UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE LA CORRELACIÓN Y DE LA CORRENTROPÍA CRUZADA, UTILIZANDO FPGA

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

FRANCISCO JAVIER RIVERA SERRANO

PROFESOR GUÍA:
PABLO ESTÉVEZ VALENCIA

MIEMBROS DE LA COMISIÓN:
RICARDO FINGER CAMUS
GONZALO RUZ HEREDIA

SANTIAGO DE CHILE
2017

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
POR: FRANCISCO JAVIER RIVERA SERRANO
FECHA: 2017
PROF. GUÍA: PABLO ESTÉVEZ VALENCIA

DISEÑO E IMPLEMENTACIÓN DE LA CORRELACIÓN Y DE LA CORRENTROPÍA CRUZADA, UTILIZANDO FPGA

La Correntropía es una medida no lineal de similitud entre dos variables aleatorias. Esta Tesis plantea una forma de implementación de la correntropía, haciendo uso de dispositivos digitales de alta integración llamados FPGA (*Field Programmable Gate Array*) los cuales permiten procesar la información directamente en hardware, logrando mejoras significativas en los tiempos de proceso.

El objetivo de esta Tesis es el diseño e implementación en hardware de la correlación cruzada y de la correntropía cruzada, utilizando FPGA. De acuerdo a lo investigado a la fecha, existen trabajos previos en la implementación de la correlación pero no así para la correntropía en la forma como aquí se plantea. Para poder comparar lo obtenido con correntropía, se implementó también la correlación cruzada, utilizando los mismos dispositivos FPGA.

En base a lo anterior, se desarrolló un diseño considerando la obtención de la menor latencia posible para el cálculo de la Correntropía, siendo la latencia el retardo producido entre la entrada y la salida para producir un resultado esperado. Se supone que la latencia de un FPGA es menor entre uno y dos órdenes de magnitud, comparado con un procesador, lo cual se demuestra en este trabajo.

En esta Tesis, con el fin de implementar el hardware en base a dispositivos FPGA, se ha desarrollado una metodología de diseño en Sistemas Digitales, basada en Máquinas de Estado Finito que separa claramente el diseño de la implementación y puede ser aplicada para abordar sistemas digitales complejos y de gran envergadura.

Para desarrollar esta Tesis se decidió utilizar la tarjeta de desarrollo Nexys4 de Xilinx la cual utiliza la herramienta de software VIVADO. Dentro de VIVADO, el lenguaje de descripción de hardware (HDL) utilizado fue SystemVerilog.

En relación al desarrollo del proyecto, éste se dividió en dos etapas: la primera contempló el diseño e implementación de la Correlación Cruzada, utilizando un FPGA. Se utilizó la definición de correlación en el dominio de la frecuencia. Esto implicó utilizar módulos que calculan la Transformada de Fourier para cada una de las entradas. La segunda etapa del proyecto contempló el diseño e implementación de la Correntropía Cruzada, propiamente tal, utilizando un FPGA. El enfoque de diseño es diferente al aplicado a la correlación, dado que la definición de correntropía incluye un *Kernel* Gaussiano.

En ambas etapas del proyecto se lograron los resultados esperados: salidas del diseño implementado para FPGA, idénticas a las salidas dadas por la herramienta MATLAB, considerando diferentes tipos de entradas: señales sinusoidales de distinto tipo dado que son más fáciles de implementar y visualizar, series de tiempo de señales electromagnéticas de Astronomía y eventos de husos de sueño en registros de electroencefalogramas (EEG). Se confirma, además, la menor latencia, de al menos un orden de magnitud, de las salidas de la herramienta VIVADO en comparación a lo obtenido con la herramienta MATLAB, obteniéndose menores latencias para la Correlación que para la Correntropía.

A mi esposa Jeanette y a mis hijos Francisco, Catalina y Bárbara. A mi primo Luis Silva Rivera quién, a pesar de los años transcurridos y de su temprana partida, mantuvo vivo en mí el deseo de hacer estudios de Postgrado.

Agradecimientos

Especiales agradecimientos al Profesor Pablo Estévez por su apoyo constante e incondicional desde los inicios de este proyecto. Al grupo de Inteligencia Computacional por la ayuda recibida. A los Profesores del programa de Magíster del DIE por el apoyo para cumplir con esta tarea.

Tabla de Contenido

Página

Índice de Tablas **ix**

Índice de Ilustraciones **x**

Introducción **1**

1. TECNOLOGÍA UTILIZADA **4**
1.1. FPGA: *Field Programmable Gate Array* 4
1.2. Tarjeta de Desarrollo NEXYS4 de DIGILENT 7

2. DISEÑO E IMPLEMENTACIÓN EN HARDWARE DE LA CORRELACIÓN UTILIZANDO FPGA **8**
2.1. Metodología de Diseño 8
2.2. Diseño e Implementación de la Correlación Cruzada 10
2.2.1. Consideraciones en relación al Diseño de la Correlación Cruzada . . . 10
2.2.2. Diagrama en Bloques Simplificado del Diseño del *Correlator* 11
2.2.3. Diagrama de Flujo Simplificado del Controlador del *Correlator* 15
2.2.4. Diagrama en Bloques Detallado del Diseño del *Correlator* 15
2.2.5. Diagrama de Flujo Detallado del Controlador del *Correlator* 18
2.2.6. Diseño e Implementación Final del *Correlator* 18
2.3. Resultados Obtenidos para la Correlación Cruzada 22
2.4. Resultados de la Implementación del *Correlator* en el FPGA 29

3. DISEÑO E IMPLEMENTACIÓN EN HARDWARE DE LA CORRENTROPÍA UTILIZANDO FPGA **30**
3.1. Metodología de Diseño 30
3.2. Diseño e Implementación de la Correntropía Cruzada 30
3.2.1. Consideraciones en relación al Diseño de la Correntropía Cruzada . . 30
3.2.2. Diagrama en Bloques Simplificado del Diseño del *CorrentropyTor* . . 31
3.2.3. Diagrama de Flujo Simplificado del Controlador del *CorrentropyTor* . 33
3.2.4. Diagrama en Bloques Detallado del Diseño del *CorrentropyTor* 35
3.2.5. Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* . . 37
3.2.6. Diseño e Implementación Final del *CorrentropyTor* 38
3.3. Resultados Obtenidos para la Correntropía Cruzada 43
3.3.1. Comentarios Previos 43

3.3.2. Presentación y Análisis de los Resultados del Diseño del *Correntropy-Tor*, utilizando entradas sinusoidales 45
3.3.3. Presentación y Análisis de los Resultados del Diseño del Correntropy-Tor, utilizando entradas no sinusoidales 51
3.4. Resultados de la Implementación del *CorrentropyTor* en el FPGA 62

4. ANÁLISIS COMPARATIVO DE LOS RESULTADOS OBTENIDOS PARA LA CORRELACIÓN Y LA CORRENTROPÍA CRUZADA IMPLEMENTADAS EN FPGA 64

Conclusiones y Trabajo Futuro 65

Glosario 67

Bibliografía 70

Anexos 72
Anexo A.1: Tarjeta de Desarrollo Nexys4™ de DIGILENT 73
Anexo A.2: Detalle Entradas/Salidas Módulo: "Fast Fourier Transform LogiCORE IP Xilinx v9.0" 75
Anexo A.3: Detalle Entradas/Salidas Módulo: "Complex Multiplier LogiCORE IP Xilinx v6.0" 77
Anexo A.4: Diagramas de Flujo Detallados del Controlador del *Correlator* 78
Anexo A.5: Resultados del Diseño del *Correlator* para Entradas Sinusoidales 82
Anexo A.6: Detalle Entradas/Salidas Módulo: "CORDIC LogiCORE IP Xilinx v6.0" 87
Anexo A.7: Diagramas de Flujo Detallados del Controlador del *CorrentropyTor* 88
Anexo A.8: Resultados del Diseño del *CorrentropyTor* para Entradas Sinusoidales 92
Anexo A.9: Implementación de la Función Correntropía en MATLAB . 115
Anexo A.10: Representación de Números Binarios en Formato Punto-Fijo 116
Anexo A.11: Conversión de un Diagrama de Flujo a un Diagrama MDS 117
Anexo A.12: Programa en SystemVerilog del CONTROLLER del *CorrentropyTor* 124

Índice de Tablas

3.1. Definición de Parámetros para mostrar los Resultados de la Correntropía . . 46

4.1. Parámetros Iniciales Análisis Comparativo 64
4.2. Análisis Comparativo de Latencias para la Correlación 65
4.3. Análisis Comparativo de Latencias para la Correntropía 65
A1. Ejemplo de Números Binarios con Formato $1Q7$ o fix_{9_7} 116
A2. Ejemplo de Números Binarios con Formato $2Q6$ o fix_{9_6} 116

Índice de Ilustraciones

1.1. Esquema y Aplicación de un PLA 4
1.2. Diagrama Esquemático de un CPLD 5
1.3. Diagrama Esquemático de un FPGA 6
1.4. CLB: *Configurable Logic Block* 6

2.1. Diagrama en Bloques Diseño Inicial de un Sistema Digital 9
2.2. Comparación Multiplicaciones en la Correlación, tomada de [26] 11
2.3. Primer Diagrama en Bloques Simplificado del *Correlator* 12
2.4. Diagrama en Bloques FFT LogiCORE IP v9.0 [32] 13
2.5. Diagrama en Bloques módulo COMPLEX MULTIPLIER de Xilinx [31] . . . 14
2.6. Diagrama de Flujo Simplificado del CONTROLLER del *Correlator* 16
2.7. Segundo Diagrama en Bloques del *Correlator* 17
2.8. Diagrama MDS del Controlador del *Correlator* (1 de 2) 19
2.9. Diagrama MDS del Controlador del *Correlator* (2 de 2) 20
2.10. Módulos del Diseño del *Correlator* ingresados en VIVADO 21
2.11. Flujo Diseño–Implementación en VIVADO 21
2.12. Ventana *Flow Navigator* de VIVADO 22
2.13. Resultado para el *Correlator*: entradas iguales de 8 bits y 256 muestras ($FREQ1 = 5,2Hz$, $FREQ2 = 0Hz$) 24
2.14. Resultado para el *Correlator*: entradas iguales de 16 bits y 256 muestras ($FREQ1 = 5,2Hz$, $FREQ2 = 0Hz$) 25
2.15. Magnitud del Espectro de Frecuencia de la Correlación (salida FPGA) . . . 25
2.16. Resultado para el *Correlator* para señal de Husos de Sueño 26
2.17. Magnitud del Espectro de Frecuencia de la Correlación de un HS 26
2.18. Resultado para el *Correlator* para señal de una Curva de Luz Sintética . . . 27
2.19. Resultado para el Correlator para señal de una Curva de Luz Sintética con la media descontada 28
2.20. Magnitud del Espectro de Frecuencia de la Correlación de una Curva de Luz Sintética (Figura 2.18) 28
2.21. Nivel de ocupación del Diseño del *Correlator* en el FPGA 29

3.1. Primer Diagrama en Bloques Simplificado del *CorrentropyTor* 31
3.2. Diagrama en Bloques CORDIC LogiCORE IP v6.0 [35] 32
3.3. Diagrama de Flujo Simplificado del CONTROLLER del *CorrentropyTor* . . 34
3.4. Diagrama en Bloques Detallado del *CorrentropyTor* 36
3.5. Diagrama MDS del Diseño del Controlador del *CorrentropyTor* (1 de 3) . . . 39
3.6. Diagrama MDS del Diseño del Controlador del *CorrentropyTor* (2 de 3) . . . 40

3.7. Diagrama MDS del Diseño del Controlador del *CorrentropyTor* (3 de 3) . . . 41
3.8. Módulos del Diseño del *CorrentropyTor* ingresados en VIVADO 42
3.9. Salida 4 del *CorrentropyTor* según Tabla 3.1 47
3.10. Salida 4 del *CorrentropyTor* según Tabla 3.1 con la media descontada 48
3.11. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 4, Figura 3.9) 48
3.12. Salida 5 del *CorrentropyTor* según Tabla 3.1 49
3.13. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 5, Figura 3.12) 50
3.14. Salida 6 del *CorrentropyTor* según Tabla 3.1 50
3.15. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 6, Figura 3.14) 51
3.16. Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,90$ 52
3.17. Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,90$ y restada la media 53
3.18. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.16) 53
3.19. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.17) 54
3.20. Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,36$ 55
3.21. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.20) 55
3.22. Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,18$ 56
3.23. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.22) 56
3.24. Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,80$ 57
3.25. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.24) 58
3.26. Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,80$ y media descontada 58
3.27. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.26) 59
3.28. Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,32$ 60
3.29. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.28) 60
3.30. Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,16$ 61
3.31. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.30) 61
3.32. Nivel de Ocupación del Diseño del *CorrentropyTor* en el FPGA 62
3.33. Utilización de Recursos y Disipación de Energía del FPGA entregados por VIVADO [29] 63

A1. Diagrama de Flujo Detallado del Controlador del *Correlator* (1 de 4) 78
A2. Diagrama de Flujo Detallado del Controlador del *Correlator* (2 de 4) 79
A3. Diagrama de Flujo Detallado del Controlador del *Correlator* (3 de 4) 80
A4. Diagrama de Flujo Detallado del Controlador del *Correlator* (4 de 4) 81
A5. Salida *Correlator*: entradas iguales ($FREQ1 = 5,2Hz, FREQ2 = 0Hz$) de 16 bits y 1024 muestras 82

A6. Salida *Correlator*: entradas iguales ($FREQ1 = 2,6Hz, FREQ2 = 0Hz$) de 16 bits y 1024 muestras 83
A7. Salida *Correlator*: entradas diferentes (entrada A: $FREQ1 = 2,6Hz$ y $FREQ2 = 0Hz$, entrada B: $FREQ1 = 5,2Hz$ y $FREQ2 = 0Hz$) de 16 bits y 1024 muestras 84
A8. Salida *Correlator*: entradas iguales de dos frecuencias diferentes (entradas A y B: $FREQ1 = 2,6Hz$ y $FREQ2 = 23,2Hz$) de 16 bits y 1024 muestras . . 85
A9. Salida *Correlator*: entradas diferentes de dos frecuencias diferentes (entrada A: $FREQ1 = 2,6Hz$ y $FREQ2 = 23,2Hz$, entrada B: $FREQ1 = 5,2Hz$ y $FREQ2 = 46,4Hz$) de 16 bits y 1024 muestras 86
A10. Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (1 de 4) . 88
A11. Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (2 de 4) . 89
A12. Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (3 de 4) . 90
A13. Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (4 de 4) . 91
A14. Salida 1 del *CorrentropyTor* según Tabla 3.1 92
A15. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 1, figura A14) 93
A16. Salida 1 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía 94
A17. Salida 2 del *CorrentropyTor* según Tabla 3.1 94
A18. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 2, figura A17) 95
A19. Salida 3 del *CorrentropyTor* según Tabla 3.1 96
A20. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 3, figura A19) 96
A21. Salida 7 del *CorrentropyTor* según Tabla 3.1 97
A22. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 7, figura A21) 98
A23. Salida 7 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía 99
A24. Salida 8 del *CorrentropyTor* según Tabla 3.1 100
A25. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 8, figura A24) 100
A26. Salida 9 del *CorrentropyTor* según Tabla 3.1 101
A27. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 9, figura A26) 102
A28. Salida 10 del *CorrentropyTor* según Tabla 3.1 103
A29. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 10, figura A28) 103
A30. Salida 10 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía 104
A31. Salida 11 del *CorrentropyTor* según Tabla 3.1 105
A32. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 11, figura A31) 105
A33. Salida 12 del *CorrentropyTor* según Tabla 3.1 106
A34. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 12, figura A33) 107
A35. Salida 13 del *CorrentropyTor* según Tabla 3.1 108

A36. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 13, figura A35) . . . 108
A37. Salida 13 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía . . . 109
A38. Salida 14 del *CorrentropyTor* según Tabla 3.1 . . . 110
A39. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 14, figura A38) . . . 110
A40. Salida 15 del *CorrentropyTor* según Tabla 3.1 . . . 111
A41. Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 15, figura A38) . . . 112
A42. Salida 16 del *CorrentropyTor* según Tabla 3.1 . . . 113
A43. Salida 17 del *CorrentropyTor* según Tabla 3.1 . . . 113
A44. Salida 18 del *CorrentropyTor* según Tabla 3.1 . . . 114
A45. Ejemplo Simbología utilizada en el Diagrama MDS . . . 117
A46. Ejemplo Simbología utilizada en el Diagrama MDS . . . 118
A47. Ejemplo 1 de Conversión Bloques de Proceso a Diagrama MDS . . . 118
A48. Ejemplo 2 de Conversión Bloques de Proceso a Diagrama MDS . . . 119
A49. Caminos de Decisión . . . 119
A50. Múltiples Caminos de Decisión . . . 120
A51. Diagramas de Flujo y MDS con dos Entradas Asíncronas . . . 120
A52. Diagrama de Flujo y MDS con dos Entradas Asíncronas sin *Carreras Críticas* 121
A53. Especificación de una Salida Incondicional . . . 121
A54. Especificación de una Salida Condicional . . . 122
A55. Salida Incondicional con una Dependencia del Tiempo de Duración de una Entrada . . . 122
A56. Salida Condicional con una Dependencia del Tiempo de Duración de una Entrada 123

Introducción

1. Motivación

Uno de los grandes problemas que enfrenta el área de procesamiento de la información es cómo extraer, de la mejor forma posible, la información contenida en los datos. Para ello, existe una gran variedad de técnicas y metodologías. Una característica común a todas estas técnicas y metodologías es que son muy demandantes de recursos computacionales, es decir, tiempo de proceso de un computador con uno o más procesadores. Esta Tesis plantea una forma de implementación sin utilizar procesadores, haciendo uso de dispositivos digitales de alta integración llamados FPGA (*Field Programmable Gate Array*) [16] los cuales permiten procesar la información directamente en hardware, logrando mejoras significativas en los tiempos de proceso.

En esta Tesis se aborda el Diseño e Implementación en Hardware de la Correntropía[1] Cruzada [18], utilizando FPGA. Para poder comparar lo obtenido con Correntropía, se implementó también la Correlación Cruzada, utilizando los mismos dispositivos FPGA.

En relación al diseño de la solución, motiva el desarrollo de una metodología de diseño en Sistemas Digitales, basada en Máquinas de Estado Finito (FSM: *Finite State Machine*) [28] que separe, claramente, el diseño de la implementación y pueda ser aplicada al diseño de sistemas digitales complejos y de gran envergadura.

Los resultados de esta Tesis podrían ser aplicados en distintos ámbitos, uno de ellos es la radio-astronomía y particularmente lo relacionado a una alternativa a la medida de la Correlación de señales electromagnéticas. Actualmente, el dispositivo que realiza esta función en un observatorio radio-astronómico (por ejemplo, ALMA) es el Correlacionador (*Correlator* en inglés). El objetivo de medir la correlación es poder determinar, en general, el grado de similitud entre dos señales o entre una señal y la misma, retardada en una cierta cantidad de tiempo. Otro ámbito importante donde se pueden aplicar los resultados de esta Tesis, corresponde a la detección y caracterización de eventos de husos de sueño (*sleep-spindle* en inglés) en registros de electroencefalogramas (EEG).

2. Problema a Abordar

Desarrollo en hardware del cálculo de la Correntropía en base a un FPGA. Se trata de calcular la Correntropía Cruzada entre dos entradas aleatorias y donde el tiempo de proceso es un objetivo importante.

La función Correntropía Cruzada para procesos aleatorios discretos, se define como [18]:

[1] La palabra "Correntropía" no existe en la lengua castellana y se ha definido para expresar un nuevo concepto que se forma al combinar los conceptos de Correlación y Entropía.

$$\widehat{V_{xy}}[l] = \frac{1}{(N-l+1)} \sum_{n=l}^{N} G_\sigma(x_n - x_{n-l}) \tag{1}$$

dónde: σ es el ancho del *Kernel* y G es el *Kernel* Mercer Gaussiano, definido como:

$$G_\sigma(x-y) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-\frac{\|x-y\|^2}{2\sigma^2}} \tag{2}$$

Podríamos decir que la forma convencional de implementar esta función sería utilizando una expansión en una serie de Taylor.

Así como al Correlacionador se le llama normalmente *Correlator*, al dispositivo que calcula la Correntropía lo hemos llamado *CorrentropyTor*.

3. Hipótesis

La implementación en hardware de la Correntropía (ecuaciones 1 y 2), utilizando FPGA, permite obtener un menor tiempo de proceso de, a lo menos, un orden de magnitud en comparación al cálculo realizado en un procesador. Esto se debe a que el FPGA procesa directamente en sus componentes básicas ("Flip-flops" y compuertas) y no necesita ejecutar instrucciones como en el caso de un procesador. Para lograr dicha mejora se debe desarrollar un diseño en sistemas digitales basado en máquinas de estado finito [28].

4. Objetivos

(a) Objetivo General

El objetivo general es el diseño e implementación en hardware de la correlación cruzada y de la correntropía cruzada, utilizando FPGA.

(b) Objetivos Específicos

Los objetivos específicos definidos son los siguientes:

i. Desarrollar un diseño que permita obtener la menor latencia posible para el cálculo de la Correntropía. Definiremos latencia como el retardo producido entre la entrada y la salida para producir un resultado esperado. Por ejemplo, en el caso de un procesador, la latencia tiene que ver con el retardo producido por la ejecución de las instrucciones básicas de éste. En el caso de un FPGA, el retardo tiene que ver con el de sus componentes básicas que son compuertas y "Flip-flops". Se supone que la latencia obtenida con un FPGA es menor, entre uno y dos órdenes de magnitud, comparado con un procesador.

ii. Desarrollar un diseño lo más parametrizado posible que permita realizar, fácilmente, cambios y definiciones que ayuden a obtener el objetivo anterior. También se debe considerar minimizar dicha cantidad de parámetros.

iii. Validar el diseño obtenido considerando entradas sinusoidales de distinto tipo y también con otro tipo de entradas como, por ejemplo, señales simuladas de astronomía (curvas de luz sintéticas) y eventos de husos de sueño en registros de electroencefalogramas (EEG).

iv. Comparar resultados obtenidos para la Correlación con los de la Correntropía en FPGA y en MATLAB.

v. Investigar y desarrollar la implementación en el FPGA del *Kernel* Gaussiano, utilizado en la Correntropía.

5. Revisión Bibliográfica

Como resultado de la revisión bibliográfica se determinaron tres tipos de referencias que fueron importantes para el desarrollo de esta Tesis. La primera de ellas tiene que ver con las referencias relacionadas con los conceptos de Correlación, Entropía y Correntropía. El segundo tipo de referencia se relaciona con el diseño e implementación de sistemas digitales de mayor envergadura. Aquí fue importante contar con referencias que ayudaron en el diseño e implementación en base a un FPGA. Por último, tenemos las referencias vinculadas a los dispositivos de alta integración, FPGA. Aquí contamos con referencias relacionadas con experiencias de diseño e implementación de diferentes conceptos como: filtros, correlación, entropía, etc., y referencias del proveedor del FPGA que utilizamos en el desarrollo de esta Tesis.

(a) **Bibliografía vinculada a Correlación, Entropía y Correntropía**: para el concepto de entropía la referencia básica es [23]. Para el concepto de Correlación se utilizaron las referencias [15] y [26]. Para el concepto básico de Correntropía y su relación con el concepto de Correlación, tenemos las referencias [14], [18], [20], [21] y [38] que son documentos claves para comprender los conceptos básicos que fueron necesarios para el desarrollo de la Tesis.

(b) **Bibliografía vinculada al Diseño de Sistemas Digitales Complejos**: aunque existe mucha bibliografía relacionada con diseño de sistemas digitales, no es fácil encontrar referencias que aborden el problema de enfrentar diseños de mayor complejidad en sistemas digitales. Una referencia clave, en ese aspecto, es [11]. En este caso, además, se ha utilizado la experiencia personal adquirida con los años en la enseñanza en el diseño de sistemas digitales de mayor envergadura. Para los conceptos básicos de diseño de sistemas digitales se tiene [19] y [28]. Para el diseño, incorporando el lenguaje HDL Verilog, se tienen las referencias [1] y [4]. Para SystemVerilog (lenguaje HDL utilizado finalmente) se tienen las referencias [10] y [25].

(c) **Bibliografía vinculada al diseño e implementación con FPGA**: afortunadamente existe bastante bibliografía relacionada a la implementación de diferentes tipos de diseños utilizando dispositivos FPGA. Por ejemplo, hay experiencias en la implementación de filtros FIR e IIR [4], transformada de Fourier [7], correlación [26] y "Look-Up Tables" [8] y [12]. Para la implementación de la Correntropía, utilizando un FPGA, existen muy pocas referencias, por ejemplo, [5] y [6], con un enfoque muy particular aplicado al procesamiento de señales. Por último, se tienen las referencias relacionadas con la herramienta que vamos a utilizar en el desarrollo de este proyecto. En este caso utilizaremos la línea de FPGA de Xilinx con su herramienta de diseño de última generación llamada Vivado [29] [33]. Para la Tarjeta de Desarrollo Nexys4 de Digilent, tenemos el Anexo A.1 y la referencia [9].

Capítulo 1

TECNOLOGÍA UTILIZADA

A continuación se entregan algunos antecedentes básicos de los FPGA y de la Tarjeta de Desarrollo Nexys4 utilizada en esta Tesis.

1.1. FPGA: *Field Programmable Gate Array*

Los FPGA son dispositivos electrónicos de alta integración que se utilizan actualmente para realizar diseños digitales de alta complejidad y donde los requerimientos de alta velocidad son prioritarios. Estos circuitos integrados permiten la implementación de diseños complejos en un solo circuito integrado o "chip" aprovechando las bondades del diseño a bajo nivel como el paralelismo al nivel de compuertas y "Flip-flops".

De acuerdo a lo anterior, un FPGA presenta las siguientes características y/o bondades:

1. Es un dispositivo basado en Lógica Programable como son los PLA: *Programmable Logic Array* [16]. Un esquema básico de un PLA se muestra en la figura 1.1.

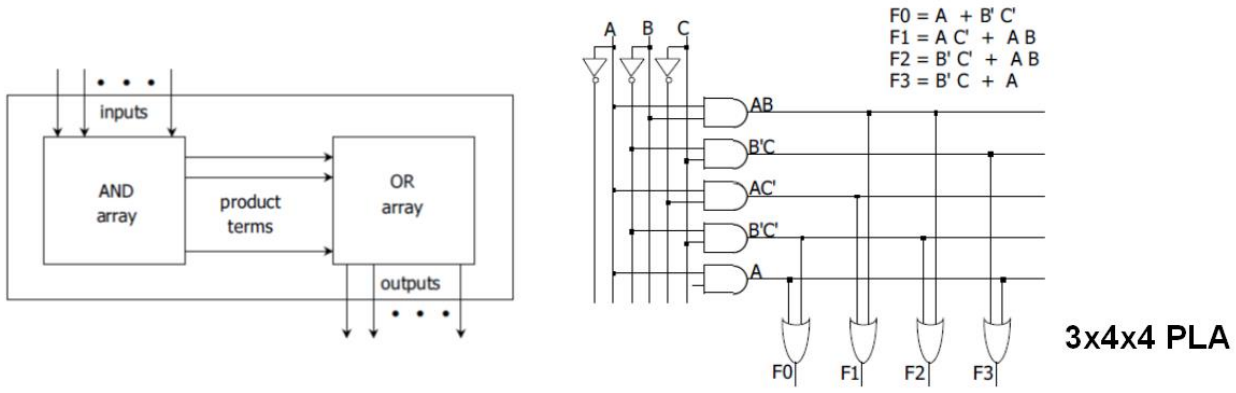

Figura 1.1: Esquema y Aplicación de un PLA

   Como se puede apreciar en la figura 1.1, los PLA permiten la programación de expresiones booleanas de cierta cantidad de entradas y con cierta cantidad de términos productos.

2. Permiten un procesamiento paralelo en comparación a un procesador en el cual se deben ejecutar instrucciones secuencialmente. Aunque esto permite obtener ventajas en velocidad de procesamiento, presenta la desventaja que su implementación es bastante

más compleja.

3. Se pueden implementar Sistemas Combinacionales y Sistemas Secuenciales y estos últimos pueden ser bastante complejos con muchas componentes básicas involucradas.
4. Aunque la forma de ingresar el diseño en un FPGA puede ser a través de esquemáticos, la forma normal y recomendada es utilizar Lenguajes de Descripción de Hardware (HDL) como VHDL y Verilog (o SystemVerilog [25]).
5. Es diferente al ASIC (*Application Specific Integrated Circuits*). Se puede decir que el ASIC podría ser la forma final de implementación de un diseño complejo que se justificaría si se requiere un volumen importante del sistema diseñado. Dada su capacidad de ser un dispositivo programable, el FPGA sería el elemento a utilizar en la etapa de diseño del sistema cuya implementación final sería el ASIC. Este último no es programable por el usuario.

Además del PLA que se muestra en la figura 1.1, se tiene la familia de los CPLD (*Complex Programmable Logic Device*) que son dispositivos que cuentan con una combinación de arreglos AND/OR completamente programables y un banco de Macro-celdas. Las Macro-celdas son bloques funcionales que realizan lógica Combinacional y Secuencial. Un diagrama esquemático de un CPLD se muestra en la figura 1.2:

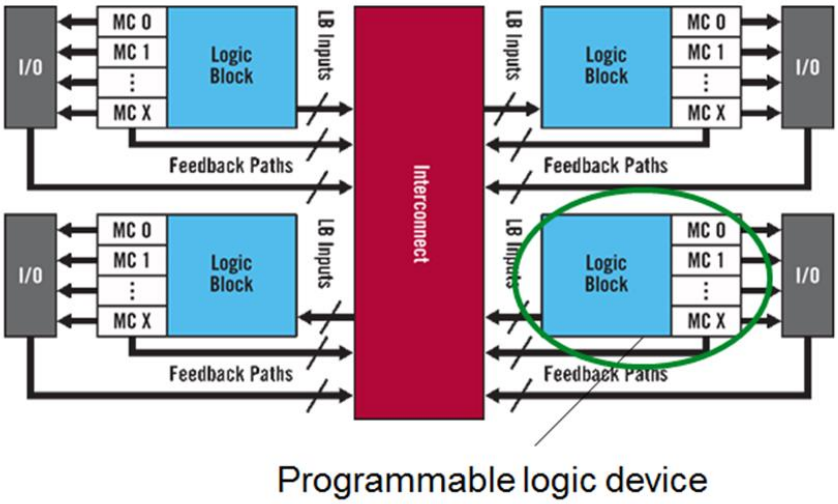

Figura 1.2: Diagrama Esquemático de un CPLD

Y, finalmente, tenemos los FPGA, propiamente tal, que presentan una estructura diferente a los CPLD como se muestra en la figura 1.3.

Los primeros dispositivos PLA aparecen en los comienzos de la década de los ochenta. Un par de años después, aparecen los dispositivos CPLD. El primer FPGA comercial, el $XC2064$, que tenía 64 CLB (*Configurable Logic Block*) y dos LUT (*Look-Up Table* de tres entradas), fue liberado por la empresa Xilinx en 1985.

De acuerdo al diagrama esquemático de la figura 1.3, los FPGA presentan una matriz de bloques lógicos configurables y conectados entre sí, llamados CLB. Estos son dispositivos re-configurables.

Inicialmente los FPGA estaban compuestos solamente de CLB e IOB (*Input/Output Blocks*)

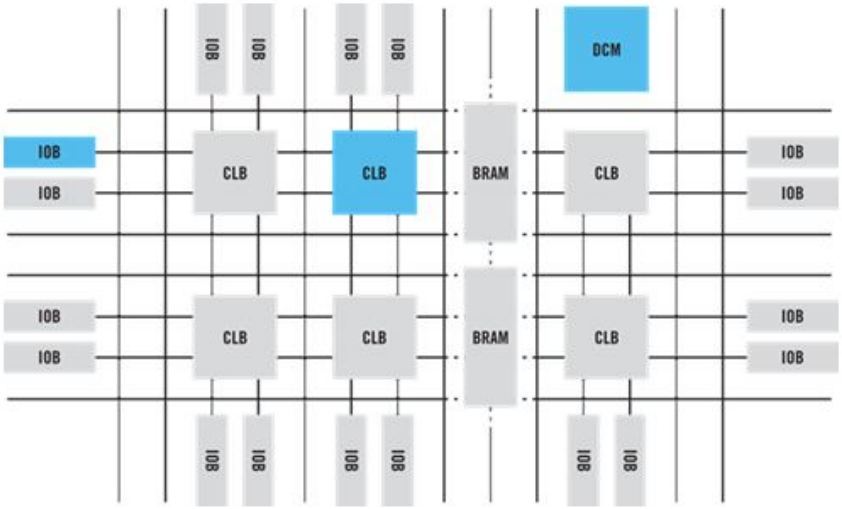

Figura 1.3: Diagrama Esquemático de un FPGA

pero a medida que fueron evolucionando, aparecieron también bloques con funciones específicas como los DCM (*Digital Clock Manager*) y los BRAM (*Block RAM*), todo ello gracias a la capacidad de poner cada vez más componentes básicas en una misma área, lo cual fue predicho por la Ley de Moore. Esto facilita enormemente el diseño de sistemas complejos.

La Figura 1.4 muestra el detalle de un CLB para un FPGA típico. El elemento básico lo constituye el LUT.

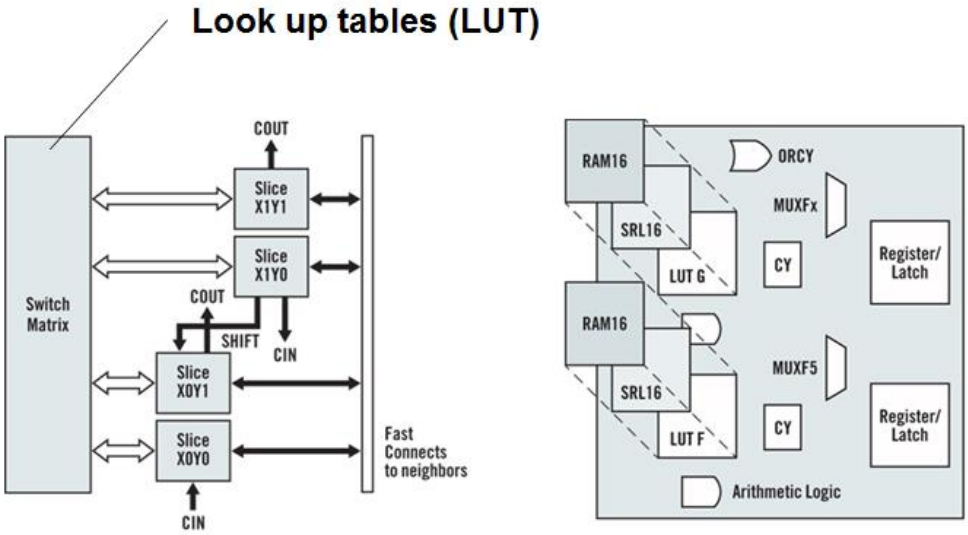

Figura 1.4: CLB: *Configurable Logic Block*

1.2. Tarjeta de Desarrollo NEXYS4 de DIGILENT

La tarjeta de desarrollo Nexys4 [9] es una plataforma completa de desarrollo, diseñada para utilizar tecnología de circuitos digitales, basada en un FPGA $Artix - 7^{TM}$ [30], de última generación de Xilinx. Contiene un FPGA de gran capacidad (Xilinx, número de parte $XC7A100T - 1CSG324C$), con recursos importantes de memorias externas y una colección de puertas USB, Ethernet y otras.

La tarjeta Nexys4 puede implementar diseños que van desde circuitos combinacionales hasta potentes procesadores embebidos. Tiene incorporada varios periféricos, incluyendo un acelerómetro, un sensor de temperatura, un micrófono digital MEM, un amplificador de altavoz y varios dispositivos de E/S que permiten a la tarjeta Nexys4 ser utilizada para una amplia gama de diseños sin necesidad de ningún otro componente.

En el Anexo A.1, se entrega información más detallada de esta tarjeta de desarrollo.

Capítulo 2

DISEÑO E IMPLEMENTACIÓN EN HARDWARE DE LA CORRELACIÓN UTILIZANDO FPGA

2.1. Metodología de Diseño

Uno de los problemas que se ha presentado con el diseño de sistemas digitales es que se ha volcado más a la implementación, dada la gran variedad de hardware disponible para implementar los diseños. Los diseñadores tienden a seleccionar *a priori* el hardware que van a utilizar para la implementación del sistema, saltándose el procedimiento de diseño y sin poder validar si el hardware seleccionado es lo más apropiado para el problema planteado. La única forma de asegurarse que el hardware seleccionado es el apropiado es desarrollando un completo procedimiento de diseño, independiente de la futura implementación en hardware.

En esta Tesis se plantea una metodología de diseño que está basada en [11], un texto publicado en 1980. Aunque en esos años no se contaba con los Lenguajes de Descripción de Hardware (HDL), la metodología planteada a partir de [11] se complementa muy bien para ayudar a obtener los programas en HDL (Verilog, SystemVerilog o VHDL) que requieren las herramientas modernas, como VIVADO [33] de Xilinx, para el diseño, síntesis e implementación de sistemas digitales complejos, utilizando FPGA.

De acuerdo a la metodología aquí planteada, todo diseño de un Sistema Digital de cierta complejidad (por ejemplo, que su control central necesite más de 10 estados) debe partir con un Diagrama en Bloques como se muestra en la figura 2.1.

Desde la partida, se debe tener claro que cualquier sistema digital consta de un Sistema Controlado y un Sistema Controlador, como se indica en la figura 2.1. En seguida, se debe comenzar a dividir el Sistema Controlado en bloques más pequeños y simples que van a permitir simplificar el diseño (aquí se aplica el dicho: *"dividir para conquistar"*). A continuación se describen los diferentes diagramas que se utilizan en esta metodología de diseño.

1. **Diagrama en Bloques Simplificado**: este es el primer diagrama en bloques (después del diagrama de la figura 2.1) que se obtiene de acuerdo a la descripción y especificaciones del diseño. Se definen los primeros bloques que habría que incluir en el diseño. Por ejemplo, en nuestro diseño del *Correlator*, el Diagrama en Bloques Simplificado se muestra en la figura 2.3. Para el caso del diseño del *CorrentropyTor* su Diagrama en Bloques Simplificado se muestra en la figura 3.1. El término "Simplificado" indica que el diagrama muestra solamente la conexión entre los bloques y el Sistema Controlador sin detallar todavía las señales específicas involucradas.
2. **Diagrama de Flujo Simplificado del Sistema Controlador**: el Sistema Contro-

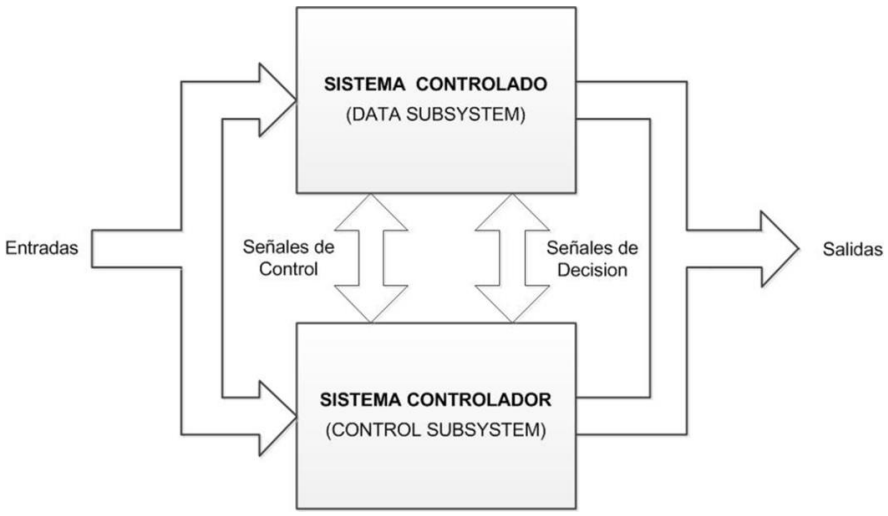

Figura 2.1: Diagrama en Bloques Diseño Inicial de un Sistema Digital

lador (o simplemente Controlador), como su nombre lo indica, es el módulo que va a controlar todo el sistema. Para ello, necesita ejecutar un algoritmo que indica el funcionamiento global del sistema. Para la implementación de dicho algoritmo, se utiliza una Máquina de Estado Finito (FSM). La mejor herramienta para describir el algoritmo de un FSM es un simple Diagrama de Flujo. El término "Simplificado" indica que el diagrama muestra, en palabras, la descripción del algoritmo, sin referirse todavía a las señales específicas involucradas. En nuestro diseño del *Correlator*, el Diagrama de Flujos Simplificado se muestra en la figura 2.6. Para el caso del diseño del *CorrentropyTor* su Diagrama de Flujos Simplificado se muestra en la figura 3.3.

3. **Diagrama en Bloques Detallado**: se obtiene en base al Diagrama en Bloques Simplificado donde se han incorporado las señales específicas entre los módulos y el Controlador. Esto hace que en el módulo del Controlador aparezcan una cantidad significativa de señales de entrada y salida. En nuestro diseño del *Correlator*, el Diagrama en Bloques Detallado se muestra en la figura 2.7. Para el caso del diseño del *CorrentropyTor* su Diagrama en Bloques Detallado se muestra en la figura 3.4.
4. **Diagrama de Flujo Detallado del Sistema Controlador**: se obtiene en base al Diagrama de Flujo Simplificado pero ahora se incorporan las variables de decisión (esquematizadas en los rombos del diagrama de flujo) de acuerdo a señales específicas. Cada uno de los bloques rectangulares del Diagrama de Flujo Detallado, constituyen los estados de la máquina de estado finito. En nuestro diseño del *Correlator*, el Diagrama de Flujo Detallado se muestra en las figuras A1 a A4 del Anexo A.4. Para el caso del diseño del *CorrentropyTor* su Diagrama de Flujo Detallado se muestra en las figuras A10 a A13 del Anexo A.7.
5. **Diagrama MDS (Mnemonic Documented State diagram)**: una vez que el Diagrama de Flujo Detallado está completo con todos los estados identificados, se construye el Diagrama MDS [11], similar a un Diagrama de Estado utilizado en el diseño de sistemas digitales basados en "Flip-Flops" y compuertas. Es una traducción directa desde el Diagrama de Flujo Detallado. El Anexo A.11 muestra el procedimiento para convertir un Diagrama de Flujo Detallado en un Diagrama MDS. Este diagrama corresponde al

paso anterior a la fase inicial de la implementación de hardware. A partir de este diagrama es posible implementar con "Flip-Flops" y compuertas (Integración de Pequeña Escala: SSI por sus siglas en inglés), con Contadores, Multiplexores y Decodificadores (Integración de Mediana Escala: MSI por sus siglas en inglés) y con Microcontroladores y FPGA (Integración de Gran Escala, Muy Gran Escala y Ultra Gran Escala: LSI, VLSI y UVLSI por sus siglas en inglés, respectivamente).

