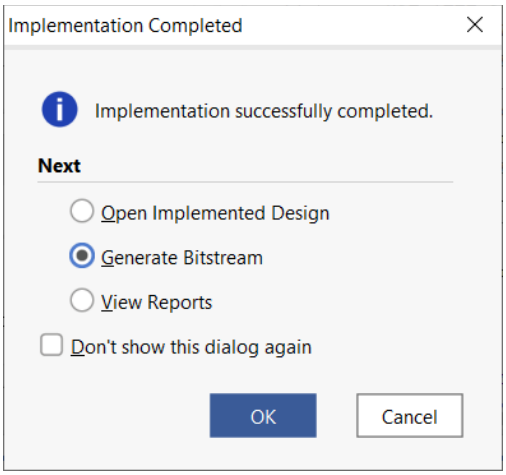


Figura 2.21. Implementación exitosa

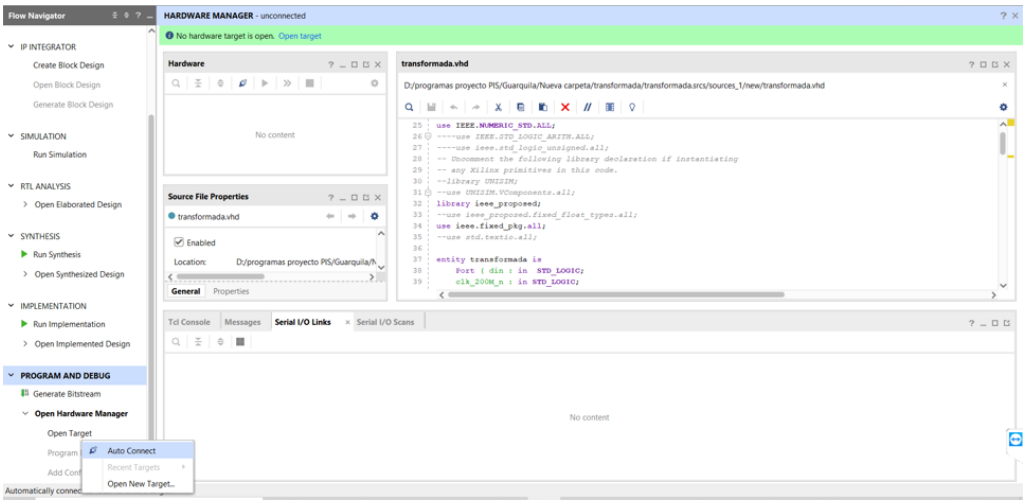


Figura 2.22. Verificación de la tarjeta FPGA

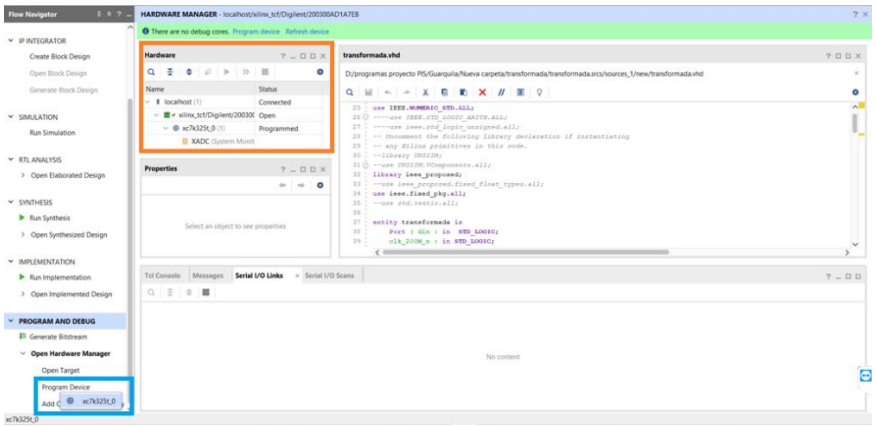


Figura 2.23. Tipo de tarjeta FPGA conectada

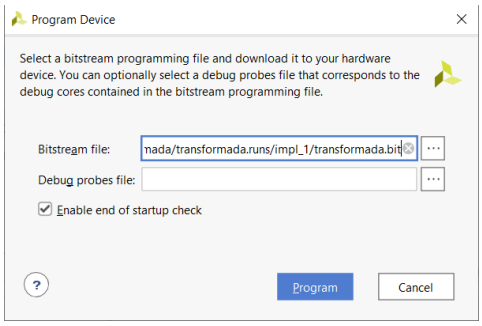


Figura 2.24. Archivo a cargar en la FPGA

2.5.1. INTERFAZ GRÁFICA EN MATLAB PARA VERIFICACIÓN DEL BLOQUE FFT DE LA FPGA

Para la ejecución del algoritmo en la FPGA se envían y reciben 2048 muestras de una señal; estas muestras generadas a una frecuencia de muestreo determinada se envían desde una computadora con Matlab, y se reciben en la FPGA, la cual envía el resultado obtenido de la FFT mediante el algoritmo Split-Radix y compara el resultado calculado en la FPGA con el obtenido en la computadora mediante Matlab, dicha comparación se evalúa mediante el parámetro de error de cuantización, para esto se usa la SQNR [25] (relación señal a ruido de cuantización), expresión que se muestra en la ecuación (2.2):

$$SQNR|_{dB} = 10\ log\ \left[\frac{\frac{1}{N}\sum_{-\infty}^{\infty} x(n)}{\frac{1}{N}\sum_{-\infty}^{\infty} e(n)}\right] \tag{2.2}$$

Donde *x(n)* es la señal cuantificada y *e(n)* es el error de cuantización.

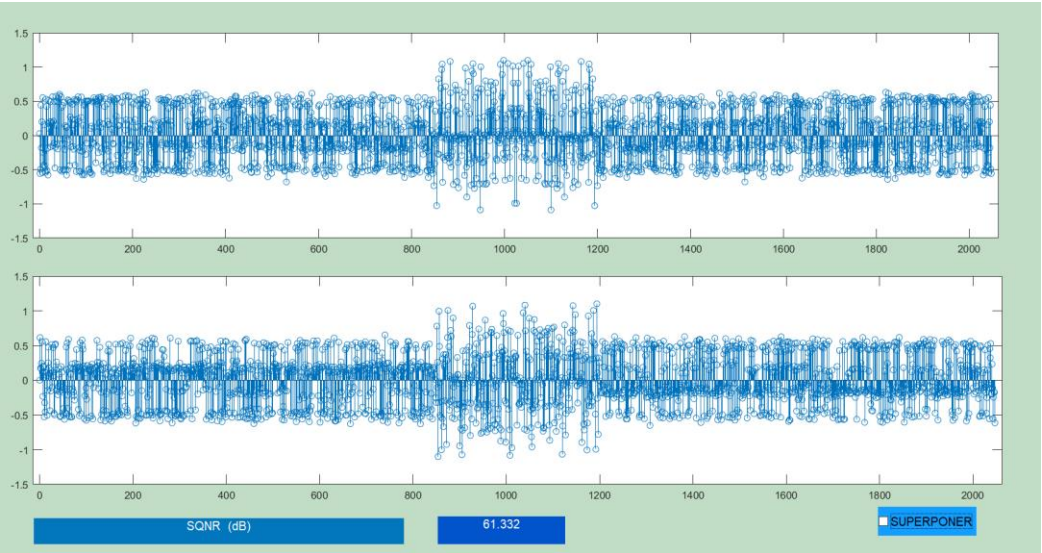


Figura 2.25. Señal FFT obtenida de la FPGA

Los datos recibidos son graficados separando parte real e imaginaria, tal como se muestra en la figura 2.25, programa que se adjunta en el anexo F de este proyecto, de la misma manera se presenta el parámetro SQNR de la señal recibida. La interfaz tiene la opción de superponer las gráficas, es decir, permite graficar en la misma interfaz la señal de la FFT obtenida con la FPGA y la de "referencia" calculada en Matlab, tal como se muestra en la figura 2.26.

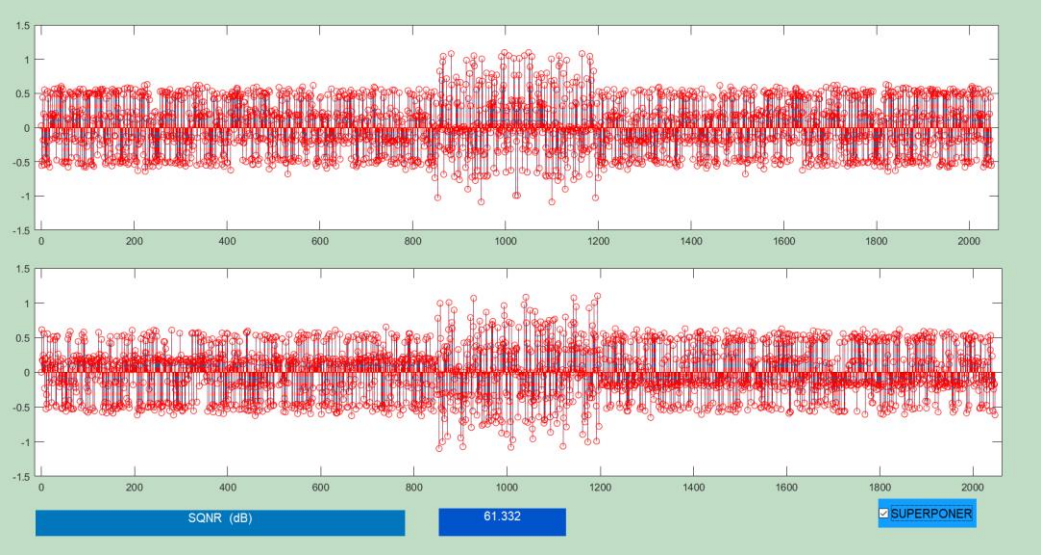


Figura 2.26. Señales FFT de la FPGA y Matlab superpuestas

De la figura 2.26 se observa que el parámetro SQNR de la señal es de 61.32 medido en dB (decibelios), por lo que se tiene una gran aproximación en el cálculo de la FFT obtenida en la FPGA y la función FFT de Matlab. Este parámetro puede variar dependiendo del tipo de señal y ruido que contenga la señal de prueba.

3. RESULTADOS Y DISCUSIÓN

3.1. INTERFAZ GRÁFICA EN PYHTON PARA EL ANALIZADOR DE ESPECTRO.

La interfaz gráfica del analizador de espectro implementado en el proyecto se muestra en la figura 3.1, en la que se pueden variar parámetros de frecuencia central, span, frecuencia de muestreo y resolución, estos 2 últimos parámetros se seleccionan paralelamente. La FPGA se conecta a un computador mediante un cable USB tanto para la programación de la tarjeta como para la transmisión y recepción de los datos. Cada uno de los parámetros de la interfaz se describe a continuación.