En nuestro diseño del *Correlator*, el Diagrama MDS se muestra en las figuras 2.8 y 2.9. Para el caso del diseño del *CorrentropyTor*, su Diagrama MDS se muestra en las figuras 3.5 a 3.7.

Una vez obtenido el Diagrama MDS, resulta bastante simple el obtener el programa en HDL para el Sistema Controlador dado que se pueden utilizar plantillas escritas en Verilog, SystemVerilog o VHDL, disponibles para el usuario. Se pueden utilizar otros tipos de diagramas, como por ejemplo, Diagramas de Tiempo Simplificado y Detallado para indicar especificaciones y restricciones de tiempo para ciertas señales de entrada y salida. En nuestro caso no fue necesario utilizar ese tipo de diagramas.

2.2. Diseño e Implementación de la Correlación Cruzada

2.2.1. Consideraciones en relación al Diseño de la Correlación Cruzada

La Correlación Cruzada en el dominio del tiempo, está dada por [26]:

$$r(j) = \frac{1}{N} \sum_{n=0}^{N-1} x_a(n) \cdot x_b(n+j), \quad j = 0, 1, \ldots, N-1 \tag{2.1}$$

dónde: a, b son las señales a ser correlacionadas. N es el número de muestras (por ejemplo: 1024).

Esta es una forma fácil y directa de implementar la Correlación Cruzada pero requiere una gran cantidad de recursos computacionales. De la ecuación 2.1 se puede ver que se requieren N^2 multiplicaciones.

Ahora bien, la Correlación Cruzada en el dominio de la frecuencia, está dada por [26]:

$$r = \frac{1}{N} F^{-1}[F(x_a) \cdot F(x_b)^*] \tag{2.2}$$

dónde: a, b son las señales a ser correlacionadas, N es el número de muestras (por ejemplo: 1024), F indica la Transformada Discreta de Fourier, F^{-1} indica la Transformada Inversa Discreta de Fourier y $*$ indica el Complejo Conjugado.

Esta es una forma más compleja de implementar la Correlación Cruzada pero requiere una cantidad significativamente menor de recursos computacionales (se puede demostrar [15] que se requieren $\frac{N}{2} log_2(N)$ multiplicaciones).

Una comparación en la cantidad de multiplicaciones entre la versión en el dominio del tiempo y en el dominio de la frecuencia [26], se muestra en la figura 2.2.

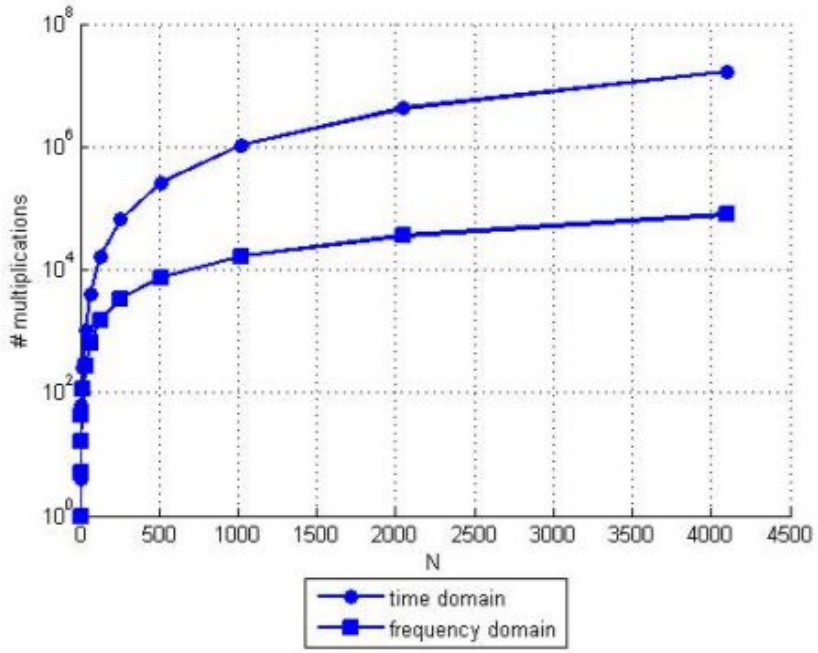

Figura 2.2: Comparación Multiplicaciones en la Correlación, tomada de [26]

De la figura 2.2 se puede constatar que para 2048 muestras de datos, el cálculo de la Correlación Cruzada efectuada en el dominio de la frecuencia, tiene una reducción en la cantidad de multiplicaciones en un factor de 100, aproximadamente. Considerando lo anterior, se ha decidido implementar la Correlación Cruzada utilizando la solución en el dominio de la frecuencia. Cabe destacar, además, que esta última solución permite una mejor *performance* para el *Correlator*.

2.2.2. Diagrama en Bloques Simplificado del Diseño del *Correlator*

De acuerdo a la metodología descrita en el punto 2.1 se obtuvo un primer Diagrama en Bloques Simplificado que permite visualizar, a grandes rasgos, lo que se quiere obtener finalmente. La figura 2.3 muestra este primer Diagrama en Bloques Simplificado para nuestro *Correlator* a ser implementado en la Tarjeta de Desarrollo Nexys4.

A continuación, se describen cada uno de los módulos de la figura 2.3:

1. **CIRCULAR BUFFER**: este es un módulo de entrada; se trata de un *buffer* tipo FIFO (*First-Input First-Output*) que permite adecuar los datos de entrada A y B, en tiempo real, con los datos ingresados al siguiente módulo. Esto permite tener un flujo continuo de datos dependiendo del tiempo que tome a los demás módulos procesar los datos. En todo caso, se pretende asegurar que no haya pérdida en la información de entrada. En un principio se pensó utilizar un módulo IP (*Intellectual Property*, también llamado *Virtual Component*) disponible en la herramienta VIVADO pero al final se desarrolló un módulo *ad-hoc* en SystemVerilog [10] [25].
2. **FFT MODULE**: Este es un módulo IP, disponible para los usuarios en la herramienta VIVADO, con dos canales para poder manejar cada una de las entradas, que a su vez,

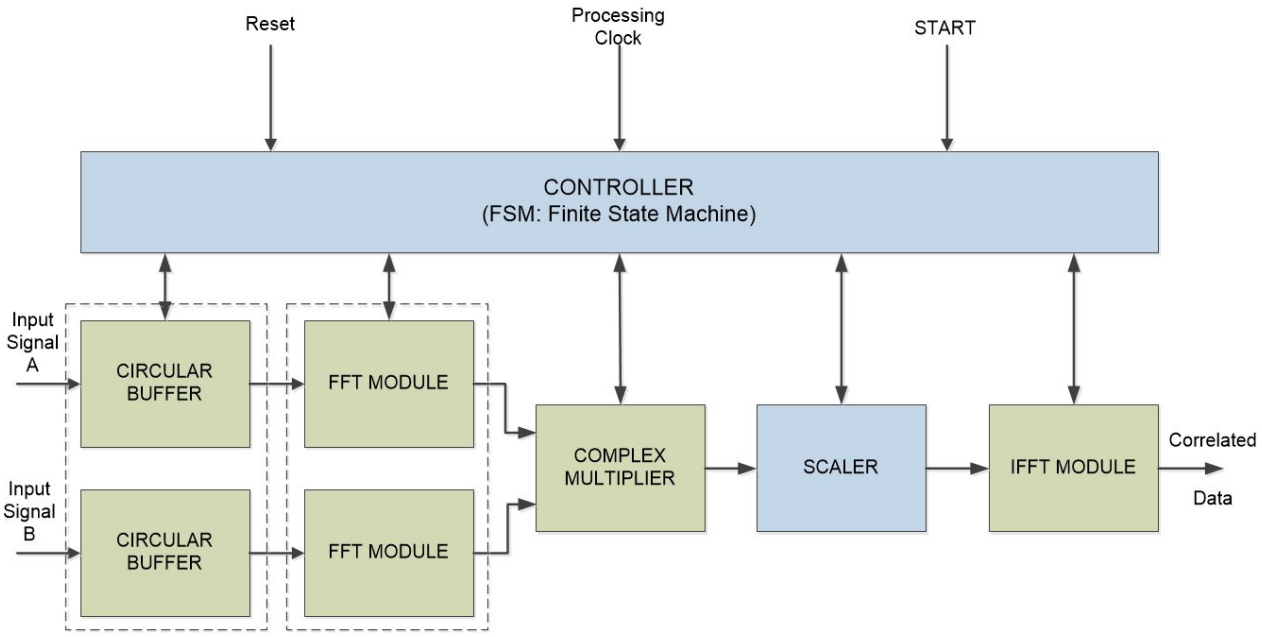

Figura 2.3: Primer Diagrama en Bloques Simplificado del *Correlator*

constan de una parte real y otra imaginaria.

En este caso, nos interesa utilizar la Transformada Discreta de Fourier, desarrollada con el algoritmo FFT (*Fast Fourier Transform*). El IP utilizado en este caso es el *LogiCORE* IP de Xilinx en su versión 9.0 [32]; última actualización realizada en 2015.

Lo interesante de este IP es que el módulo FFT fue desarrollado utilizando el algoritmo de Cooley–Tukey [7] que, de acuerdo a la literatura, es el método computacional más eficiente que existe para implementar la Transformada Discreta de Fourier (DFT).

Un Diagrama en Bloques del IP que vamos a utilizar, se muestra en la figura 2.4 donde se detallan sus entradas y salidas. El detalle descriptivo se encuentra en la Tabla 2.1 del documento "Fast Fourier Transform v9.0, LogiCORE IP Product Guide" [32]. Esta Tabla se reproduce en el Anexo A.2.

El FFT es un algoritmo computacionalmente eficiente para calcular la Transformada Discreta de Fourier (DFT: *Discrete Fourier Transform*) de muestras cuyo tamaño es un número entero, potencia de 2. El DFT $X(k), k = 0, \ldots, N-1$ de una secuencia $x(n), n = 0, \ldots, N-1$ es definido como [32]:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, \mathrm{e}^{-\frac{jnk2\pi}{N}}, \quad k = 0, \ldots, N-1 \tag{2.3}$$

dónde: N es el tamaño de la transformada y $j = \sqrt{-1}$. La DFT inversa (IDFT) está dada por [32]:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, \mathrm{e}^{\frac{jnk2\pi}{N}}, \quad n = 0, \ldots, N-1 \tag{2.4}$$

En cuanto a los algoritmos utilizados, el módulo FFT utiliza las descomposiciones *Radix-4* y *Radix-2* para el cálculo del DFT [32]. Para las arquitecturas *Burst I/O*

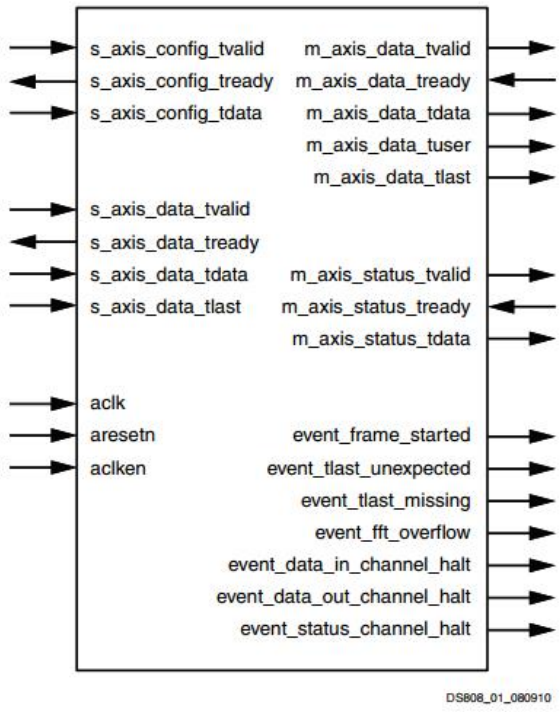

Figura 2.4: Diagrama en Bloques FFT LogiCORE IP v9.0 [32]

se utiliza el método *decimation-in-time* (DIT) [7], mientras que para la arquitectura *Pipelined Streaming I/O* se utiliza el método *decimation-in-frequency* (DIF) [7].

Cuando se utiliza la descomposición *Radix-4*, el FFT de N puntos consiste de $log_4(N)$ etapas, con cada etapa formada por $N/4$ *butterflies Radix-4*. Los tamaños de puntos que no son una potencia de 4, necesitan una etapa extra del tipo *Radix-2* para combinar los datos. Un FFT de N puntos que utiliza una descomposición *Radix-2*, tiene $log_2(N)$ etapas con cada etapa formada por $N/2$ *butterflies Radix-2* [32].

La FFT inversa (IFFT) es calculada conjugando los factores de fase de la correspondiente FFT normal o no inversa.

3. **COMPLEX MULTIPLIER**: este es otro módulo IP disponible en VIVADO para los usuarios. Este módulo permite realizar la multiplicación compleja entre las salidas del módulo FFT. De acuerdo al diagrama en bloques (figura 2.3), este último módulo entrega las transformadas de cada una de las entradas (A y B). De acuerdo a la ecuación 2.2, el módulo COMPLEX MULTIPLIER debe realizar la multiplicación de la primera salida del módulo FFT con el complejo conjugado de la segunda salida del módulo FFT.

   Un Diagrama en Bloques del IP que vamos a utilizar, se muestra en la figura 2.5 donde se detallan sus entradas y salidas. El detalle descriptivo se encuentra en la Tabla 2.1 del documento "Complex Multiplier v6.0, LogiCORE IP Product Guide" [31]. Esta Tabla se reproduce en el Anexo A.3.

   Al igual que el módulo FFT, este módulo tiene diferentes formas de ser configurado. En nuestro diseño nos interesa enfatizar la *performance* por sobre el uso de recursos.

En este caso, por ejemplo, se ha seleccionado el modo *Non-Blocking* en lugar del modo *Blocking* dado que el primero permite utilizar el módulo a máxima velocidad de cálculo.

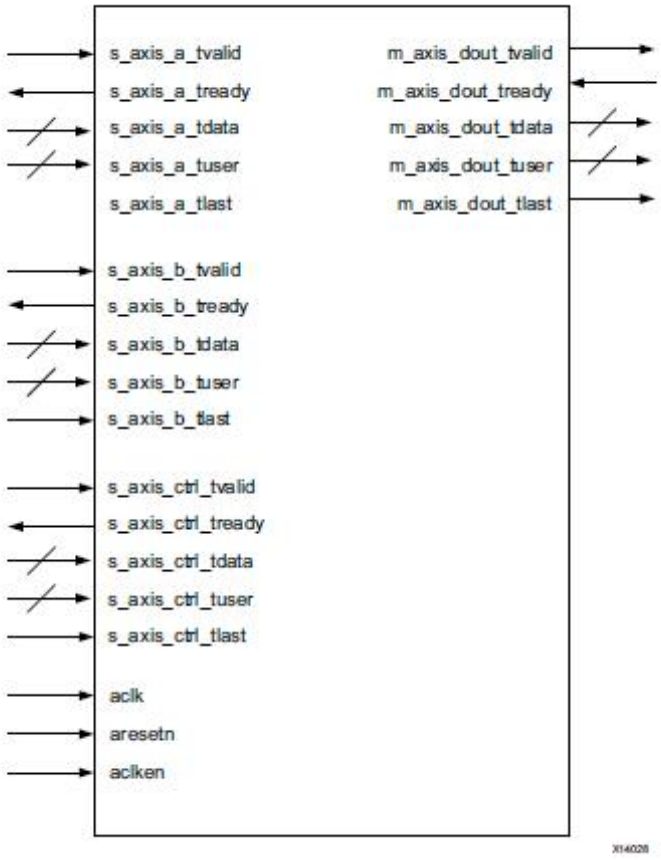

Figura 2.5: Diagrama en Bloques módulo COMPLEX MULTIPLIER de Xilinx [31]

4. **SCALER**: este módulo permite realizar una adaptación entre formatos de los datos de salida del COMPLEX MULTIPLIER y entrada al módulo IFFT MODULE.
5. **IFFT MODULE**: este módulo corresponde al mismo módulo FFT MODULE anterior pero configurado para realizar la Transformada Inversa de Fourier.
6. **CONTROLLER**: este módulo es el controlador del sistema y se ha implementado como una Máquina de Estado Finito. Este módulo permite que los demás módulos se ejecuten en secuencia, en un esquema "pipeline", de acuerdo a como se indica en la figura 2.3. Por lo tanto, este módulo ejecuta un algoritmo que se puede representar en un Diagrama de Flujo.

   La siguiente etapa en el proceso de diseño de nuestro *Correlator*, es obtener un Diagrama en Bloques Detallado donde queden especificados completamente los módulos que, en definitiva, van a formar el diseño final del *Correlator*. Una vez logrado dicho diagrama, se puede obtener el Diagrama de Flujo Detallado del Controlador donde se consideran las señales definitivas de cada uno de los módulos que forman el diseño. Obviamente, se va a llegar a un Diagrama de Flujo mucho más extenso. Esto se describe en los puntos 2.2.4 y 2.2.5 siguientes.

2.2.3. Diagrama de Flujo Simplificado del Controlador del *Correlator*

La figura 2.6 muestra un Diagrama de Flujo Simplificado del algoritmo a ser ejecutado por el CONTROLLER.

La señal *START* es una entrada al sistema global que cuando se activa ("1" lógico) se inicia el proceso del cálculo de la correlación.

De acuerdo a lo indicado en la figura 2.6, el Controlador comienza ingresando datos al FIFO_BUFFER. Una vez que estos datos están disponibles, se pasan al módulo FFT que calcula la transformada de Fourier. Luego, la salida de estos datos, son multiplicados en el módulo COMPLEX_MULT que es un multiplicador de números complejos. El resultado de esta multiplicación es ingresada al módulo IFFT que realiza la transformada inversa de Fourier lo cual corresponde a la etapa final del proceso de correlación.

2.2.4. Diagrama en Bloques Detallado del Diseño del *Correlator*

El diseño planteado en el Diagrama en Bloques del punto 2.2.2, aunque es fácil de entender en relación a los módulos básicos a ser considerados, no puede ser implementado en el FPGA por una limitación básica de este dispositivo. Tal como se ha planteado, tenemos que la entrada de nuestro diseño está principalmente dada por los datos, A y B en el Diagrama en Bloques. Si suponemos que cada una de estas entradas son de 16 bits, para la parte real, 16 bits para la parte imaginaria y 1024 muestras, tendríamos: $(16 + 16) * 2 * 1024 = 65{,}536$ pines que tendríamos que tener disponible en el chip FPGA. Esto no es posible si el objetivo es leer los datos de entrada en paralelo (un esquema serial sería demasiado lento) por lo cual tenemos que modificar nuestro esquema.

Para obviar el problema anterior, se decidió generar las entradas dentro de nuestro diseño por lo cual ya no se necesita contar con 65.536 pines de entrada en el FPGA. Dado que el problema también se presenta en la salida, se decidió mostrar las salidas en los "display" de 7 segmentos con que cuenta la tarjeta de desarrollo Nexsys4.

Considerando lo anterior, se obtuvo el Diagrama en Bloques Detallado de la figura 2.7. Se trata de un diagrama detallado porque incorpora las diferentes señales de los distintos módulos. Se puede ver ahora que las entradas al diseño serían solamente *aclk*, *reset* y *start* y la salida sería *led [15:0]*. En este diagrama, además, hay que destacar los siguientes módulos adicionales a los de la figura 2.3:

1. **INPUT_A**: corresponde al módulo que genera las entradas A para el *Correlator*. Como primeras entradas, se trabaja con expresiones trigonométricas [3] que son de más fácil generación y también para obtener diferentes alternativas de entradas. Las expresiones, escritas en SystemVerilog, son las siguientes:

$$\begin{aligned} theta &= [real'(\mathrm{i})/real'(MAX\_SAMPLES)] * FREQ1 * 2{,}0 * PI \\ theta2 &= [real'(\mathrm{i})/real'(MAX\_SAMPLES)] * FREQ1 * 2{,}0 * PI \\ re\_real &= cosine(-theta) + cosine(-theta2) - 1{,}0 \\ im\_real &= sine(-theta) + sine(-theta2) \end{aligned} \tag{2.5}$$

   Los ángulos están dados por las expresiones de *theta* y *theta2* que, a su vez, dependen de

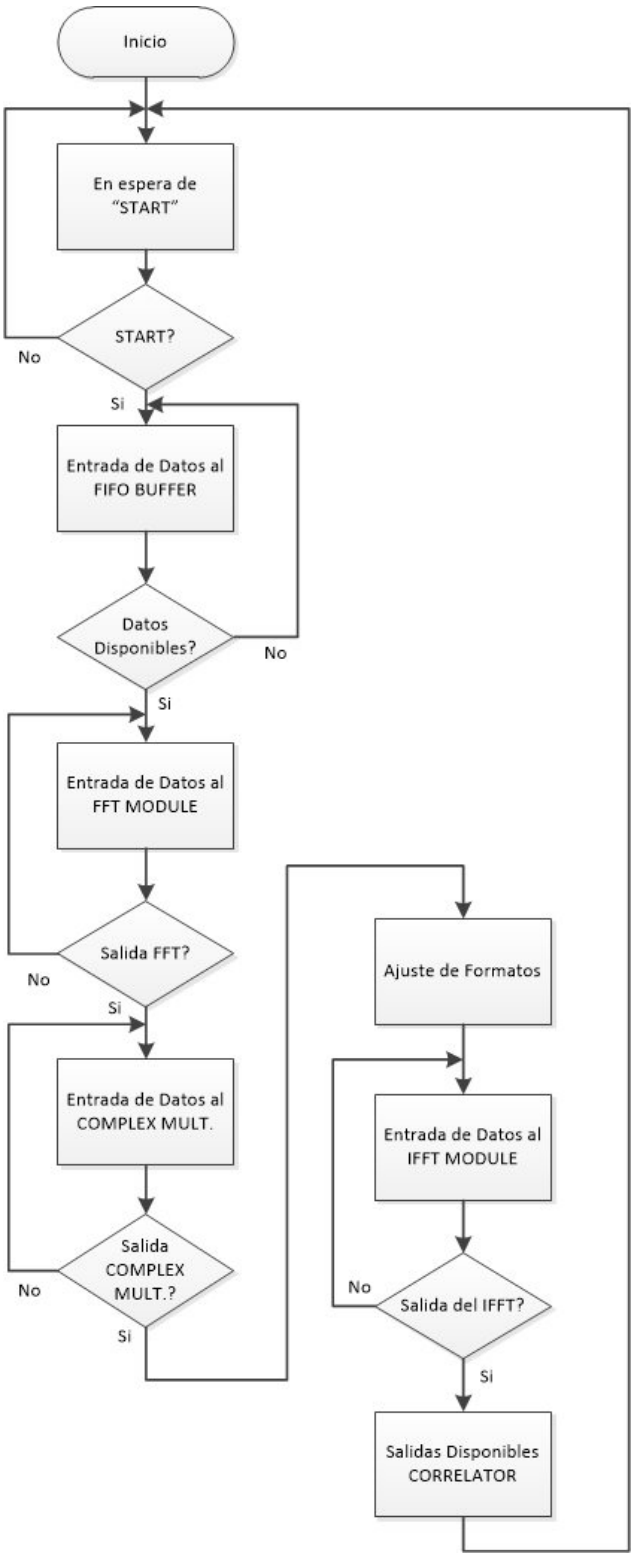

Figura 2.6: Diagrama de Flujo Simplificado del CONTROLLER del *Correlator*

las frecuencias $FREQ1$ y $FREQ2$, medidas en Hz, que son parámetros que se definen más adelante para lograr una buena visualización de las sinusoides. $MAX\_SAMPLES$ corresponde a la cantidad de muestras que también se maneja como parámetro. La

variable i se mueve entre 0 y $MAX_SAMPLES - 1$. Las expresiones re_real e im_real corresponden a la parte real y parte imaginaria de la entrada generada. PI corresponde a la constante π.

2. **INPUT_B**: corresponde a la entrada B y es análoga a la entrada A descrita anteriormente.
3. **COUNTER**: es un módulo contador que es necesario agregar para manejar los LED de los "display" de 7 segmentos que se van a utilizar en la salida.
4. **LED_DECODER**: es el módulo que permite desplegar la salida en los "display" de 7 segmentos con que cuenta la tarjeta de desarrollo Nexys4.

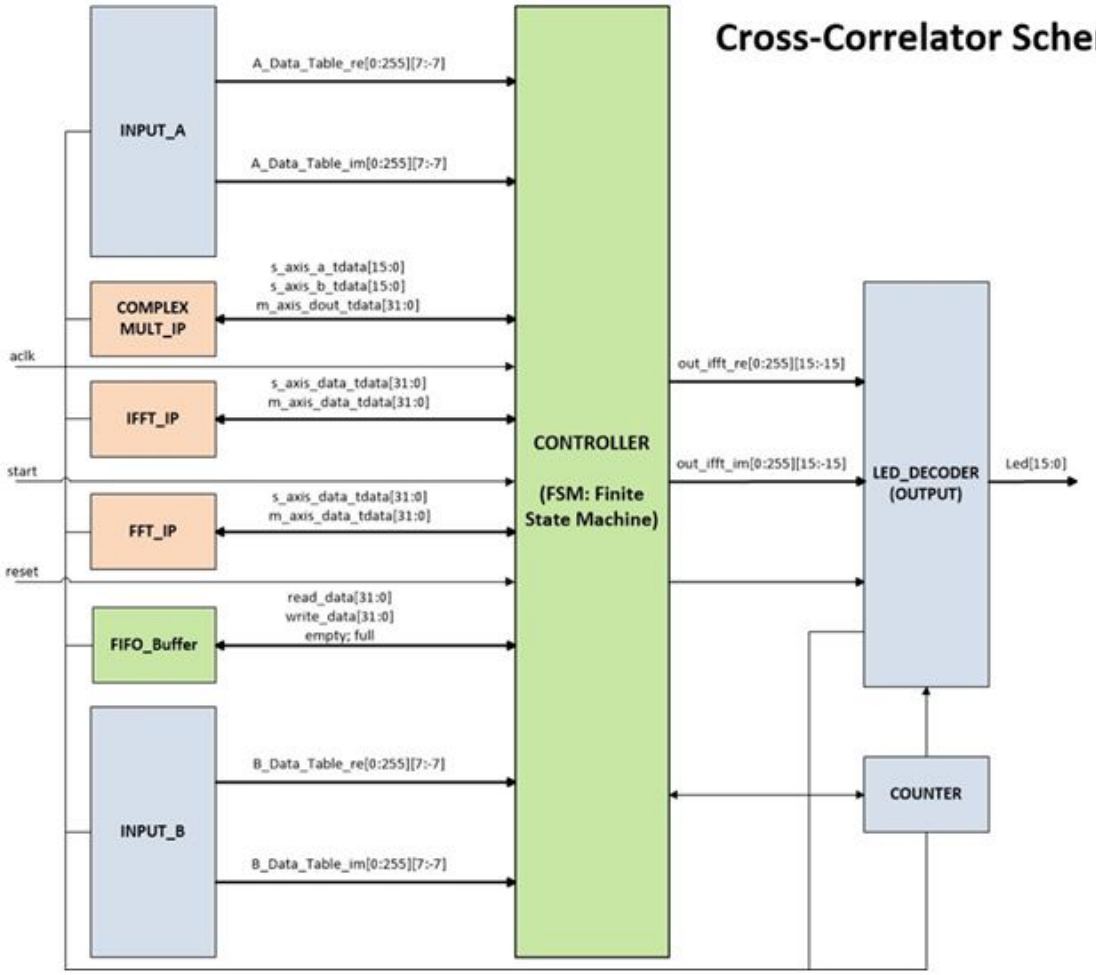

Figura 2.7: Segundo Diagrama en Bloques del *Correlator*

En relación a la figura 2.7, cabe destacar que lo que está en café claro (COMPLEX_MULT_IP, IFFT_IP y FFT_IP), corresponde a los módulos IP que se están utilizando en este diseño. Lo que está en verde (FIFO_Buffer y CONTROLLER) corresponde a los módulos desarrollados completamente en SystemVerilog y lo que está en gris (INPUT_A, INPUT_B, COUNTER y LED_DECODER) también fueron desarrollados en SystemVerilog pero, en estricto rigor, no forman parte del *Correlator*; fueron agregados para poder implementar el diseño dentro de la tarjeta de desarrollo Nexys4.

2.2.5. Diagrama de Flujo Detallado del Controlador del *Correlator*

De acuerdo al Diagrama de Flujo Simplificado (figura 2.6) y considerando el Diagrama en Bloques Detallado de la figura 2.7, se obtiene el Diagrama de Flujo Detallado del Controlador del *Correlator* que se muestra en las figuras A1 a A4 del Anexo A.4.

Los diagramas de flujo obtenidos resultan muy auto-explicativos. Por ejemplo, para el primer diagrama de flujo (figura A1), los bloques que representan los estados $S2$ al $S13$ corresponden a la configuración del módulo FFT de acuerdo a la información entregada en [32]. La entrega de información al módulo FFT y su posterior entrega de resultados, corresponde a la secuencia representada por los estados $S21$ a $S26$ del segundo diagrama de flujo (figura A2).

2.2.6. Diseño e Implementación Final del *Correlator*

De acuerdo al punto 2.1, corresponde desarrollar el Diagrama MDS para el diseño del Controlador del *Correlator*. Las figuras 2.8 y 2.9 muestran dicho diagrama. Como se indicó anteriormente, el Diagrama MDS se obtiene directamente del Diagrama de Flujo Detallado de acuerdo al procedimiento descrito en el Anexo A.11. Este diagrama MDS es similar a lo que se conoce en los textos como diagrama ASM [4] (*Algorithmic State Machine*). El MDS es un diagrama más fácil de entender que un ASM. Además, con la información contenida en el diagrama MDS es muy fácil obtener el programa en un lenguaje HDL; SystemVerilog en nuestro caso.

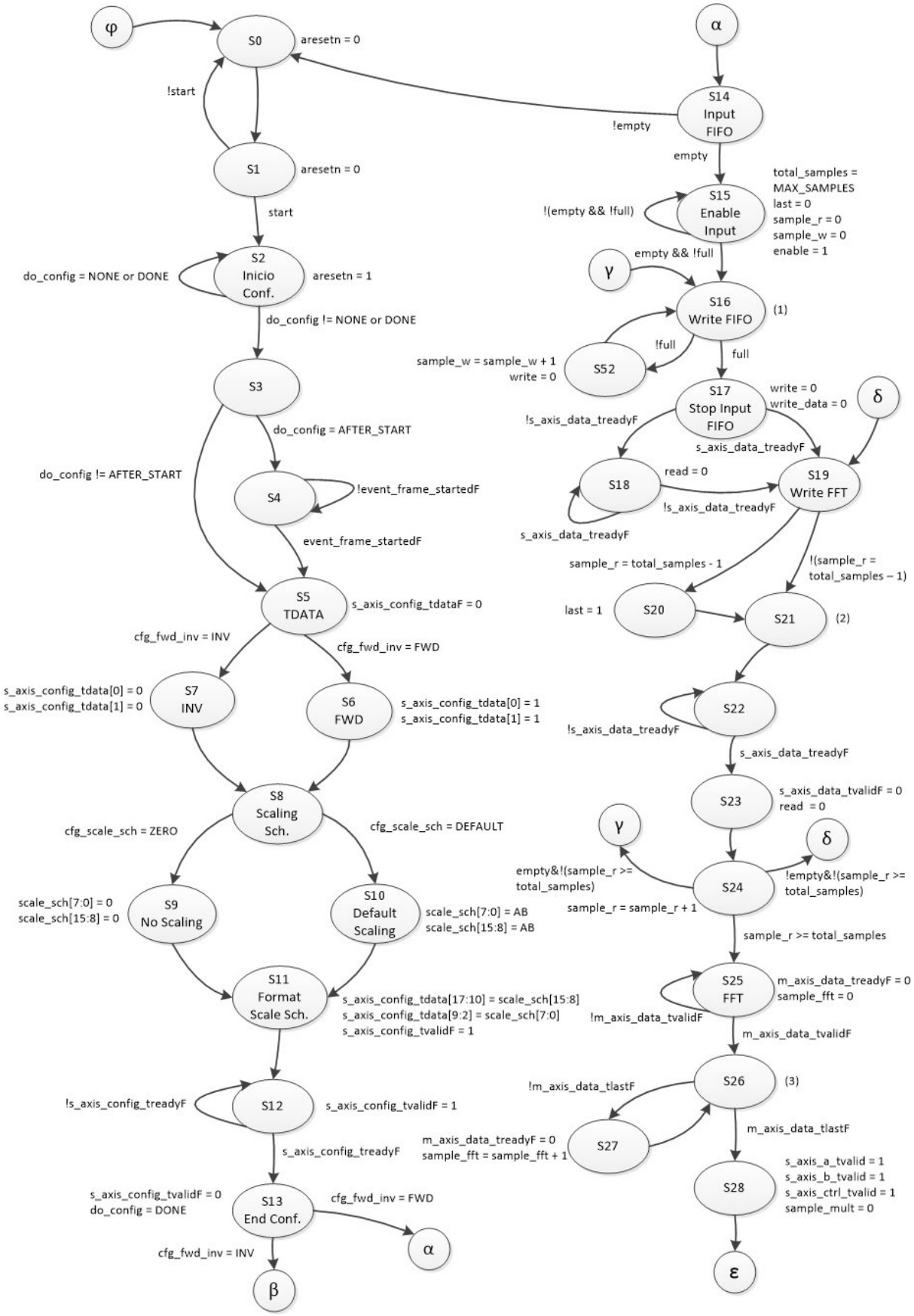

Figura 2.8: Diagrama MDS del Controlador del *Correlator* (1 de 2)

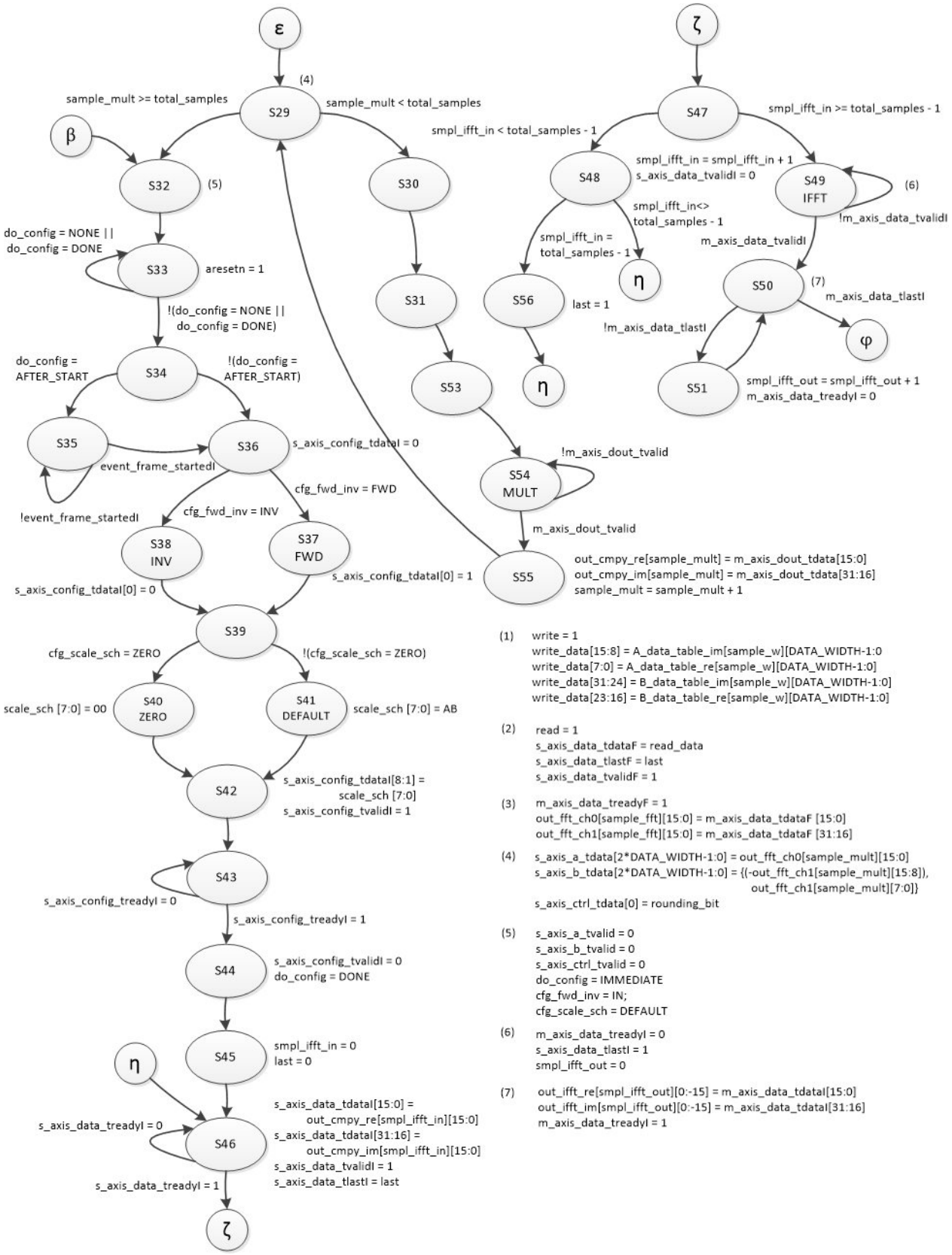

Figura 2.9: Diagrama MDS del Controlador del *Correlator* (2 de 2)

De acuerdo al Diagrama en Bloques Detallado (figura 2.7) existen 9 bloques, de los cuales,

tres son bloques IP que fueron utilizados en este diseño. Los 6 bloques restantes fueron desarrollados en SystemVerilog e ingresados en la herramienta VIVADO [34] [36]. La figura 2.10 muestra parte de la pantalla principal de la herramienta con todos los módulos ingresados. El programa *Correlator_ Top* es el programa de mayor jerarquía que aglutina a los 9 módulos del Diagrama en Bloques Detallado y el encargado de manejar las entradas globales (*aclk*, *reset* y *start*) y las salidas globales (*led [15:0]*).

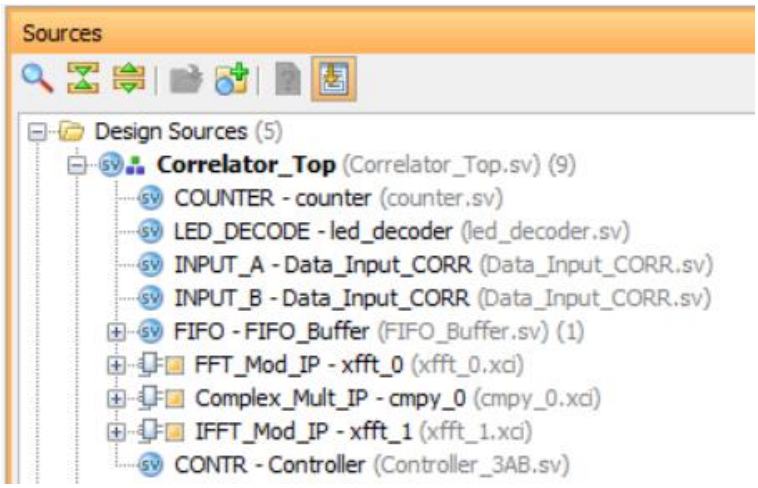

Figura 2.10: Módulos del Diseño del *Correlator* ingresados en VIVADO

Al ingresar los módulos en la herramienta VIVADO, ésta los compila automáticamente, generando mensajes cuando detecta problemas en la sintaxis. La figura 2.11 muestra la secuencia de procesos que se deben realizar en la herramienta VIVADO desde el ingreso de los programas en SystemVerilog (etapa *RTL Source*) hasta la carga del diseño final en la tarjeta de desarrollo (etapa *Bitstream File*), pasando por las etapas de sintetización (*Synthesized Design*) e implementación (*Implemented Design*).

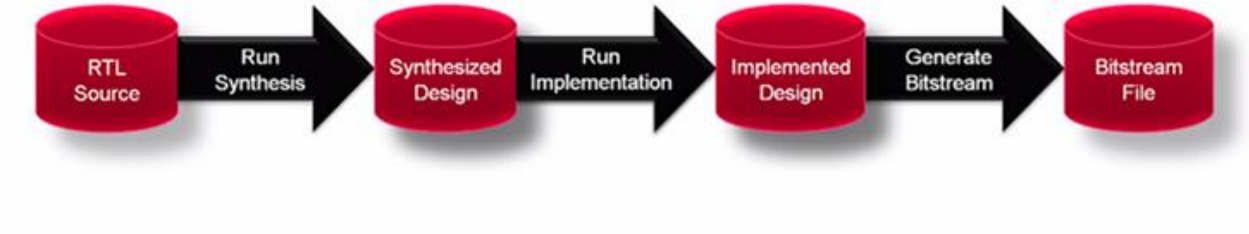

Figura 2.11: Flujo Diseño–Implementación en VIVADO

La figura 2.12 muestra la ventana (*Flow Navigator*) de VIVADO donde se operan las etapas descritas anteriormente (figura 2.11). También esta ventana incluye la operación de la simulación de todo el diseño.

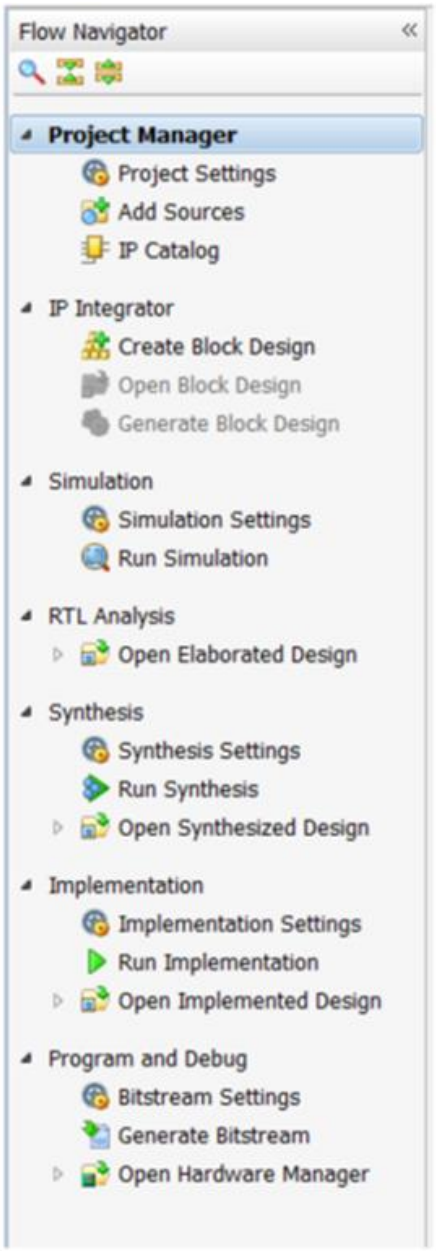

Figura 2.12: Ventana *Flow Navigator* de VIVADO

2.3. Resultados Obtenidos para la Correlación Cruzada

Antes de mostrar los resultados obtenidos para la Correlación Cruzada, es necesario hacer los siguientes comentarios:

1. Dada la gran cantidad de parámetros existentes en este proyecto, fundamentalmente los parámetros que requieren los módulos IP utilizados, se hizo una serie de simulaciones para poder encontrar los parámetros claves en este caso. Se detectó que los parámetros más importantes son la cantidad de bits para la representación de los datos de entrada y la cantidad de muestras a utilizar en el cálculo de la correlación.
2. Para la configuración de los tres módulos IP utilizados (transformada de Fourier, mul-

tiplicador complejo y transformada inversa de Fourier), se aplicó un enfoque orientado a obtener mayor *performance* por sobre la optimización en el uso de los recursos del FPGA.

3. Como se indicó anteriormente, dada la limitación de la tarjeta de desarrollo Nexys4 en la cantidad de pines de entrada/salida, se decidió incluir la generación de los datos dentro del diseño como también el mostrar los datos de salida en un conjunto de "display" de 7 segmentos con que cuenta la tarjeta. En la figura 2.10, el módulo INPUT_A-Data_Input_CORR genera los datos para la entrada A y el módulo INPUT_B-Data_Input_CORR, genera los datos para la entrada B. El módulo LED_DECODE-led_decoder, junto con el módulo COUNTER-counter, se encargan de mostrar los datos de salida en los "display" de 7 segmentos.
4. Además del problema del punto anterior, se encontró otra limitante del FPGA de la tarjeta de desarrollo, relacionada con el tamaño del diseño completo. Se pudo ingresar al FPGA un diseño que contempla entradas de 8 bits y 256 muestras. Al ingresar valores superiores, como 16 bits y/o 1024 muestras, la herramienta entrega un mensaje dando cuenta que se han sobrepasado los límites del dispositivo.

En cuanto a los resultados obtenidos, podemos decir que hubo dos etapas en el proceso de búsqueda de resultados. La primera de ellas, fue trabajar con entradas de 8 bits para la parte real y 8 bits para la parte imaginaria. Junto con ello se consideraron 256 muestras. Con estos parámetros se logró no solamente simular el diseño sino que también realizar el proceso completo (figura 2.11) que incluye la carga del diseño final en la tarjeta de desarrollo y su ejecución en ella.

La figura 2.13 (generada en MATLAB [13]) muestra un primer resultado obtenido, considerando 8 bits de entrada (parte real y parte imaginaria). En este caso, las dos entradas de la misma frecuencia y 256 muestras.

Las dos señales de la parte superior de la figura 2.13, corresponden a las entradas A y B respectivamente (iguales en este caso). Estas señales son generadas por los módulos INPUT_A-Data_Input_CORR e INPUT_B-Data_Input_CORR respectivamente (figura 2.10). Las frecuencias utilizadas en este caso son $FREQ1 = 5,2Hz$, $FREQ2 = 0Hz$ de acuerdo a las expresiones 2.5.

Dentro del módulo *testbench* del diseño (*Correlator_Top_tb.sv*), se imprimen estas entradas en un archivo el cual es leído desde MATLAB para generar el gráfico de la figura 2.13.

La tercera señal de la figura 2.13 corresponde a la salida generada en MATLAB, correspondiente a la siguiente expresión [13]:

$$X = \mathrm{i}fft(fft(complex(A\_real, A\_\mathrm{i}mag), L).*conj(fft(complex(B\_real, B\_\mathrm{i}mag), L)))/L, \tag{2.6}$$

dónde: A_real y A_imag corresponden a la parte real y parte imaginaria de la entrada A, respectivamente. B_real y B_imag corresponden a la parte real y parte imaginaria de la entrada B, respectivamente. L es la cantidad de muestras. La ecuación 2.6 corresponde exactamente al cálculo que realiza nuestro *Correlator* de acuerdo a la ecuación 2.2, dada en la sección 2.2.1.

La cuarta señal de la figura 2.13 es la salida obtenida por nuestro diseño. Esta salida también

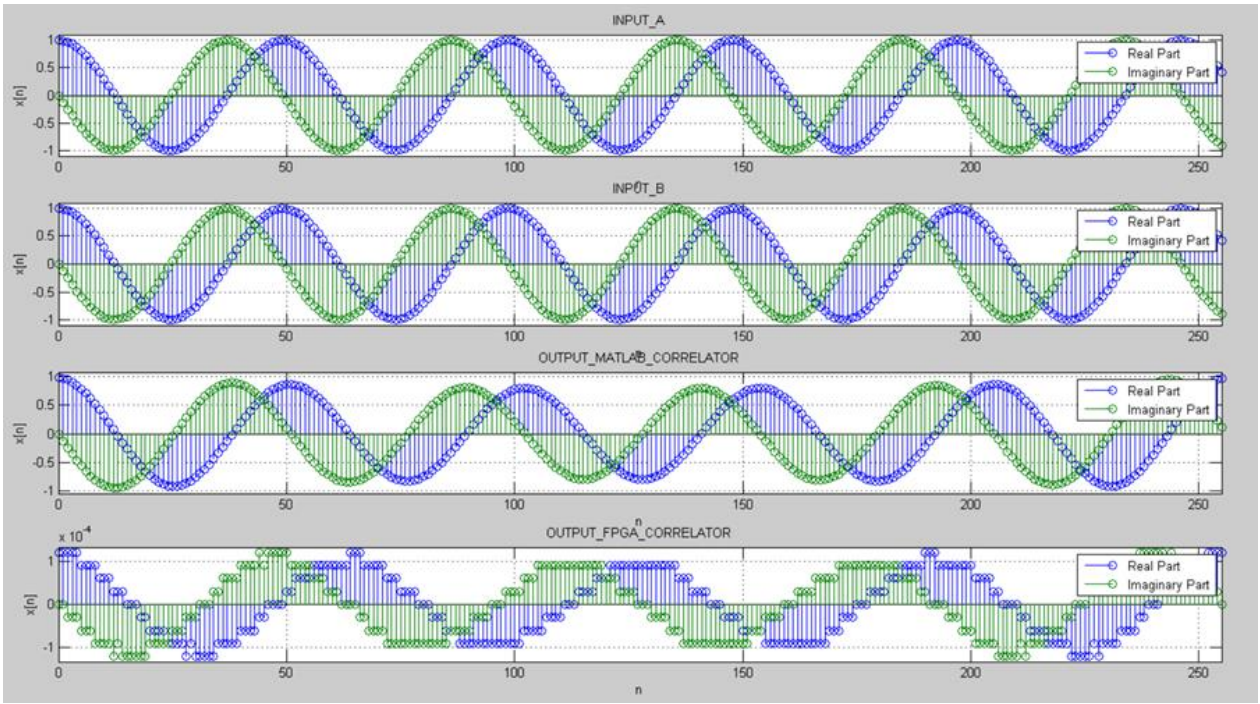

Figura 2.13: Resultado para el *Correlator*: entradas iguales de 8 bits y 256 muestras ($FREQ1 = 5,2Hz$, $FREQ2 = 0Hz$)

es grabada en un archivo que luego es graficada con MATLAB. Como se puede ver no corresponde exactamente a lo entregado por MATLAB de acuerdo a la tercera señal de la figura.

Se hicieron muchas simulaciones para tratar de entender el porqué de los malos resultados obtenidos por el diseño. En un principio se pensó que el problema estaba en la cantidad de muestras. Se aumentó a 1024 muestras pero el resultado fue similar. Se hicieron cambios en el diseño, pensando que había errores en el ingreso de datos y en la lectura de resultados en los módulos IP utilizados en el diseño. No se lograron mejoras significativas.

Finalmente, se encontró que el problema se debía a la cantidad de bits utilizados para representar las entradas A y B de nuestro diseño. Al aumentar a 16 bits los resultados cambiaron drásticamente.

En efecto, la figura 2.14 muestra lo mismo que la figura 2.13 pero considerando 16 bits para las entradas A y B y 256 muestras. Como se puede ver, el resultado obtenido de nuestro diseño, corresponde exactamente a lo indicado por MATLAB.

La figura 2.15 muestra la magnitud del espectro de frecuencia para la correlación (salida FPGA) de la figura 2.14. Se ve claramente la frecuencia involucrada ($FREQ1$).

En el Anexo A.5 se muestran una serie de ejemplos para diferentes entradas sinusoidales con diferentes frecuencias de las señales y para 256 y 1024 muestras. En todos ellos se puede constatar que los resultados obtenidos son idénticos a los entregados por MATLAB.

A continuación, se muestran resultados para entradas no sinusoidales. El primero corresponde a Husos de Sueños (HS) obtenidos de electroencefalogramas (EEG). Se tratan de registros polisomnográficos tomados de niños sanos (10 años de edad), adquiridos en el laboratorio

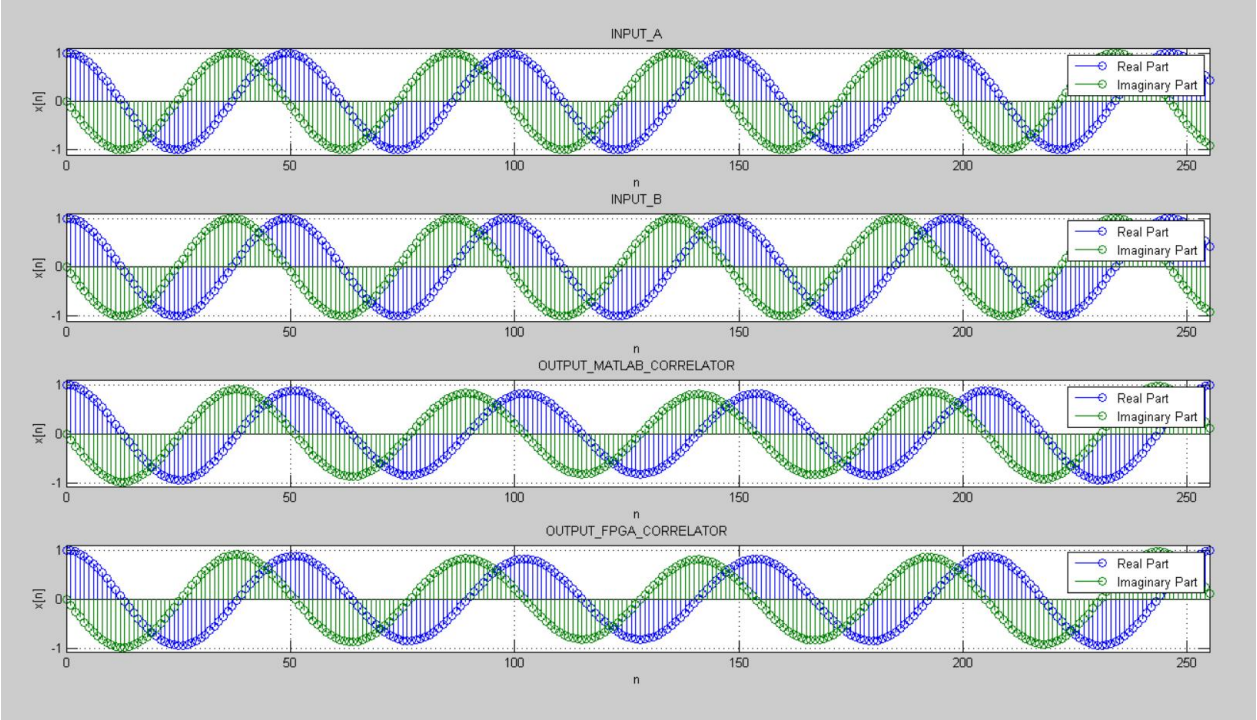

Figura 2.14: Resultado para el *Correlator*: entradas iguales de 16 bits y 256 muestras ($FREQ1 = 5,2Hz$, $FREQ2 = 0Hz$)

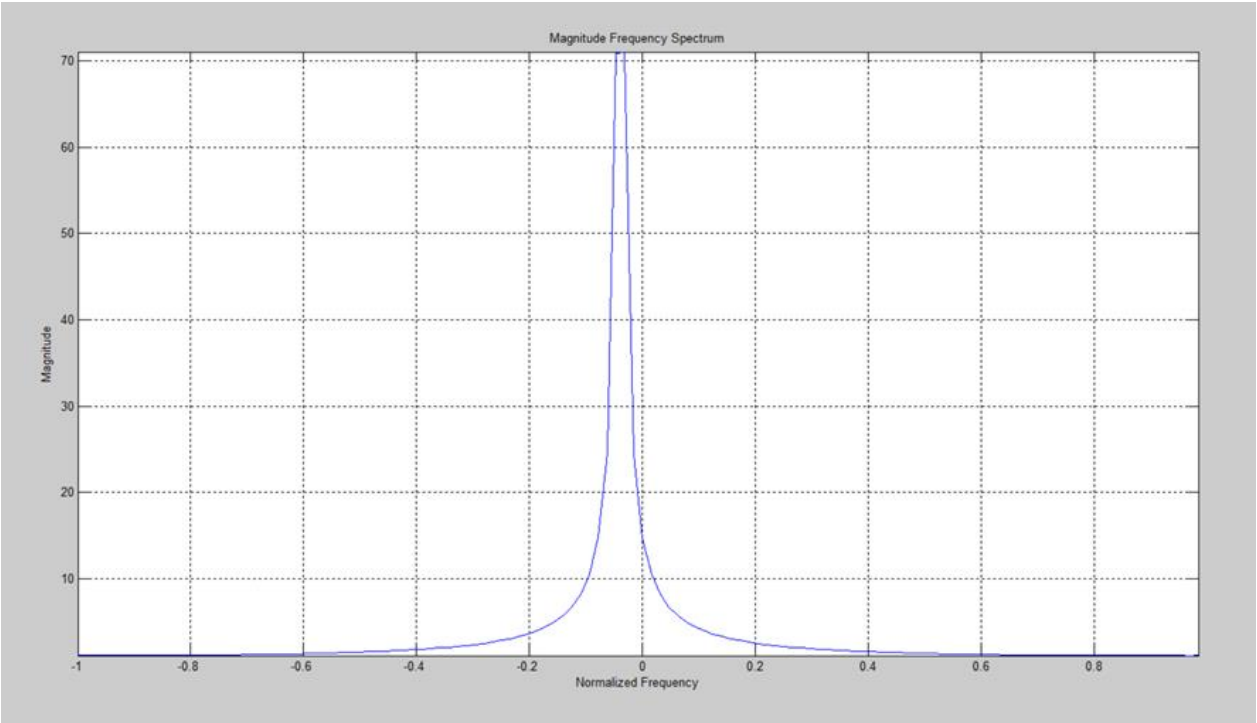

Figura 2.15: Magnitud del Espectro de Frecuencia de la Correlación (salida FPGA)

del sueño del Instituto de Nutrición y Tecnología de los Alimentos INTA, Universidad de Chile. Para la adquisición de datos se empleó un polígrafo modelo EEG-II de 32 canales. Estos registros tienen la característica de estar muestreados a 200 [Hz] con un pre-filtrado de

0,5 a 60 [Hz]. La figura 2.16 se muestra la salida del *Correlator* para una señal de Husos de Sueños. Aquí también se obtiene una salida idéntica a la obtenida con MATLAB.

La figura 2.17 muestra la magnitud del espectro de frecuencia para la correlación obtenida en la figura 2.16 donde se puede apreciar que destacan claramente un par de frecuencias.

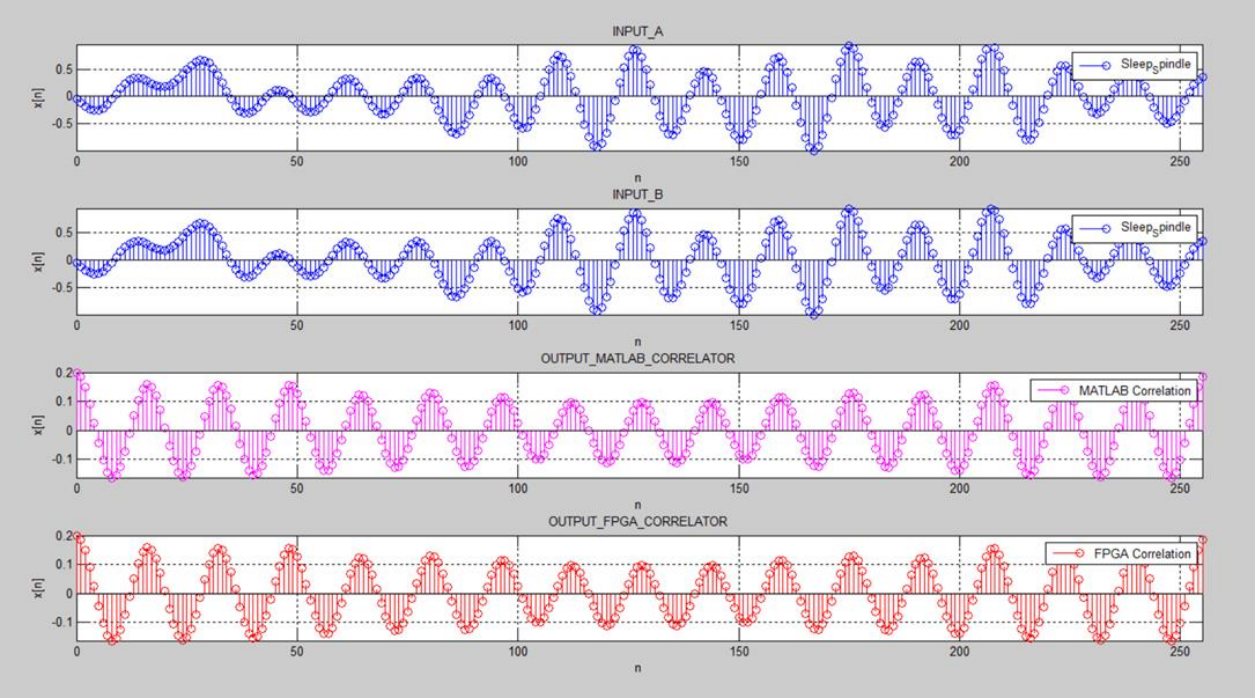

Figura 2.16: Resultado para el *Correlator* para señal de Husos de Sueño

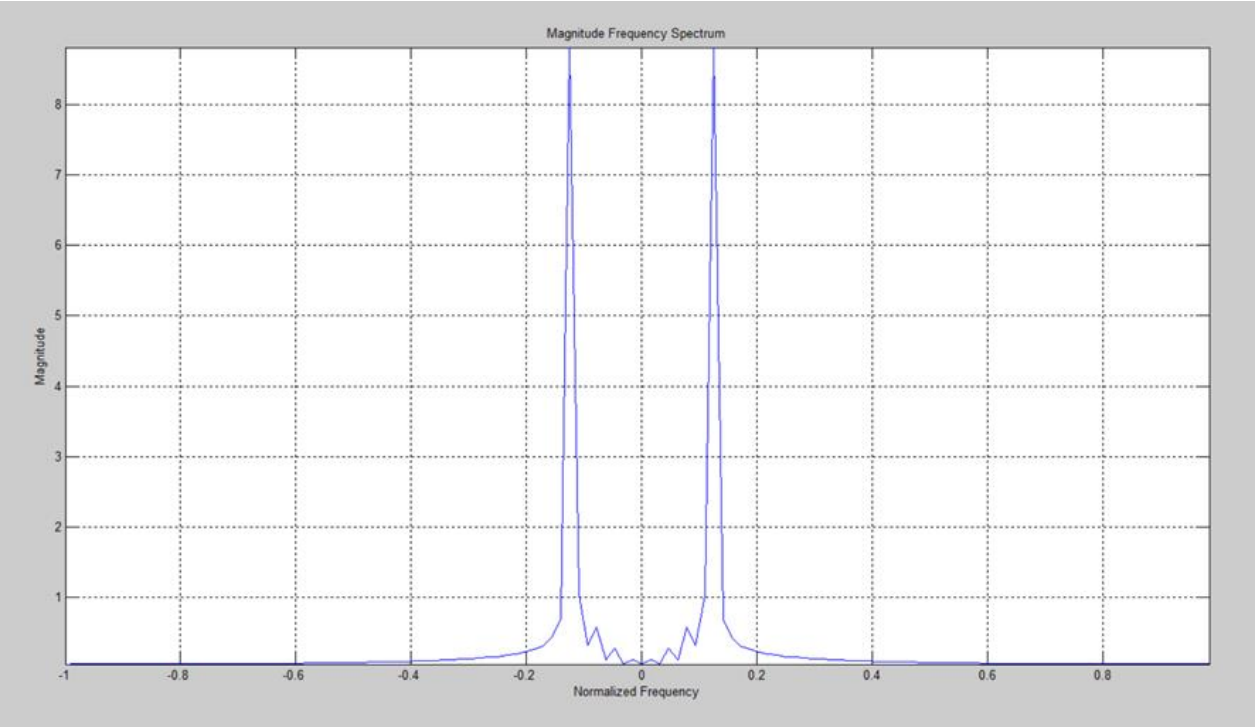

Figura 2.17: Magnitud del Espectro de Frecuencia de la Correlación de un HS

Finalmente, para una aplicación en la Astronomía, la figura 2.18 muestra el resultado obtenido para una entrada correspondiente a una Curva de Luz Sintética. Esta señal corresponde a una estrella variable de tipo RR Lyrae, llamadas así porque el prototipo de estrella es RR de la Lyra (de la constelación de la Lira). Son estrellas de tipo espectral A al F que tienen cambios en su radio (pulsaciones radiales) con períodos de 0,2 a 1,2 días y amplitudes (cambios de brillo) de 0,2 a 2 magnitudes. La utilizada en este ejemplo tiene un período de 0,5 días. Son astros intrínsecamente bastante brillantes: su magnitud absoluta es próxima a 0,50 (compárese con la del Sol que es igual a 4,81). Tradicionalmente, se denomina también a las RR Lyrae "cefeidas de corto período" o "variables de cúmulo", por aparecer en gran cantidad en cualquier cúmulo globular.

Las entradas A (INPUT_A) y B (INPUT_B) que se muestran en la figura 2.18, los ejes de las ordenadas están invertidos. Esto está de acuerdo con la convención utilizada en Astronomía que dice que mientras más brillante es la señal ésta es más negativa.

Con el fin de obtener más detalle del resultado entregado en la figura 2.18, la figura 2.19 muestra el mismo resultado pero donde se ha restado el valor medio de la señal obtenida en la figura 2.18.

Igualmente al resultado de los ejemplos anteriores, se logra una salida de acuerdo a lo indicado por MATLAB.

La figura 2.20 muestra la magnitud del espectro de frecuencia para la señal original, mostrada en la figura 2.18.

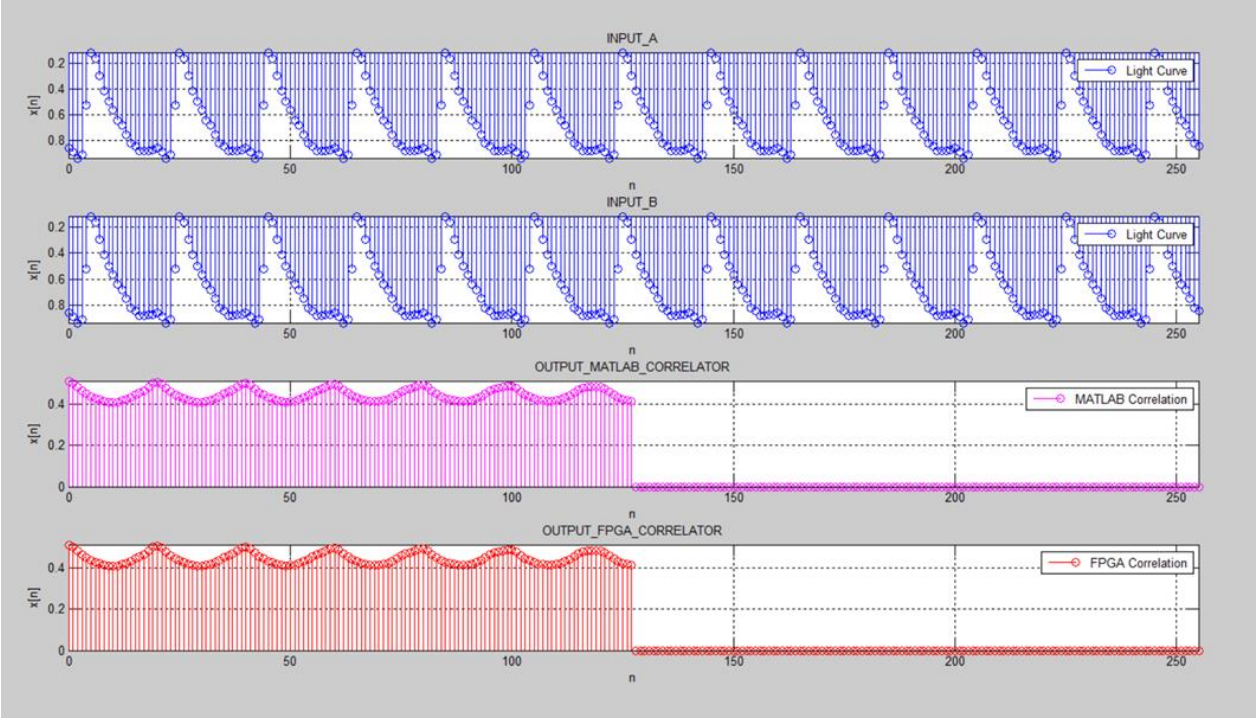

Figura 2.18: Resultado para el *Correlator* para señal de una Curva de Luz Sintética

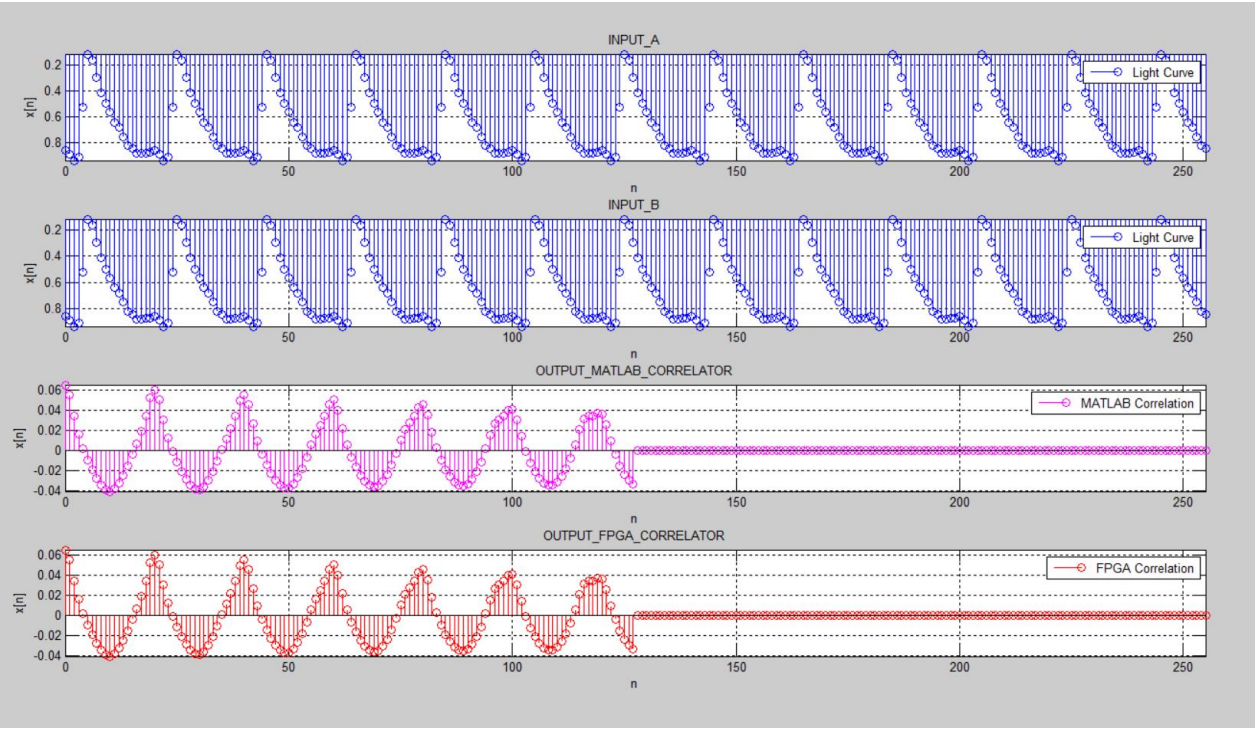

Figura 2.19: Resultado para el Correlator para señal de una Curva de Luz Sintética con la media descontada

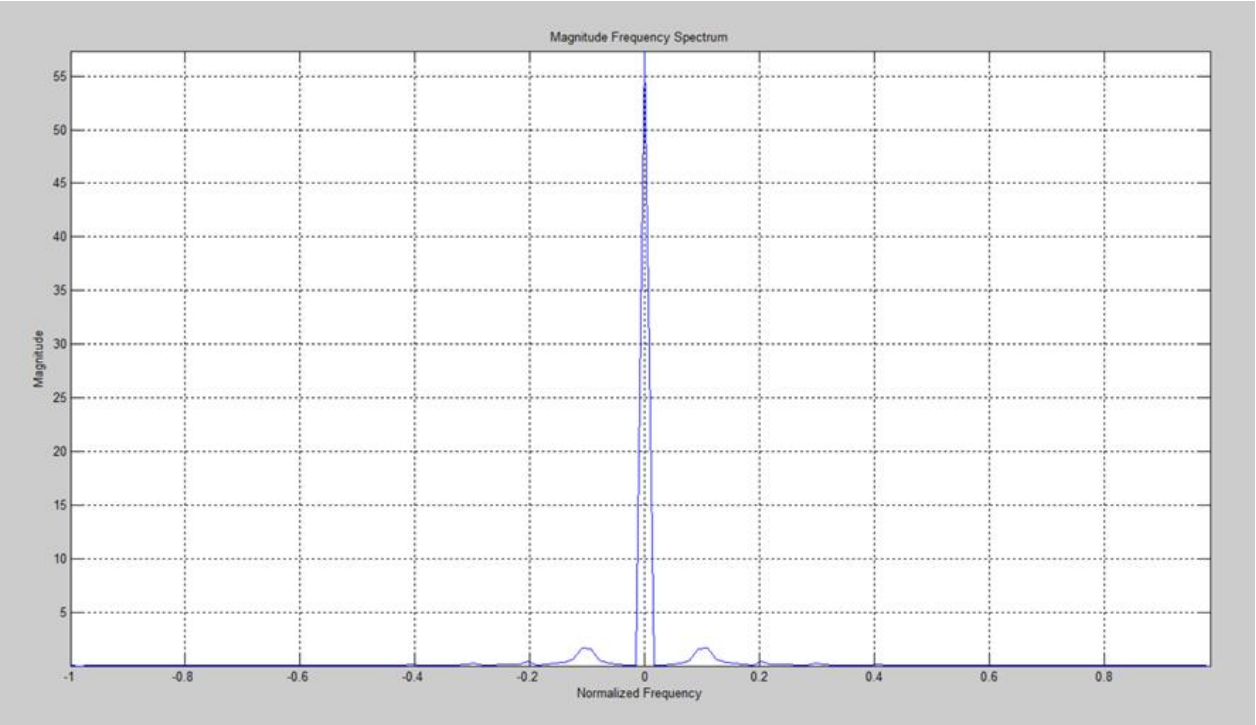

Figura 2.20: Magnitud del Espectro de Frecuencia de la Correlación de una Curva de Luz Sintética (Figura 2.18)

2.4. Resultados de la Implementación del *Correlator* en el FPGA

Como se indicó anteriormente, tenemos una limitante en la capacidad del FPGA en la tarjeta de desarrollo. Se pudo simular, sintetizar, implementar y cargar el diseño en la tarjeta de desarrollo, solamente para el caso de representar en 8 bits las entradas y considerando 256 muestras. Para valores mayores a los indicados, VIVADO entrega un mensaje indicando que se sobrepasaron los límites del dispositivo. La figura 2.21 muestra el nivel de ocupación (el fondo celeste o más claro) del FPGA cuando se implementa un diseño con 8 bits para las entradas y 256 muestras.

Como se puede ver, el nivel de ocupación es del orden del 90 %. Por lo menos fue posible verificar, utilizando 8 bits para las entradas y 256 muestras, que el diseño funciona correctamente en la tarjeta de desarrollo, entregando los resultados cuya simulación se muestra en la figura 2.13.

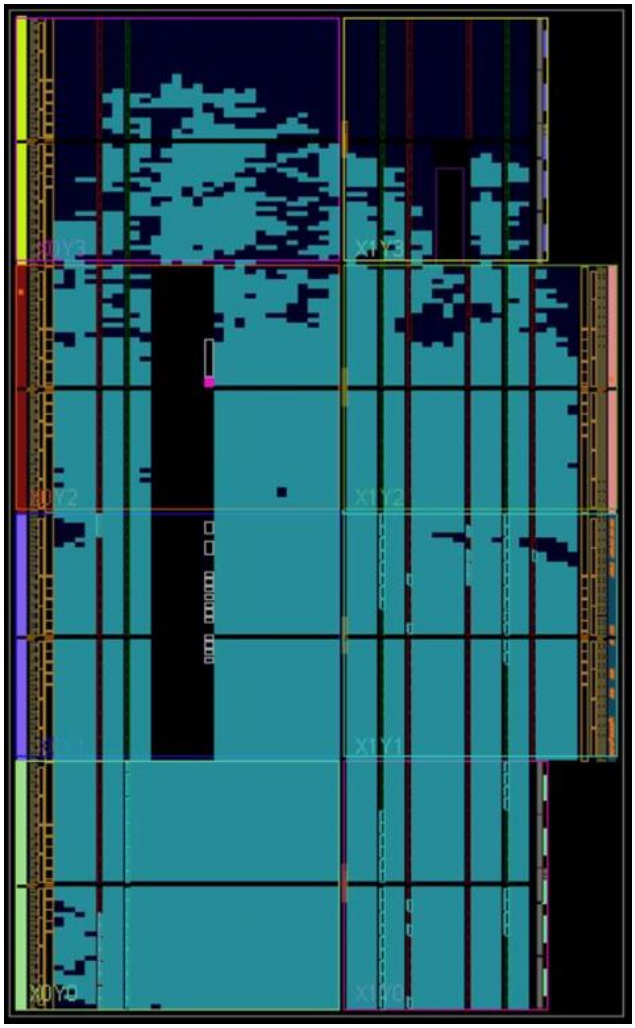

Figura 2.21: Nivel de ocupación del Diseño del *Correlator* en el FPGA

Capítulo 3

DISEÑO E IMPLEMENTACIÓN EN HARDWARE DE LA CORRENTROPÍA UTILIZANDO FPGA

3.1. Metodología de Diseño

La metodología de diseño para abordar sistemas digitales complejos se planteó en el punto 2.1 y es la misma que se aplicará para el diseño del *CorrentropyTor*. En este caso el Sistema Controlado no contará con módulos IP (salvo el módulo CORDIC que se utilizará en una de las opciones para la implementación del *Kernel* Gaussiano) como en el caso del diseño del *Correlator*; todos tendrán que ser desarrollados en SystemVerilog.

3.2. Diseño e Implementación de la Correntropía Cruzada

3.2.1. Consideraciones en relación al Diseño de la Correntropía Cruzada

Como se indicó en el punto 2 de la Introducción, la función Correntropía cruzada para procesos aleatorios discretos, se define según lo indican las ecuaciones 1 y 2.

Podríamos decir que la forma convencional de implementar esta función sería utilizando una expansión en una serie de Taylor [17] [22]. Aunque se trabajó en esta opción, finalmente, se decidió desarrollar en SystemVerilog las expresiones de las ecuaciones 1 y 2 considerando el programa obtenido en la aplicación MATLAB. La función Correntropía desarrollada en MATLAB, se muestra en el Anexo A.9.

Analizando las ecuaciones 1 y 2 de la Introducción, podemos darnos cuenta que hay dos bucles (*loops*) que tenemos que considerar en nuestro diseño: uno de ellos tiene que ver con la sumatoria implícita en la ecuación 1. El segundo bucle, que sería un bucle interno con respecto al anterior, lo constituye el cálculo de la norma al cuadrado dentro de la función exponencial en la ecuación 2, correspondiente al *Kernel* Gaussiano.

Otro aspecto importante a destacar, es la representación interna que vamos a utilizar para el cálculo de la Correntropía. Para el diseño con FPGA es recomendable utilizar una representación en Punto Fijo [2]. En el Anexo A.10 se describe la representación de números binarios en formato Punto-Fijo. La otra alternativa es utilizar Punto Flotante, en base al estándar IEEE 754 [24], que es utilizado internamente por todos los procesadores disponibles actualmen-

te en el mercado (Intel, AMD, MIPS, ARM, por nombrar los más conocidos). Esta última representación requiere de muchos recursos para su implementación y no es recomendable utilizarla en el diseño con FPGA.

3.2.2. Diagrama en Bloques Simplificado del Diseño del *CorrentropyTor*

Como se vio en el diseño del *Correlator*, para enfrentar un diseño complejo en sistemas digitales, es recomendable partir con un diagrama llamado: Diagrama en Bloques Simplificado. Este diagrama permite visualizar en bloques y a grandes rasgos, lo que se quiere obtener finalmente. La figura 3.1 muestra un primer Diagrama en Bloques Simplificado para nuestro sistema a ser implementado en la tarjeta de desarrollo Nexys4. Este primer diagrama se obtuvo considerando la siguiente información:

1. Se utilizará un esquema de diseño como el utilizado para el diseño del *Correlator*: un conjunto de bloques que realizan funciones específicas y todos ellos manejados por un Controlador como se muestra en la Figura 3.1.

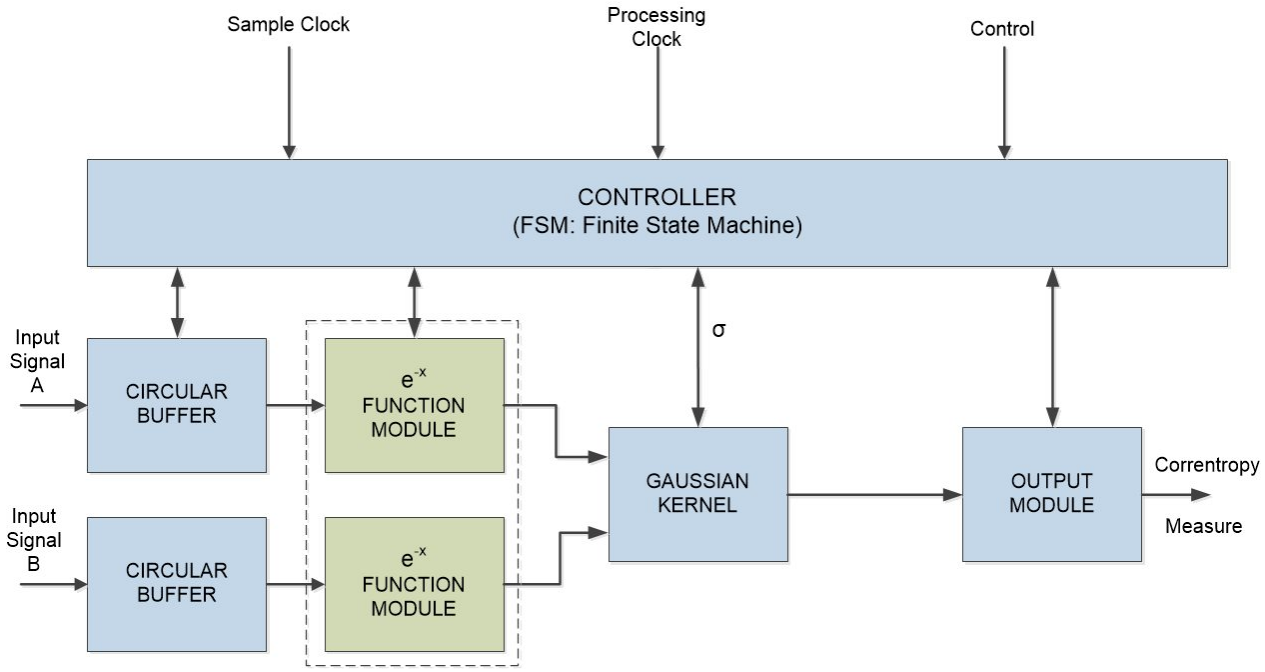

Figura 3.1: Primer Diagrama en Bloques Simplificado del *CorrentropyTor*

De acuerdo a la ecuación 2 de la Introducción, el *Kernel* Gaussiano incluye una función exponencial, función que no existe en el repertorio de funciones del lenguaje de descripción de hardware SystemVerilog. Para generar esta función, una de las alternativas es utilizar un módulo IP de Xilinx [35], disponible para los usuarios sin costo adicional. Este módulo utiliza un algoritmo llamado CORDIC (*COordinate Rotation DIgital Computer*) [27] para calcular *sinh* (seno hiperbólico) y *cosh* (coseno hiperbólico) y, con ello, obtener la función exponencial de acuerdo a la siguiente ecuación de Euler [3]:

$$e^{-x} = cosh(x) - sinh(x) \tag{3.1}$$

La otra alternativa es obtener la función exponencial a través del desarrollo de una serie de Taylor [22]. Se implementaron ambas alternativas como parte del objetivo de analizar diferentes formas de diseño e implementación del *Kernel* Gaussiano.

A continuación, se describen cada uno de los módulos de la figura 3.1:

CIRCULAR BUFFER: este es un módulo de entrada; se trata de un *buffer* tipo FIFO, similar al utilizado en el *Correlator*, que sincroniza los datos de entrada A y B, en tiempo real, con los datos ingresados al siguiente módulo. Esto permite tener un flujo continuo de datos dependiendo del tiempo que tome, a los demás módulos, procesar los datos. En todo caso, se pretende asegurar la no pérdida de la información de entrada. En un principio se pensó utilizar un módulo IP disponible en la herramienta VIVADO pero al final se desarrolló un módulo *ad-hoc* en SystemVerilog.

e^{-x} **FUNCTION MODULE**: Este es un módulo que calcula la función exponencial y puede ser un módulo IP, disponible para los usuarios en la herramienta VIVADO, con una entrada (en radianes), correspondiente al valor de un ángulo y dos salidas que corresponden al cálculo del seno y coseno hiperbólico. El IP utilizado en ese caso es el " CORDIC LogiCORE IP" de Xilinx en su versión 6.0 [35]; última actualización realizada en Octubre 5, 2016.