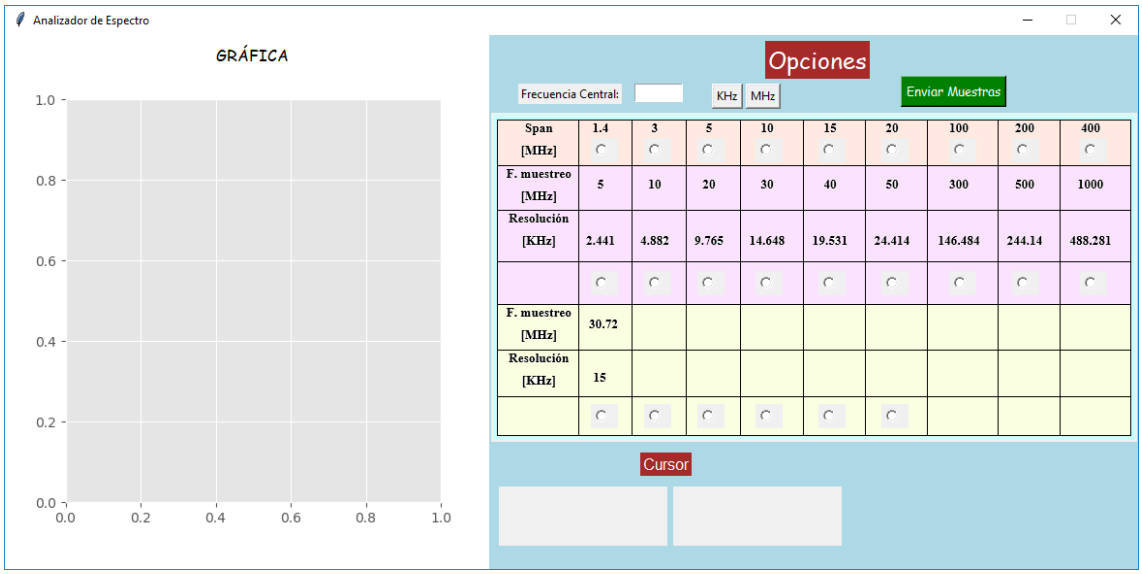


Figura 3.1. Interfaz gráfica del analizador de espectro

- **Frecuencia central:** Valor de la frecuencia que se ingresa por teclado y es el centro del rango de frecuencias mostrado.
- **SPAN:** Rango de frecuencias que se muestra en la gráfica, es decir, este define el límite inferior y superior dependiendo de la frecuencia central.

$$\boldsymbol{limite\ inferior\ =\ frecuencia\ central\ -\ SPAN/2}$$

$$\boldsymbol{limite\ superior\ =\ frecuencia\ central\ +\ SPAN/2}$$

- **Frecuencia de muestreo y resolución:** estos dos parámetros se relacionan entre sí, ya que dependiendo de la frecuencia de muestreo se obtiene la resolución. La frecuencia de muestreo varía dependiendo de la señal que se desea observar, ya que la frecuencia máxima de la señal a observar debe ser menor o igual a la mitad de la frecuencia de muestreo. En cambio, la resolución

representa el tamaño del paso del espectro del cual se presenta de forma gráfica el nivel de señal y su fórmula se incluye a continuación.

$$\boldsymbol{Resolución} = \frac{\boldsymbol{Frecuencia\ de\ muestreo}}{\boldsymbol{2048}} \quad (3.1)$$

Donde: 2048 es la longitud del algoritmo Split-Radix.

- **Botón Enviar Muestras:** Este botón lee el archivo .txt que contiene las muestras de la señal que se encuentra a la salida de la etapa de radiofrecuencia (salida del mixer) y las envía a la FPGA por la interfaz serial. Así mismo, se reciben los datos o los resultados de la transformada de Fourier que se encuentran concatenados de la siguiente manera: 8 tramas recibidas contienen parte real e imaginaria de la FFT calculada en la FPGA. Como indicación al usuario este botón se pinta de color gris cuando está enviando o recibiendo datos.

Además, la interfaz permite el ingreso de cursores o marcadores a la gráfica en la cual se pueden presentar hasta 6 marcadores mediante un clic izquierdo.

3.2. PRUEBAS

Para realizar las pruebas se deben enviar por comunicación serial los datos muestreados y leídos de un archivo .txt en Python, programa que se adjunta en el anexo G de este proyecto. Seguidamente deben ser procesados en la tarjeta FPGA para obtener la FFT. A continuación, se deben enviar los resultados obtenidos por la FPGA a través de la interfaz serial y deben ser leídos nuevamente en Python.

Una vez recibido el resultado de la FFT calculada en la FPGA, se debe escoger la frecuencia de muestreo de acuerdo a la especificada para generar los datos del archivo .txt, span y la frecuencia central. Se debe mencionar que una selección adecuada de los parámetros anteriores permitirá presentar correctamente los resultados en la interfaz gráfica. Al momento de ingresar la frecuencia central y escoger el botón KHz o MHz se toman los valores previamente configurados y se muestra el espectro tomando en cuenta los parámetros escogidos.

Para la ejecución de las pruebas de funcionamiento del analizador de espectros implementado se requieren los siguientes elementos:

- Computadora con Spyder[12].

[12] Spyder: Software utilizado para crear el código Pyhton [24].

- Tarjeta de entrenamiento XC7K325T-2FFG900C.
- Fuente de alimentación de la tarjeta de entrenamiento.
- Cable JTAG – USB.
- Cable UART – USB.

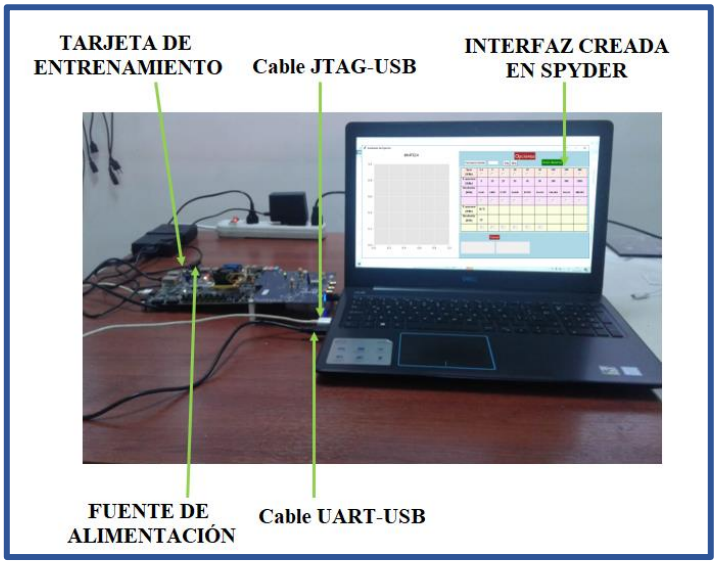


Figura 3.2. Elementos para pruebas del analizador de espectro

En la figura 3.2 se muestran las conexiones que se requieren para el funcionamiento del analizador de espectros.

A continuación, se muestran los resultados obtenidos con diferentes señales.

Señal de prueba 1.-

Señales senoidales puras con ruido, parámetros configurados en el analizador de espectros: SPAN=400MHz y frecuencia central=2.6GHz.

La expresión en alta frecuencia se muestra en la ecuación (3.2):

$$y(t) = 0.001.* \sin(2 * pi * f1.* t) + 0.001.* \sin(2 * pi * f2.* t) + 0.001.* \cos(2 * pi * f3.* t) + 0.001.* rand(1,2048) \quad (3.2)$$

Donde:

f1=2.5GHz, f2=2.7GHz y f3=2.75GHz

Esta señal se mezcla con un mixer que tiene un oscilador local que se configura de manera automática a 2.4GHz (valor de frecuencia que depende de la frecuencia central y del span), trasladando la señal a baja frecuencia. De esta manera los valores de las frecuencias de la señal son las siguientes: f1=100MHz, f2=300MHz y f3=350MHz. Esta señal es muestreada en baja frecuencia a 1000MHz.

En la figura 3.3 se muestra la señal de baja frecuencia (salida del mixer) muestreada, mientras que en la figura 3.4 se presenta el espectro de la señal obtenido con el analizador de espectros.

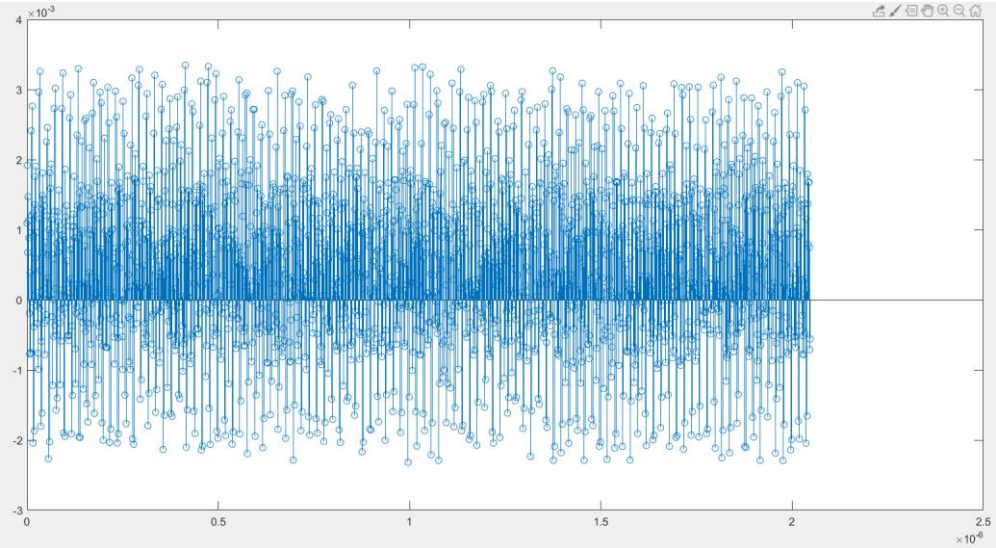


Figura 3.3. Muestras de la señal de prueba 1 tomadas en baja frecuencia

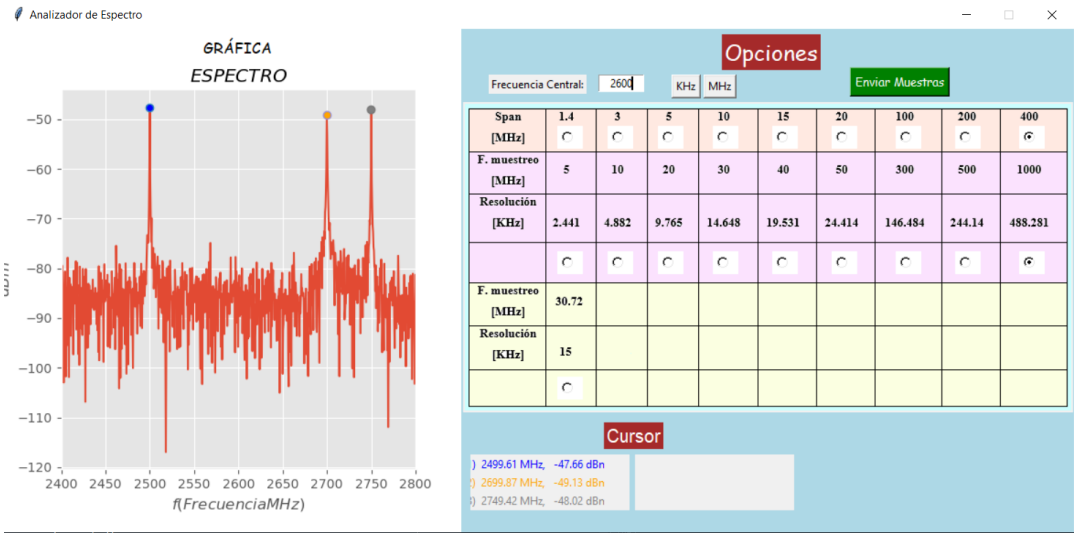


Figura 3.4. Espectro de señal de prueba 1 con ruido

Señal de prueba 2.-

Señal AM con doble banda lateral con frecuencia de portadora de 15 MHz, frecuencia de la señal moduladora de 20 KHz, parámetros configurados en el analizador de espectros: Span=1.4MHz y frecuencia central=16MHz.

La expresión de la señal de prueba 2 en alta frecuencia se muestra en la ecuación (3.3):

$$y = (1 + \sin(2 * pi * Fm * t)) * \sin(2 * pi * fc * t) \quad (3.3)$$

Donde:

Fm=20KHz y fc=15MHz

Esta señal se mezcla en un mixer con la señal de un oscilador local que se configura de manera automática a 15.3MHz, de esta manera los valores de la frecuencia de modulación y portadora son las siguientes: Fm=20KHz y fc=300KHz. Esta señal es muestreada en baja frecuencia a 5MHz y se muestra en la figura 3.5, en la figura 3.6 se muestra el espectro de la señal.

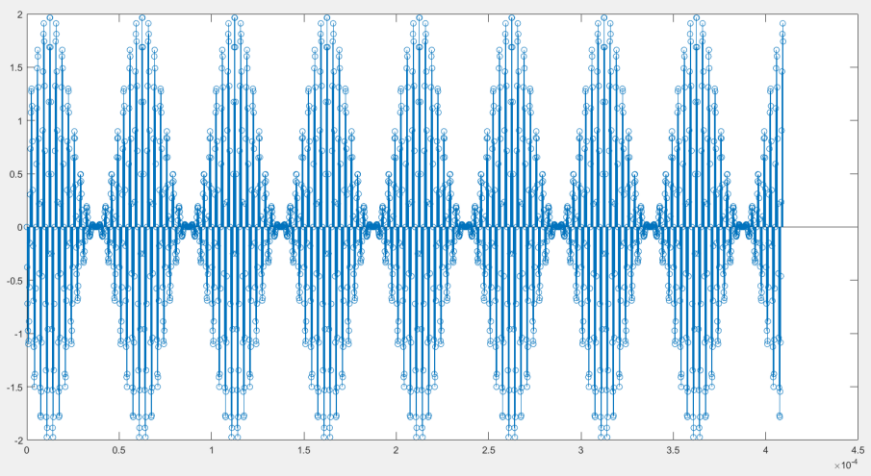


Figura 3.5. Muestras tomadas de señal AM de prueba 2

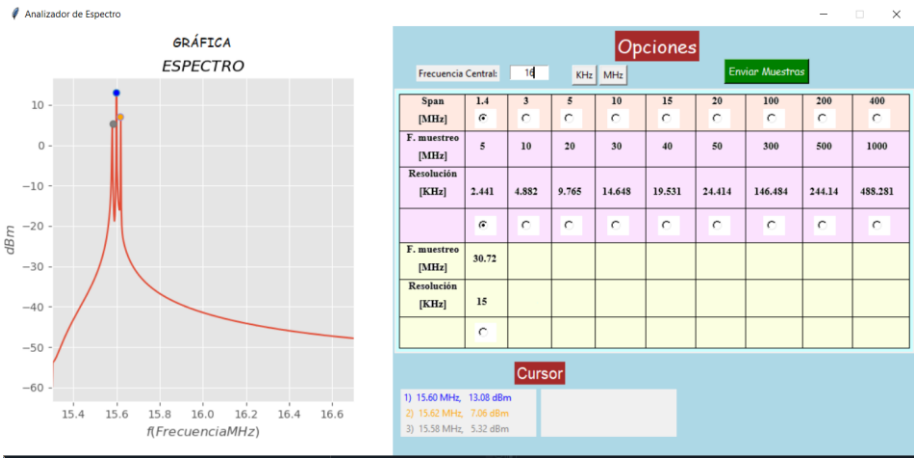


Figura 3.6. Espectro de la señal AM muestreada

Señal de prueba 3.-

Señal AM con doble banda lateral agregada ruido con frecuencia de portadora de 15 MHz, frecuencia de la señal moduladora de 20 KHz, parámetros configurados en el analizador de espectros: Span=3MHz y frecuencia central=16MHz.

La expresión en alta frecuencia se muestra en la ecuación (3.4):

$$y = (1 + \sin(2 * pi * Fm * t)) * \sin(2 * pi * fc * t) \quad (3.4)$$

Donde:

Fm=20KHz y fc=15MHz

Esta señal se mezcla en un mixer con la señal de un oscilador local que se configura de manera automática a 14.5MHz, de esta manera los valores de la frecuencia de la moduladora y portadora son las siguientes: Fm=20KHz y fc=500KHz.

Esta señal es muestreada en baja frecuencia a 10MHz y se muestra en la figura 3.7, en la figura 3.8 se muestra el espectro de la señal.

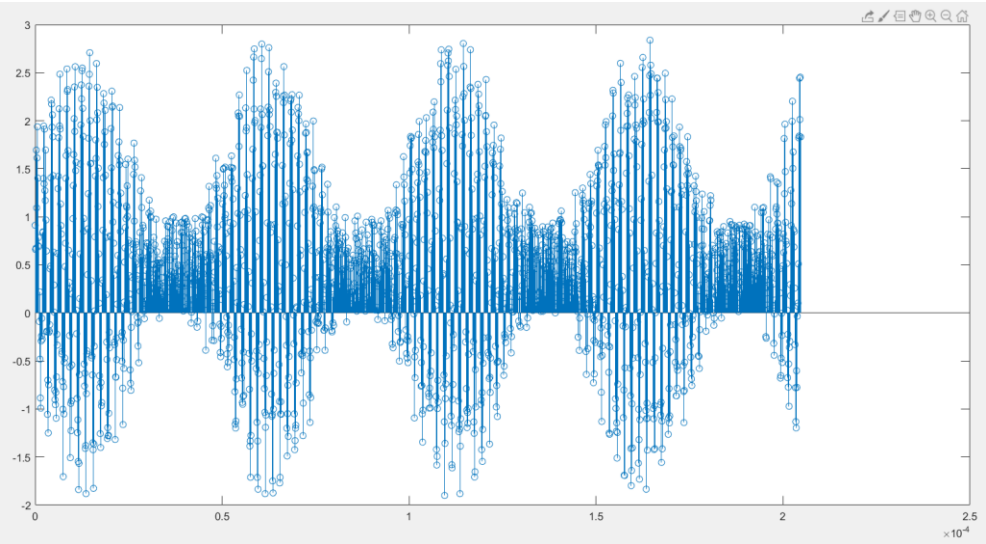


Figura 3.7. Muestras tomadas de señal AM con ruido prueba 3

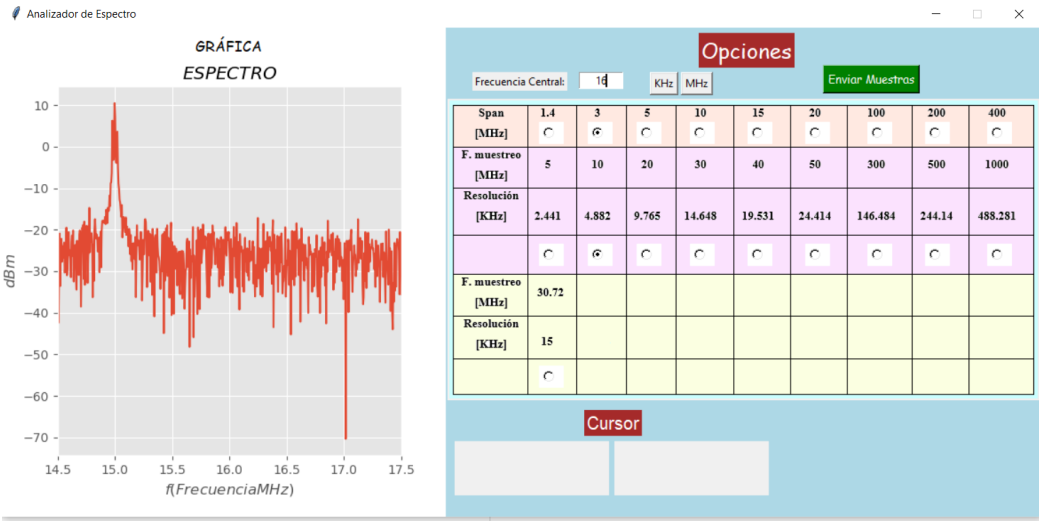


Figura 3.8. Espectro de la señal AM muestreada

Señal de prueba 4.-

Señal FM con índice de modulación igual a uno con frecuencia de portadora de 104.5 MHz, frecuencia de la señal moduladora de 50 KHz, parámetros configurados en el analizador de espectros: Span=1.4MHz y frecuencia central=104.7MHz.

La expresión en alta frecuencia se muestra en la ecuación (3.5):

$$y = \sin(2 * pi * fc * t + \sin(2 * pi * Fm * t)) \quad (3.5)$$

Donde:

Fm=50KHz y fc=104.5MHz.

Esta señal se mezcla en un mixer con la señal de un oscilador local que se configura de manera automática a 104 MHz, de esta manera los valores de la frecuencia de la moduladora y portadora son las siguientes: Fm=50KHz y fc=500KHz. Esta señal es muestreada en baja frecuencia a 5 MHz y se muestra en la figura 3.9, en la figura 3.10 se muestra el espectro de la señal.

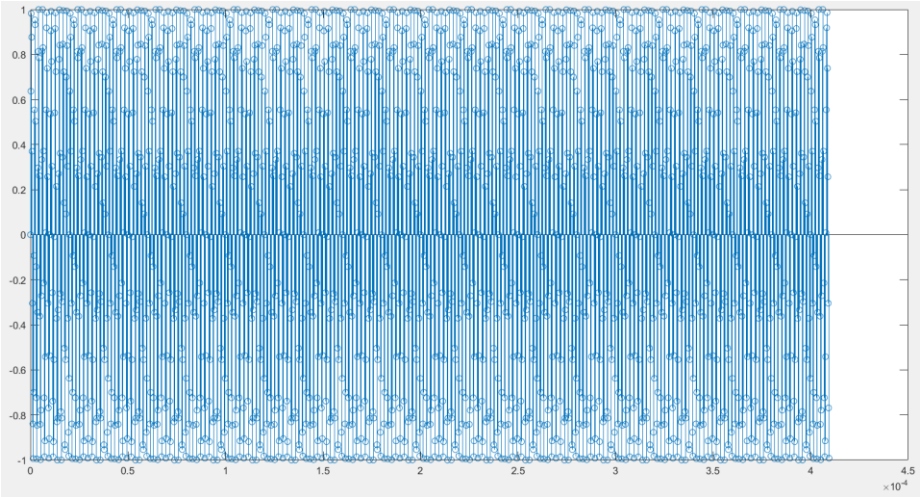


Figura 3.9. Muestras tomadas de la señal FM generada en Matlab

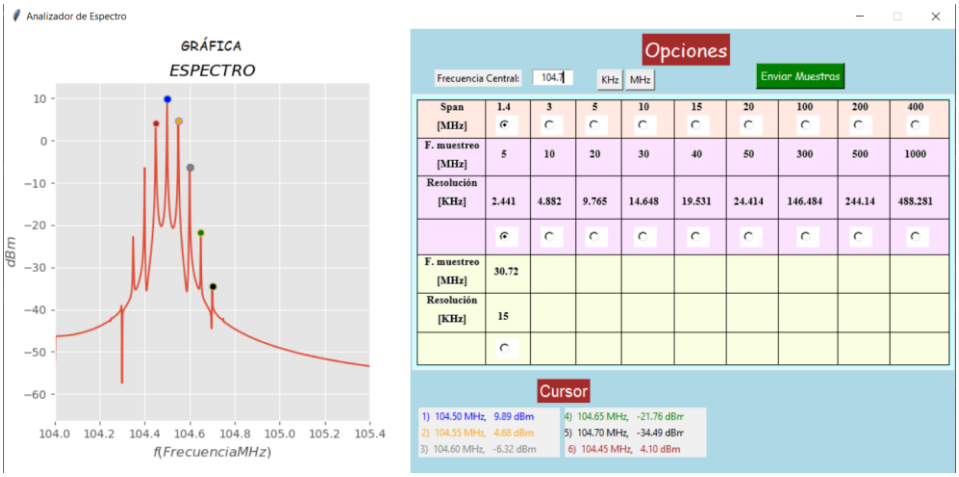


Figura 3.10. Espectro de la señal FM muestreada

Señal de prueba 5.-

Señal senoidal pura sin ruido, parámetros configurados en el analizador de espectros: SPAN=3MHz y frecuencia central=1900MHz.