Un Diagrama en Bloques del IP CORDIC, se muestra en la figura 3.2 donde se detallan sus entradas y salidas. El detalle descriptivo se encuentra en la Tabla 2.1 del documento: PG105 - "LogiCORE IP CORDIC v6.0 Product Guide" [35]. Esta Tabla se reproduce en el Anexo A.6.

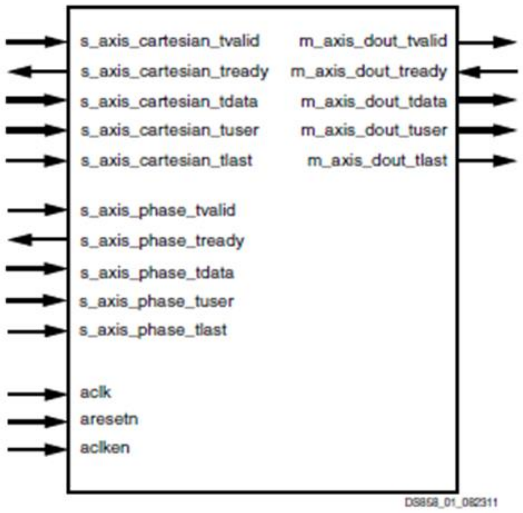

Figura 3.2: Diagrama en Bloques CORDIC LogiCORE IP v6.0 [35]

La otra alternativa para obtener la función exponencial fue a través del desarrollo de una serie de Taylor [22], programada en SystemVerilog.

GAUSSIAN KERNEL: En este módulo se incluyen todas las operaciones de Punto-Fijo que se requieren ejecutar para calcular la Correntropía incluyendo el *Kernel* Gaussiano. Para este último se deben utilizar las salidas del módulo e^{-x} FUNCTION MODULE para obtener la función exponencial.

OUTPUT MODULE: Este módulo permite realizar una adaptación entre formatos de los datos de salida del GAUSSIAN KERNEL y la salida propiamente tal del diseño.

CONTROLLER: Este módulo es el controlador del sistema y se ha de implementar como una Máquina de Estado Finito [28]. Este módulo permite que los demás módulos se ejecuten en secuencia, en un esquema "pipeline", de acuerdo a como se indica en la figura 3.1. Por lo tanto, este módulo ejecuta un algoritmo que se puede representar en un Diagrama de Flujo. La figura 3.3 muestra un Diagrama de Flujo Simplificado del algoritmo a ser ejecutado por el CONTROLLER.

3.2.3. Diagrama de Flujo Simplificado del Controlador del *CorrentropyTor*

Continuando con nuestra metodología de diseño, obtenida de [11], nos corresponde desarrollar el Diagrama de Flujo Simplificado del Controlador de acuerdo al Diagrama en Bloques Simplificado obtenido en la figura 3.1.

Este diagrama de flujo muestra, en forma muy sucinta, el algoritmo que tiene que ejecutar el controlador, llamado CONTROLLER en nuestro Diagrama en Bloques Simplificado (figura 3.1), para hacer funcionar todos los bloques y obtener el resultado esperado. La idea es identificar los grandes procesos que este controlador debería ejecutar.

El Diagrama de Flujo obtenido es el que se muestra en la Figura 3.3. En este diagrama, la señal "START" es una entrada al sistema global que cuando se activa ("1" lógico) se inicia el proceso del cálculo de la Correntropía. La señal "STOP" es también una entrada al sistema global que cuando se activa, permite detener la secuencia de despliegue de las salidas en los "display" de 7 segmentos de la tarjeta de desarrollo Nexys4.

De acuerdo a lo indicado en la figura 3.3, el Controlador comienza ingresando datos al FIFO_BUFFER. Una vez que estos datos están disponibles, se pasan al módulo e^{-x} FUNCTION MODULE que calcula la función exponencial del *Kernel* Gaussiano. Luego, la salida de estos datos, son utilizados para completar el cálculo de la correntropía de acuerdo a las ecuaciones 1 y 2 de la Introducción. También se realizan ajustes de formatos para obtener el resultado final del *CorrentropyTor*.

La siguiente etapa en el proceso de diseño de nuestro *CorrentropyTor*, es verificar que es posible implementar el Diagrama en Bloques Simplificado en nuestra tarjeta de desarrollo Nexys4. Veremos, como en el caso del diseño del *Correlator*, que tenemos que adecuar dicha implementación y se entrega, en el punto siguiente, el diseño definitivo a ser utilizado.

Una vez definido el Diagrama en Bloques Simplificado, se obtiene un Diagrama en Bloques Detallado donde queden especificados completamente los módulos que van a formar el diseño final del *CorrentropyTor*. Una vez obtenido dicho diagrama, se puede obtener el Diagrama de Flujo Detallado del Controlador donde se consideran las señales definitivas de cada uno de los módulos que forman el diseño. Obviamente, se va a obtener un Diagrama de Flujo mucho más extenso. Esto se describe en los puntos 3.2.4 y 3.2.5 siguientes.

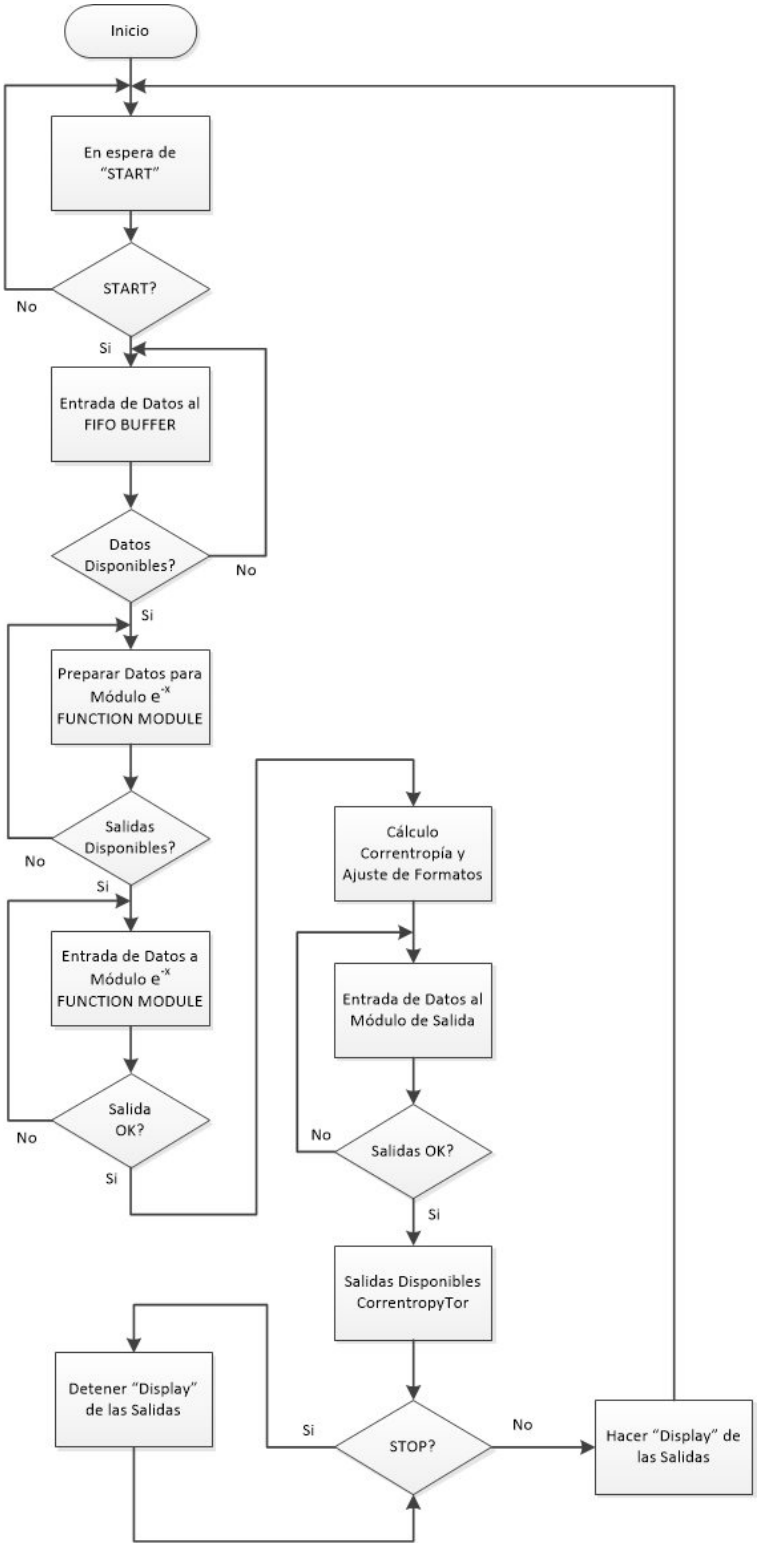

Figura 3.3: Diagrama de Flujo Simplificado del CONTROLLER del *CorrentropyTor*

3.2.4. Diagrama en Bloques Detallado del Diseño del *Correntropy-Tor*

El diseño planteado en el Diagrama en Bloques de la Figura 3.1, tiene un problema para ser implementado en el FPGA por una limitación básica en la cantidad de pines de entrada y salidas de este dispositivo, tal como se explicó en el punto 2.2.4.

Para obviar el problema anterior, se decidió, como en el caso del diseño del *Correlator*, generar las entradas dentro de nuestro diseño por lo cual ya no se necesita contar con 65.536 pines de entrada en el FPGA. Dado que el problema también se presenta en la salida, se decidió mostrar las salidas en los "display" de 7 segmentos con que cuenta la tarjeta de desarrollo Nexsys4.

Considerando lo anterior, se modificó el Diagrama en Bloques de la figura 3.1 y se obtuvo el Diagrama en Bloques Detallado de la figura 3.4. Se trata de un diagrama detallado porque incorpora las diferentes señales de los distintos módulos. Se puede ver ahora que las entradas globales al diseño serían solamente *aclk*, *reset*, *start* y *stop* y las salidas serían *counter*[7:0] y *led[15:0]* (26 pines en total). En este diagrama, además, hay que destacar los siguientes módulos adicionales a los de la figura 3.1:

INPUT_A: corresponde al módulo que genera las entradas A para el *CorrentropyTor*. Se trabaja con expresiones trigonométricas que son de más fácil generación y también para obtener diferentes alternativas. Para compatibilizar con lo que se hizo para el *Correlator*, las entradas son de 16 bits, tanto para la parte real como para la parte imaginaria. Las expresiones en SystemVerilog son las mismas entregadas en las ecuaciones 2.5.

INPUT_B: corresponde a la entrada B y es análoga a la entrada A descrita anteriormente.

COUNTER: es un módulo contador que es necesario agregar para manejar los LED de los "display" de 7 segmentos que se van a utilizar en la salida.

LED_DECODER: es el módulo que permite desplegar la salida en los "display" de 7 segmentos con que cuenta la tarjeta de desarrollo Nexys4. Con el fin de cubrir todos los casos, de acuerdo al valor de sigma, se definieron 32 bits para la salida.

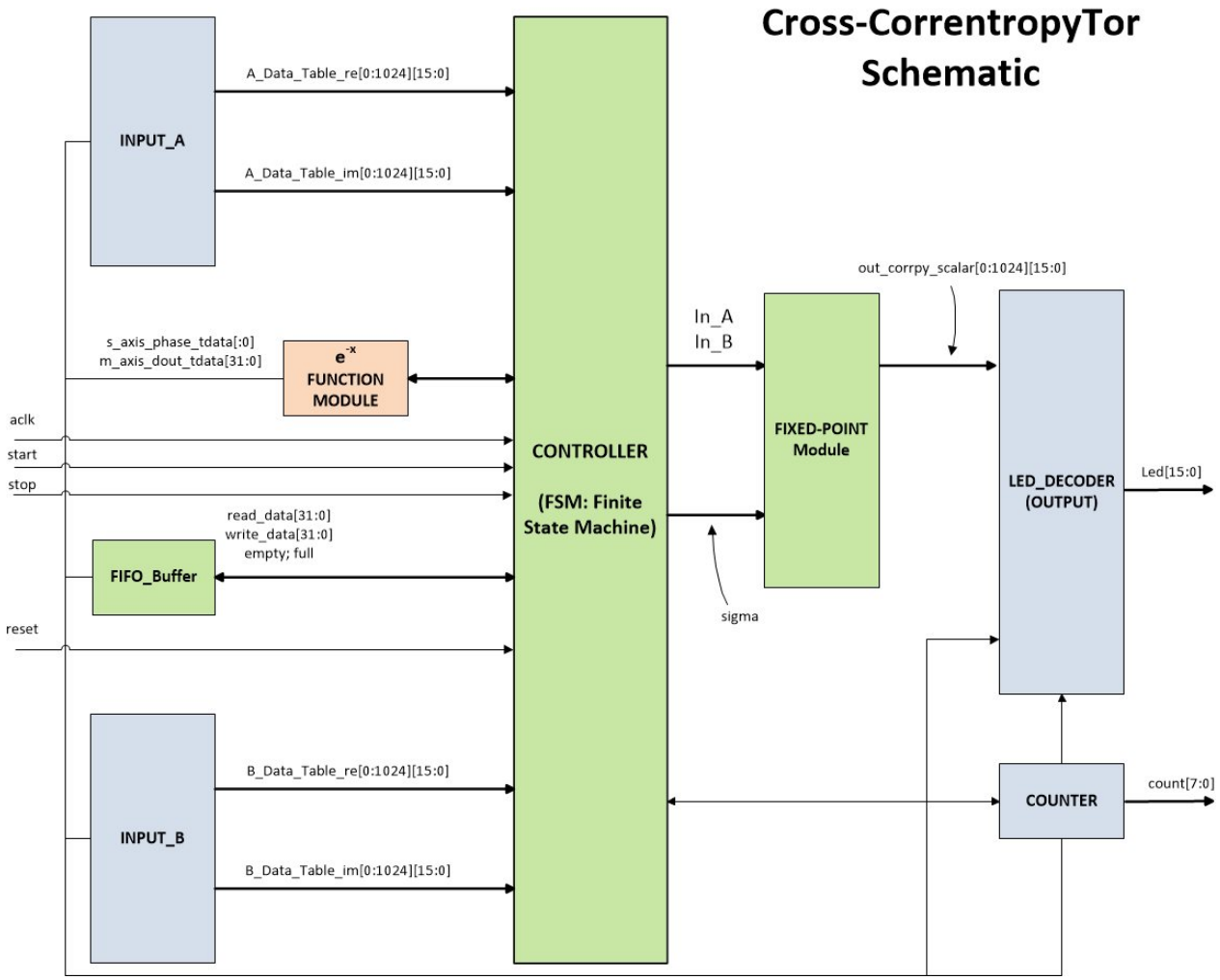

Figura 3.4: Diagrama en Bloques Detallado del *CorrentropyTor*

FIXED-POINT Module: Corresponde al módulo que realiza las operaciones en Punto-Fijo. Contiene, a su vez, los módulos que calculan las sumas, multiplicaciones y divisiones en Punto-Fijo que se requieren para el cálculo del *Kernel* Gaussiano y de la Correntropía. Estos módulos son:

1. **ADD_FIX_N_Q_RE**: Módulo SUMA que calcula la diferencia en Punto-Fijo, en la parte real, entre cada uno de los elementos de x e y en la ecuación 2 de la Introducción, correspondiente al *Kernel* Gaussiano.
2. **ADD_FIX_N_Q_IM**: Módulo SUMA que calcula la diferencia en Punto-Fijo, en la parte imaginaria, entre cada uno de los elementos de x e y en la ecuación 2 de la Introducción, correspondiente al *Kernel* Gaussiano.
3. **MUL_FIX_N_Q_RE**: Módulo MULTIPLICADOR que calcula el cuadrado en Punto-Fijo (parte real) de la salida del módulo ADD_FIX_N_Q_RE.
4. **MUL_FIX_N_Q_IM**: Módulo que calcula el cuadrado en Punto-Fijo (parte imaginaria) de la salida del módulo ADD_FIX_N_Q_IM.
5. **ADD_FIX2_N_Q**: Módulo SUMA que calcula la suma en Punto-Fijo de los

cuadrados de x e y, correspondiente a la suma de las salidas de los módulos MUL_FIX_N_Q_RE y MUL_FIX_N_Q_IM.

6. **MUL_FIX_N_Q_SIG2**: Módulo MULTIPLICADOR que calcula el cuadrado en Punto-Fijo de σ que se utiliza en la expresión del *Kernel* Gaussiano (ecuación 2 de la Introducción).
7. **MUL_FIX_N_Q_RC2PI**: Módulo MULTIPLICADOR que calcula la expresión $\sqrt{2\pi}\sigma$ en Punto-Fijo y que se utiliza en la expresión del *Kernel* Gaussiano (ecuación 2 de la Introducción).
8. **DIV_FIX_N_Q_NARG**: Módulo DIVISOR que calcula en Punto-Fijo el argumento de la función exponencial. Esto corresponde a: $e^{-\frac{\|x-y\|^2}{2\sigma^2}}$ en la expresión del *Kernel* Gaussiano (ecuación 2 de la Introducción).
9. **MUL_FIX_N_Q_FACPI**: Módulo MULTIPLICADOR que adecúa en Punto-Fijo el formato de la salida del módulo MUL_FIX_N_Q_NARG a la entrada del módulo CORDIC (en caso de ser utilizado) el cual requiere un formato fix_{32_29} (véase Anexo A.10).
10. **ADD_COSH_SINH_N_Q**: Módulo SUMA que calcula en Punto-Fijo la expresión de la ecuación 3.1. *Cosh* y *sinh* se obtienen de las salidas del módulo CORDIC.
11. **MUL_FIX_N_Q_RC2PIxSIGxSMPL**: Módulo MULTIPLICADOR que calcula en Punto-Fijo la expresión: $\sqrt{2\pi}\sigma(L-l+1)$, donde L es la cantidad de muestras y l los retardos. Esta expresión se utiliza en la ecuación 1 de la Introducción para el cálculo de la Correntropía.
12. **ADD_RTEMP_N_Q**: Módulo SUMA que calcula en Punto-Fijo la sumatoria en la ecuación 1 de la Introducción para el cálculo de la Correntropía.
13. **DIV_FIX_N_Q_CORR**: Módulo DIVISOR que calcula en Punto-Fijo la expresión final de la Correntropía de acuerdo a la ecuación 1 de la Introducción.

En relación a la figura 3.4, cabe destacar que lo que está en café claro ("e^{-x} FUNCTION MODULE"), corresponde al módulo que calcula la función exponencial. Lo que está en verde ("CONTROLLER", "FIFO_Buffer" y "FIXED-POINT Module") corresponde a los módulos desarrollados completamente en SystemVerilog y lo que está en gris ("INPUT_A", "INPUT_B", "LED_DECODER" y "COUNTER") también fueron desarrollados en SystemVerilog pero, en estricto rigor, no forman parte del *CorrentropyTor*; fueron agregados para poder implementar el diseño dentro de la tarjeta de desarrollo Nexys4.

3.2.5. Diagrama de Flujo Detallado del Controlador del *CorrentropyTor*

Una vez obtenido el Diagrama en Bloques Detallado (figura 3.4), corresponde ahora, de acuerdo al procedimiento de diseño planteado en el punto 2.1, obtener el Diagrama de Flujo Detallado del controlador (módulo CONTROLLER). Este diagrama incorpora todas las señales de entrada y salida de los módulos incluidos en el Diagrama en Bloques Detallado (figura 3.4).

Dada su extensión, se utilizaron 4 figuras, figuras A10 a A13 del Anexo A.7, para mostrar el Diagrama de Flujo Detallado completo.

Cada uno de los bloques rectangulares del Diagrama de Flujo Detallado, constituyen los estados de la máquina de estado finito. El diagrama completo tiene 63 estados.

Como en el diseño del *Correlator*, los diagramas de flujo obtenidos resultan muy auto-explicativos. Por ejemplo, para el primer diagrama de flujo (figura A10), los bloques que representan los estados $S2$ al $S8$ corresponde a la escritura y lectura de datos en el módulo FIFO_BUFFER. Los bloques que representan los estados $S80$, $S81$ y $S82$ corresponden al cálculo de σ^2 y los bloques que representan los estados $S83$, $S84$ y $S85$ corresponden al cálculo de $\sqrt{2\pi}\sigma$.

Una vez que el Diagrama de Flujo Detallado está completo con todos los estados identificados, se construye el Diagrama MDS, similar a un Diagrama de Estado utilizado en el diseño de sistemas basados en "Flip-Flops" y compuertas. Es una traducción directa desde el Diagrama de Flujo. El Anexo A.11 muestra el procedimiento para convertir un Diagrama de Flujo Detallado en un Diagrama MDS.

3.2.6. Diseño e Implementación Final del *CorrentropyTor*

Las figuras 3.5 a 3.7 muestran el Diagrama MDS completo, correspondiente al Diagrama de Flujo Detallado de las figuras A10 a A13 del Anexo A.7. Estos diagramas constituyen el resultado final del proceso de diseño y contienen toda la información para iniciar la etapa de implementación en hardware. En nuestro caso, para la implementación, vamos a utilizar un FPGA pero estos diagramas podrían utilizarse para la implementación con otros dispositivos (por ejemplo, con dispositivos de un nivel de integración de menor escala).

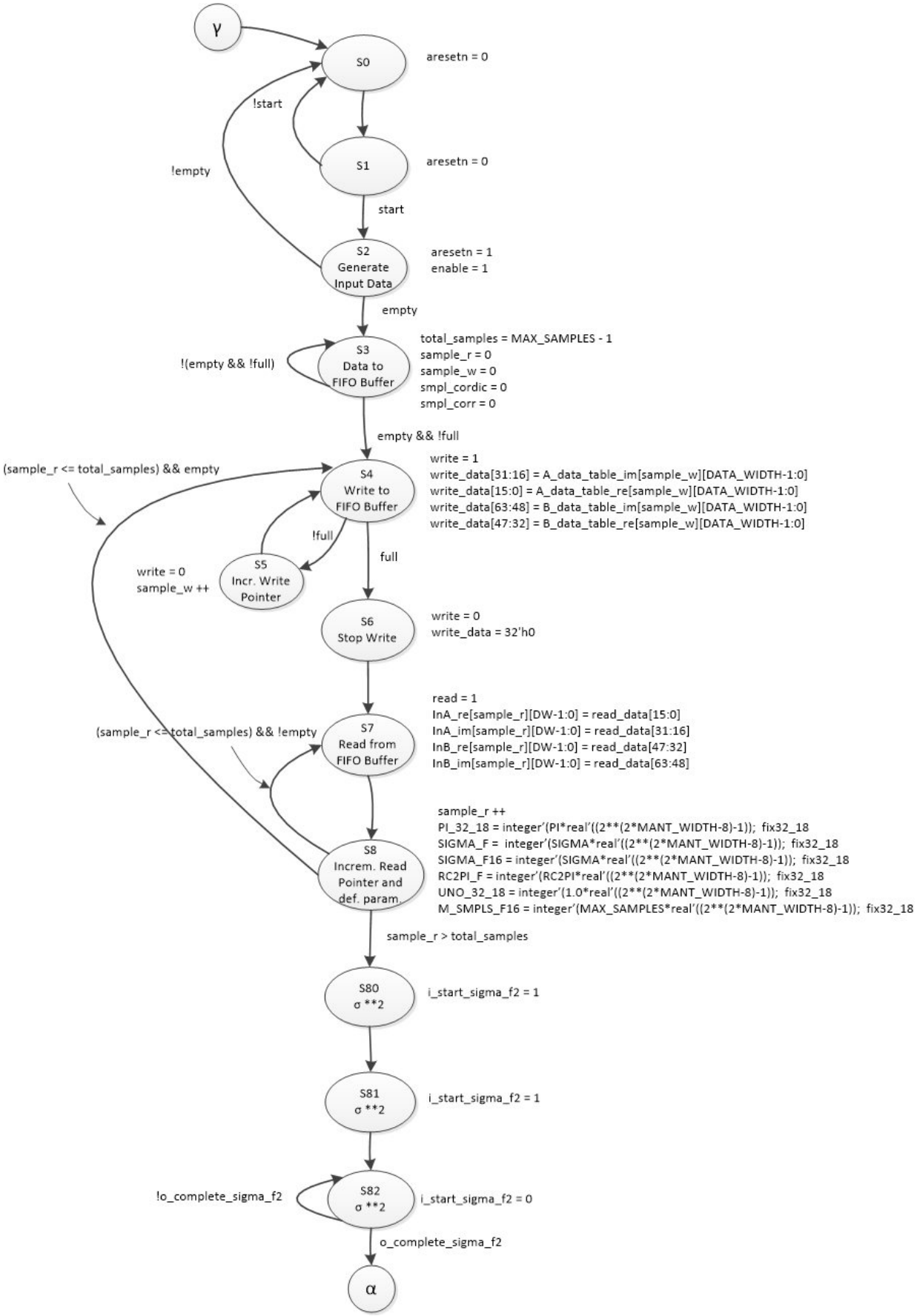

Figura 3.5: Diagrama MDS del Diseño del Controlador del *CorrentropyTor* (1 de 3)

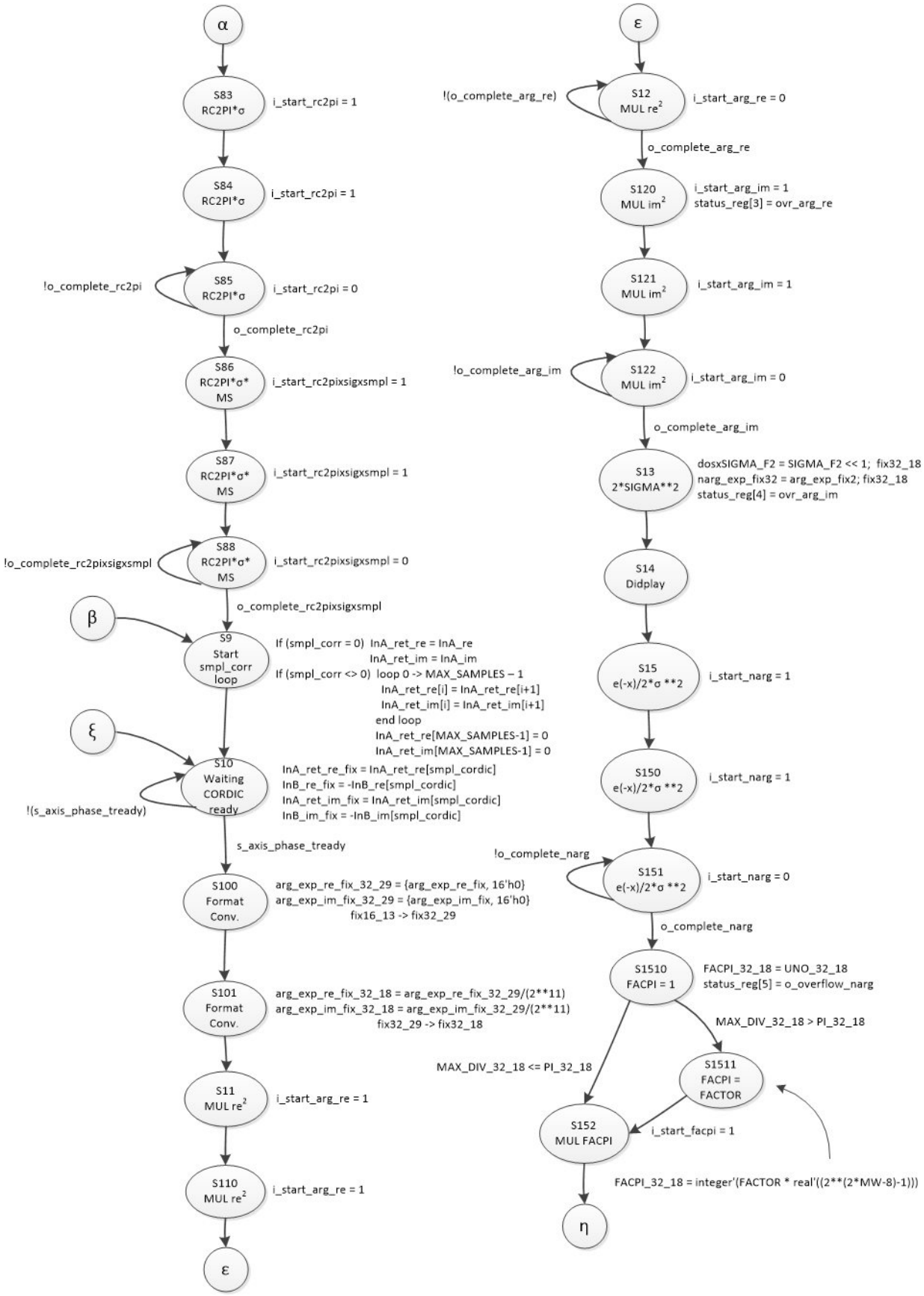

Figura 3.6: Diagrama MDS del Diseño del Controlador del *CorrentropyTor* (2 de 3)

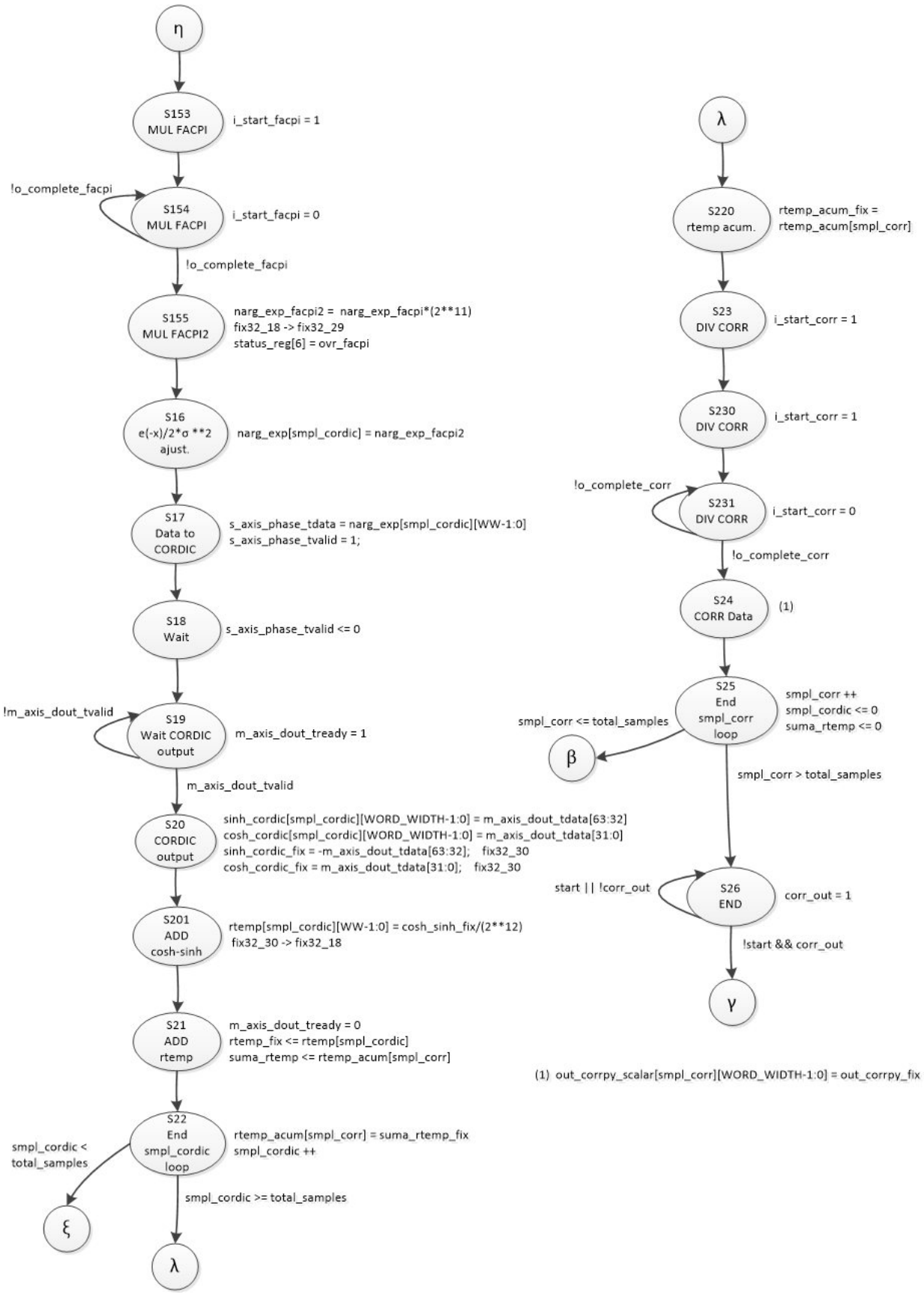

Figura 3.7: Diagrama MDS del Diseño del Controlador del *CorrentropyTor* (3 de 3)

De acuerdo al Diagrama en Bloques Detallado (figura 3.4) tenemos 8 bloques, de los cuales, uno puede ser un bloque IP (CORDIC), de Xilinx, que fue utilizado en este diseño como una de las alternativas para obtener la función exponencial del *Kernel* Gaussiano. Los 7 bloques restantes fueron desarrollados en SystemVerilog e ingresados en la herramienta VIVADO. El bloque FIXED-POINT Module, a su vez, está formado por 13 módulos. La figura 3.8 muestra parte de la pantalla principal de la herramienta VIVADO [29] de Xilinx con todos los módulos ingresados. El programa *CorrentropyTor_ Top* es el programa de mayor jerarquía que aglutina a los 8 módulos del Diagrama en Bloques Detallado y el encargado de manejar las entradas globales (*aclk*, *reset*, *start* y *stop*) y las salidas globales (*led [15:0]* y *counter [7:0]*).

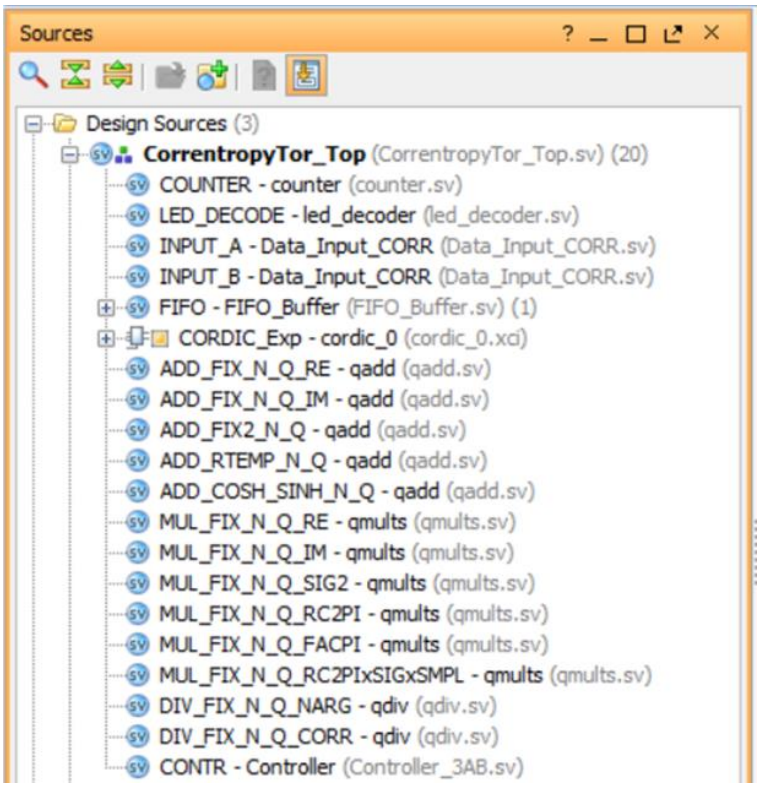

Figura 3.8: Módulos del Diseño del *CorrentropyTor* ingresados en VIVADO

Todos estos módulos fueron implementados en SystemVerilog. Al ingresar los módulos en la herramienta VIVADO, ésta los compila automáticamente, generando mensajes cuando detecta problemas en la sintaxis.

A modo de ejemplo, en el Anexo A.12, se adjunta el programa " CONTR-Controller" de la figura 3.8, escrito en SystemVerilog y que se obtuvo directamente del Diagrama MDS de las figuras 3.5 a 3.7. Este programa corresponde al Sistema Controlador ("CONTROLLER") del *CorrentropyTor* (véase figura 3.4) y tiene alrededor de 550 líneas de código.

Análogamente al caso del diseño del Correlator, la figura 2.11 muestra la secuencia de procesos que se deben realizar en la herramienta VIVADO, para el diseño del *CorrentropyTor*, desde el ingreso de los programas en SystemVerilog (etapa *RTL Source*) hasta la carga del diseño final en la tarjeta de desarrollo (etapa *Bitstream File*), pasando por las etapas de sintetización (*Synthesized Design*) e implementación (*Implemented Design*).

3.3. Resultados Obtenidos para la Correntropía Cruzada

3.3.1. Comentarios Previos

Antes de mostrar los resultados obtenidos para la Correntropía Cruzada, es necesario hacer los siguientes comentarios:

1. En este caso, los parámetros claves en el diseño del *CorrentropyTor* son: el ancho del *Kernel* (σ), la cantidad de bits para la representación de los datos de entrada y la cantidad de muestras a utilizar en el cálculo de la Correntropía.
2. Dentro del diseño, un objetivo importante a desarrollar fue encontrar una solución para el cálculo del *Kernel* Gaussiano. Éste incorpora una función exponencial que no se encuentra en el repertorio de funciones en el lenguaje de descripción de hardware SystemVerilog utilizado. Se manejaron dos alternativas: la primera, y más directa, fue desarrollar la función aplicando una serie de Taylor. La segunda alternativa fue utilizar un módulo IP, basado en el algoritmo CORDIC, disponible en la herramienta VIVADO. Este módulo calcula la función exponencial en base a las funciones hiperbólicas seno y coseno. Esta opción presentó el inconveniente que la entrada al módulo debe estar acotada al rango $(-\frac{\pi}{4}, +\frac{\pi}{4})$. Para entradas acotadas a dicho rango, el módulo IP funciona correctamente, obteniendo los valores deseados para la Correntropía. Finalmente, se decidió utilizar la primera opción dado que no presentaba ningún tipo de restricción y funcionó correctamente.
3. Para la configuración del módulo IP (CORDIC) utilizado (para la obtención del seno y coseno hiperbólico), y de acuerdo a uno de los objetivos específicos estipulados, se aplicó un enfoque de obtener mayor *performance* por sobre la optimización en el uso de los recursos del FPGA.
4. Como se destacó en el punto 3.2.1, el proceso de cálculo de la Correntropía, implica ejecutar dos bucles; uno dentro del otro. Cada uno de estos bucles va a iterar la cantidad de muestras dadas. En el proceso de simulación trabajamos con 256 y 1024 muestras. Para el primer caso, el tiempo de simulación fue de aproximadamente 20 minutos (256*256 = 65.536 iteraciones) y para 1024 muestras el tiempo fue de alrededor de 7 horas (1024*1024 = 1.048.576 iteraciones).
5. De acuerdo a lo indicado anteriormente, dada la limitación del FPGA de la tarjeta de desarrollo Nexys4 en la cantidad de pines de entrada/salida, se decidió incluir la generación de los datos dentro del diseño como también el mostrar los datos de salida en un conjunto de "display" de 7 segmentos con que cuenta la tarjeta. En la figura 3.8, el módulo INPUT_A-Data_Input_CORR, genera los datos (en base a expresiones sinusoidales) para la entrada A y el módulo INPUT_B-Data_Input_CORR, genera los datos para la entrada B (también en base a expresiones sinusoidales). El módulo LED_DECODE-led_decoder, junto con el módulo COUNTER-counter, se encargan de mostrar los datos de salida en los "display" de 7 segmentos. Este enfoque facilitó mucho la obtención de los resultados por lo fácil de implementar y de visualizar las salidas. Además, los recursos utilizados fueron poco significativos comparado con cualquier otra solución que signifique la instalación de módulos como una Soft CPU (por ejemplo, MicroBlaze de Xilinx [37]).

6. Además del problema del punto anterior, se encontró otra limitante del FPGA de la tarjeta de desarrollo, relacionada con el tamaño del diseño completo. Tanto para el diseño del *Correlator* como para el *CorrentropyTor*, es posible implementar solamente diseños que contemplan entradas de 8 bits y 256 muestras. Al ingresar valores superiores, como 1024 muestras, la herramienta entrega un mensaje dando cuenta que se han sobrepasado los límites del dispositivo. En todo caso, cabe destacar que no hubo problema alguno para realizar la síntesis de cualquier configuración de diseño.

3.3.2. Presentación y Análisis de los Resultados del Diseño del *CorrentropyTor*, utilizando entradas sinusoidales

Para generar los resultados, un tipo de entradas que se utilizaron fueron entradas sinusoidales de acuerdo a las ecuaciones 2.5 según se indica en el punto 3.2.4 (INPUT_A). Dichas expresiones nos permiten generar cualquier onda sinusoidal que pueda ser visualizada correctamente en la entrada y obtener salidas también fáciles de visualizar. Los parámetros que estas expresiones utilizan son: $FREQ1$, $FREQ2$, $MAX\_SAMPLES$ y PI. Los parámetros $FREQ1$ y $FREQ2$ nos permiten determinar las diferentes frecuencias, medidas en Hz, a utilizar para cada una de las entradas. $MAX\_SAMPLES$ corresponde al número de muestras (en nuestro caso 256 o 1024) y PI corresponde al valor de la constante π. Todas las entradas son de 16 bits.

La Tabla 3.1 muestra los valores definidos para los parámetros anteriores para cada una de las salidas obtenidas. A esta Tabla se han agregado los valores de sigma (σ) utilizados.

Para determinar un valor de sigma de referencia, se recomienda utilizar Silverman [18]. Se hizo un análisis para cada una de las combinaciones de frecuencia de la Tabla 3.1. Para 256 muestras se obtuvo un ancho del *Kernel* (σ) de Silverman de aproximadamente 0,40. En base a este valor se definieron tres valores para sigma: 0,40 (una vez Silverman), 0,80 (dos veces Silverman) y 2,00 (cinco veces Silverman). Para 1024 muestras se obtuvo un sigma de Silverman de 0,32. En este caso, los tres valores definidos son: 0,32, 0,64 y 1,60, respectivamente.

Los resultados que se muestran a continuación, corresponden a las salidas 4, 5 y 6 de la Tabla 3.1. El resto de las salidas de la tabla se muestran en el Anexo A.8.

La figura 3.9 (generada en MATLAB) muestra un primer resultado (Salida 4 en la Tabla 3.1) obtenido, considerando que ambas entradas tienen la misma frecuencia (5.2 Hz), 16 bits de entrada (con parte real y parte imaginaria), un sigma de 2,00 y 256 muestras.

Las dos señales de la parte superior de la figura 3.9, corresponden a las entradas A y B, respectivamente (iguales en este caso). Cada una de estas entradas consta de una parte real y una parte imaginaria como se indica en la figura 3.9. Estas señales son generadas por los módulos INPUT_A - Data_Input_CORR e INPUT_B - Data_Input_CORR respectivamente (figura 3.8). Estas entradas se imprimen, dentro del módulo *testbench* del diseño (" Correlator_Top_tb.sv", forma parte de los archivos de la figura 3.8), en un archivo el cual es leído desde MATLAB para generar el gráfico de la figura 3.9.