La expresión en alta frecuencia se muestra en la ecuación (3.6):

$$y(t) = 0.001.* \sin(2 * pi * f1.* t) \quad (3.6)$$

Donde

f1=1900MHz.

Esta señal se mezcla con un mixer que tiene un oscilador local que se configura de manera automática a 1898.5 MHz (valor de frecuencia que depende de la frecuencia

central y del span), de esta manera el valor de la frecuencia es la siguiente: f1=1.5MHz. Esta señal es muestreada en baja frecuencia a 10MHz.

En la figura 3.11 se muestra la señal de baja frecuencia (salida del mixer) muestreada, mientras que en la figura 3.12 se presenta el espectro de la señal obtenido con el analizador de espectros.

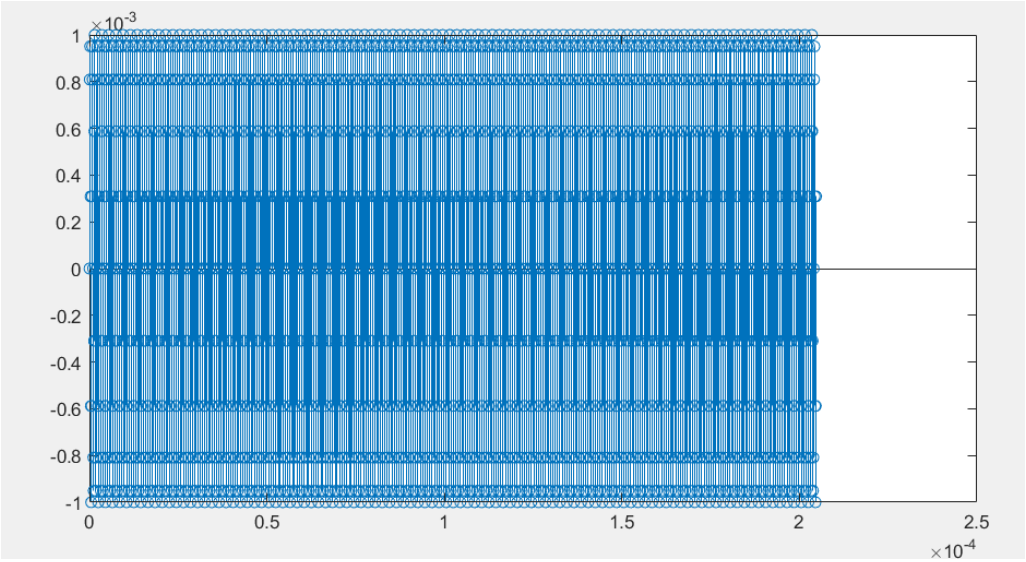


Figura 3.11. Muestras tomadas de la señal senoidal generada en Matlab

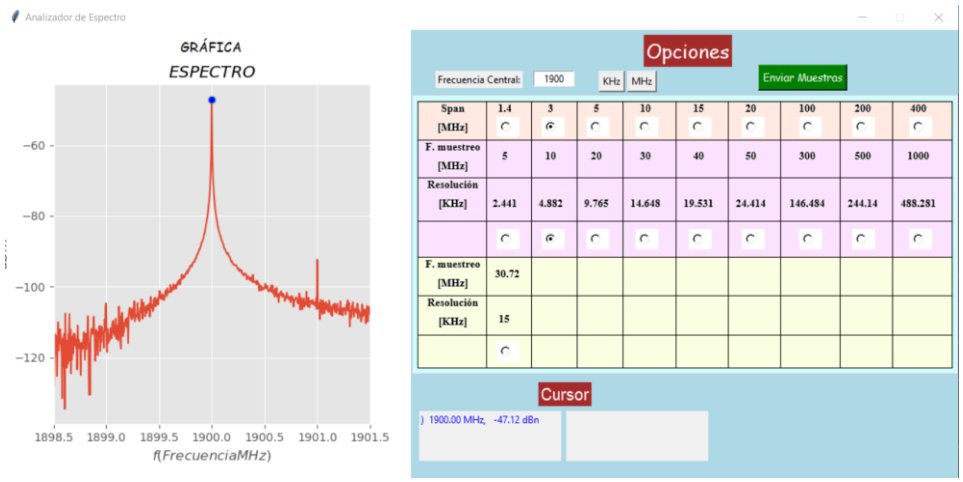


Figura 3.12. Espectro de la señal senoidal muestreada

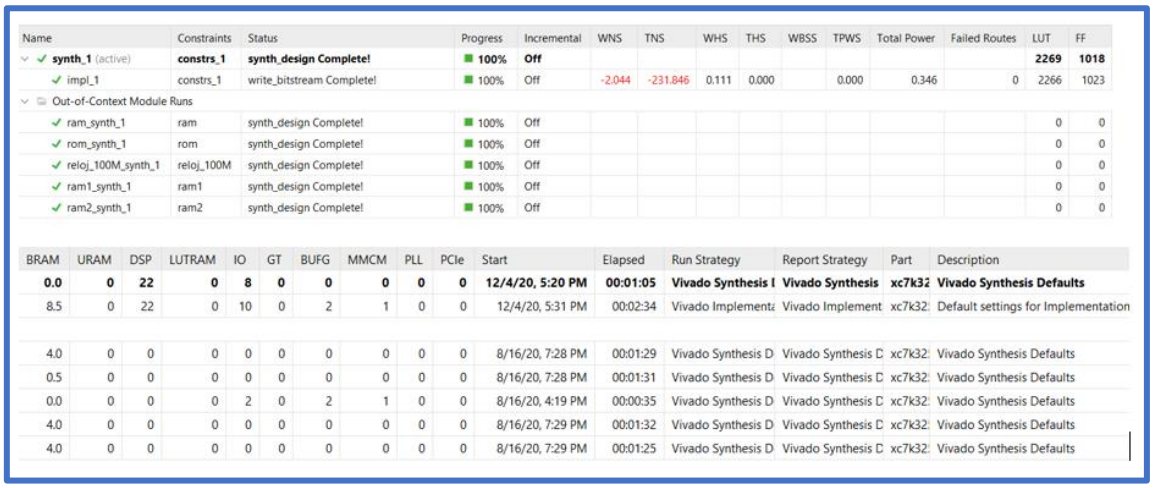
3.3. ANÁLISIS DE LOS RESULTADOS

Después de compilar la síntesis y ejecutar la implementación Vivado muestra una tabla de reporte figura (3.13) donde se indica los recursos utilizados para implementar el código realizado de la arquitectura.

Los parámetros WNS (Worst Negative Slack), TNS (Total Negative Slack), WHS (Worst Hold Slack) y THS (Total Hold Slack) se derivan de los requerimientos de celda funcionales los cuales son [26]:

- Tiempo de configuración: el tiempo antes del cual los nuevos datos deben estar disponibles antes del siguiente cambio de estado de reloj activo para poder ser capturados de forma segura.
- Requisito de retención: la cantidad de tiempo que los datos deben permanecer estables después de cambio de estado de reloj activo para evitar capturar un valor no deseado.
- Tiempo de recuperación: El tiempo mínimo requerido entre el momento en que la señal de reinicio asíncrona ha cambiado a su estado 0L y el siguiente estado de reloj de reloj 1L.
- Tiempo de eliminación: el tiempo mínimo después de un cambio de estado de reloj 1L antes de que la señal de reinicio asíncrona se pueda cambiar de forma segura a su estado 0L.

Las LUT (tablas de consultas) son tablas que almacena valores lógicos y son mayormente usados cuando se crean las memorias RAM. El proyecto implementado utiliza 2269 LUTs en síntesis y 1018 Flip-Flops. Así mismo utiliza 2266 LUTs y 1023 Flip-Flops para implementación tal como se muestra en la figura 3.13.



Name	Constraints	Status	Progress	Incremental	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	LUT	FF
synth_1 (active)	**constrs_1**	**synth_design Complete!**	**100%**	**Off**									**2269**	**1018**
impl_1	constrs_1	write_bitstream Complete!	100%	Off	-2.044	-231.846	0.111	0.000		0.000	0.346	0	2266	1023
Out-of-Context Module Runs														
ram_synth_1	ram	synth_design Complete!	100%	Off									0	0
rom_synth_1	rom	synth_design Complete!	100%	Off									0	0
reloj_100M_synth_1	reloj_100M	synth_design Complete!	100%	Off									0	0
ram1_synth_1	ram1	synth_design Complete!	100%	Off									0	0
ram2_synth_1	ram2	synth_design Complete!	100%	Off									0	0

BRAM	URAM	DSP	LUTRAM	IO	GT	BUFG	MMCM	PLL	PCIe	Start	Elapsed	Run Strategy	Report Strategy	Part	Description
0.0	**0**	**22**	**0**	**8**	**0**	**0**	**0**	**0**	**0**	**12/4/20, 5:20 PM**	**00:01:05**	**Vivado Synthesis I**	**Vivado Synthesis**	**xc7k32**	**Vivado Synthesis Defaults**
8.5	0	22	0	10	0	2	1	0	0	12/4/20, 5:31 PM	00:02:34	Vivado Implementa	Vivado Implement	xc7k32	Default settings for Implementation
4.0	0	0	0	0	0	0	0	0	0	8/16/20, 7:28 PM	00:01:29	Vivado Synthesis D	Vivado Synthesis D	xc7k32	Vivado Synthesis Defaults
0.5	0	0	0	0	0	0	0	0	0	8/16/20, 7:28 PM	00:01:31	Vivado Synthesis D	Vivado Synthesis D	xc7k32	Vivado Synthesis Defaults
0.0	0	0	0	2	0	2	1	0	0	8/16/20, 4:19 PM	00:00:35	Vivado Synthesis D	Vivado Synthesis D	xc7k32	Vivado Synthesis Defaults
4.0	0	0	0	0	0	0	0	0	0	8/16/20, 7:29 PM	00:01:32	Vivado Synthesis D	Vivado Synthesis D	xc7k32	Vivado Synthesis Defaults
4.0	0	0	0	0	0	0	0	0	0	8/16/20, 7:29 PM	00:01:25	Vivado Synthesis D	Vivado Synthesis D	xc7k32	Vivado Synthesis Defaults

Figura 3.13. Recursos utilizados por la FPGA para el bloque FFT

El algoritmo Split-Radix implementado en la tarjeta FPGA tiene un valor muy alto de SQNR, es decir, el error del cálculo realizado es muy bajo. Hay que tomar en cuenta que el espectro aparece como un espejo conjugado en las muestras 1024 a 2047 de las

muestras 0 a 1023 por lo que la presentación del espectro se realiza únicamente considerando los valores de frecuencia discreta de la 0 a 1023.

Se debe considerar que cuando se cambian los parámetros Span y Frecuencia central se deben volver a procesar las muestras, ya que depende de estos el valor del oscilador local. Además, hay que tomar en cuenta que al trasladar a baja frecuencia la señal puede ser muestreada a diferentes frecuencias dependiendo de la resolución que se desea observar.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

El algoritmo Split-Radix asimétrico implementado en la FPGA realiza un menor número de operaciones complejas en comparación de los algoritmos Radix-2 y Radix-4. Esto se debe a la combinación de los dos algoritmos que se implementa en Split-Radix.

La utilización de los recursos de la FPGA depende mayormente de la cantidad de memorias RAM y ROM que se implementan en el código. Este proyecto hace uso de tres memorias RAM para recursividad del algoritmo Split-Radix. También se debe tomar en cuenta que la longitud de cada localidad de memoria RAM es de 64 bits para tener una mejor exactitud de la FFT obtenida, de esta manera se logra tener un SQNR alrededor de 60 dB.

El mezclador de frecuencia que se implementa como parte de la etapa de radiofrecuencia del analizador de espectro cuenta con un Oscilador Local, el cual se configura dependiendo del Span y de la Frecuencia central escogida, trasladando la señal de alta frecuencia a baja frecuencia y de esta manera poder muestrear. Posteriormente, se debe tomar en cuenta el valor de Oscilador Local para la representación gráfica del espectro en la interfaz.

Para procesar la señal en el mezclador de frecuencia es necesario hacer un filtrado de la señal en alta frecuencia, para observar únicamente las frecuencias en el rango del parámetro Span escogido. Así mismo, se debe seleccionar adecuadamente la frecuencia de muestreo para observar correctamente el espectro en la ventana escogida.

El analizador de espectros implementado está basado en el algoritmo de Split-Radix de 2048 puntos, y por esto, es posible alcanzar una resolución mínima de 2,441KHz. La resolución del analizador de espectro se determina en base al número de muestras que admite para el procesamiento dicho algoritmo, por lo que si se aumenta el número de puntos de procesamiento del algoritmo se podrá tener un menor valor de resolución.

4.2 RECOMENDACIONES

El tiempo de recepción y transmisión de datos entre la tarjeta FPGA y el computador se puede disminuir aumentando la velocidad de transmisión. Se recomienda generar una señal de la cual se pueda obtener una señal de 9600 baudios o 115200 baudios.

Si se emplean componentes CORE IP de vivado, se recomienda revisar la documentación de la componente y realizar simulaciones del funcionamiento, que permitan conocer cuántos ciclos de reloj necesita la componente para entregar la respuesta correcta. Como ejemplo se pueden mencionar las simulaciones realizadas de la componente RAM, la cual se llegó a determinar que requiere de tres ciclos de reloj para guardar el dato.

Como mejora a este proyecto de titulación, se podría aumentar el número de muestras que se necesitan para ejecutar el algoritmo Split-Radix de manera que se tenga una menor resolución en el analizador de espectros y un mayor rango de frecuencia de detección.

Para complementar el presente proyecto se sugiere realizar la implementación de las etapas de radio frecuencia, tales como: etapa de recepción de señales (antena), filtrado, mixer y convertidor análogo digital. Se debe tomar en cuenta también la implementación de un oscilador local de frecuencia variable dependiente de los parámetros de Span y Frecuencia central escogido.

5. REFERENCIAS BIBLIOGRÁFICAS:

[1] K. Kanwal, G. A. Safdar, and M. UrRehman, “Energy Efficiency Led reduced CO2 Emission in Green LTE Networks,” *EAI Endorsed Trans. Energy Web*, vol. 4, no. 14, 2017, doi: 10.4108/eai.4-10-2017.153160.

[2] C. Sisterna, “FIELD PROGRAMMABLE GATE ARRAYS (FPGAS).” [Online]. Available: http://dea.unsj.edu.ar/sisdig2/Field Programmable Gate Arrays_A.pdf. [Accessed: 17-Jun-2020].

[3] DIGILENT, “Placa de desarrollo FPGA Xilinx Kintex-7 - Digilent.” [Online]. Available: https://store.digilentinc.com/genesys-2-kintex-7-fpga-development-board/. [Accessed: 16-Sep-2020].

[4] J. Kikkert, “Mixers.” [Online]. Available: http://mwl.diet.uniroma1.it/people/pisa/SISTEMI_RF/MATERIALE INTEGRATIVO/Kikkert_RF_Electronics_Course/07-RF_Electronics_Kikkert_Ch5_Mixers.pdf. [Accessed: 21-Oct-2020].

[5] C. Vega, “Amplificadores, Osciladores y Mezcladores.” [Online]. Available: https://personales.unican.es/perezvr/pdf/CH4ST_Web.pdf. [Accessed: 21-Oct-2020].

[6] Mini-Circuit, “Frequency Mixer.” [Online]. Available: https://www.minicircuits.com/pdfs/ZEM-4300+.pdf. [Accessed: 21-Oct-2020].

[7] B. Gimeno and E. Tutores, “Diseño de un Convertidor Analógico-Digital de Aproximaciones Sucesivas de bajo consumo y área reducida Proyecto Fin de Carrera.” [Online]. Available: https://riunet.upv.es/bitstream/handle/10251/19255/EnriqueBuenoGimeno.pdf. [Accessed: 11-Nov-2020].

[8] J. Bernal, P. Gómez, and J. Bobadilla, “UNA VISIÓN PRÁCTICA EN EL USO DE LA TRANSFORMADA DE FOURIER COMO HERRAMIENTA PARA EL ANÁLISIS ESPECTRAL DE LA VOZ.”

[9] R. M. Olalla, “Tratamiento Digital de la Señal,” 2011.

[10] Universidad Nacional del Sur, “Métodos rápidos para el cálculo de la TDF,” pp. 1–84.

[11] M. Lara, “Algoritmos para el Filtrado Eficiente de Señales Reales basados en la DFT.” [Online]. Available: http://bibing.us.es/proyectos/abreproy/12300/fichero/PFC_Lara+Martin-17_09_2015.pdf. [Accessed: 06-Jul-2020].

[12] P. Rossi Sancho, “Análisis De Las Arquitecturas De La Transformada Rápida De Fourier.”

[13] F. Semiconductor, “Software Optimization of FFTs and IFFTs Using the SC3850 Core,” 2008. [Online]. Available: https://www.nxp.com/docs/en/application-note/AN3666.pdf. [Accessed: 28-Apr-2020].

[14] P. Duhamel, “Split radix FFT algorithm,” *Electron. Lett.*, vol. 41, no. 2, pp. 40–41, 2005, doi: 10.1049/el.

[15] C. Watanabe, “Diseño De La Transformada Rápida De Fourier Con Algoritmo Split-Radix En Fpga,” 2009.

[16] M. Sánchez-Élez, “Introducción a la Programación en VHDL 1 F. Informática (UCM).”

[17] Universidad de la república, “Representación en punto flotante.” [Online]. Available: https://eva.udelar.edu.uy/pluginfile.php/808915/mod_resource/content/3/Clase 18.pdf. [Accessed: 02-Jun-2020].

[18] XILINX, “Vivado Design Suite User Guide Synthesis UG901 (v2018.1),” 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug901-vivado-synthesis.pdf. [Accessed: 12-Jun-2020].

[19] C. Bran, “DISEÑO DE MAQUINAS DE ESTADO FINITO CON VHDL | Sistemas Embebidos.” [Online]. Available: http://systemonfpga.blogspot.com/2015/04/diseno-de-maquinas-de-estado-finito-con.html. [Accessed: 12-Jun-2020].

[20] D. Bishop, “Fixed point algorithmic math package user’s guide.” [Online]. Available: http://www.vhdl.org/fphdl/vhdl2008c.zip. [Accessed: 12-Jun-2020].

[21] R. G. Duque, “Python PARA TODOS.” [Online]. Available: http://mundogeek.net/tutorial-python/. [Accessed: 14-Sep-2020].

[22] C. Husillos and V. Terrón, “Módulos Introducción a PYTHON Abril de 2014,” 2014. [Online]. Available: https://docplayer.es/10483021-Introduccion-a-python-cesar-husillos-victor-terron-abril-de-2014.html. [Accessed: 14-Sep-2020].

[23] E. Bahit, “CURSO: PYTHON PARA PRINCIPIANTES.” [Online]. Available: http://46.101.4.154/Libros/El Lenguaje Python.pdf. [Accessed: 14-Sep-2020].

[24] Spyder, “Sitio web de Spyder.” [Online]. Available: https://www.spyder-ide.org/. [Accessed: 14-Sep-2020].

[25] P. Rossi Sancho, “Estudio de la cuantización.” [Online]. Available: http://bibing.us.es/proyectos/abreproy/11014/fichero/Volumen+1%252F6.-+Estudio+de+la+cuantizacion.pdf. [Accessed: 30-Sep-2020].

[26] XILINX, “UltraFast Design Methodology Guide for the Vivado Design Suite,” 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug949-vivado-design-methodology.pdf#page=200&zoom=100,122,89. [Accessed: 08-Dec-2020].

ANEXOS

ANEXO A. Desarrollo de una compuerta and en vivado.

ANEXO B. Código en Matlab para crear vectores de giro.

ANEXO C. Código Python de la interfaz gráfica del analizador de espectro.

ANEXO D. Código vhdl del algoritmo split-radix de 2048 puntos.

ANEXO E. Archivo para definición de pines de la entidad transformada.

ANEXO F. Scrip de Matlab para el cálculo del SQNR de la FFT obtenida en la FPGA.

ANEXO G. Scrip de Matlab para generar muestras con extensión .txt.

ANEXO A

DESARROLLO DE UNA COMPUERTA AND EN VIVADO

Primero se debe elegir crear un Nuevo Proyecto en la ventana de inicio del Vivado. Donde se abrirá una ventana la cual indica que se debe configurar varios parámetros para el nuevo proyecto, así como: nombre del proyecto, localización, tipo de tarjeta y fuentes creadas previamente. La ventana principal se muestra en la figura A.1. Seguidamente se abre una ventana para asignar un nombre al proyecto y la localización de este tal como se muestra en la figura A.2.

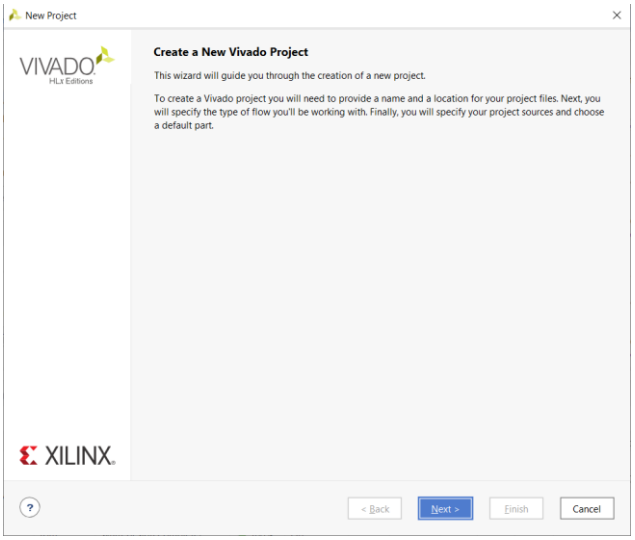


Figura A.1. Ventana de introducción para crear un proyecto

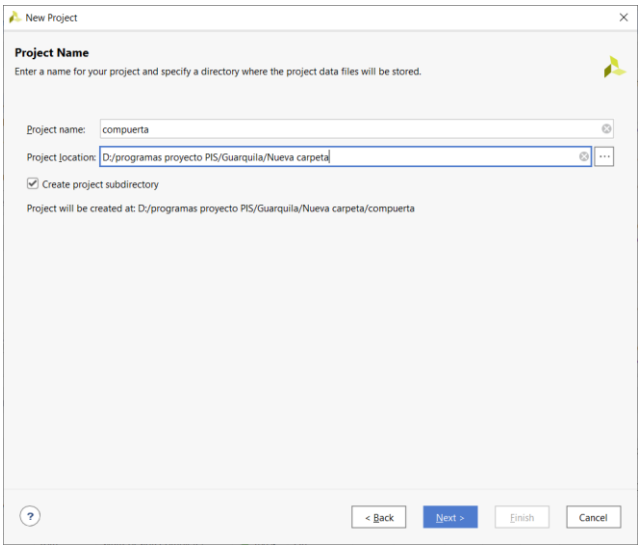


Figura A.2. Ventana para asignar nombre y localización del proyecto

La figura A.3 muestra que tipo de proyecto se desea crear como:

- **RTL Proyect:** en esta opción se puede agregar fuentes creadas previamente, bloques IP y correr la síntesis, implementación, planificación de pines, análisis e implementación.
- **Post synthesis Proyect:** en esta opción se puede agregar fuentes creadas previamente y correr la síntesis, implementación, planificación de pines, análisis e implementación.
- **I/O Planning Proyect:** no se puede crear proyecto únicamente crear el archivo para asignación de pines.
- **Importar Proyecto**
- **Ejemplo de Proyecto**

La opción más recomendada y completa para crear el proyecto es RTL Proyect.