Salida	**Entrada A**		**Entrada B**		**Sigma (σ)**	$MAX\_SAMPLES$
	$FREQ1$ (Hz)	$FREQ2$ (Hz)	$FREQ1$ (Hz)	$FREQ2$ (Hz)		
1	2,6	0,0	2,6	0,0	2,00	256
2	2,6	0,0	2,6	0,0	0,80	256
3	2,6	0,0	2,6	0,0	0,40	256
4	5,2	0,0	5,2	0,0	2,00	256
5	5,2	0,0	5,2	0,0	0,80	256
6	5,2	0,0	5,2	0,0	0,40	256
7	2,6	0,0	5,2	0,0	2,00	256
8	2,6	0,0	5,2	0,0	0,80	256
9	2,6	0,0	5,2	0,0	0,40	256
10	2,6	23,2	2,6	23,2	2,00	256
11	2,6	23,2	2,6	23,2	0,80	256
12	2,6	23,2	2,6	23,2	0,40	256
13	2,6	23,2	5,2	46,4	2,00	256
14	2,6	23,2	5,2	46,4	0,80	256
15	2,6	23,2	5,2	46,4	0,40	256
16	2,6	23,2	5,2	46,4	2,00	1024
17	2,6	23,2	5,2	46,4	0,80	1024
18	2,6	23,2	5,2	46,4	0,40	1024

Tabla 3.1: Definición de Parámetros para mostrar los Resultados de la Correntropía

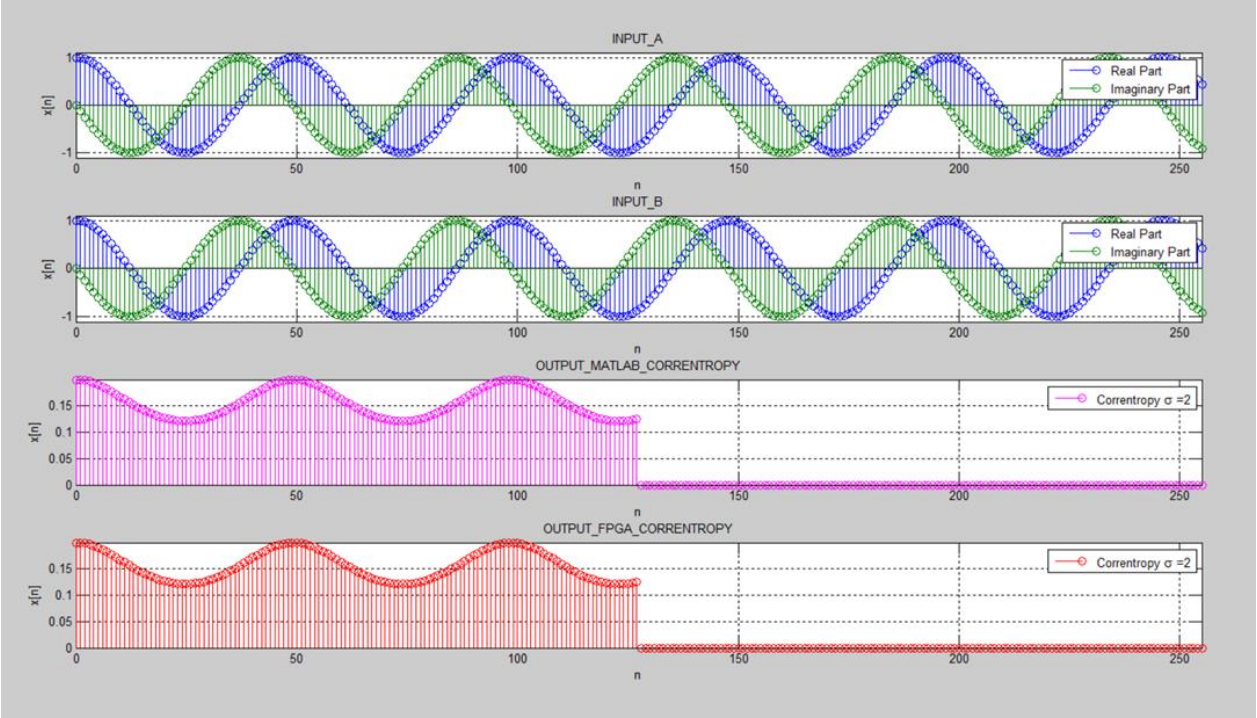

Figura 3.9: Salida 4 del *CorrentropyTor* según Tabla 3.1

La tercera señal de la figura 3.9 corresponde a la salida generada en MATLAB, correspondiente a la expresión de la Correntropía dada en el Anexo A.9. Se muestran solamente los primeros 128 retardos. De acuerdo a [18], se puede constatar que la Correntropía es un escalar positivo.

La cuarta señal de la figura 3.9 es la salida obtenida por nuestro diseño. Esta salida también es grabada en un archivo que luego es leída con MATLAB. Al igual que para la tercera señal, aquí también se muestran los primeros 128 retardos. Como se puede ver, el resultado obtenido corresponde exactamente a lo entregado por MATLAB de acuerdo a la tercera señal de la figura 3.9.

Con el fin de obtener más detalle del resultado entregado en la figura 3.9, la figura 3.10 muestra el mismo resultado pero donde se ha restado el valor medio de la señal obtenida en la figura 3.9. Normalmente, la Correntropía incorpora una media que aumenta cuando se incrementa el valor de sigma.

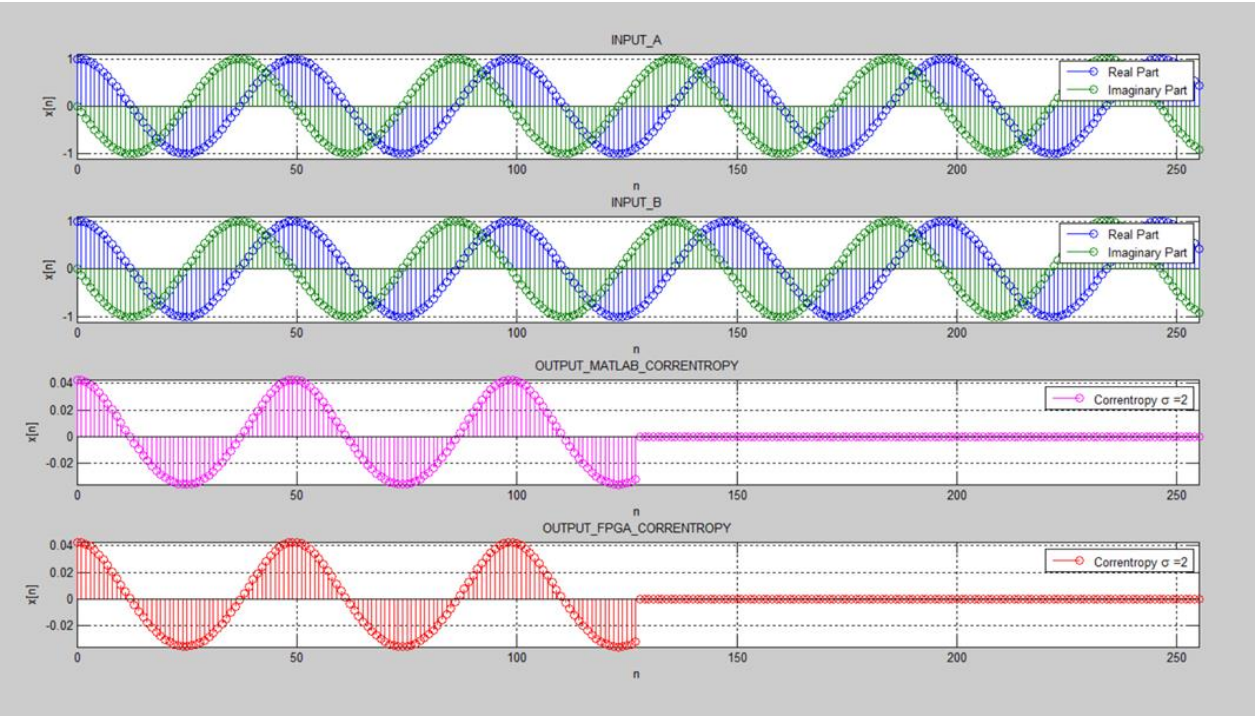

Figura 3.10: Salida 4 del *CorrentropyTor* según Tabla 3.1 con la media descontada

En la figura 3.11 se entrega la magnitud del espectro de frecuencia para la Correntropía (salida FPGA) de la figura 3.9.

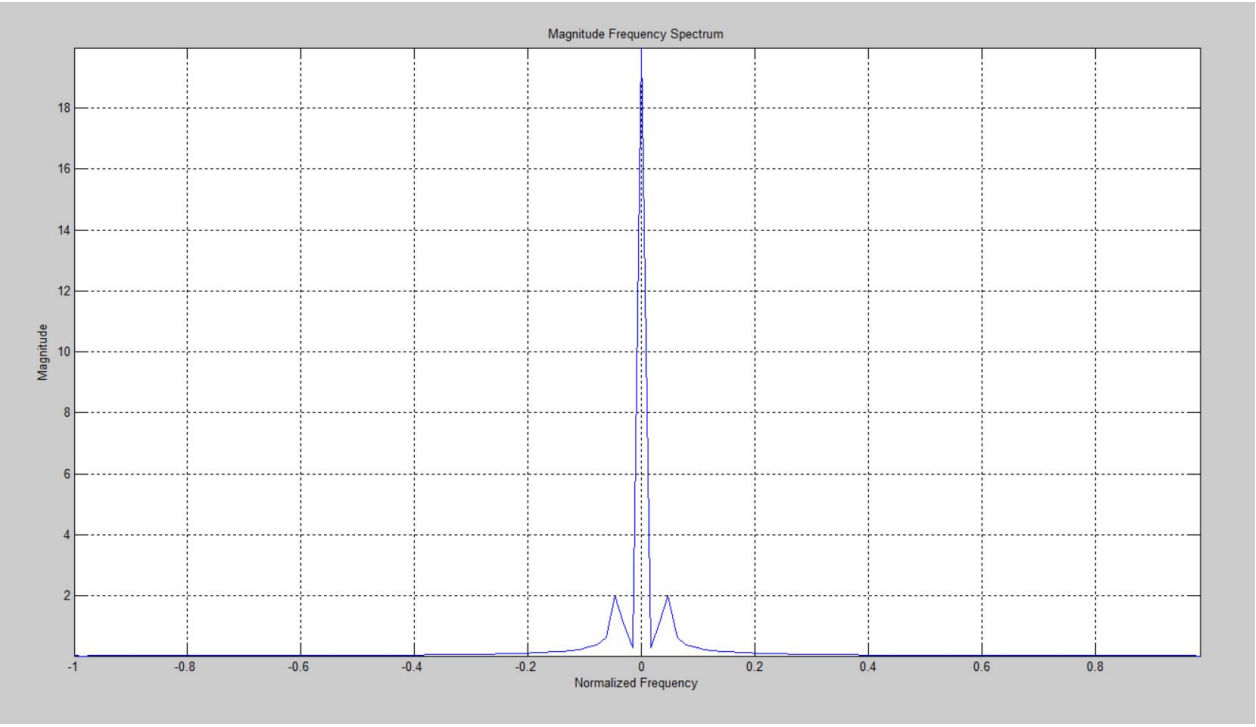

Figura 3.11: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 4, Figura 3.9)

Una de las propiedades de la Correntropía [18] es que para valores altos de sigma, ésta tiende a parecerse a la Correlación. Dado que un valor de 2,00 (cinco veces Silverman para este ejemplo) es un valor alto de sigma, si comparamos las figuras 2.14 y 3.10 vemos que tienen una gran similitud. También podemos comparar las magnitudes de los espectros de frecuencia (figuras 2.15 y 3.11). Vemos que para la Correlación (figura 2.15), el pulso, que indica la frecuencia fundamental, es más ancho que el de la Correntropía (figura 3.11) y, además, en el caso de la Correntropía se visualizan pequeñas harmónicas.

La figura 3.12 corresponde a la Salida 5 de la Tabla 3.1. En este caso el ancho del *Kernel* (sigma) es 0,80 (dos veces Silverman), manteniéndose el resto de los parámetros con respecto a la Salida 4.

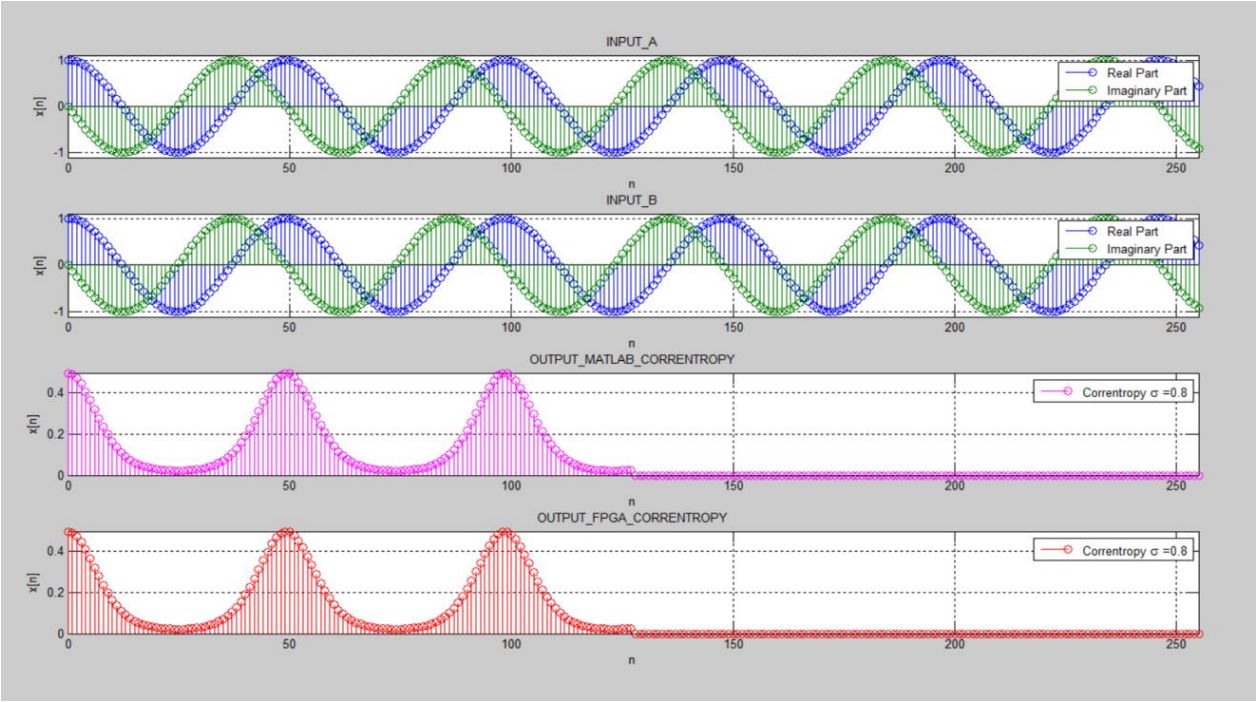

Figura 3.12: Salida 5 del *CorrentropyTor* según Tabla 3.1

Como se puede ver, el resultado obtenido de nuestro diseño corresponde exactamente a lo indicado por MATLAB.

La magnitud del espectro de frecuencia para este caso (Salida 5, Tabla 3.1) se muestra en la figura 3.13. Comparando con el resultado de la figura 3.11, se ve que aparecen más harmónicas y de mayor magnitud.

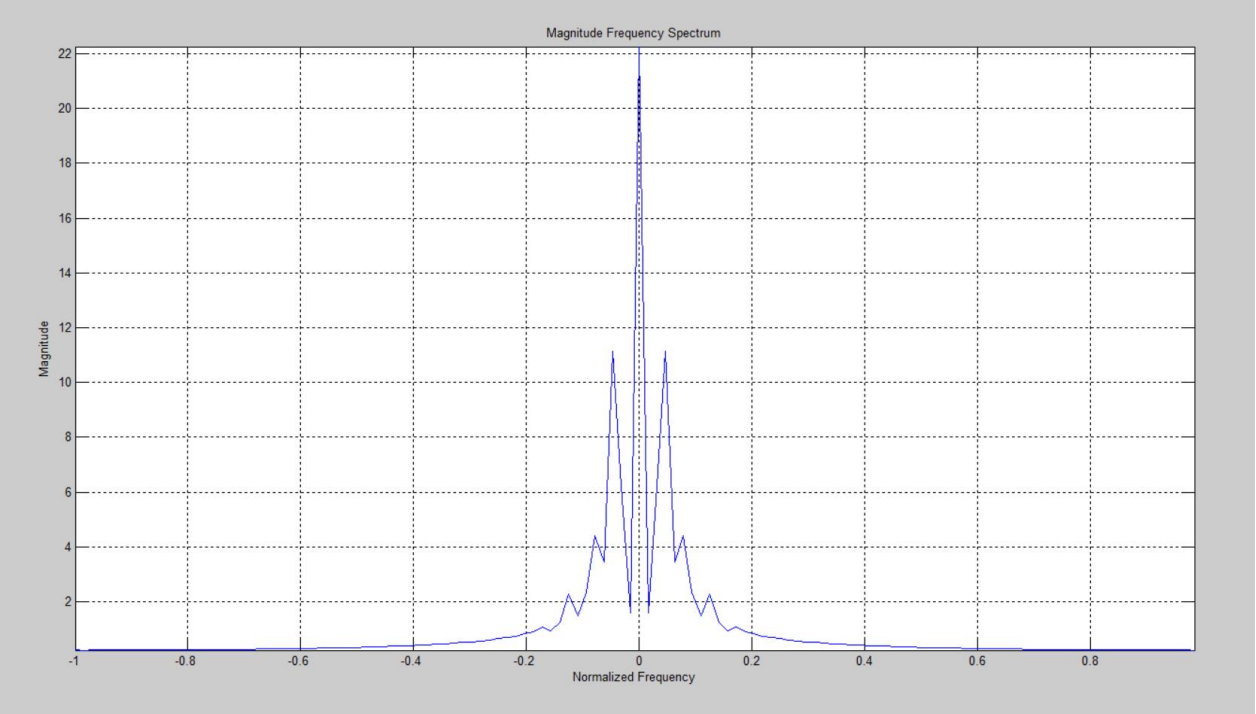

Figura 3.13: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 5, Figura 3.12)

La figura 3.14 muestra el resultado obtenido para la Salida 6 de la Tabla 3.1. En este caso lo único que cambia, nuevamente, es sigma que toma el valor de 0,40 (una vez Silverman). Igualmente, la salida obtenida corresponde exactamente a lo indicado por MATLAB.

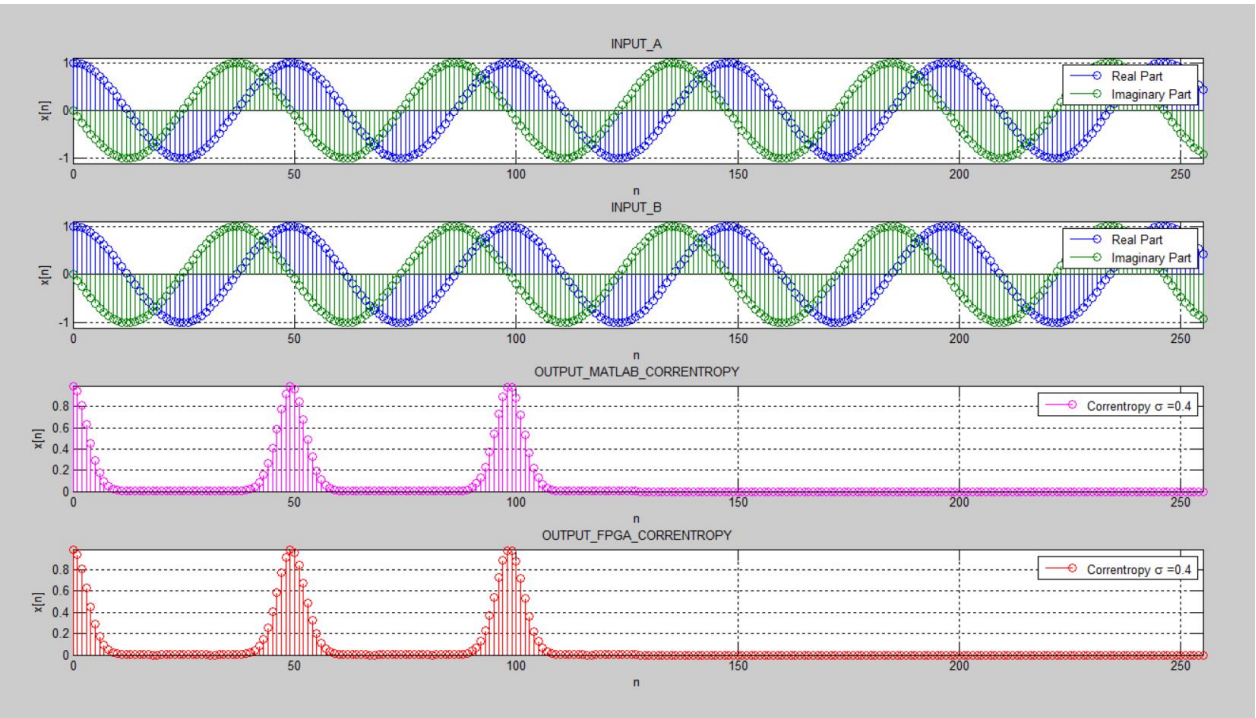

Figura 3.14: Salida 6 del *CorrentropyTor* según Tabla 3.1

La magnitud del espectro de frecuencia para este caso (Salida 6, Tabla 3.1) se muestra en la figura 3.15. Comparando con los resultados de la figura 3.11 y 3.13, se ve que aparecen más harmónicas y de mayor magnitud.

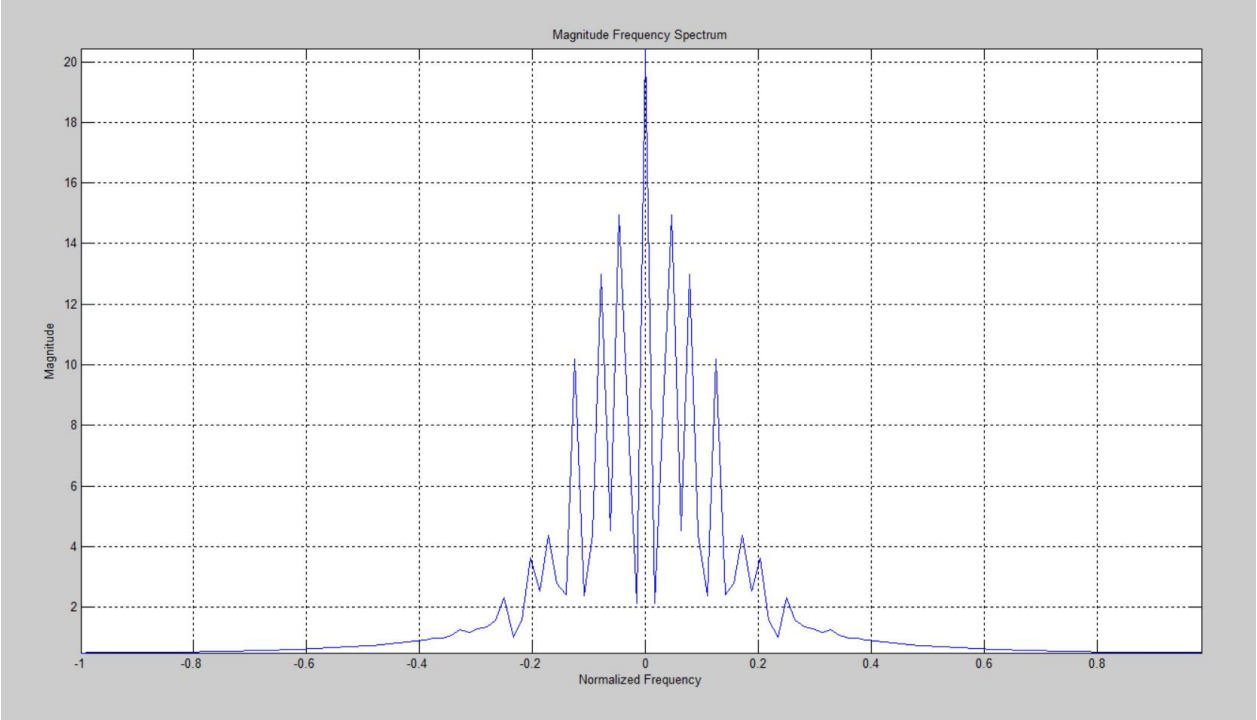

Figura 3.15: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 6, Figura 3.14)

3.3.3. Presentación y Análisis de los Resultados del Diseño del CorrentropyTor, utilizando entradas no sinusoidales

Como entradas no sinusoidales, se utilizaron datos de husos de sueños obtenidos de un electroencefalograma (EEG) y datos de una curva de luz astronómica. Para la descripción de estas señales, véase punto 2.3.

Para ambos casos se obtuvieron los sigmas de Silverman [18] como valores de referencia. Para el primer caso, husos de sueño, se obtuvo un sigma de Silverman de aproximadamente 0,18 y para la curva de luz astronómica, se obtuvo un sigma de Silverman de aproximadamente 0,16. Considerando lo anterior, para el caso de los husos de sueño, se consideraron valores de sigma de 0,90 (cinco veces Silverman), 0.36 (dos veces Silverman) y 0.18 (una vez Silverman). Para el caso de la curva de luz astronómica, se consideraron valores de sigma de 0.80 (cinco veces Silverman), 0.32 (dos veces Silverman) y 0.16 (una vez Silverman).

La figura 3.16 muestra el resultado para una curva de husos de sueños, considerando un $\sigma = 0,90$ (cinco veces Silverman) y se puede apreciar que la salida obtenida del diseño coincide exactamente a lo obtenido con MATLAB.

Con el fin de obtener más detalle del resultado entregado en la figura 3.16, la figura 3.17 muestra el mismo resultado pero donde se ha restado el valor medio de la señal obtenida en

la figura 3.16.

En la figura 3.18 se entrega la magnitud del espectro de frecuencia para la Correntropía (salida FPGA) de la figura 3.16.

En la figura 3.19 se entrega la magnitud del espectro de frecuencia para la Correntropía (salida FPGA) de la figura 3.17.

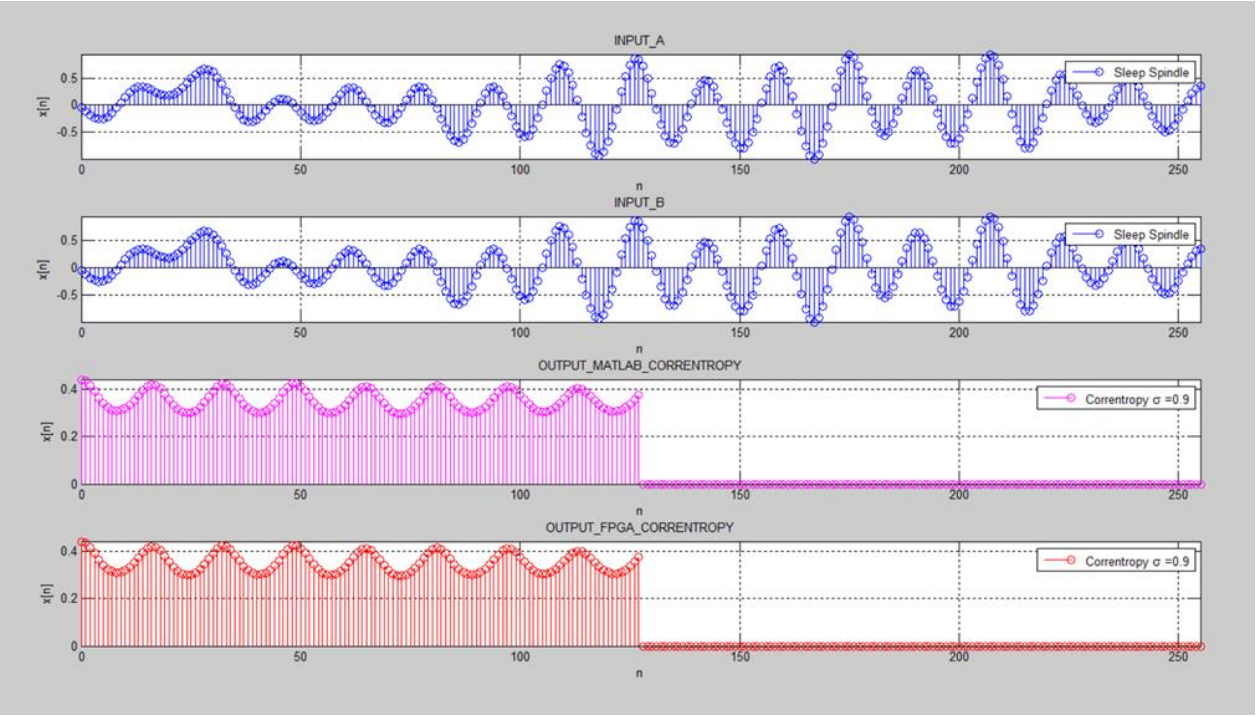

Figura 3.16: Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,90$

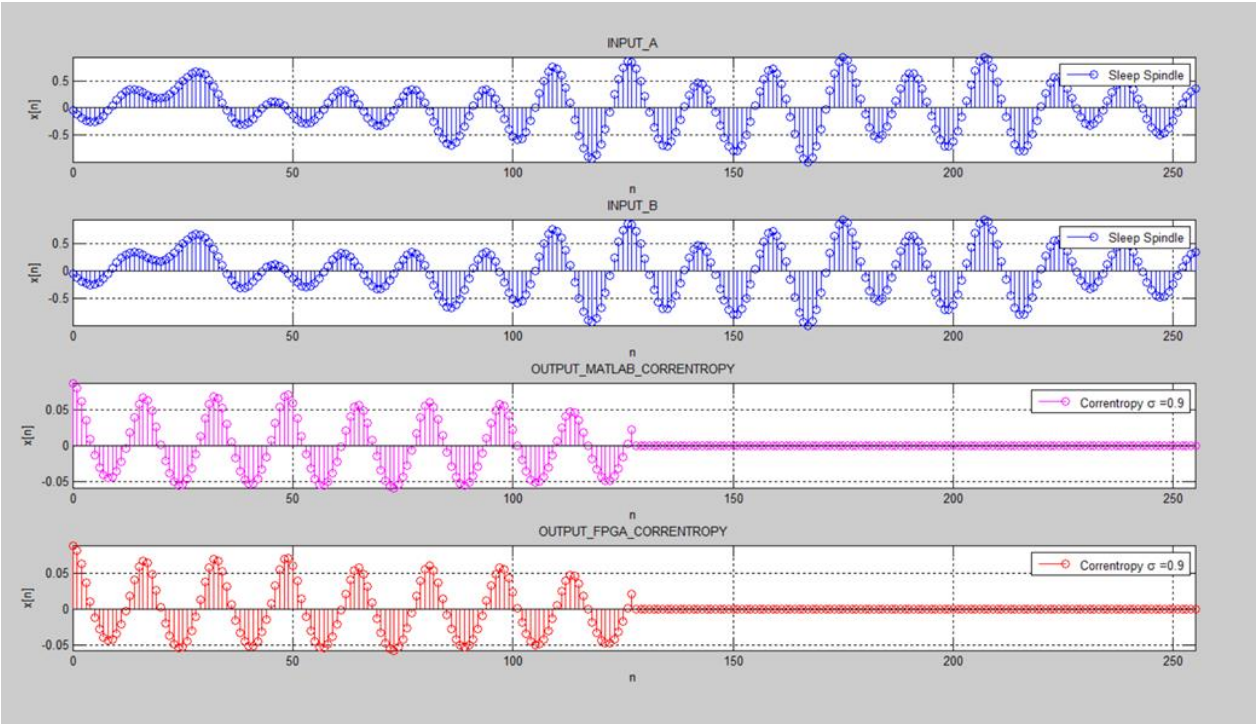

Figura 3.17: Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,90$ y restada la media

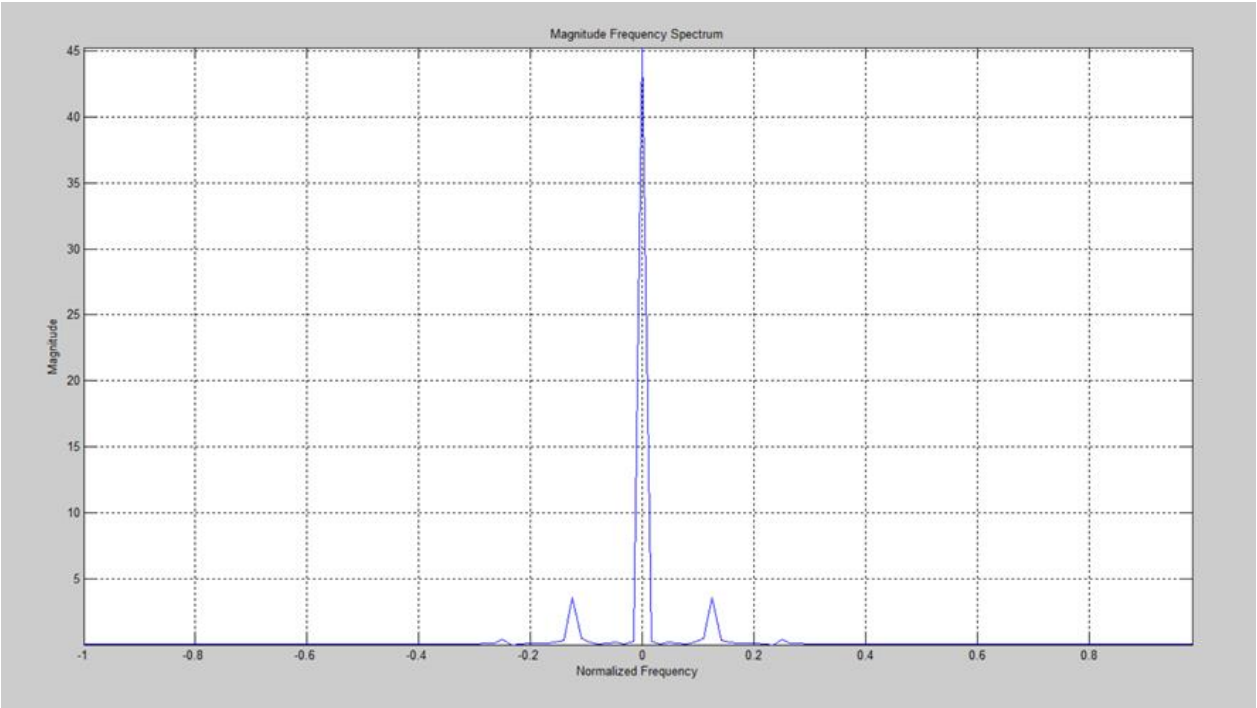

Figura 3.18: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.16)

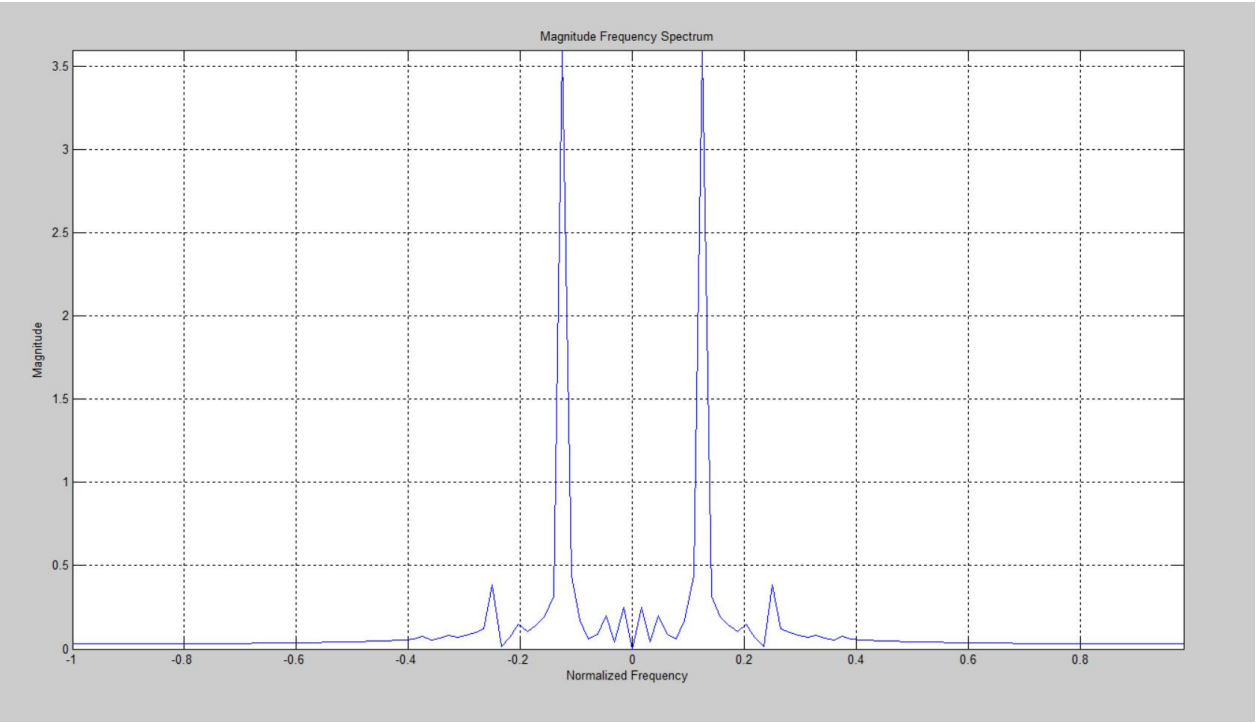

Figura 3.19: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.17)

Como se indicó anteriormente, para un valor alto de sigma, la Correntropía tiende a parecerse a la Correlación. Dado que un valor de 0,90 (cinco veces Silverman para este ejemplo) es un valor alto de sigma, si comparamos las figuras 2.16 y 3.17 vemos que tienen una gran similitud. En relación a las magnitudes de los espectros de frecuencia hay que comparar las figuras 2.17 y 3.19; también tenemos una gran similitud.

La figura 3.20 muestra el resultado para un valor de sigma de 0,36 (dos veces Silverman). Igualmente al ejemplo anterior, la salida entregada por el diseño coincide exactamente con lo obtenido con la aplicación MATLAB. La figura 3.21 muestra la magnitud del espectro de frecuencia para el resultado (salida FPGA) de la figura 3.20.

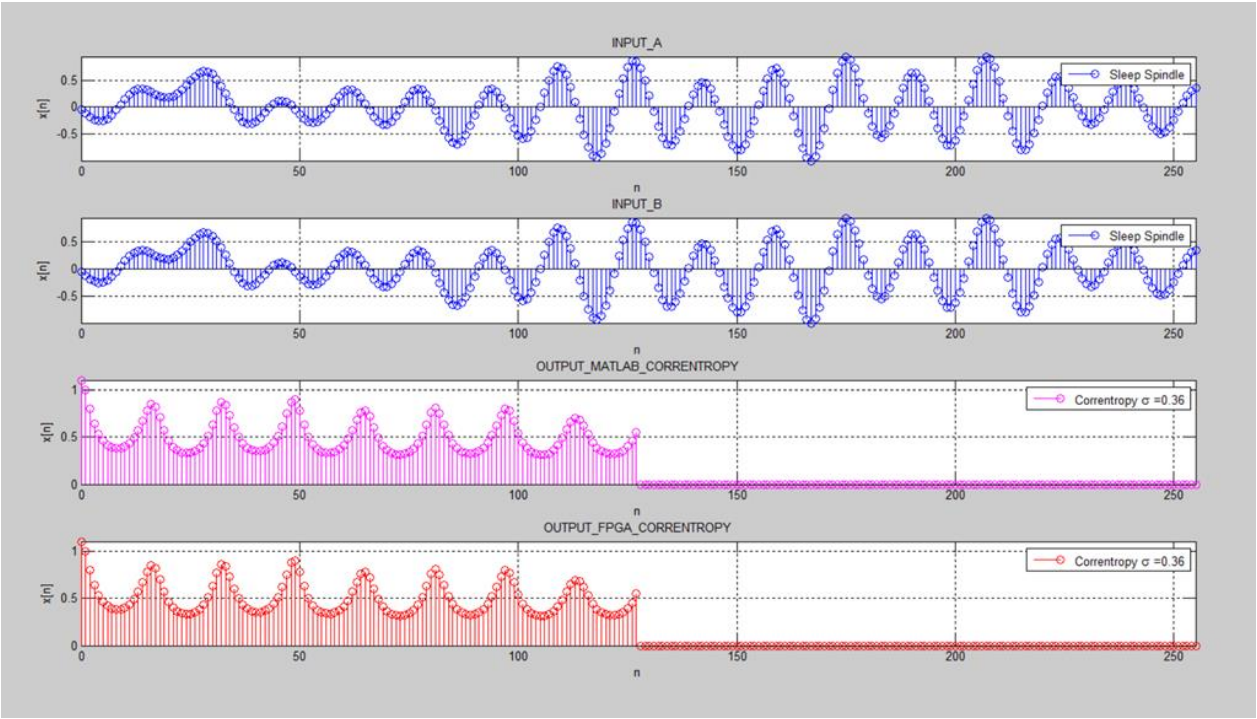

Figura 3.20: Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,36$

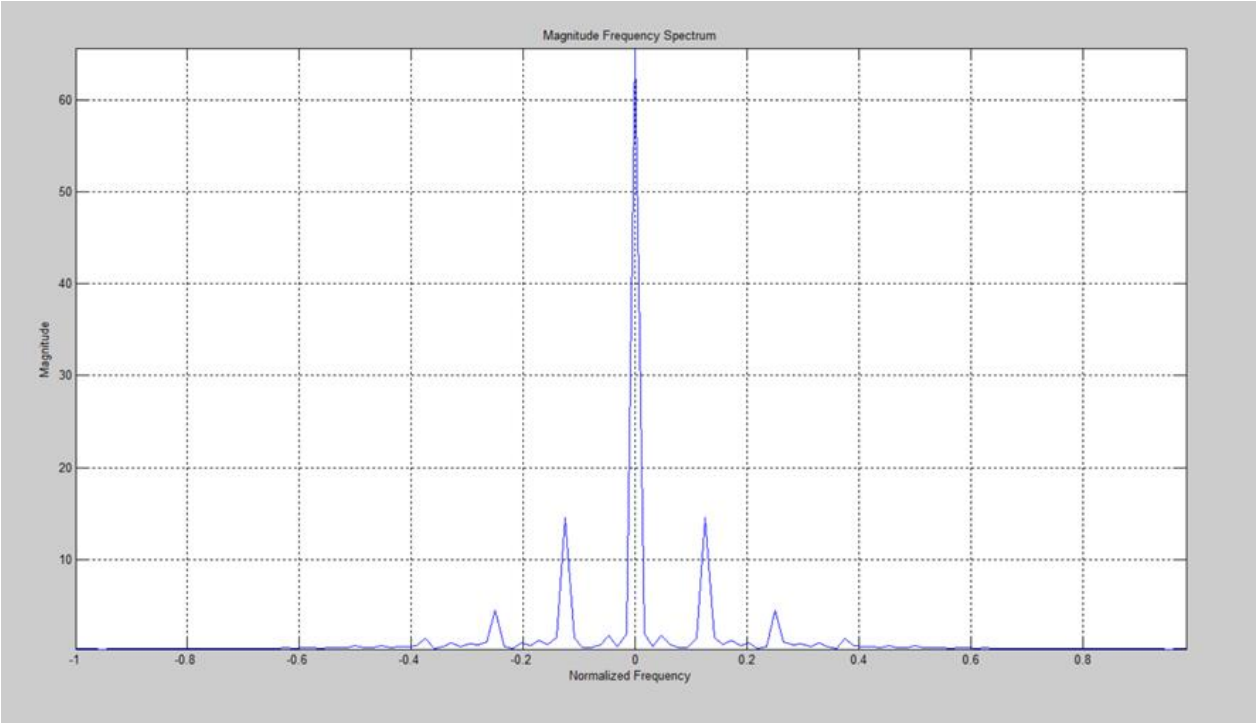

Figura 3.21: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.20)

La figura 3.22 muestra el resultado para un valor de sigma de 0,18 (una vez Silverman). Igualmente al ejemplo anterior, la salida entregada por el diseño coincide exactamente con

lo obtenido con la aplicación MATLAB. La figura 3.23 muestra la magnitud del espectro de frecuencia para el resultado (salida FPGA) de la figura 3.22.