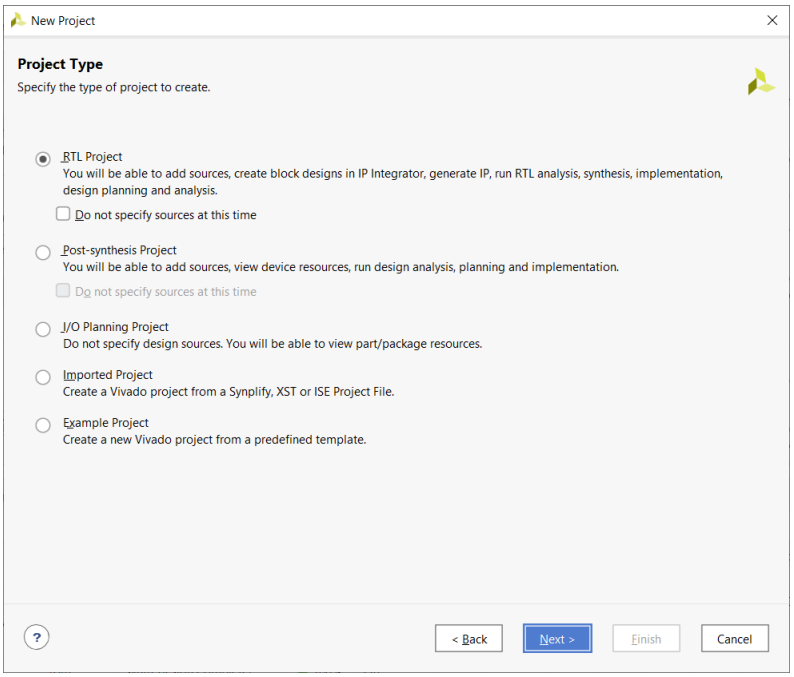


Figura A.3. Ventana para elegir el tipo de proyecto

Seguidamente nos muestra una ventana en la que podemos añadir proyectos creados previamente o crear la carpeta del nuevo proyecto. También es válido crear o añadir archivos posteriormente por lo que dejaremos en blanco esta ventana. La ventana se muestra en la figura A.4. Opcionalmente se muestra la figura A.5 la cual una ventada para añadir archivos de asignación de pines los cuales tienen una extensión .xdc.

A continuación, se muestra la figura A.6 la cual es la ventana para escoger el tipo de tarjeta a programar que en nuestro caso es la tarjeta XC7K325T-2FFG900C. El tipo de

familia de esta tarjeta es Kintex-7, paquete FFG-900 y velocidad -2. Seguidamente se muestra la figura A.7 que es la ventana de resumen del proyecto que se está creando.

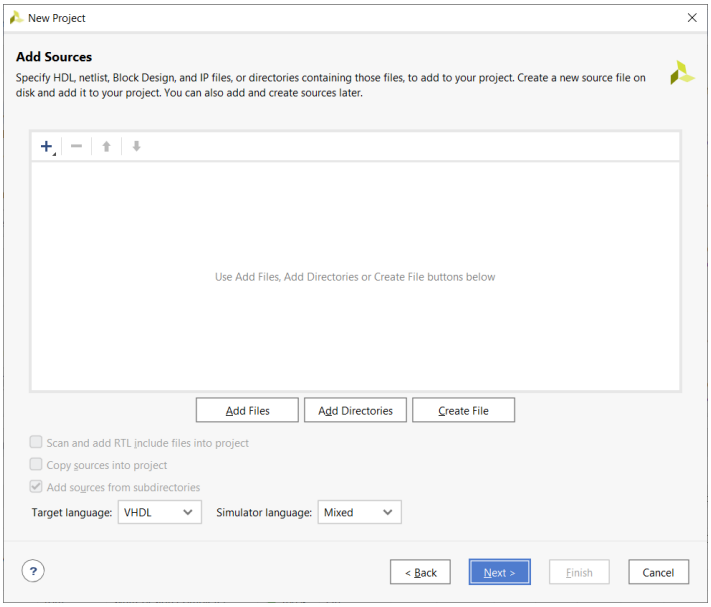


Figura A.4. Ventana para añadir proyectos

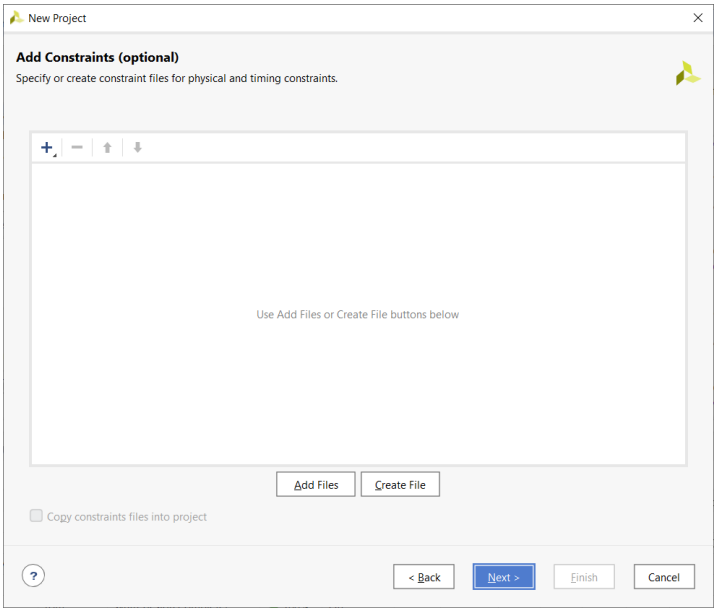


Figura A.5. Ventana para añadir archivos de asignación de pines

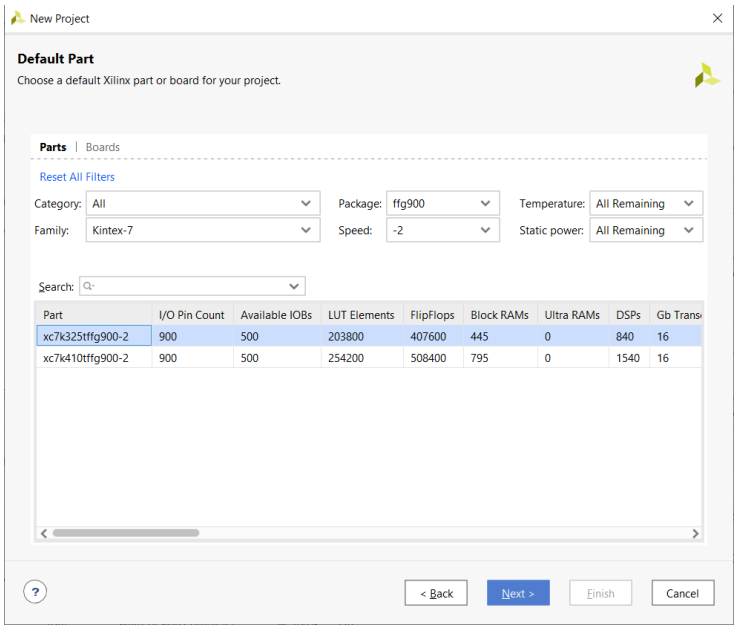


Figura A.6. Ventana para elegir la tarjeta a programar

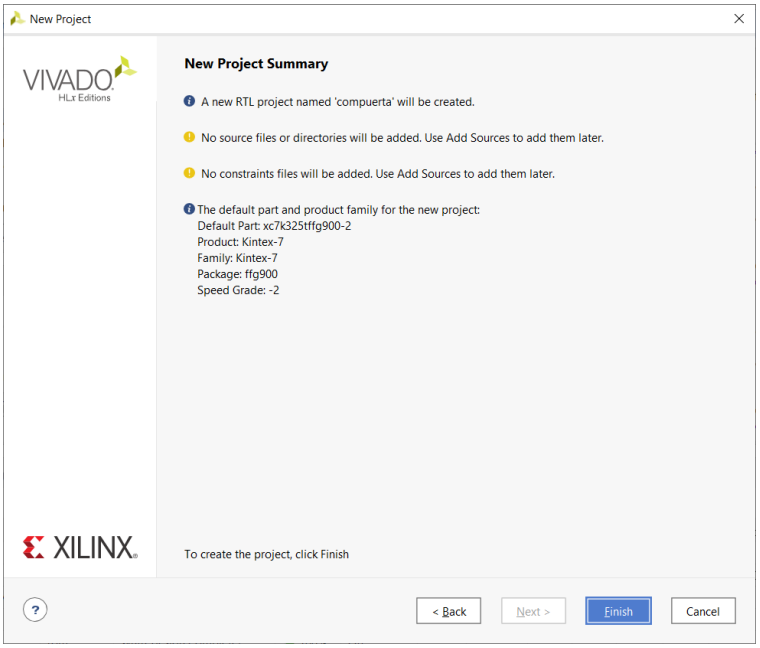


Figura A.7. Ventana de resumen del proyecto

A continuación, en la subventana Proyect Manager escogemos la opción Add Source misma que se encuentra remarcada de color celeste en la figura A.8, para crear la carpeta que contendrá al archivo vhd. Debemos escoger la opción *Add or create desing source* misma que se encuentra remarcada de color celeste en la figura A.9.

Seguidamente escogemos el tipo de lenguaje, el nombre de la carpeta y la localización, tal como se muestra en la figura A.10.

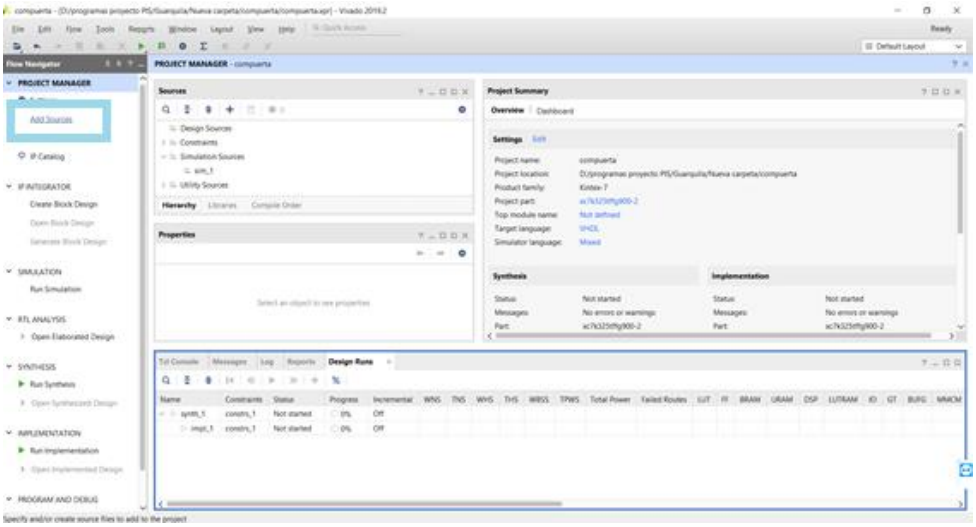


Figura A.8. Ventana de opciones de vivado

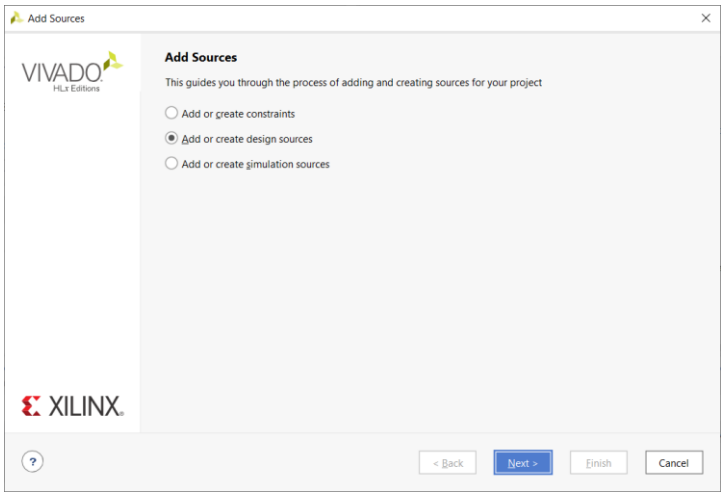


Figura A.9. Ventana para añadir o crear fuentes, simulaciones o definición de pines

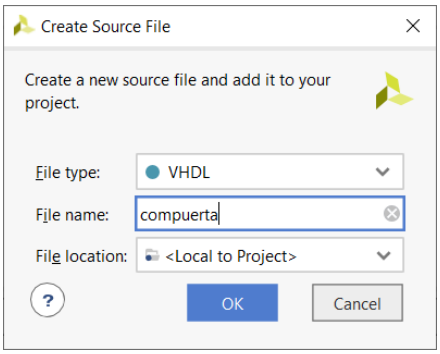


Figura A.10. Ventana para asignar nombre y localización del archivo a crear

En la ventana *Add Source* se debe escoger la opción *Create File* como se muestra en la figura A.11, mostrando así la ventana *Define Module* donde se asigna el nombre de la entidad y opcionalmente se agrega pines de entrada y salida tal como se muestra en la figura A.12.