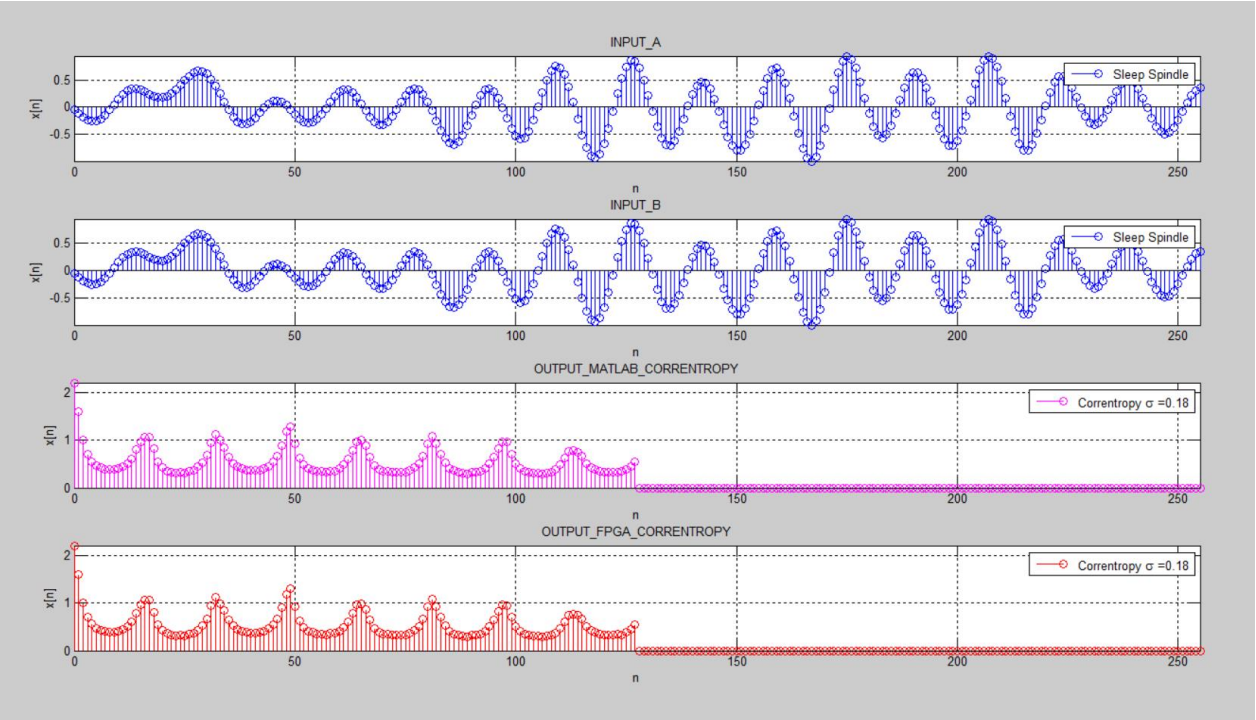

Figura 3.22: Resultado del *CorrentropyTor* para un Huso de Sueño con $\sigma = 0,18$

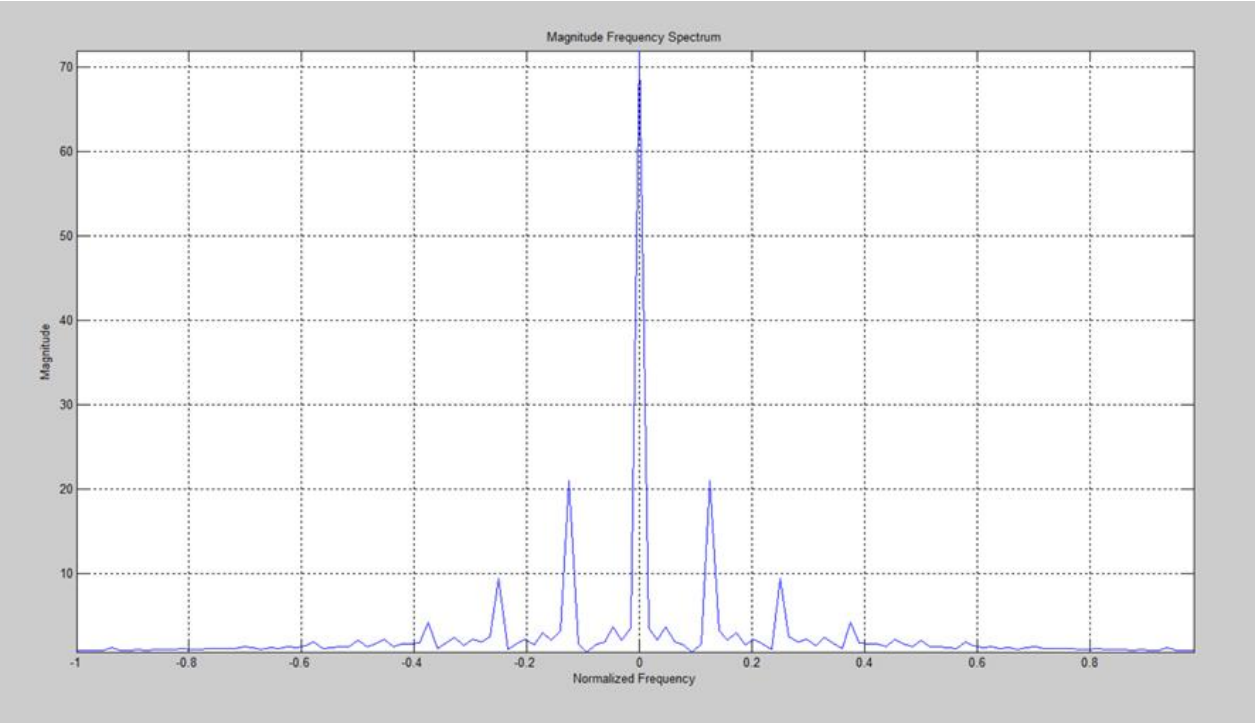

Figura 3.23: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.22)

El segundo ejemplo de entradas no sinusoidales a desarrollar, corresponde a una Curva de Luz Astronómica, descrita en el punto 2.3. Como se indicó anteriormente, esta señal tiene un sigma de Silverman de aproximadamente 0,16. De acuerdo a ello, se definieron tres valores para el ancho del *Kernel*: 0,80 (cinco veces Silverman), 0,32 (dos veces Silverman) y 0,16 (una vez Silverman).

La figura 3.24 muestra el resultado para la curva de luz, considerando un sigma de 0,80 (cinco veces Silverman). Se puede ver que la salida de nuestro diseño (salida FPGA) coincide exactamente a lo indicado por MATLAB.

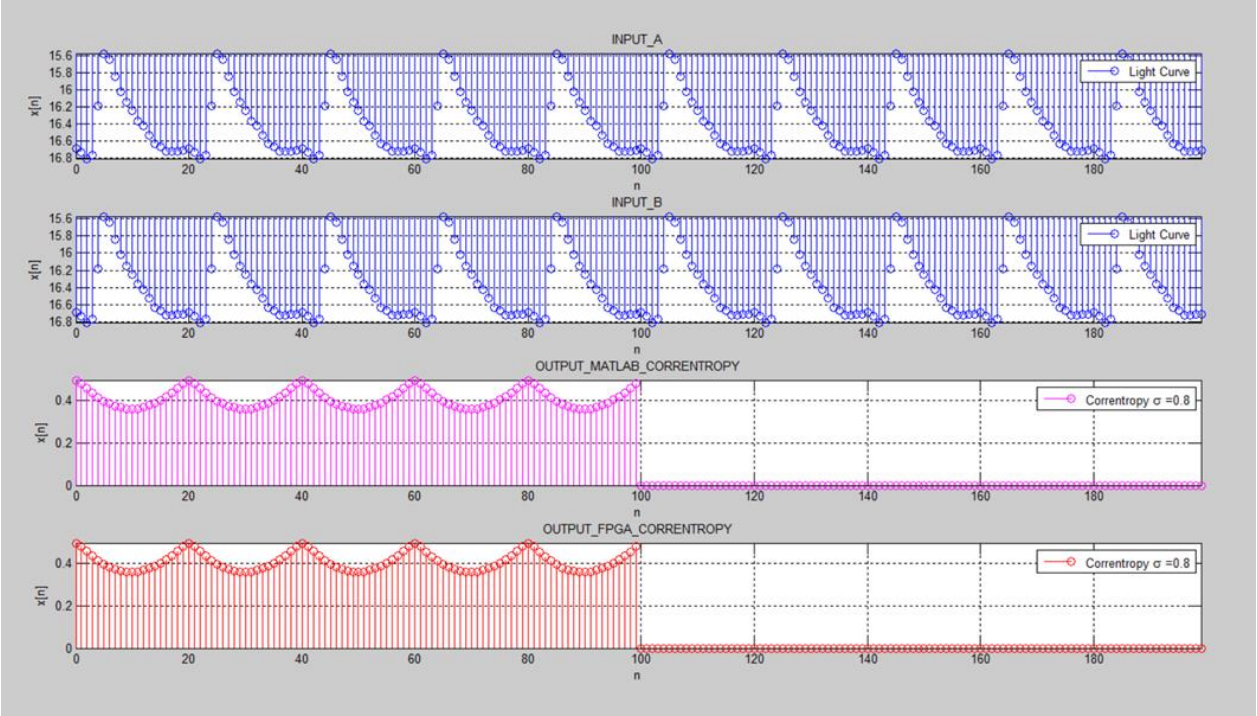

Figura 3.24: Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,80$

La magnitud del espectro de frecuencia para la salida FPGA de la figura 3.24, se muestra en la figura 3.44.

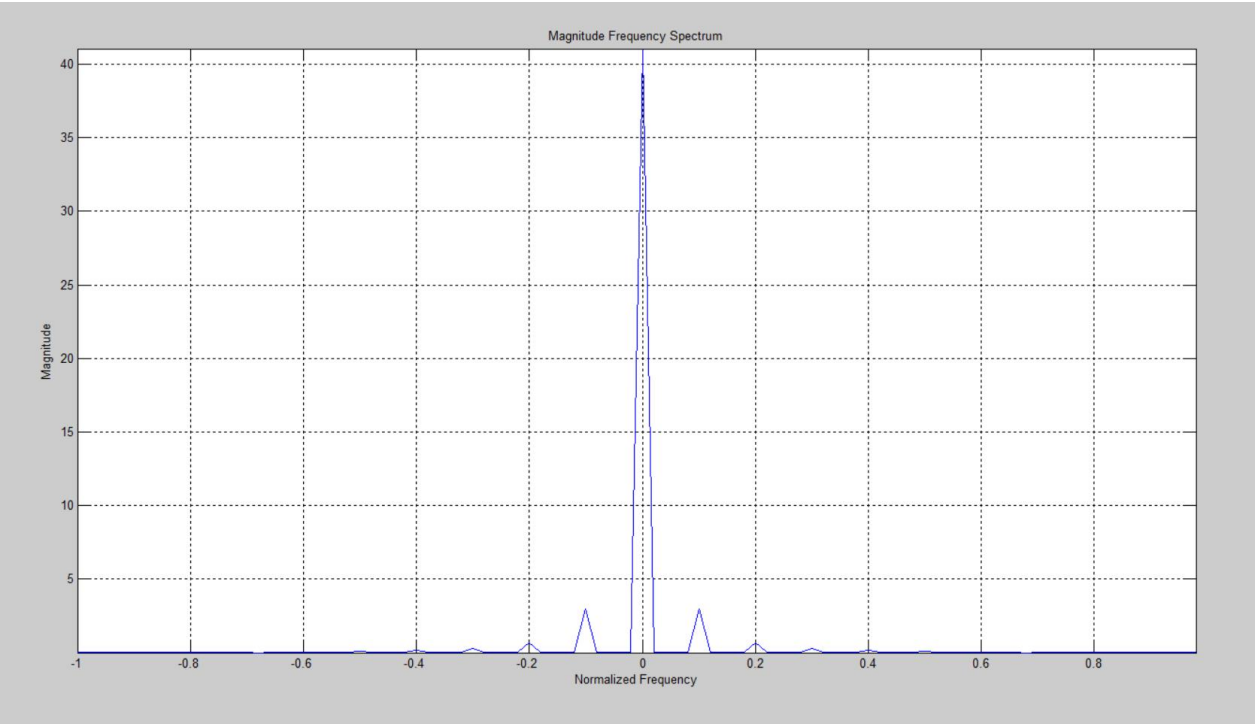

Figura 3.25: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.24)

La figura 3.26 muestra el resultado de la figura 3.24 al cual se ha descontado el valor medio de la correntropía. Su espectro de frecuencia se muestra en la figura 3.27.

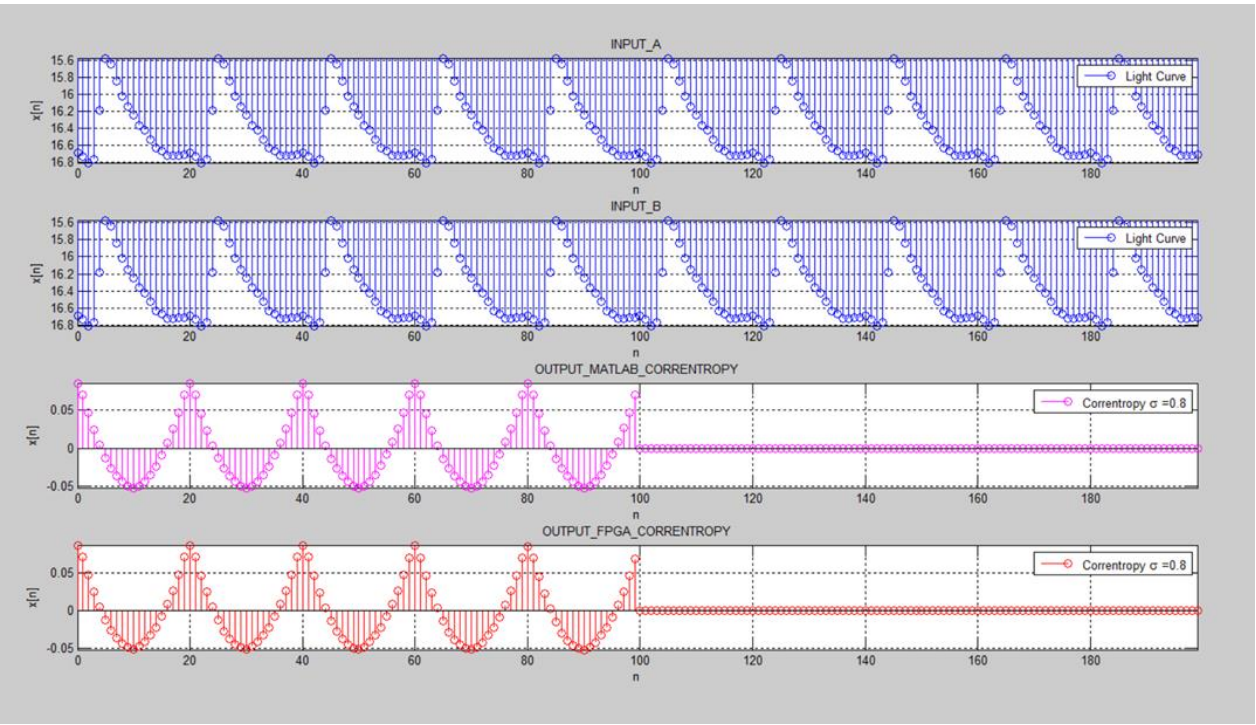

Figura 3.26: Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,80$ y media descontada

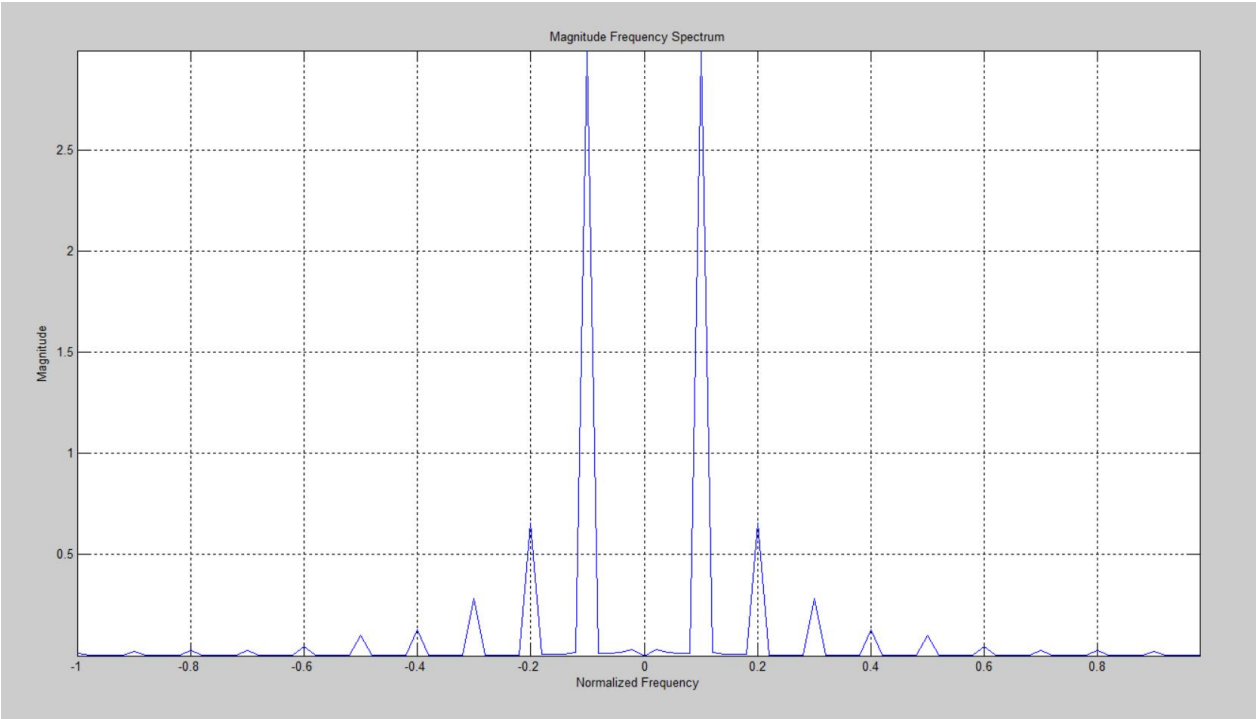

Figura 3.27: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.26)

Dado que un valor de 0,80 (cinco veces Silverman para este ejemplo) es un valor alto de sigma, si comparamos las figuras 2.18 y 3.24 vemos que tienen una gran similitud. En relación a las magnitudes de los espectros de frecuencia hay que comparar las figuras 2.20 y 3.25; también tenemos una gran similitud.

La figura 3.28 muestra el resultado para un valor de sigma de 0,32 (dos veces Silverman). Igualmente al ejemplo anterior, la salida entregada por el diseño coincide exactamente con lo obtenido con la aplicación MATLAB. La figura 3.29 muestra la magnitud del espectro de frecuencia para el resultado (salida FPGA) de la figura 3.28.

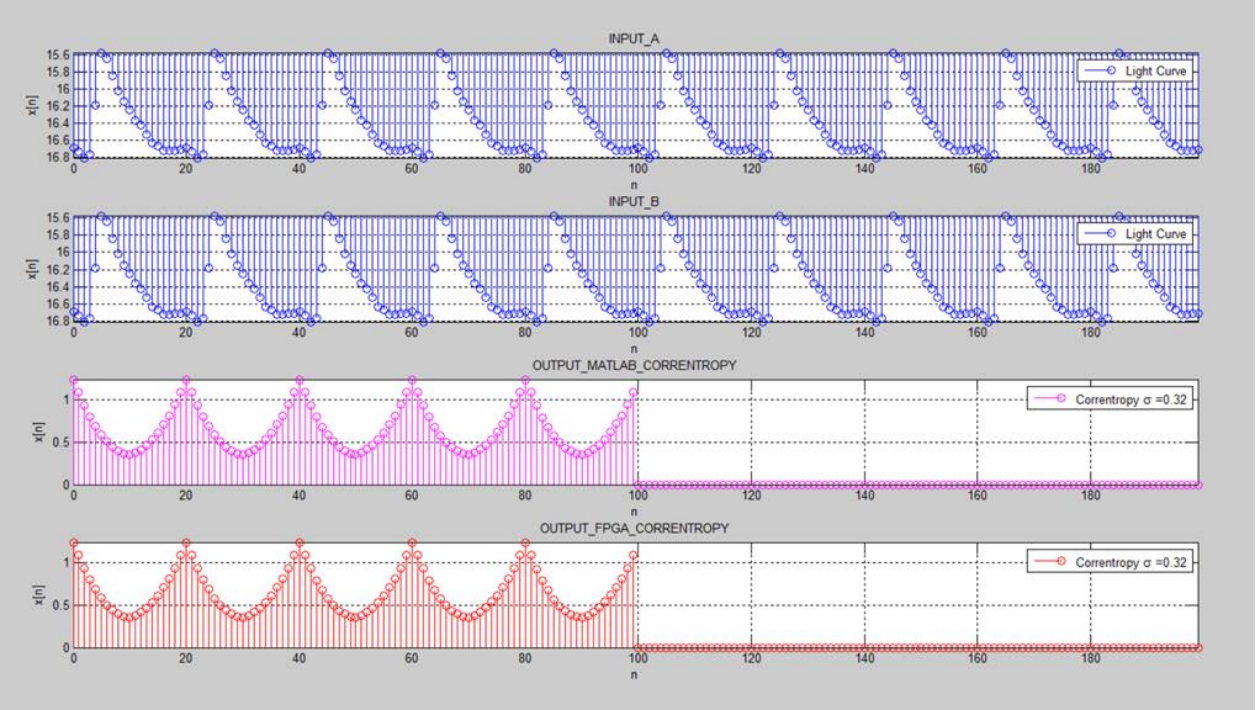

Figura 3.28: Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,32$

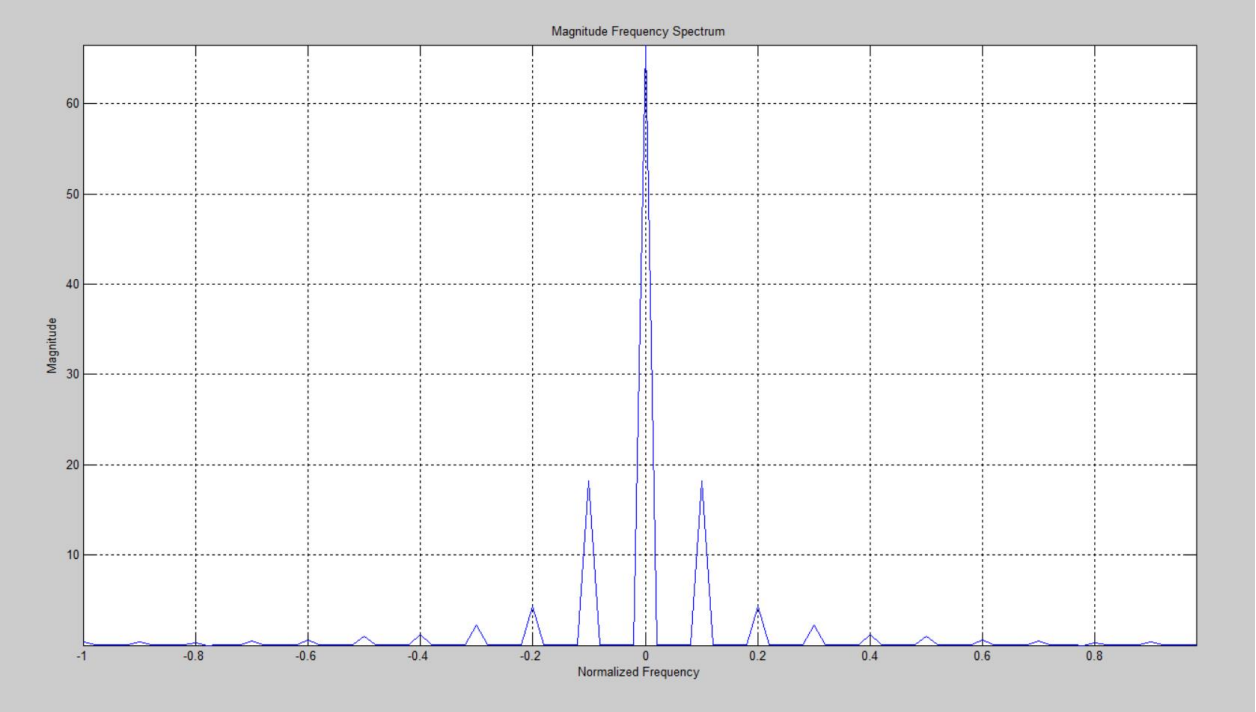

Figura 3.29: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.28)

La figura 3.30 muestra el resultado para un valor de sigma de 0,16 (una vez Silverman). Igualmente al ejemplo anterior, la salida entregada por el diseño coincide exactamente con

lo obtenido con la aplicación MATLAB. La figura 3.31 muestra la magnitud del espectro de frecuencia para el resultado (salida FPGA) de la figura 3.30.

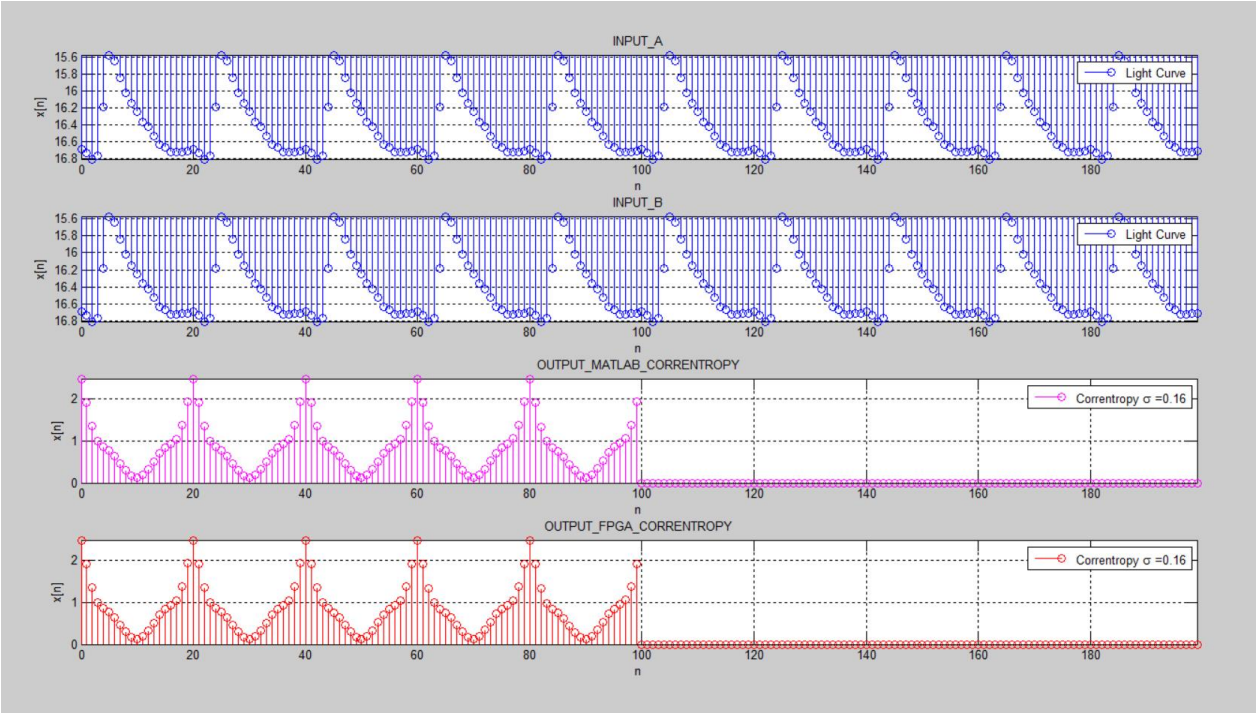

Figura 3.30: Salida del *CorrentropyTor* para Curva de Luz con $\sigma = 0,16$

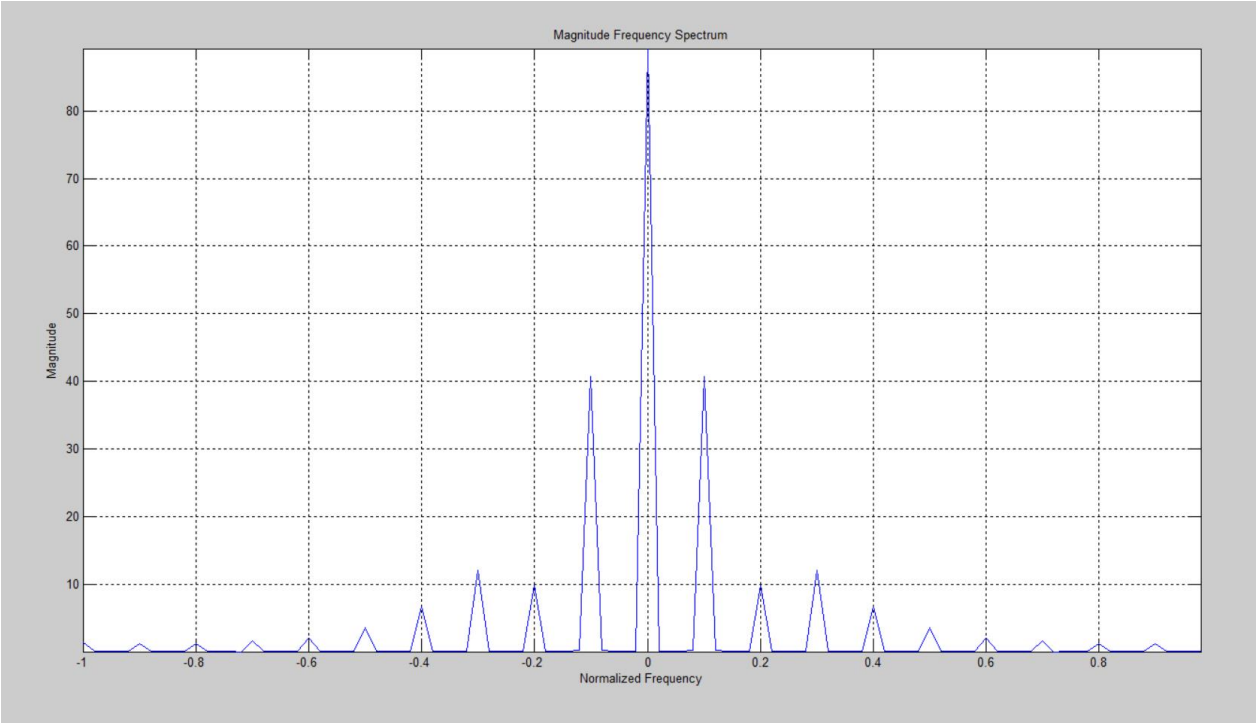

Figura 3.31: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Figura 3.30)

3.4. Resultados de la Implementación del *Correntropy-Tor* en el FPGA

Como se indicó anteriormente, tenemos una limitante en la capacidad del FPGA en la tarjeta de desarrollo Nexsys4. Se pudo simular, sintetizar, implementar y cargar el diseño en la tarjeta de desarrollo, solamente para el caso de representar las entradas en 8 bits y considerando 256 muestras. Para valores mayores en la cantidad de muestras, VIVADO entrega un mensaje indicando que se sobrepasaron los límites del dispositivo. La figura 3.32 muestra el nivel de ocupación del FPGA cuando se implementa un diseño con 16 bits para las entradas y 256 muestras.

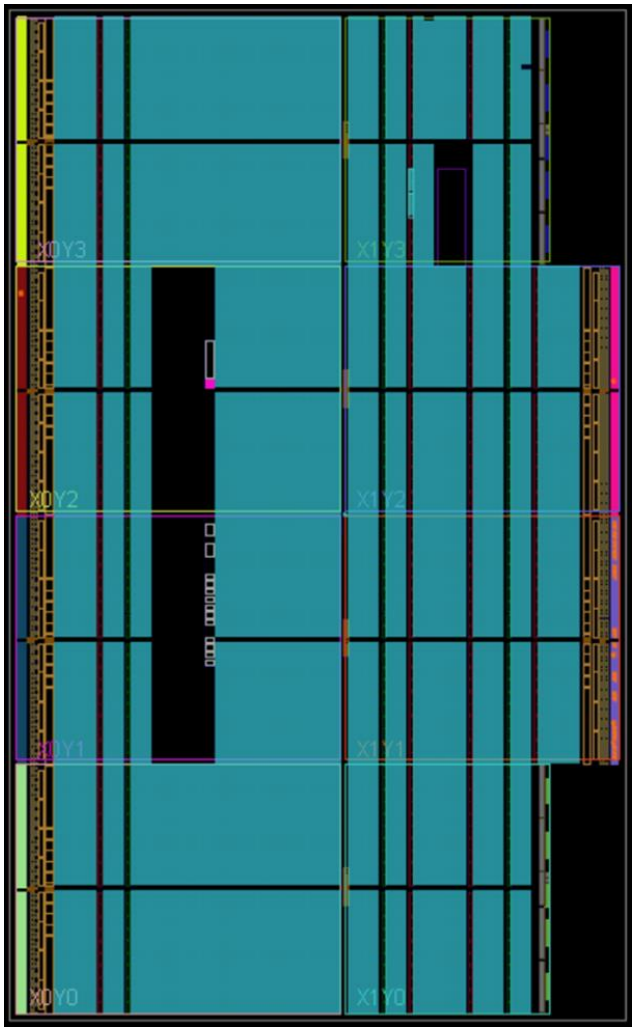

Figura 3.32: Nivel de Ocupación del Diseño del *CorrentropyTor* en el FPGA

Como se puede ver en la figura 3.32, el nivel de ocupación es casi del 100 % (área de color celeste o área más clara: los rectángulos oscuros indican áreas no ocupadas por el FPGA). Por lo menos fue posible verificar, utilizando 16 bits para las entradas y 256 muestras, que el diseño funciona correctamente en la tarjeta de desarrollo, entregando los resultados cuya simulación se muestra en la figura 3.12.

La utilización de recursos como "Flip-Flops"(FF), memorias RAM (BRAM), "Look-Up-Tables"(LUT), etc., se muestra en el lado izquierdo de la figura 3.33 y corresponde a uno de los numerosos informes que entrega la herramienta VIVADO.

El lado derecho de la figura 3.33, muestra la disipación de energía y la temperatura máxima que presentaría el FPGA para el diseño obtenido. Esto es muy importante tener en cuenta dado que es posible que un diseño tenga como resultado una disipación de energía y temperaturas que se salgan de los límites definidos para el FPGA. En ese caso se deben realizar cambios a nivel del diseño, considerando indicaciones dadas por el fabricante del FPGA, para solucionar los problemas anteriores.

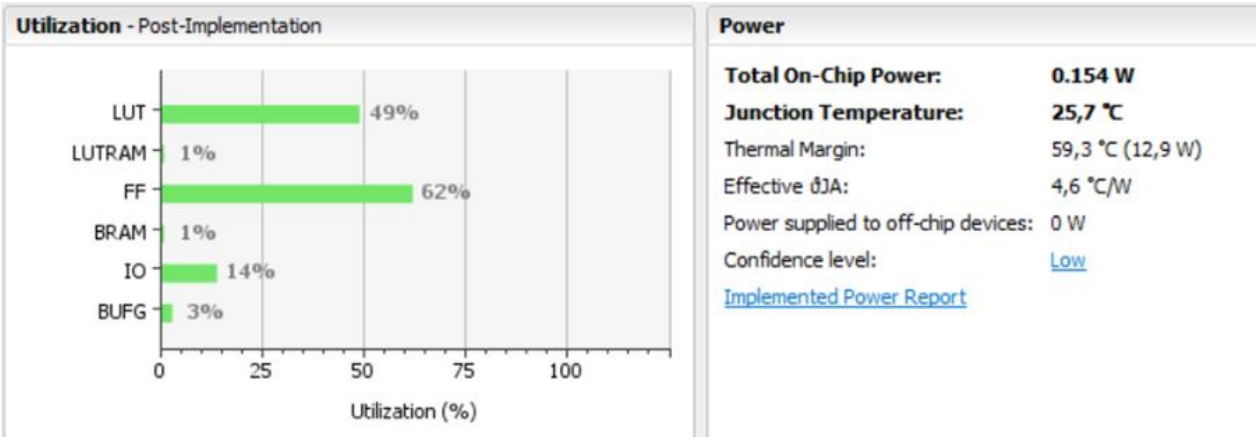

Figura 3.33: Utilización de Recursos y Disipación de Energía del FPGA entregados por VIVADO [29]

Capítulo 4

ANÁLISIS COMPARATIVO DE LOS RESULTADOS OBTENIDOS PARA LA CORRELACIÓN Y LA CORRENTROPÍA CRUZADA IMPLEMENTADAS EN FPGA

A continuación, se hace un análisis comparativo en tiempos de proceso (latencia) entre la Correlación y la Correntropía, considerando lo obtenido como resultado del diseño en comparación a lo entregado por la herramienta MATLAB. Esta herramienta representa el procesamiento utilizando procesadores y la herramienta VIVADO representa la ejecución en un FPGA.

Los parámetros iniciales a considerar para realizar esta comparación, se muestran en la Tabla 4.1. Cabe destacar que la frecuencia de 100 MHz, corresponde a la frecuencia del reloj de la tarjeta de desarrollo Nexys4 que estamos utilizando. Esta tarjeta puede operar con un reloj externo de hasta 450 MHz. Las tarjetas de desarrollo de última generación pueden operar con relojes dentro de las unidades de GHz (Giga Hertz) de frecuencia.

Bits de Entrada	16
Muestras	256 y 1024
Frecuencia	100 MHz (reloj básico Nexys4)
Tipo de Entradas	Señales Sinusoidales
FREQ1	5,2 (ambas entradas)
FREQ2	0,0 (ambas entradas)
CPU	Intel CORE i7-4500U, 1,8 GHz, 8 GB RAM

Tabla 4.1: Parámetros Iniciales Análisis Comparativo

La Tabla 4.2 muestra un cuadro comparativo entre los resultados obtenidos con VIVADO y con la herramienta MATLAB para el caso de la Correlación. Cabe recordar que el diseño de la Correlación para ser implementado en un FPGA, programado en SystemVerilog, fue realizado utilizando la expresión en el dominio de la frecuencia (ecuación 2.2, apartado 2.2.1). La expresión equivalente utilizada con la herramienta MATLAB corresponde a la ecuación 2.6 (apartado 2.3).

De la Tabla 4.2 se puede ver que la latencia obtenida para la implementación en el FPGA de la tarjeta de desarrollo Nexys4, es menor en más de un orden de magnitud en comparación con

CORRELACIÓN		
Muestras	**Latencia (segundos)**	
	VIVADO	**MATLAB**
256	$6,1\,\mu$	$260\,m$
1024	$16,3\,\mu$	$539\,m$

Tabla 4.2: Análisis Comparativo de Latencias para la Correlación

lo obtenido con MATLAB. Esta diferencia podría aumentar significativamente al incrementar la frecuencia del reloj maestro de la tarjeta de desarrollo.

La Tabla 4.3 muestra el resultado de la comparación anterior para el caso de la Correntropía donde se ha incorporado el valor del ancho del *Kernel* (σ).

CORRENTROPÍA						
Muestras	**Latencia (segundos)**					
	VIVADO			**MATLAB**		
	$\sigma = 0,40$	$\sigma = 0,80$	$\sigma = 2,00$	$\sigma = 0,40$	$\sigma = 0,80$	$\sigma = 2,00$
256	$6,11\,m$	$6,11\,m$	$6,11\,m$	$2,07$	$3,63$	$3,45$
1024	$111,90\,m$	$111,90\,m$	$111,90\,m$	$3,28$	$3,92$	$4,90$

Tabla 4.3: Análisis Comparativo de Latencias para la Correntropía

Como se aprecia en la Tabla 4.3, también hay una diferencia en la latencia de al menos un orden de magnitud entre VIVADO y MATLAB, siendo menor en el caso de la implementación en el FPGA. Si se comparan los resultados entre la Correlación (Tabla 4.2) y la Correntropía (Tabla 4.3), se confirma que el cálculo de la Correntropía presenta una mayor latencia, en al menos un orden de magnitud, que el cálculo de la Correlación. Esto era de esperarse dado que la Correntropía incorpora en su expresión el *Kernel* Gaussiano y en el caso de la Correlación, al realizar el cálculo en el dominio de la frecuencia, se logra una menor latencia.

También, cabe hacer notar que, en el caso de VIVADO, se obtienen los mismos valores de latencia, independiente del valor de σ.

Conclusiones y Trabajo Futuro

Conclusiones

Se ha logrado el diseño e implementación en hardware de la Correlación Cruzada y de la Correntropía Cruzada, utilizando un FPGA, cumpliendo a cabalidad con el objetivo general y con los objetivos específicos de esta Tesis.