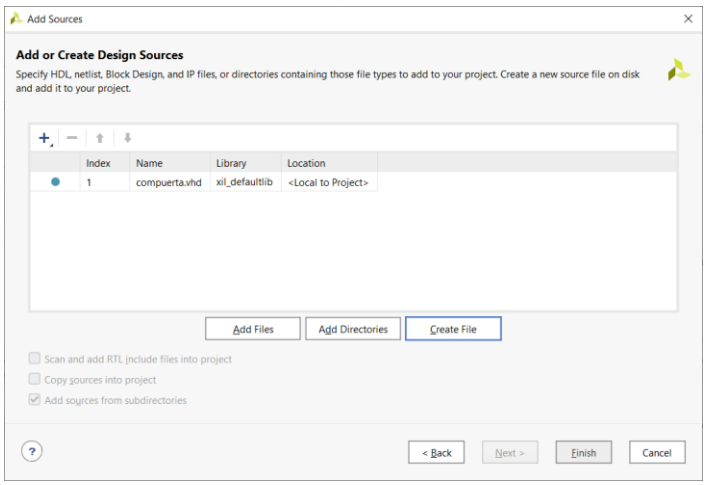


Figura A.11. Ventana para agregar o crear fuentes

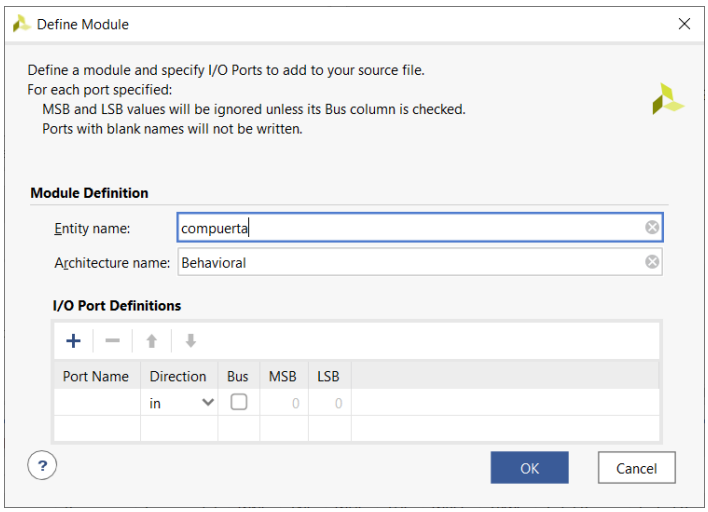


Figura A.12. Ventana para asignar nombre a la entidad

Para crear el código en la subventana Sources aparece el nombre de la entidad creada misma que se muestra remarcado de color naranja en la figura A.13. Para crear la simulación de la compuerta creada se debe hacer una síntesis del código, esta opción se encuentra remarcada de color verde en la subventana Project Manager. Una vez hecha la síntesis debe crear el test bench para la simulación de la compuerta creada, para esto debemos escoger la opción *Add Sources* en el Project Manager.

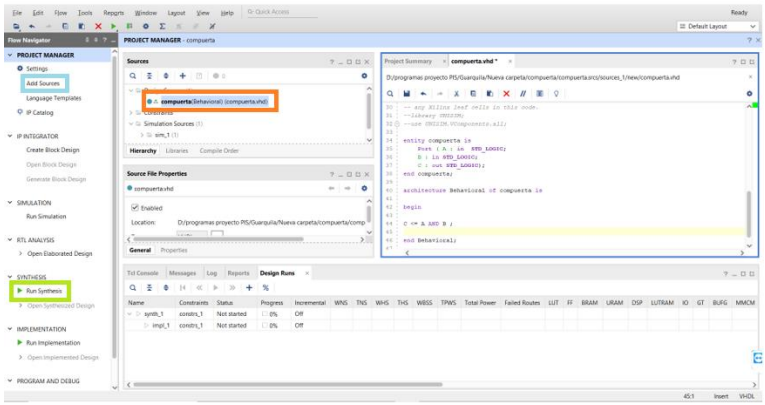


Figura A.13. Ventana de fuentes creadas en el proyecto

En la ventana Add Sources mostrada en la figura A.14 se debe escoger la opción *add or create simulation sources,* seguidamente se debe crear la carpeta del archivo de simulación escogiendo la opción *Create File* como se muestra en la figura A.15.

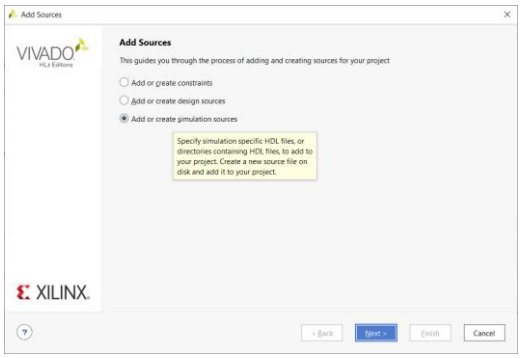


Figura A.14. Ventana para añadir simulaciones

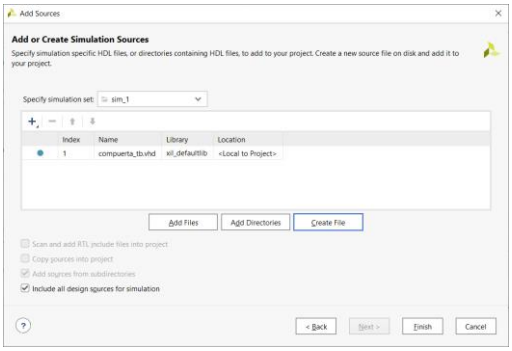


Figura A.15. Ventana para crear la carpeta del archivo de simulación

A continuación, se debe asignar el nombre del archivo de simulación tal como se muestra en la figura A.16. Para verificar el archivo en la subventana *Sources* en la carpeta Simulation Sources desplegar y dentro de este se encuentra el archivo de simulación creado previamente tal como se muestra en la figura A.17. Una vez creada los estímulos de la compuerta en la subventana *Proyect Manager* escoger *Run Simulation,* opción

remarcada de color naranja y seguidamente *Run Post-Synthesis Functional Simulation,* opción remarcada de color celeste tal como se muestra en la figura A.18.

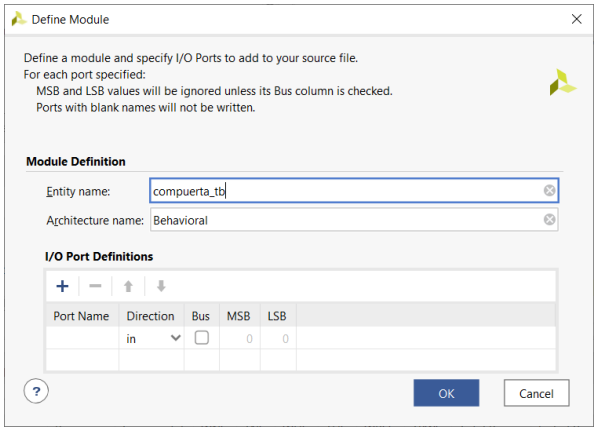


Figura A.16. Ventana para asignar nombre del archivo de simulación

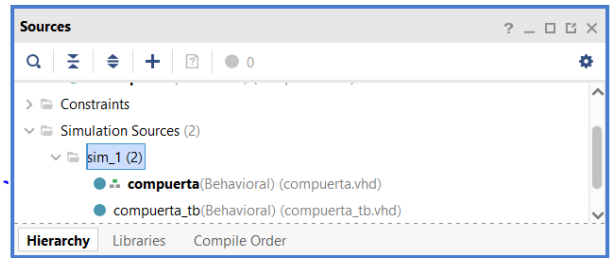


Figura A.17. Subventana de archivos creados

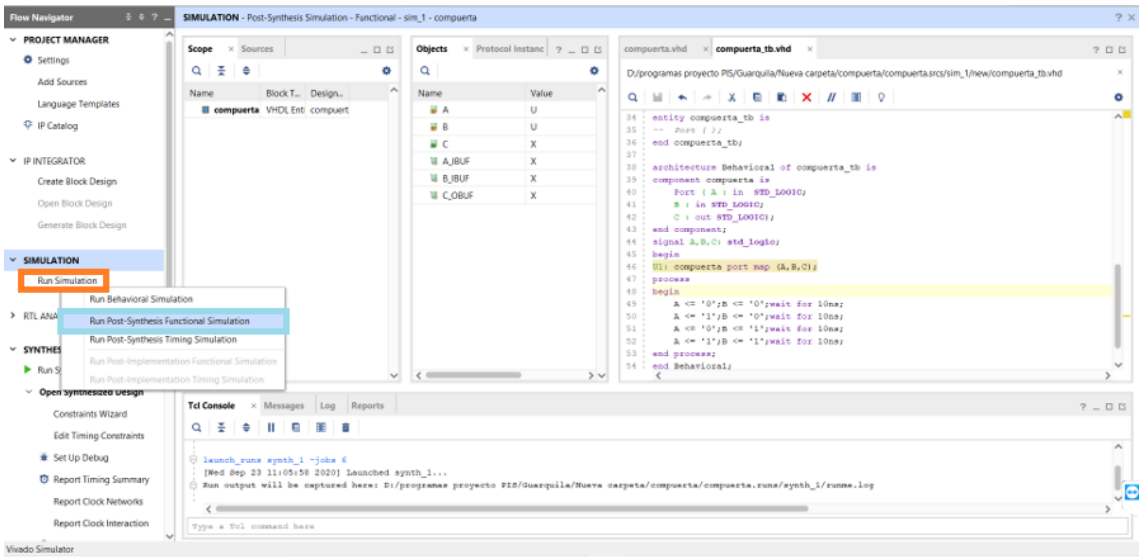


Figura A.18. Ventana para ejecutar la simulación

Una vez ejecutada la simulación se abrirá la ventana donde se muestra todas las señales del archivo compuerta tanto entradas como salidas tal como se muestra en la figura A.19.

Para la implementación del código se debe crear el archivo de definición de pines para esto se en la subventana Proyect Manager escoger la opción Open Elaborated Desing que se encuentra remarcado de color naranja en la figura A.20.