Para validar el diseño obtenido, se utilizaron diferentes tipos de entrada: entradas sinusoidales que permiten fácilmente probar distintas alternativas, series de tiempo de señales electromagnéticas de Astronomía y eventos de husos de sueño en registros de electroencefalogramas (EEG). En todos esos casos, se logran resultados idénticos a los obtenidos con la herramienta MATLAB.

Para el diseño, se desarrolló una metodología basada en un esquema compuesto por un Sistema Controlado y por un Sistema Controlador. Para este último, se desarrolló una Máquina de Estado Finito. Como herramientas de ayuda, se utilizaron Diagramas en Bloques, Simplificados y Detallados; Diagramas de Flujos, Simplificados y Detallados; y Diagramas MDS. Una vez obtenido el Diagrama MDS, fue bastante simple obtener los programas en SystemVerilog que se requerían para la implementación del diseño en el FPGA.

En el caso de la Correntropía, para la implementación del *Kernel* Gaussiano, se desarrollaron dos alternativas: la primera se basó en una serie de Taylor. La segunda alternativa fue utilizar un módulo IP, basado en el algoritmo CORDIC, disponible en la herramienta VIVADO. Finalmente, dadas las limitaciones de esta última alternativa (descrita en el apartado 3.3.1, punto 2), se decidió utilizar la primera opción considerando que no presentaba ningún tipo de restricción y funcionó correctamente.

Se realizaron medidas de latencia para comparar los tiempos de proceso de la Correlación con la Correntropía. Se constató que, en general, el cálculo de la Correlación toma menos tiempo que el cálculo de la Correntropía en, al menos, un orden de magnitud. Esto se cumple tanto al comparar los tiempos entregados por VIVADO (vinculado al FPGA) como con los tiempos entregados por MATLAB (vinculado a un procesador). También se constató que la latencia en un FPGA es menor en, al menos, un orden de magnitud comparado con la latencia vinculada a un procesador, tanto para la Correlación como para la Correntropía. Esta diferencia puede incrementarse significativamente si aumenta la frecuencia del reloj maestro de la tarjeta de desarrollo del FPGA.

Como se indicó anteriormente, se detectó una clara menor latencia en el caso de la Correlación dado que, por una parte, el proceso mismo de cálculo es más simple que el cálculo de la Correntropía y considerando que esta última incluye un *Kernel* Gaussiano. Por otra parte, al implementar la Correlación Cruzada utilizando la solución en el dominio de la frecuencia, se utilizó un módulo IP que calcula directamente la transformada de Fourier lo cual influye significativamente en la menor latencia obtenida.

Para la representación y operación de los datos en el diseño, se optó por utilizar Punto-Fijo en lugar de una representación en Punto-Flotante. Esta decisión se tomó considerando las recomendaciones de expertos que indican que trabajar con Punto-Flotante en un diseño con FPGA, implica el uso de demasiados recursos del dispositivo.

Trabajo Futuro

Dado que los FPGA han evolucionado enormemente, como trabajo a futuro, sería interesante desarrollar un diseño considerando una representación y operación de la información en Punto-Flotante. Es probable que se logre un diseño más simple y, quizás, más rápido, aun cuando signifique mayor uso de recursos del FPGA.

Para el caso de la Correntropía se podría desarrollar un módulo IP que calcule directamente la función exponencial del *Kernel* Gaussiano. Siendo aun más ambicioso, considerando el trabajo desarrollado en esta Tesis, se podría desarrollar un módulo IP que calcule directamente la Correntropía Cruzada entre dos entradas discretas y aleatorias.

Otro trabajo a futuro tiene que ver con el funcionamiento en tiempo real. Es decir, leer los datos en tiempo real y entregar los resultados también en tiempo real. Para ello es necesario desarrollar interfaces con el medio externo, tanto para la entrada como para la salida, que permitan esta operación. También sería necesario contar con una tarjeta de desarrollo que cuente con un FPGA de mayor capacidad. Actualmente, el diseño ya tiene incorporado un módulo *buffer* de entrada (FIFO_BUFFER) que fue pensado para cubrir dicho modo de funcionamiento.

Glosario

ASIC: Circuito Integrado de Aplicación Específica (*Application Specific Integrated Circuit*)

ASM: Máquina de Estado Algorítmica (*Algorithmic State Machine*)

BRAM: Bloque de RAM (Block RAM)

CLB: Bloque Lógico Configurable (*Configurable Logic Block*)

CORDIC: Algoritmo de Volder (*COordinate Rotation DIgital Computer*)

CPLD: Dispositivos Lógicos Programables Complejos (*Complex Programmable Logic Device*)

DCM: Administrador Reloj Digital (*Digital Clock Manager*)

DFT: Transformada Discreta de Fourier (*Discrete Fourier Transform*)

DIF: Algoritmo de Descomposición en las Muestras de Frecuencia para el Cálculo del FFT (*Decimation-In-Frequency*)

DIT: Algoritmo de Descomposición en las Muestras de Tiempo para el Cálculo del FFT (*Decimation-In-Time*)

EEG: Electroencefalograma (*Electroencephalogram*)

FFT: Transformada Rápida de Fourier (*Fast Fourier Transform*)

FIFO: El Primero que Entra es el Primero que Sale (*First-Input First-Output*)

FPGA: Arreglo de Compuertas Programable por el Usuario (*Field Programmable Gate Array*)

FSM: Máquina de Estado Finito (*Finite State Machine*)

HDL: Lenguaje de Descripción de Hardware (*Hardware Description Language*)

IFFT: Transformada Rápida de Fourier Inversa (*Inverse Fast Fourier Transform*)

IOB: Bloques de Entrada/Salida (*Input/Output Blocks*)

IP: Propiedad Intelectual (*Intellectual Property*)

LSI: Integración de Gran Escala (*Large Scale Integration*)

LUT: Tabla de Búsqueda (*Look-Up Table*)

MDS: Diagrama de Estado Documentado con Nemónicos (*Mnemonic Documented State diagram*)

MSI: Integración de Mediana Escala (*Medium Scale Integration*)

PLA: Arreglos Lógicos Programables (*Programmable Logic Array*)

RTL: Diseño digital a Nivel de Transferencia entre Registros (*Register Transfer Level*)

SSI: Integración de Pequeña Escala (*Small Scale Integration*)

UVLSI: Integración de Ultra Gran Escala (*Ultra Very Large Scale Integration*)

VHDL: Lenguaje de Descripción de Hardware para VHSIC (*Very High Speed Integrated Circuit*)

VLSI: Integración de Muy Gran Escala (*Very Large Scale Integration*)

Bibliografía

[1] Peter J. Ashenden. *Digital Design: An Embedded System Approach Using VERILOG.* MORGAN KAUFMANN, Massachusetts, USA, first edition, 2008.

[2] M. Becvar and P. Stukjunger. *Fixed-Point Arithmetic in FPGA.* Number 2 in Vol. 5, Num. 2. Acta Polytechnica, Czech Technical University in Prague, USA, 2005.

[3] Luis F. Chaparro. *Signal and Systems Using MATLAB®.* Academic Press, Elsevier Inc., San Diego, USA, second edition, 2015.

[4] Michael D. Ciletti. *Advanced Digital Design with the Verilog HDL.* Pearson Education Inc., Pearson Prentice Hall, New Jersey, USA, first edition, 2003.

[5] P. Cooke, J. Fowers, G. Stitt, and Hunt L. *A comparison of correntropy-based feature tracking on FPGAs and GPUs.* Architectures and Processors. IEEE 24th International Conference on Application-Specific Systems, Washington DC, USA, 2013.

[6] Patrick Cooke, Jeremy Fowers, Greg Brown, and Greg Stitt. *A tradeoff analysis of FPGAs, GPUs, and multicores for sliding-window applications.* 8, 1, Article 2. ACM Trans. Reconfig. Technol. Syst, USA, February 2015.

[7] J. W. Cooley and J. W. Tukey. *An Algorithm for the Machine Calculation of Complex Fourier Series.* Number 90 in Vol. 19. Mathematics of Computation, USA, 1965.

[8] L. Deng, C. Chakrabarti, N. Pitsianis, and X. Sun. *Automated Optimization of Look-up table Implementation for Function Evaluation on FPGAs.* Proceedings of SPIE, 2009.

[9] Digilent. *Nexys4™ FPGA Board Reference Manual, Rev. B.* Digilent, 19 November 2013.

[10] Doulos. *SystemVerilog Golden Reference Guide, Version 6.0.* Doulos, 2016.

[11] William I. Fletcher. *An Engineering Approach to Digital Design.* Pearson, Prentice Hall, USA, first edition, 1980.

[12] Robert J. Francis. *A tutorial on logic synthesis for lookup-table based FPGAs.* Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE Computer Society, Los Alamitos, CA, USA, 1992.

[13] Brian H. Hahn and Daniel T. Valentine. *Essential MATLAB for Engineers and Scientists.* Academic Press, San Diego, USA, sixth edition, 2017.

[14] P. Huijse, P. Estévez, P. Zegers, J. Principe, and P. Protopapas. *Period Estimation in Astronomical Time Series Using Slotted Correntropy.* Number 18 in IEEE Signal Processing Letters. IEEE, USA, 2011.

[15] E. C. Ifeachor and B. W. Jervis. *Digital Signal Processing: A Practical Approach.*

Addison-Wesley, New York, USA, first edition, 1998.

[16] Hubert Kaeslin. *Top-Down Digital VLSI Design – From Architectures to Gate-Lavel Circuits and FPGAs.* Elsevier, ETH Zurich, Switzerland, first edition, 2015.

[17] B. Lee and N. Burgess. *Some Results on Taylor-Series Function Approximation on FPGA.* Vol. 2. The Thirty-Seventh Asilomar Conference on Signals, Systems and Computer, IEEE Conference Publications, USA, 2003.

[18] Weifeng Liu, P.P. Pokharel, and José C. Principe. *Correntropy: Properties and Applications in Non-Gaussian Signal Processing.* Number 55 in IEEE Transaction on Signal Processing. IEEE, USA, 2007.

[19] M. Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals.* Pearson, Prentice Hall, USA, fourth edition, 2008.

[20] José C. Príncipe. *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives.* Springer, USA, first edition, 2010.

[21] I. Santamaría, P. Pokharel, and José C. Principe. *Generalized Correlation Function: Definition, Properties, and Application to Blind Equalization.* Number 54 in IEEE Transaction on Signal Processing. IEEE, USA, 2006.

[22] Rahman Shaik and Ateek Ur. *Hardware Implementation of the Exponential Function Using Taylor Series and Linear Interpolationing, LTH, Lund University.* Master Thesis, Department of Electrical and Information Technology, Faculty of Engineering, LTH, Lund University, 2014.

[23] Claude. E. Shannon. *A Mathematical Theory of Communication.* Bell System Technical Journal. Bell System Technical Journal, USA, 1948.

[24] IEEE Standars. *IEEE Std. 754-2008, IEEE Standard for Floating-Point Arithmetic.* IEEE, 2008.

[25] IEEE Standars. *IEEE Std 1800TM – 2012: Unified Hardware Design, Specification and Verification Languaje.* IEEE, 2013.

[26] S. Verslype, E. Blomme, T. Cool, R. De Craemer, F. Loret, J. Peuteman, and J. Vandenbussche. *Cross-correlation based ultrasonic multi-channel quality control using a Virtex 5 SX50T board.* Electronics-ET, Sozopol, Bulgaria, 2009.

[27] J. Volder. *The CORDIC Trigonometric Computing Technique.* Vol EC-8. IRE Trans. Electronic Computing, USA, Sept. 1959.

[28] John F. Wakerly. *Digital Design: Principles and Practices.* Pearson, Prentice Hall, USA, fourth edition, 2006.

[29] Xilinx. *Introduction to FPGA Design with Vivado High Level Design, (UG998).* Xilinx, 2013.

[30] Xilinx. *7 Series FPGAs Configuration User Guide (UG470).* Xilinx, 2015.

[31] Xilinx. *Complex Multiplier v6.0, LogiCORE IP Product Guide, Vivado Design Suite, (PG104).* Xilinx, 2015.

[32] Xilinx. *Fast Fourier Transform v9.0, LogiCORE IP Product Guide, Vivado Design Suite, (PG109).* Xilinx, 2015.

[33] Xilinx. *Vivado Design Suite User Guide: Getting Started (UG910)*. Xilinx, 2015.

[34] Xilinx. *Vivado Design Suite User Guide: Logic Simulation (UG900)*. Xilinx, 2015.

[35] Xilinx. *CORDIC v6.0, LogiCORE IP Product Guide, Vivado Design Suite, (PG105)*. Xilinx, 2016.

[36] Xilinx. *Vivado Design Suite User Guide: Programming and Debugging (UG908)*. Xilinx, 2016.

[37] Xilinx. *MicroBlaze Debug Module (MDM) v3.2 LogiCORE IP Product Guide, (PG115)*. Xilinx, 2017.

[38] J. Xu and J. Principe. *A Pitch Detector Based on Generalized Correlation Function*. Number 16 in IEEE Transaction on Audio, Speech and Language Processing. IEEE, USA, 8 November 2008.

Anexos

Anexo A.1: Tarjeta de Desarrollo Nexys4^{TM} de DIGILENT

Se entregan las especificaciones técnicas básicas para la tarjeta de desarrollo Nexys4 de DIGILENT. Para una información más detallada, véase `www.digilentinc.com`.

DIGILENT
BEYOND THEORY

1300 Henley Court
Pullman, WA 99163
509.334.6306
www.digilentinc.com

Nexys4™ FPGA Board Reference Manual

Nexys4 rev. B; Revised September 6, 2013

Overview

The Nexys4 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys4 can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, speaker amplifier and lots of I/O devices allow the Nexys4 to be used for a wide range of designs without needing any other components.

The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs. Artix-7 100T features include:

- 15,850 logic slices, each with four 6-input LUTs and 8 flip-flops
- 4,860 Kbits of fast block RAM
- Six clock management tiles, each with phase-locked loop (PLL)
- 240 DSP slices
- Internal clock speeds exceeding 450MHz
- On-chip analog-to-digital converter (XADC)

The Nexys4 also offers an improved collection of ports and peripherals, including:

- 16 user switches
- USB-UART Bridge
- 12-bit VGA output
- 3-axis accelerometer
- 16Mbyte CellularRAM
- Pmod for XADC signals
- 16 user LEDs
- Two tri-color LEDs
- PWM audio output
- Temperature sensor
- Serial Flash
- Digilent Adept USB port for programming and data
- Two 4-digit 7-segment displays
- Micro SD card connector
- PDM microphone
- 10/100 Ethernet PHY
- Four Pmod ports
- USB HID Host for mice, keyboards and memory sticks

The Nexys4 is compatible with Xilinx's new high-performance Vivado * Design Suite as well as the ISE toolset, which includes ChipScope and EDK. Xilinx offers free "Webpack" versions of these toolsets, so designs can be implemented for no additional cost.

The Nexys4 is compatible with Xilinx's new high-performance Vivado® Design Suite as well as the ISE toolset, which includes ChipScope and EDK. Xilinx offers free "Webpack" versions of these toolsets, so designs can be implemented at no additional cost.

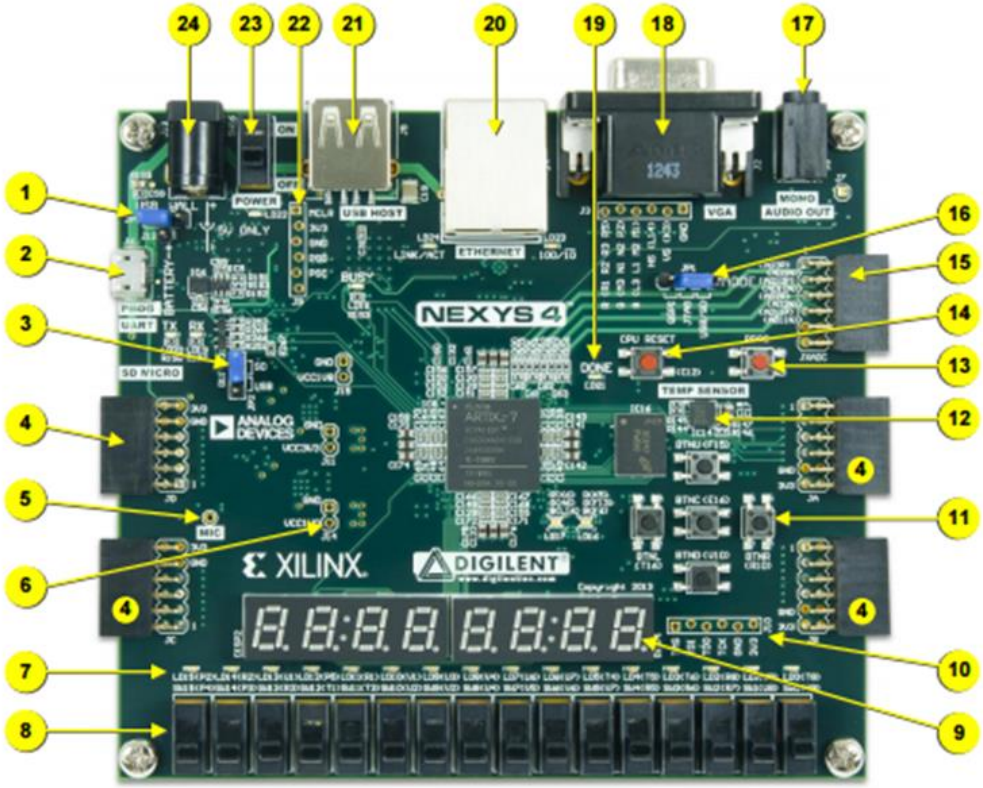

Figure 1. Nexys4 board features

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod connector (XADC)
4	Pmod connector(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

Anexo A.2: Detalle Entradas/Salidas Módulo: "Fast Fourier Transform LogiCORE IP Xilinx v9.0"

Especificaciones de las entradas y salidas del módulo IP. Tabla 2.1 del documento dado en [32].

Table 2-1: **Core Signal Pinout**

Name	Direction	Optional	Description
aclk	Input	No	Rising-edge clock.
aclken	Input	Yes	Active-High clock enable (optional).
aresetn	Input	Yes	Active-Low synchronous clear (optional, always take priority over aclken). A minimum aresetn active pulse of two cycles is required.
s_axis_config_tvalid	Input	No	TVALID for the Configuration channel. Asserted by the external master to signal that it is able to provide data.
s_axis_config_tready	Output	No	TREADY for the Configuration channel. Asserted by the core to signal that it is ready to accept data.
s_axis_config_tdata	Input	No	TDATA for the Configuration channel. Carries the configuration information: CP_LEN, FWD/INV, NFFT and SCALE_SCH. See Run Time Transfer Configuration.
s_axis_data_tvalid	Input	No	TVALID for the Data Input channel. Used by the external master to signal that it is able to provide data.
s_axis_data_tready	Output	No	TREADY for the Data Input channel. Used by the core to signal that it is ready to accept data.
s_axis_data_tdata	Input	No	TDATA for the Data Input channel. Carries the unprocessed sample data: XN_RE and XN_IM. See Data Input Channel.
s_axis_data_tlast	Input	No	TLAST for the Data Input channel. Asserted by the external master on the last sample of the frame. This is not used by the core except to generate the events event_tlast_unexpected and event_tlast_missing events
m_axis_data_tvalid	Output	No	TVALID for the Data Output channel. Asserted by the core to signal that it is able to provide sample data.
m_axis_data_tready	Input	No	TREADY for the Data Output channel. Asserted by the external slave to signal that it is ready to accept data. Only present in "Non-Realtime" mode.
m_axis_data_tdata	Output	No	TDATA for the Data Output channel. Carries the processed sample data XK_RE and XK_IM. See Data Output Channel.
m_axis_data_tuser	Output	No	TUSER for the Data Output channel. Carries additional per-sample information, such as XK_INDEX, OVFLO and BLK_EXP. See Data Output Channel.
m_axis_data_tlast	Output	No	TLAST for the Data Output channel. Asserted by the core on the last sample of the frame.

Table 2-1: **Core Signal Pinout *(Cont'd)***

Name	Direction	Optional	Description
m_axis_status_tvalid	Output	No	TVALID for the Status channel. Asserted by the core to signal that it is able to provide status data.
m_axis_status_tready	Input	No	TREADY for the Status channel. Asserted by the external slave to signal that it is ready to accept data. Only present in "Non-Realtime" mode
m_axis_status_tdata	Output	No	TDATA for the Status channel. Carries the status data: BLK_EXP or OVFLO. See Status Channel.
event_frame_started	Output	No	Asserted when the core starts to process a new frame. See event_frame_started.
event_tlast_unexpected	Output	No	Asserted when the core sees s_axis_data_tlast High on a data sample that is not the last one in a frame. See event_tlast_unexpected.
event_tlast_missing	Output	No	Asserted when s_axis_data_tlast is Low on the last data sample of a frame. See event_tlast_missing.
event_fft_overflow	Output	No	Asserted when an overflow is seen in the data samples being unloaded from the Data Output channel. Only present when overflow is a valid option. See event_fft_overflow.
event_data_in_channel_halt	Output	No	Asserted when the core requests data from the Data Input channel and none is available. See event_data_in_channel_halt.
event_data_out_channel_halt	Output	No	Asserted when the core tries to write data to the Data Output channel and it is unable to do so. Only present in "Non-Realtime" mode. See event_data_out_channel_halt.
event_status_channel_halt	Output	No	Asserted when the core tries to write data to the Status channel and it is unable to do so. Only present in "Non-Realtime" mode. See event_status_channel_halt.

Note: All AXI4-Stream port names are lowercase, but for ease of visualization, uppercase is used in this document when referring to port name suffixes, such as TDATA or TLAST.

Anexo A.3: Detalle Entradas/Salidas Módulo: "Complex Multiplier LogiCORE IP Xilinx v6.0"

Especificaciones de las entradas y salidas del módulo IP. Tabla 2.1 del documento dado en [31].

Table 2-1: **Core Signal Pinout**

Name	Direction	Optional	Description
aclk	Input	yes	Rising-edge clock. The aclk signal is optional. It is not present when FlowControl is NonBlocking and MinimumLatency = 0.
aclken	Input	yes	Active-High clock enable (optional)
aresetn	Input	yes	Active-Low synchronous clear (optional, always take priority over aclken) **Note**: aresetn should be asserted or deasserted for not less than two aclk cycles.
s_axis_a_tvalid	Input	no	TVALID for channel A
s_axis_a_tready	Output	yes	TREADY for channel A
s_axis_a_tuser[A-1:0]	Input	yes	TUSER for channel A. Width selectable from 1 to 256 bits
s_axis_a_tdata[B-1:0]	Input	no	TDATA for channel A. See TDATA Packing for internal structure and width.
s_axis_a_tlast	Input	yes	TLAST for channel A.
s_axis_b_tvalid	Input	no	TVALID for channel B
s_axis_b_tready	Output	yes	TREADY for channel B
s_axis_b_tuser[C-1:0]	Input	yes	TUSER for channel B. Width selectable from 1 to 256 bits
s_axis_b_tdata[D-1:0]	Input	no	TDATA for channel B. See TDATA Packing for internal structure and width.
s_axis_b_tlast	Input	yes	TLAST for channel B.
s_axis_ctrl_tvalid	Input	yes	TVALID for channel CTRL
s_axis_ctrl_tready	Output	yes	TREADY for channel CTRL
s_axis_ctrl_tuser[E-1:0]	Input	yes	TUSER for channel CTRL. Width selectable from 1 to 256 bits
s_axis_ctrl_tdata[7:0]	Input	yes	TDATA for channel CTRL. See TDATA Packing for internal structure and width.
s_axis_ctrl_tlast	Input	yes	TLAST for channel CTRL.
m_axis_dout_tvalid	Output	no	TVALID for channel DOUT
m_axis_dout_tready	Input	yes	TREADY for channel DOUT
m_axis_dout_tuser[G-1:0]	Output	yes	TUSER for channel DOUT. Width is the sum of the enabled TUSER fields on input channels.
m_axis_dout_tdata[H-1:0]	Output	no	TDATA for channel DOUT. See TDATA Packing internal structure.
m_axis_dout_tlast	Output	yes	TLAST for channel DOUT.

Notes:

1. All AXI4-Stream port names are lower case but for ease of visualization, upper case is used in this document when referring to port name suffixes, such as TDATA or TLAST.
2. Width constants A to H are arbitrary variables, determined by GUI or configuration parameters.

Anexo A.4: Diagramas de Flujo Detallados del Controlador del *Correlator*

Las cuatro figuras siguientes corresponden al Diagrama de Flujo Detallado completo del Controlador del *Correlator*.

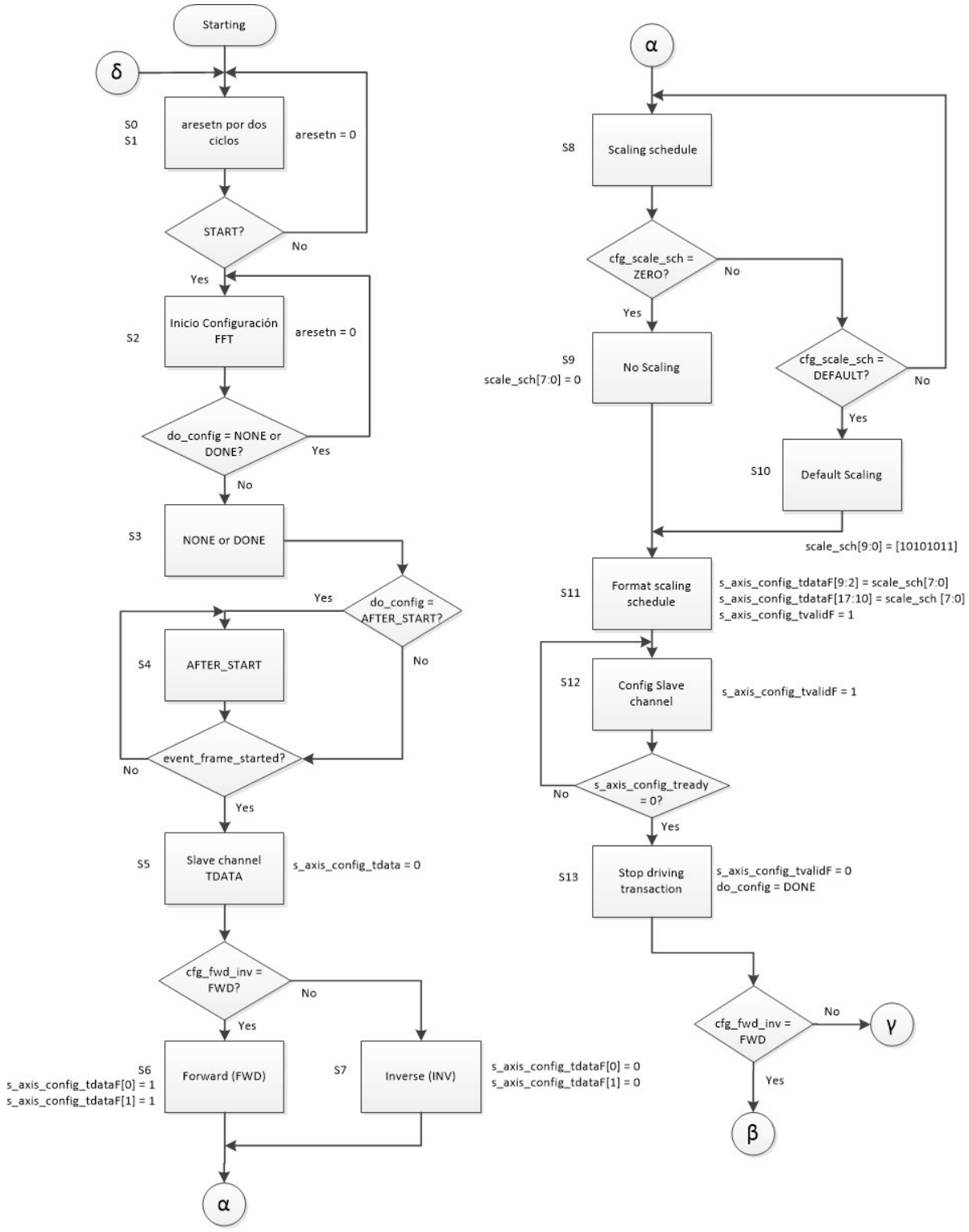

Figura A1: Diagrama de Flujo Detallado del Controlador del *Correlator* (1 de 4)

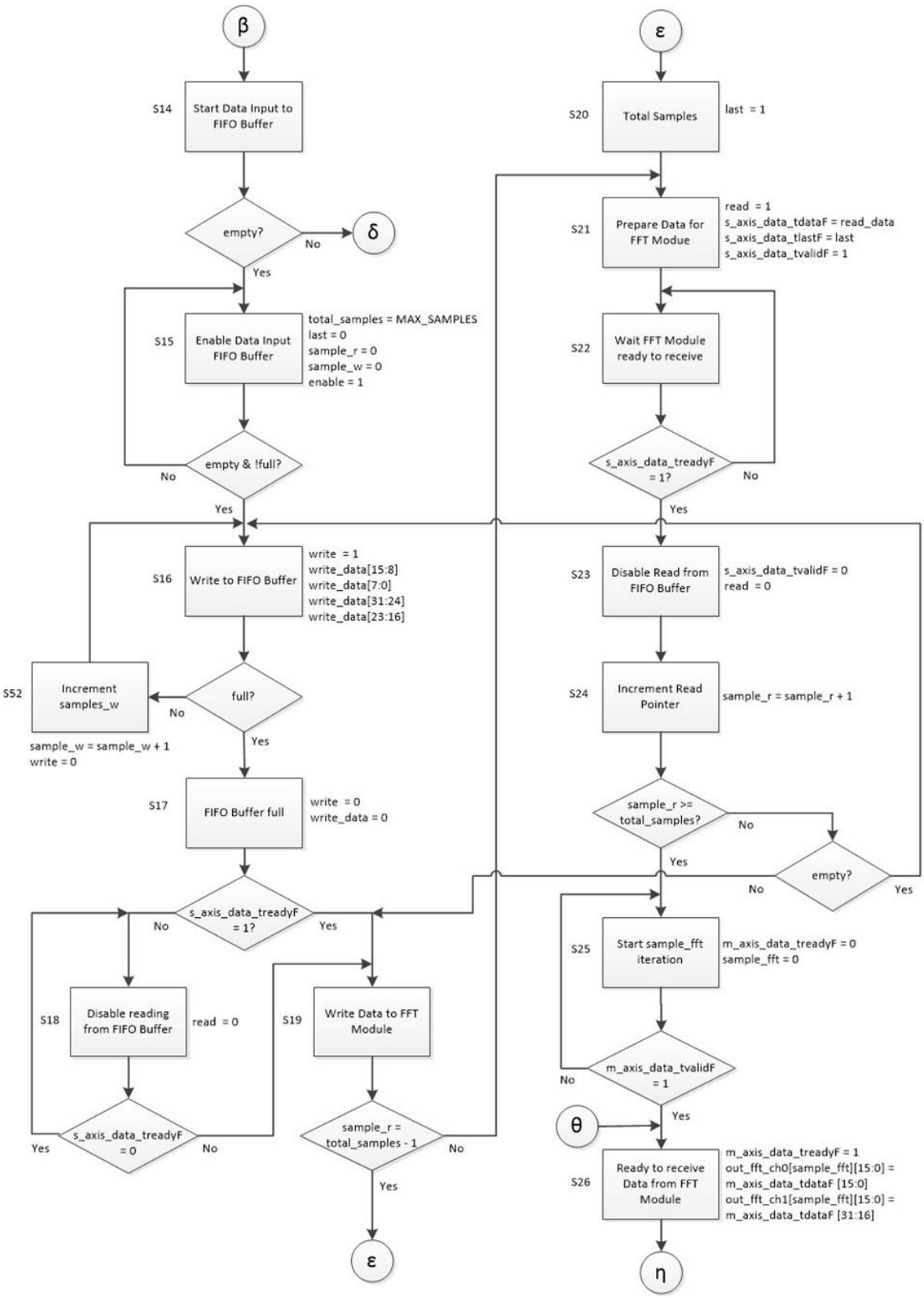

Figura A2: Diagrama de Flujo Detallado del Controlador del *Correlator* (2 de 4)

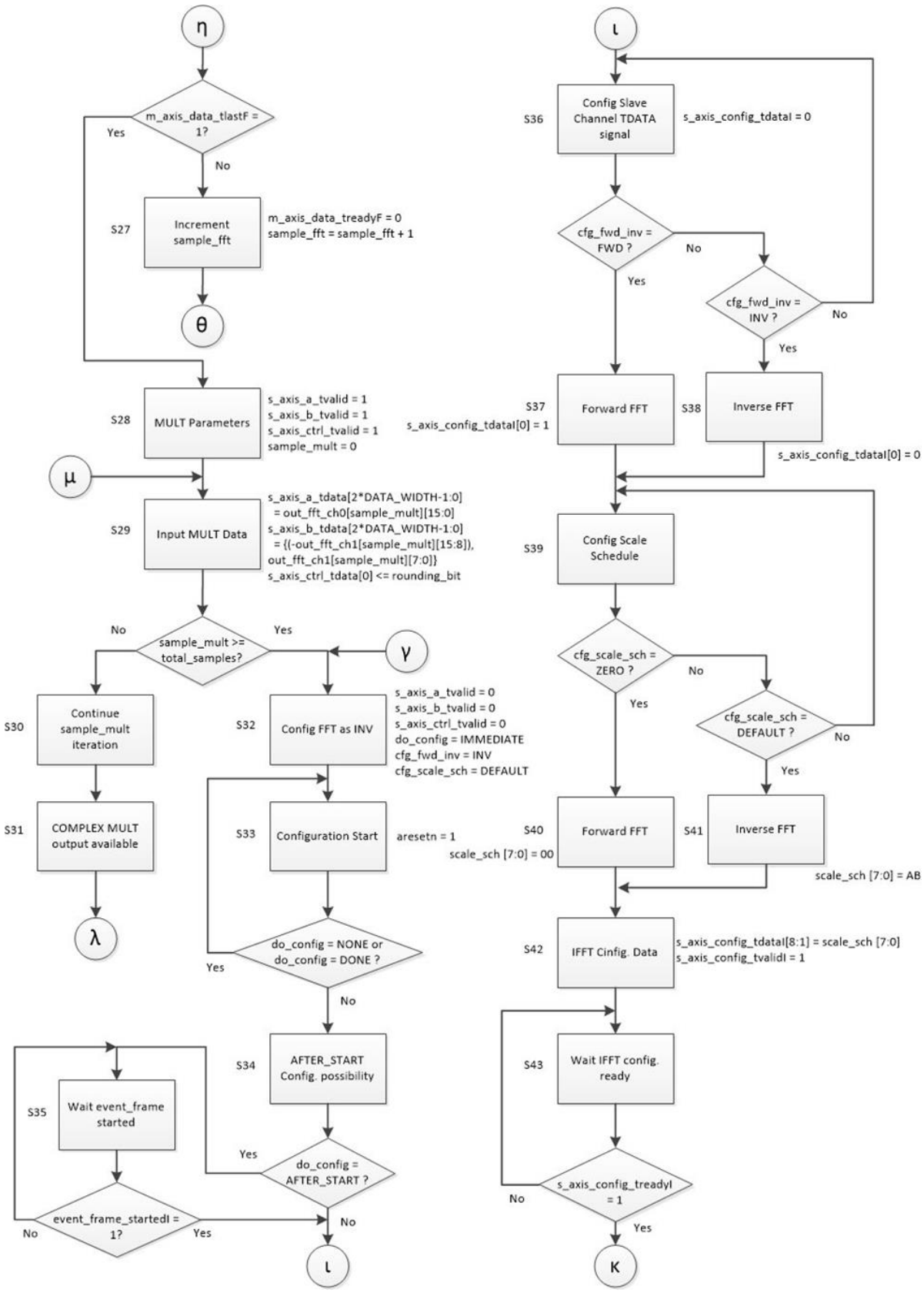

Figura A3: Diagrama de Flujo Detallado del Controlador del *Correlator* (3 de 4)

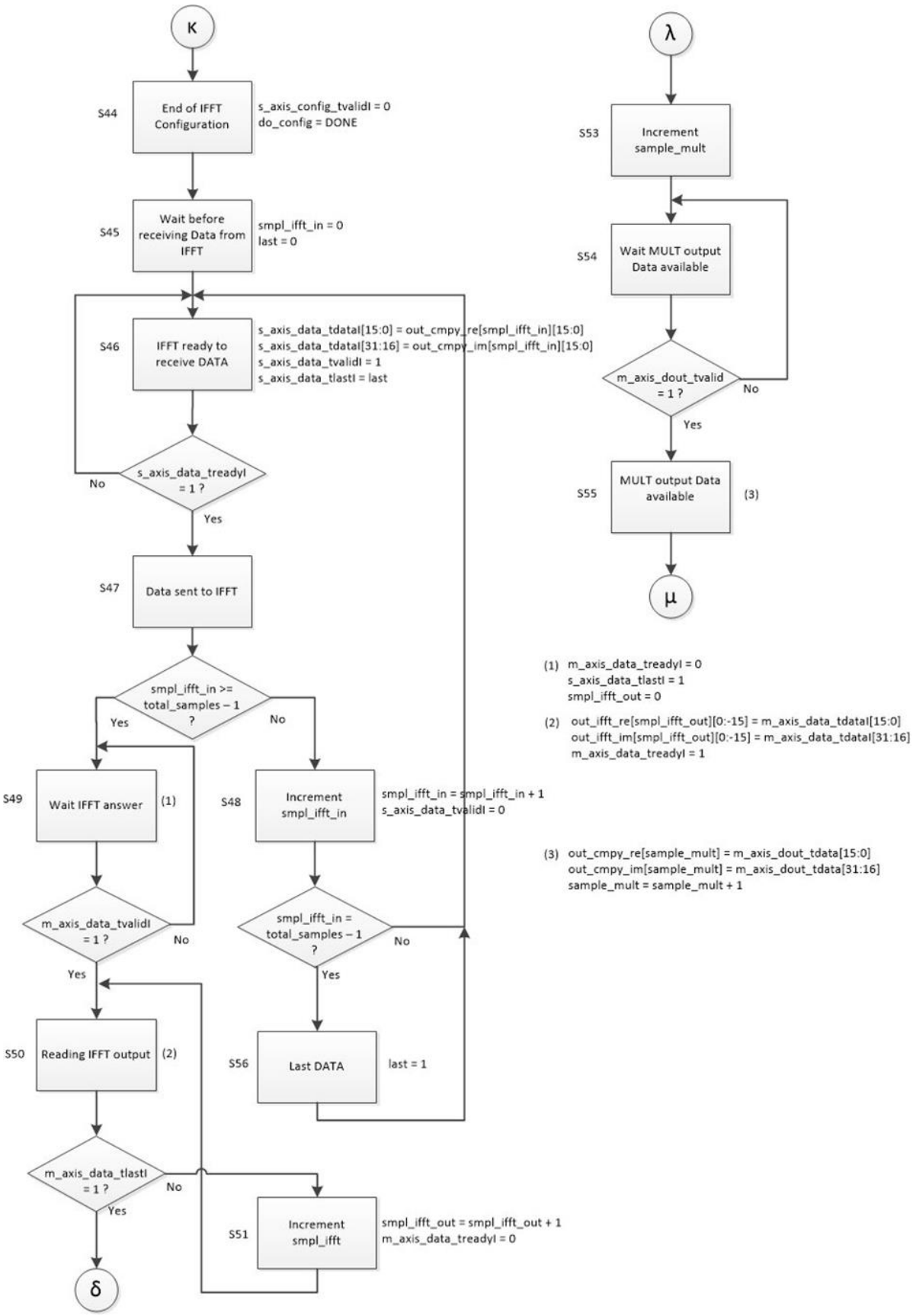

Figura A4: Diagrama de Flujo Detallado del Controlador del *Correlator* (4 de 4)

Anexo A.5: Resultados del Diseño del *Correlator* para Entradas Sinusoidales

A continuación se entregan una serie de ejemplos de entradas sinusoidales aplicadas al *Correlator* con diferentes frecuencias ($FREQ1$ y $FREQ2$) y para 1024 muestras.

La figura A5 muestra entradas iguales de 16 bits con $FREQ1 = 5,2$ Hz y $FREQ2 = 0Hz$. La cantidad de muestras es 1024. La salida obtenida del diseño corresponde exactamente a lo entregado por MATLAB.

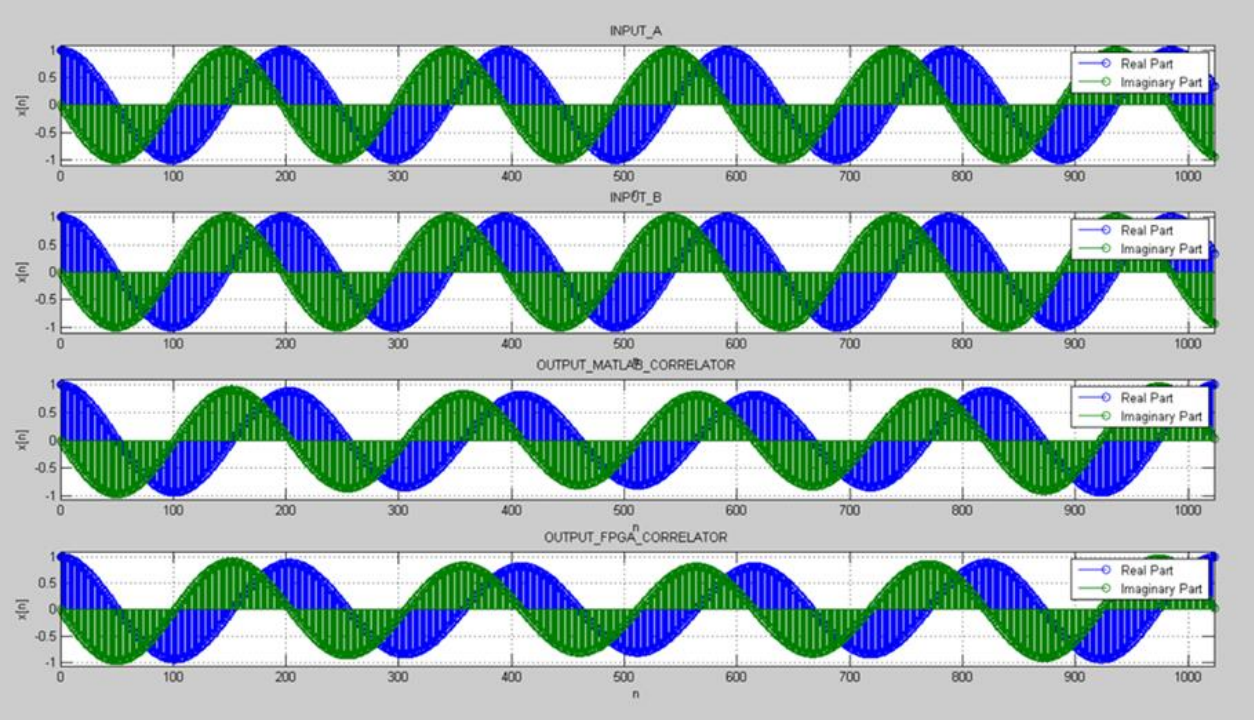

Figura A5: Salida *Correlator*: entradas iguales ($FREQ1 = 5,2Hz, FREQ2 = 0Hz$) de 16 bits y 1024 muestras

La figura A6 muestra el resultado obtenido para entradas de menor frecuencia ($FREQ1 = 2,6Hz, FREQ2 = 0Hz$) a las de la figura A5. Igualmente, la salida obtenida corresponde exactamente a lo obtenido con MATLAB.