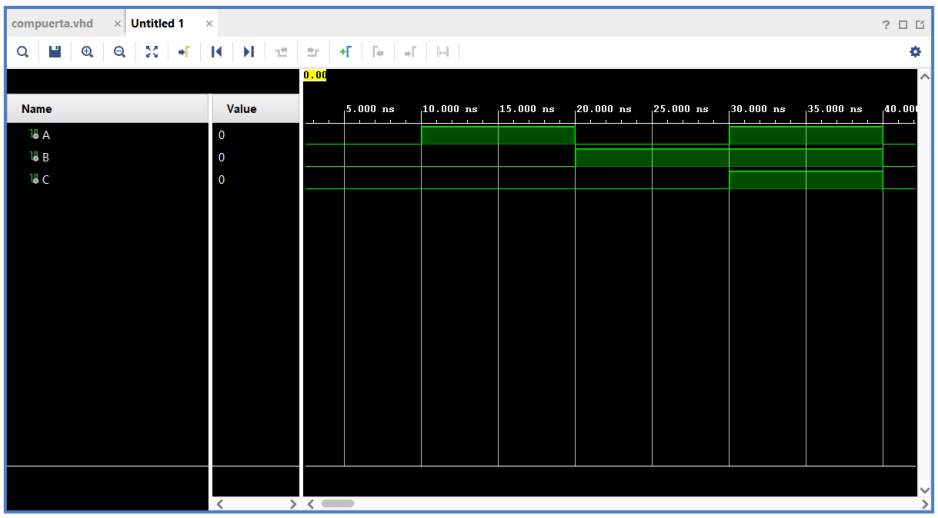


Figura A.19. Ventana de simulación

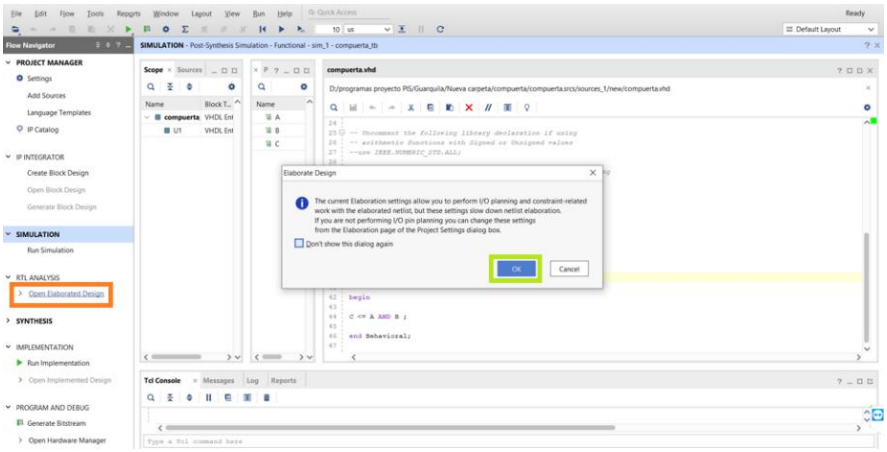


Figura A.20. Ventana para abrir la planificación de entradas y salidas

A continuación, en la carpeta Scalar ports remarcado del color naranja en la figura A.21 se encuentran los pines que se deben configurar tanto entradas como salidas, tal como se muestra en el recuadro de color verde y seguidamente guardar el archivo dando clic en el botón remarcado de color celeste. Posteriormente se abrirá una ventana para asignar un nombre y localización al archivo XDC que contiene la definición de pines tal como se muestra en la figura A.22.

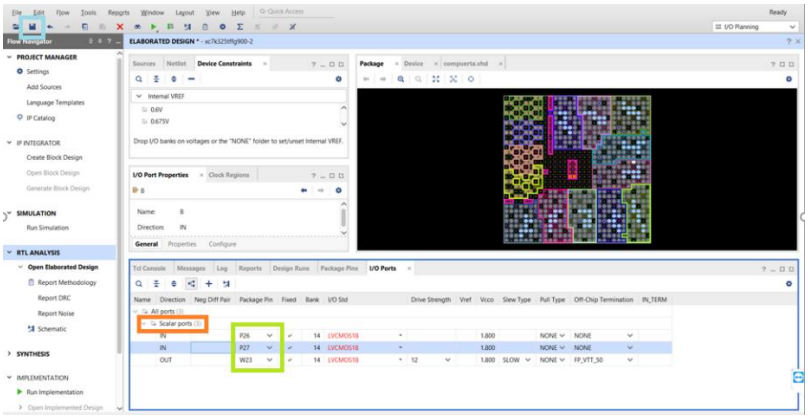


Figura A.21. Ventana para definición de entradas y salidas

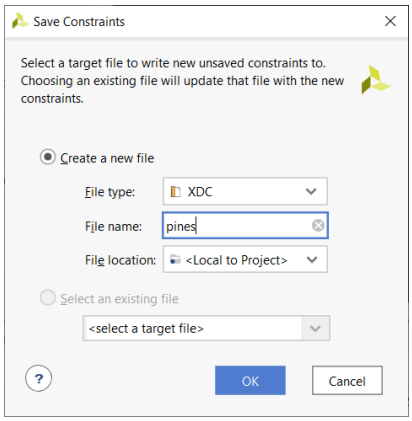


Figura A.22. Ventana para definición del nombre del archivo XDC

Una vez asignado el nombre del archivo se crea la definición tal como se muestra en la figura A.23 remarcado de color naranja. Se puede verificar el archivo en la subventana Sources dentro de la carpeta Constrains, carpeta remarcada de color verde. Seguidamente se debe ejecutar la implementación, para esto en la subventana Proyect Manager escoger la opción Run implementation remarcada de color celeste.

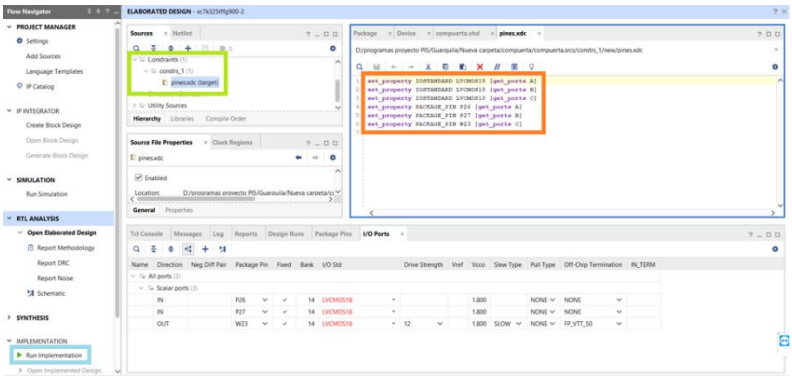


Figura A.23. Archivo XDC

A continuación, se debe generar el archivo .bit que es el que se debe cargar en la tarjeta, para esto una vez realizada la implementación se abrirá la ventana mostrada en la figura A.24 y se debe escoger la opción Generate Bitstream.

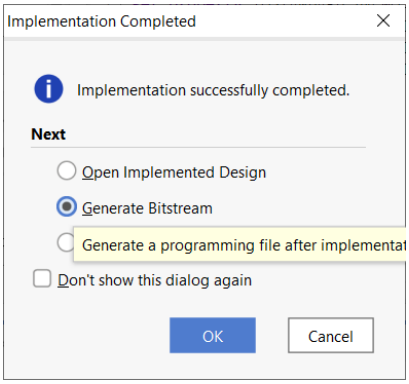


Figura A.24. Ventana para generar el archivo.bit

Posterior a la creación del archivo .bit se debe escoger en la subventana Program and Debug la opción Open Target→Auto Connect tal como se muestra en la figura A.25 remarcado de color verde. De esta manera Vivado busca la tarjeta conectada al puerto USB de la Computadora.

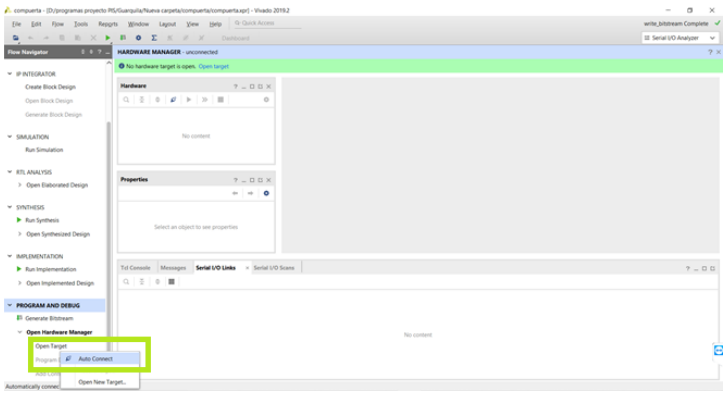


Figura A.25. Buscar FPGA conectada en la Computadora

A continuación, en la subvetana Program and Debug escoger Program Device → xc7k35t_0. Es decir, escoger la tarjeta conectada a la computadora tal como se muestra en la figura A.26 remarcado de color naranja. Después en la ventana Program Device buscar el archivo .bit generado por Vivado y dar un clic en Program para cargar el archivo .bit en la FPGA tal como se muestra en la figura A.27.

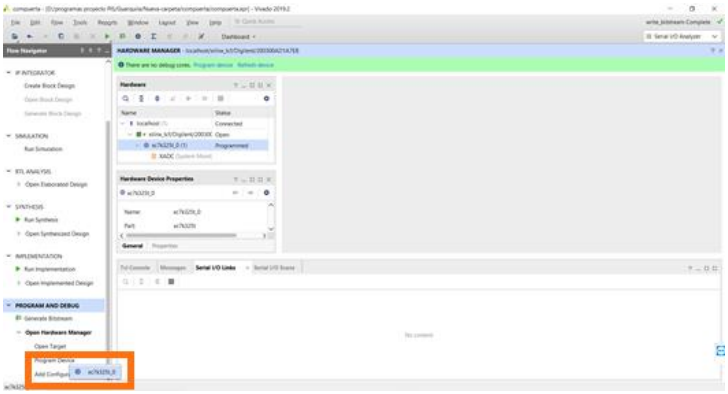


Figura A.26. Buscar FPGA a la cual se desea cargar el archivo.bit

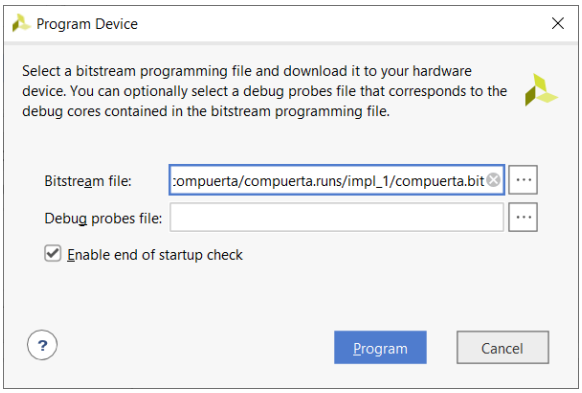


Figura A.27. Buscar archivo .bit y cargar a la FPGA

ANEXO B

CÓDIGO EN MATLAB PARA CREAR VECTORES DE GIRO.

(Se encuentra en digital**)**

ANEXO C

CÓDIGO PYTHON DE LA INTERFAZ GRÁFICA DEL ANALIZADOR DE ESPECTRO

(Se encuentra en digital**)**

ANEXO D

CÓDIGO VHDL DEL ALGORITMO SPLIT-RADIX DE 2048 PUNTOS

(Se encuentra en digital**)**

ANEXO E

ARCHIVO PARA DEFINICIÓN DE PINES DE LA ENTIDAD TRANSFORMADA

(Se encuentra en digital**)**

ANEXO F

SCRIP DE MATLAB PARA EL CÁLCULO DEL SQNR DE LA FFT OBTENIDA EN LA FPGA

(Se encuentra en digital**)**

ANEXO G

SCRIP DE MATLAB PARA GENERAR MUESTRAS CON EXTENSIÓN .txt

(Se encuentra en digital**)**

ORDEN DE EMPASTADO