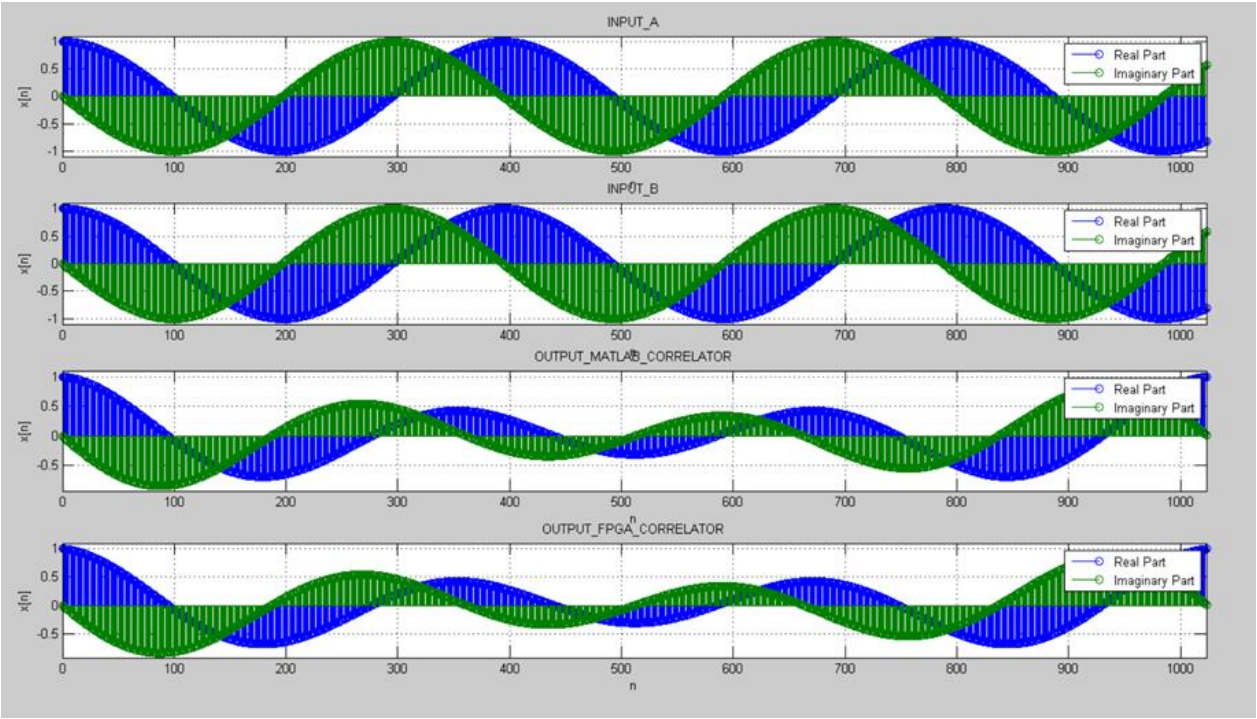

Figura A6: Salida *Correlator*: entradas iguales ($FREQ1 = 2,6Hz, FREQ2 = 0Hz$) de 16 bits y 1024 muestras

La figura A7 muestra el resultado considerando entradas de diferentes frecuencias (para la entrada A, FREQ1 = 2,6 Hz y FREQ2 = 0 Hz; para la entrada B, FREQ1 = 5,2 Hz y FREQ2 = 0 Hz).

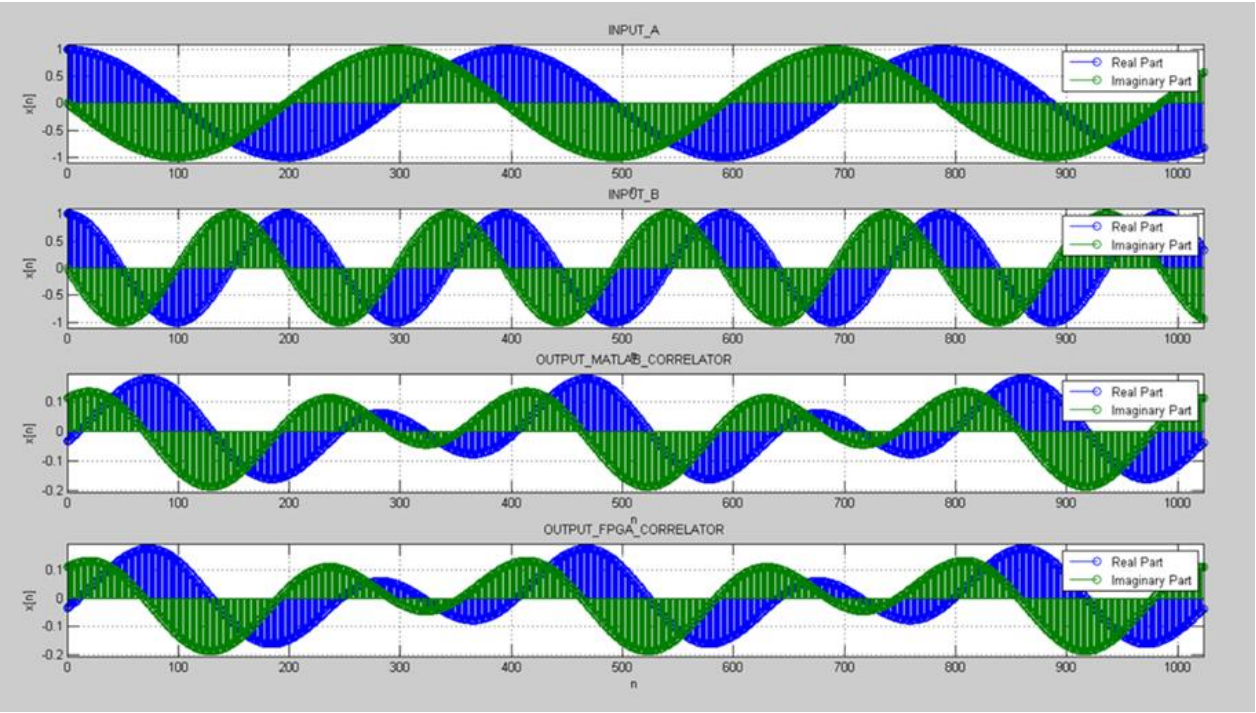

Figura A7: Salida *Correlator*: entradas diferentes (entrada A: $FREQ1 = 2,6Hz$ y $FREQ2 = 0Hz$, entrada B: $FREQ1 = 5,2Hz$ y $FREQ2 = 0Hz$) de 16 bits y 1024 muestras

La figura A8 muestra el resultado para entradas iguales que involucran dos frecuencias diferentes para cada una de las entradas (para las entradas A y B, $FREQ1 = 2,6Hz$ y $FREQ2 = 23,2Hz$).

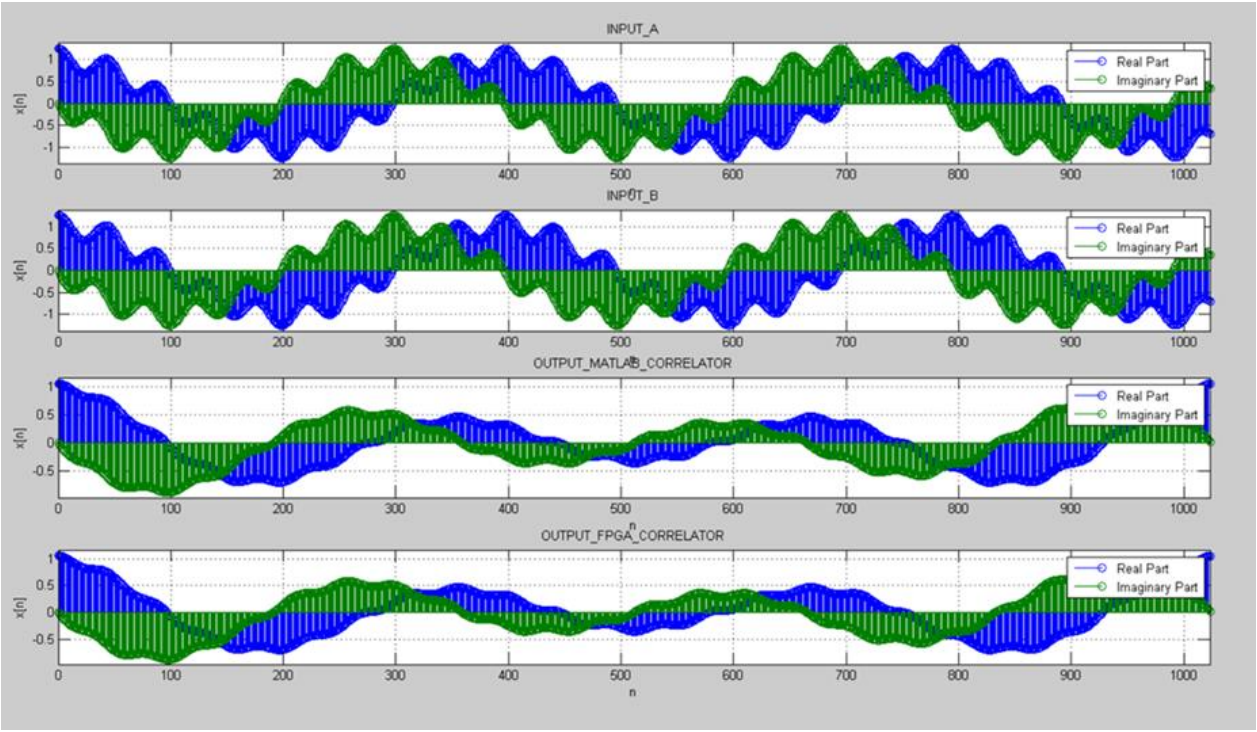

Figura A8: Salida *Correlator*: entradas iguales de dos frecuencias diferentes (entradas A y B: $FREQ1 = 2,6Hz$ y $FREQ2 = 23,2Hz$) de 16 bits y 1024 muestras

La figura A9 muestra el resultado para una situación similar a la figura A8 pero las entradas son diferentes (entrada A: $FREQ1 = 2,6Hz$ y $FREQ2 = 23,2Hz$, entrada B: $FREQ1 = 5,2Hz$ y $FREQ2 = 46,4Hz$).

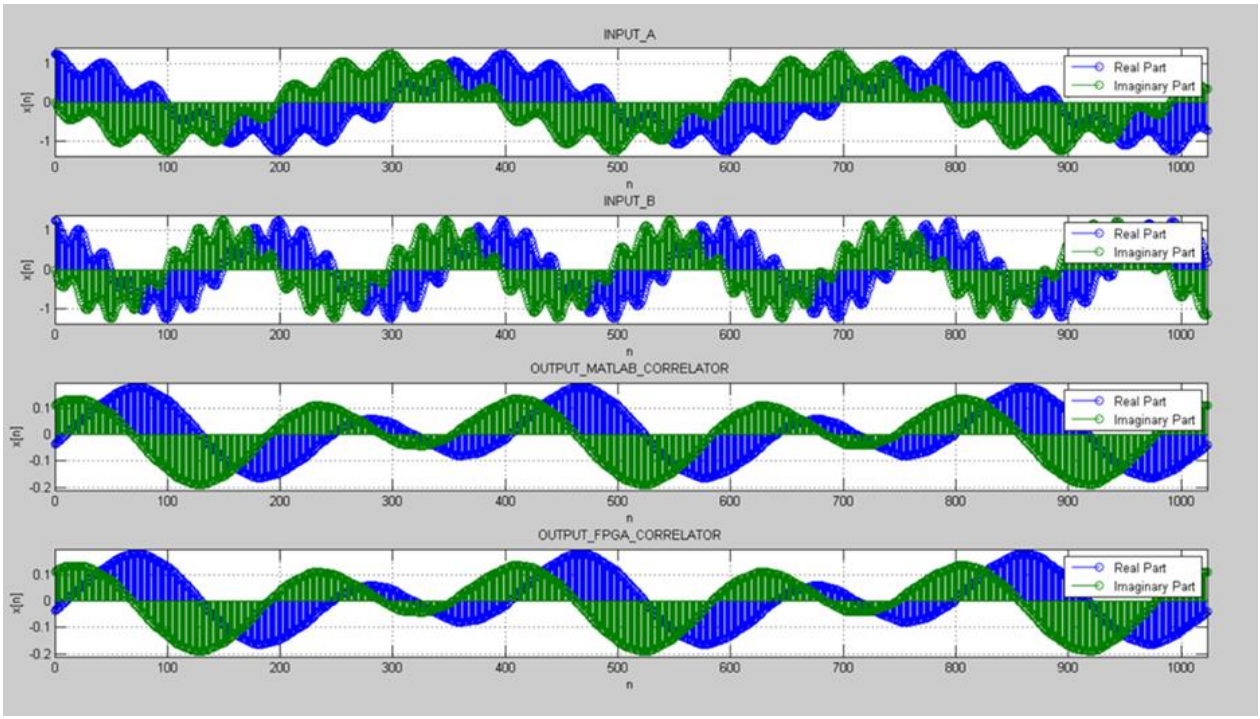

Figura A9: Salida *Correlator*: entradas diferentes de dos frecuencias diferentes (entrada A: $FREQ1 = 2,6Hz$ y $FREQ2 = 23,2Hz$, entrada B: $FREQ1 = 5,2Hz$ y $FREQ2 = 46,4Hz$) de 16 bits y 1024 muestras

Anexo A.6: Detalle Entradas/Salidas Módulo: "CORDIC LogiCORE IP Xilinx v6.0"

Especificaciones de las entradas y salidas del módulo IP. Tabla 2.1 del documento dado en [35].

Table 2-1: **Core Pinout**

Port Name	Direction	Description
aclk	In	Clock. Active rising edge.
ACLKEN	In	Clock Enable. Active-High
ARESETn	In	Synchronous Reset. Active-Low. ARESETn must be active for at least 2 clock cycles when asserted.
s_axis_cartesian_tvalid	In	Handshake signal for channel S_AXIS_CARTESIAN.(1)
s_axis_cartesian_tready	Out	Handshake signal for channel S_AXIS_CARTESIAN.(1)
s_axis_cartesian_tdata[A-1:0]	In	Depending on Functional Configuration, this port has one or two subfields; X_IN and Y_IN. These are the Cartesian operands. Each subfield is Input_Width bits wide, padded to the next byte width before being concatenated. See TDATA Packing.
s_axis_cartesian_tuser[B-1:0]	In	Data on this port is delayed with the same latency as tdata and appear on m_axis_dout_tuser.(1)
s_axis_cartesian_tlast	In	tlast is not used by the core, but is combined with s_axis_phase_tlast, or passed untouched to m_axis_dout_tlast according to TLAST_Behavior.
s_axis_phase_tvalid	In	Handshake signal for channel S_AXIS_PHASE.(1)
s_axis_phase_tready	Out	Handshake signal for channel S_AXIS_PHASE.(1)
s_axis_phase_tdata[C-1:0]	In	This port has one subfield, PHASE_IN. It is the polar operand. The subfield is Input_Width bits wide, padded to the next byte width.
s_axis_phase_tuser[D-1:0]	In	Data on this port is delayed with the same latency as tdata and appear on m_axis_dout_tuser.(1)
s_axis_phase_tlast	In	tlast is not used by the core, but is combined with s_axis_cartesian_tlast, or passed untouched to m_axis_dout_tlast according to TLAST_Behavior.
m_axis_dout_tvalid	Out	Handshake signal for channel M_AXIS_DOUT. (1)
m_axis_dout_tready	In	Handshake signal for channel M_AXIS_DOUT.(1)
m_axis_dout_tdata[E-1:0]	Out	Depending on Functional Configuration this port contains the following subfields; X_OUT, Y_OUT, PHASE_OUT. Each subfield is Output_Width bits wide, padded to the next byte width before concatenation.
m_axis_dout_tuser[F-1:0]	Out	This port contains the values input to s_axis_cartesian_tuser and/or s_axis_phase_tuser delayed by the same latency as for tdata.
m_axis_dout_tlast	Out	This port outputs s_axis_cartesian_tlast, s_axis_phase_tlast or some combination of the two delayed by the same latency as for tdata.

Notes:

1. For AXI4-Stream details see Protocol Description–AXI-4 Stream.

Anexo A.7: Diagramas de Flujo Detallados del Controlador del *CorrentropyTor*

Las cuatro figuras siguientes corresponden al Diagrama de Flujo Detallado completo del Controlador del *CorrentropyTor*.

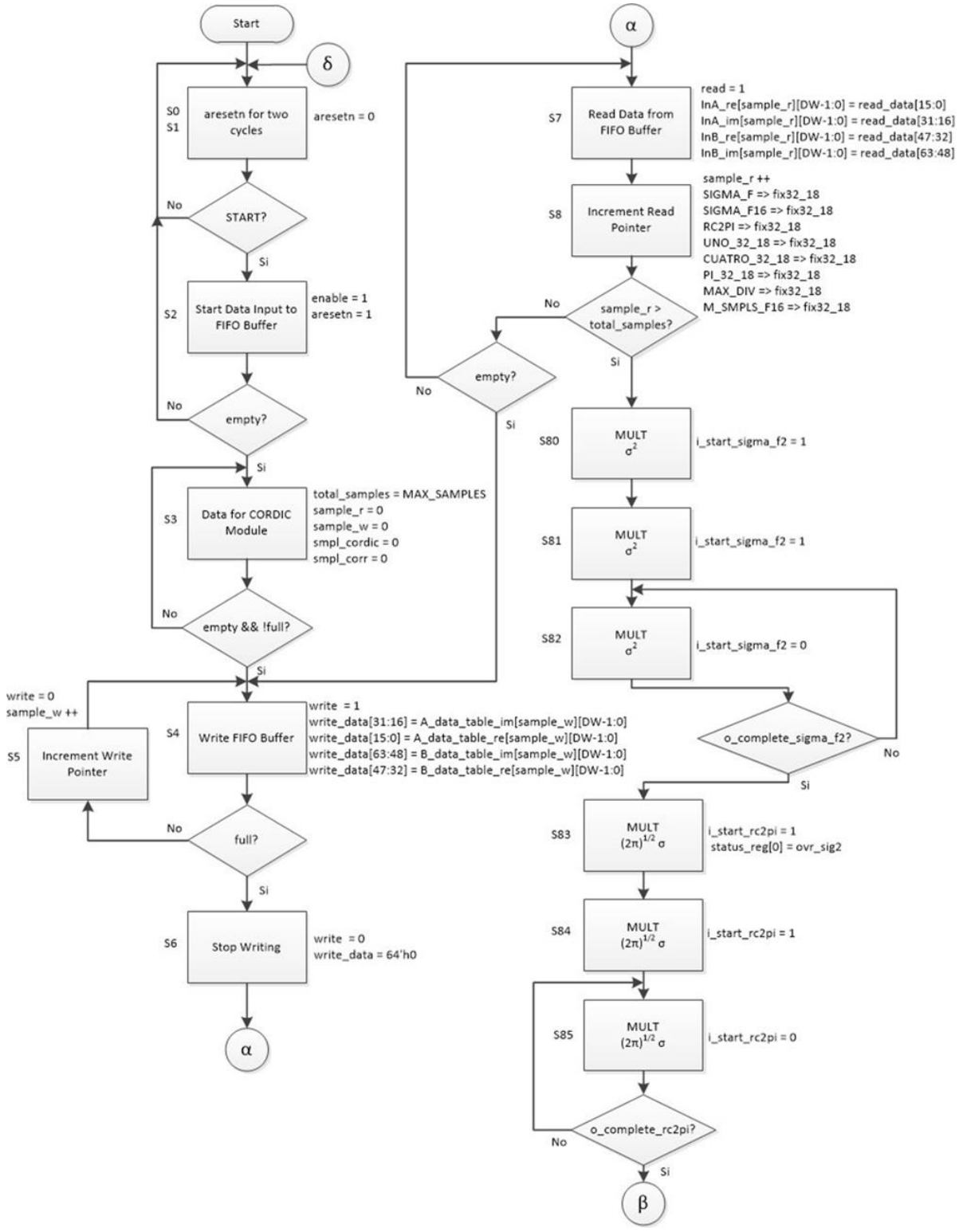

Figura A10: Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (1 de 4)

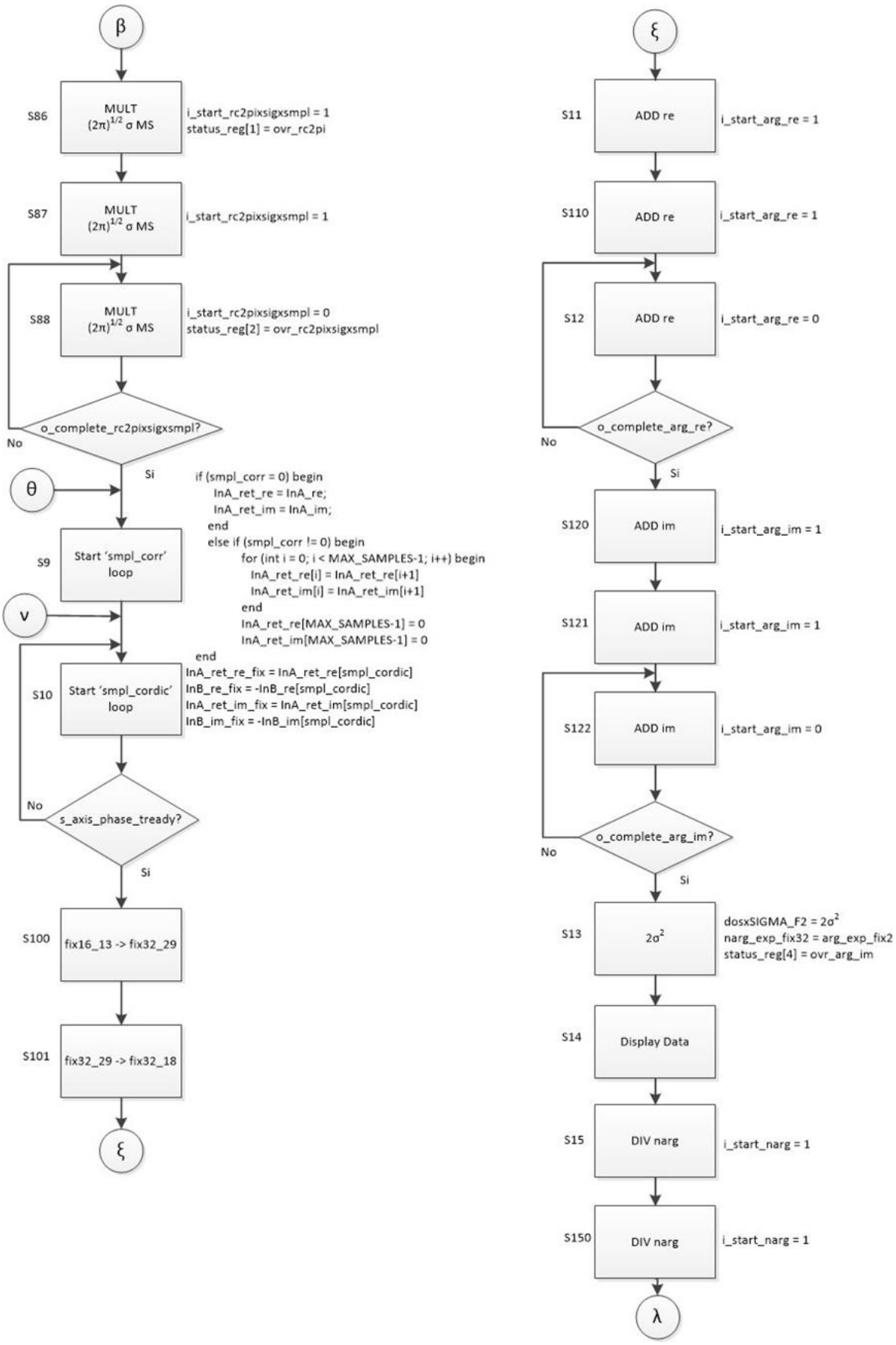

Figura A11: Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (2 de 4)

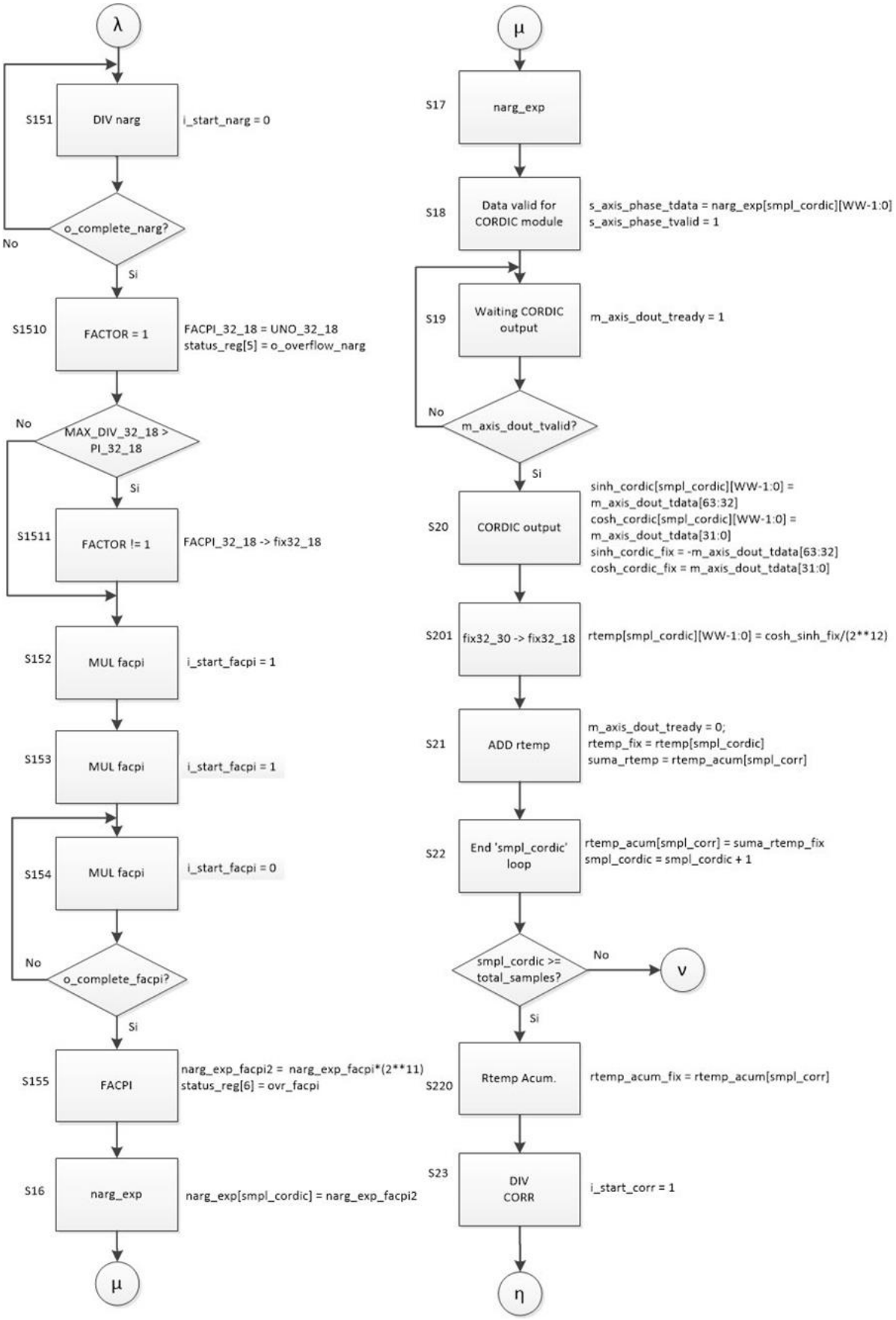

Figura A12: Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (3 de 4)

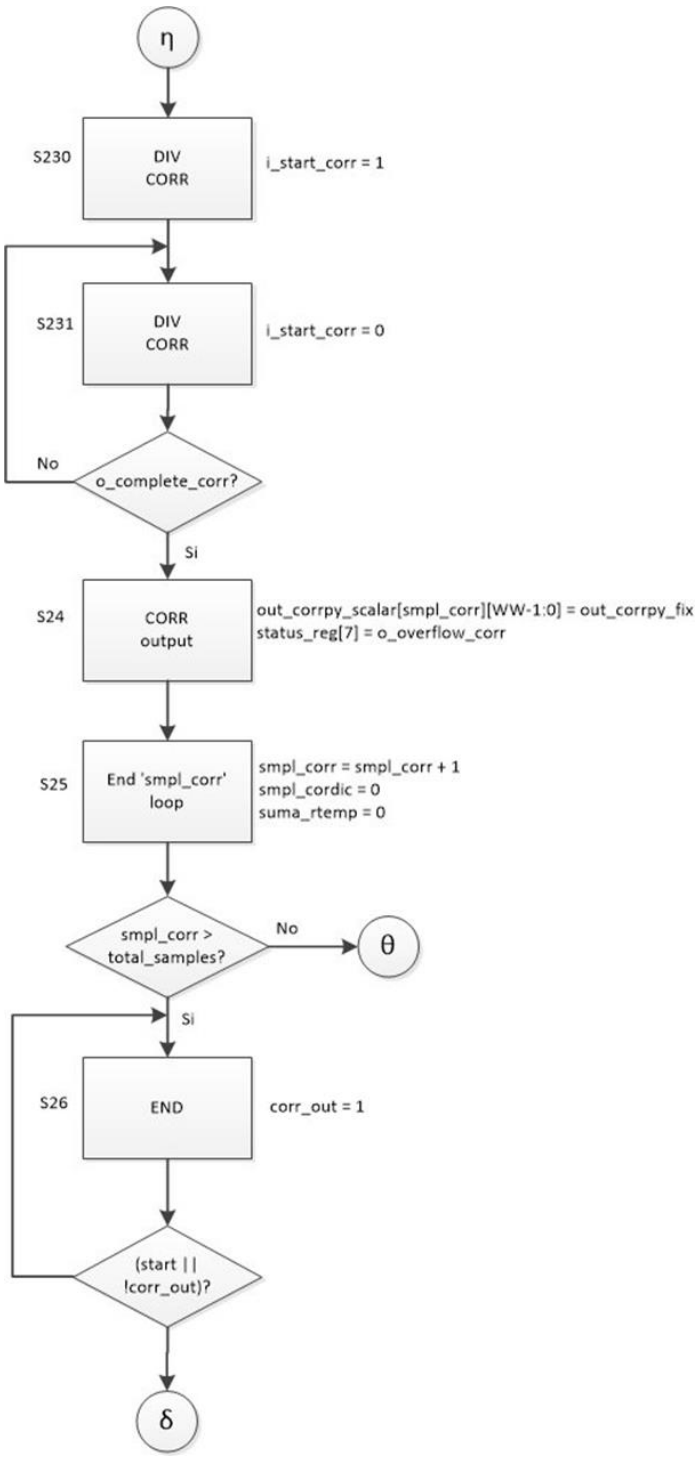

Figura A13: Diagrama de Flujo Detallado del Controlador del *CorrentropyTor* (4 de 4)

Anexo A.8: Resultados del Diseño del *CorrentropyTor* para Entradas Sinusoidales

En este Anexo se muestran los resultados de las Salidas de la Tabla 3.1 que no se mostraron en el apartado 3.3.2.

La figura A14 (generada en MATLAB) muestra un primer resultado simple (Salida 1 en la Tabla 3.1) obtenido, considerando que ambas entradas tienen la misma frecuencia (2,6 Hz), 16 bits de entrada (parte real y parte imaginaria), un sigma de 2,0 (cinco veces Silverman) y 256 muestras.

Las dos señales de la parte superior de la figura A14, corresponden a las entradas A y B, respectivamente (iguales en este caso). Cada una de estas entradas consta de una parte real y una parte imaginaria como se indica en la figura A14. Estas señales son generadas por los módulos INPUT_A - Data_Input_CORR e INPUT_B - Data_Input_CORR, respectivamente (figura 3.8). Dentro del módulo *testbench* del diseño (*Correlator_Top_tb.sv*) se imprimen estas entradas en un archivo el cual es leído desde MATLAB para generar el gráfico de la figura A14.

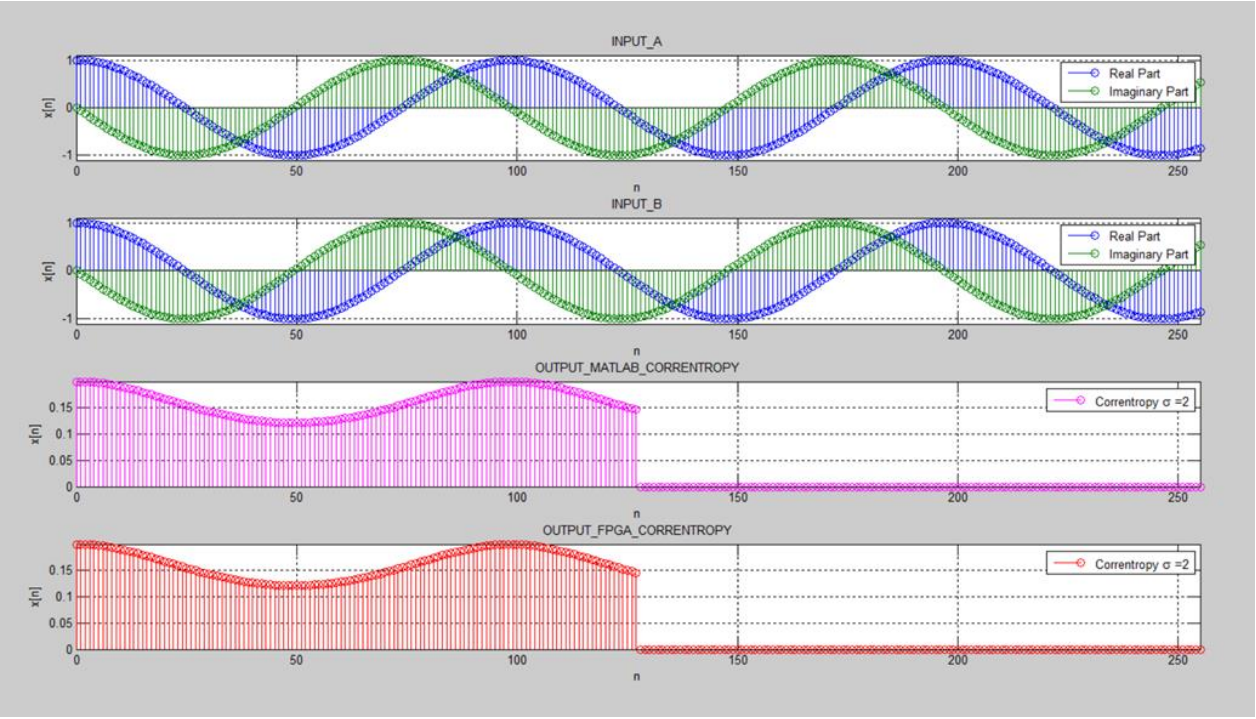

Figura A14: Salida 1 del *CorrentropyTor* según Tabla 3.1

La tercera señal de la figura A14 corresponde a la salida generada en MATLAB, correspondiente a la expresión de la Correntropía dada en el Anexo A.9.

La cuarta señal de la figura A14 es la salida obtenida por nuestro diseño. Esta salida también es grabada en un archivo que luego es graficada con MATLAB. Como se puede ver corresponde exactamente a lo entregado por MATLAB de acuerdo a la tercera señal de la figura A14.

La figura A15 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A14 donde se ve claramente la frecuencia fundamental para esta señal.

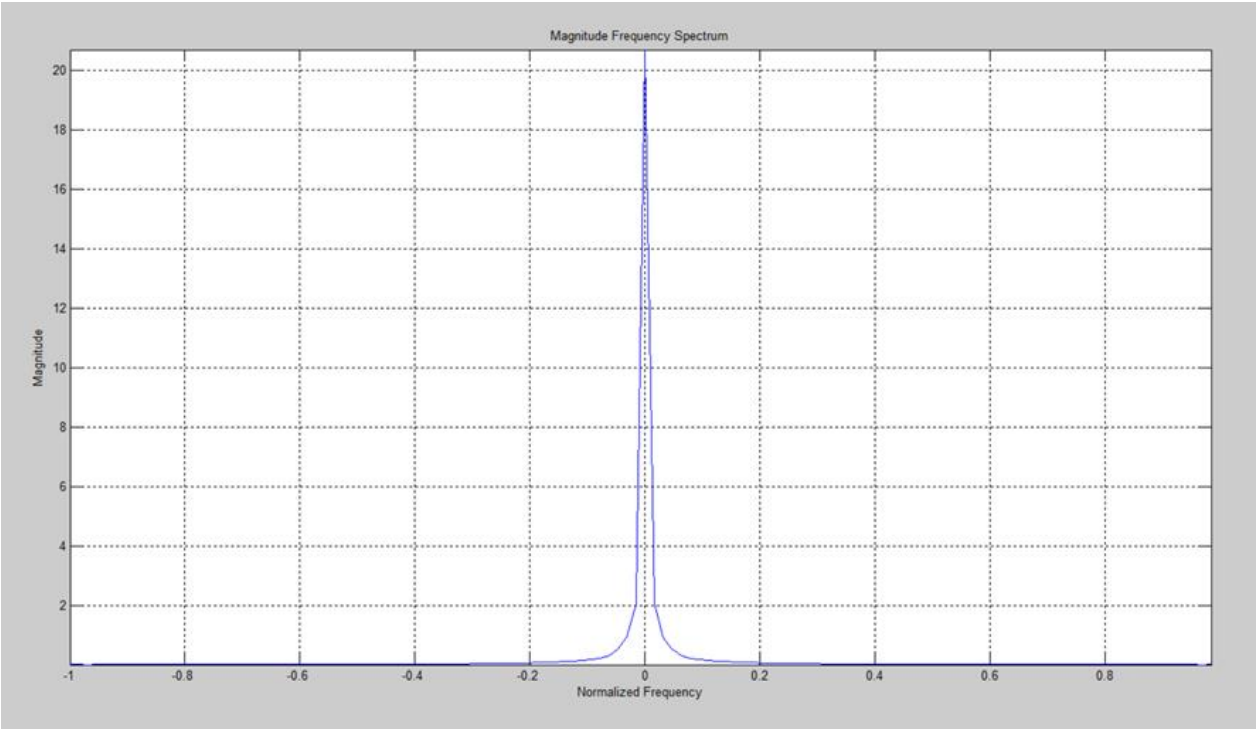

Figura A15: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 1, figura A14)

La figura A16 corresponde a la figura A14 a la cual se le ha restado el valor medio a las salidas de la Correntropía. Dado que estamos trabajando con un valor alto de sigma (2,0 = cinco veces Silverman) podemos comprobar que la señal de la Correntropía (salida FPGA) de la figura A16 sería muy similar a la salida (considerando solamente la parte real) de la Correlación obtenida en la figura A6 del Anexo A.5.

La figura A17 muestra la Salida 2 de la Tabla 3.1. Al igual que la figura A14, ambas entradas tienen la misma frecuencia (2.6 Hz), 16 bits de entrada (parte real y parte imaginaria) y 256 muestras. En este caso, el valor de sigma es 0,8 (dos veces Silverman).

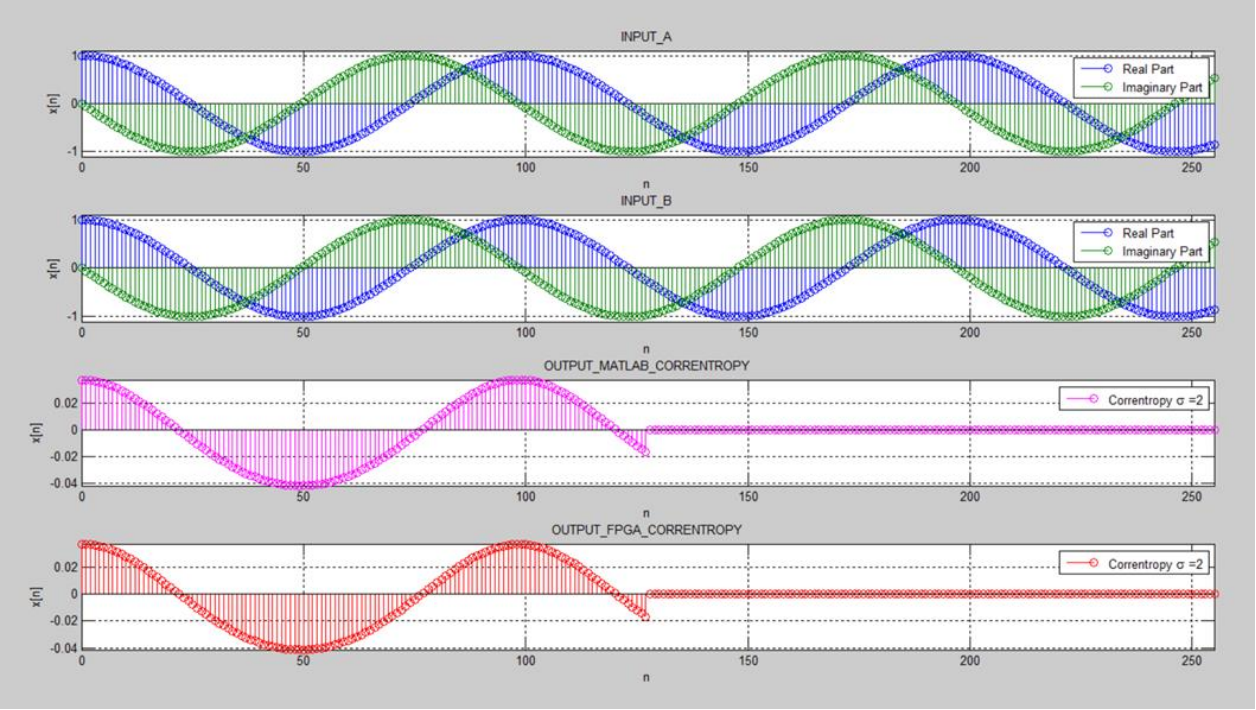

Figura A16: Salida 1 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía

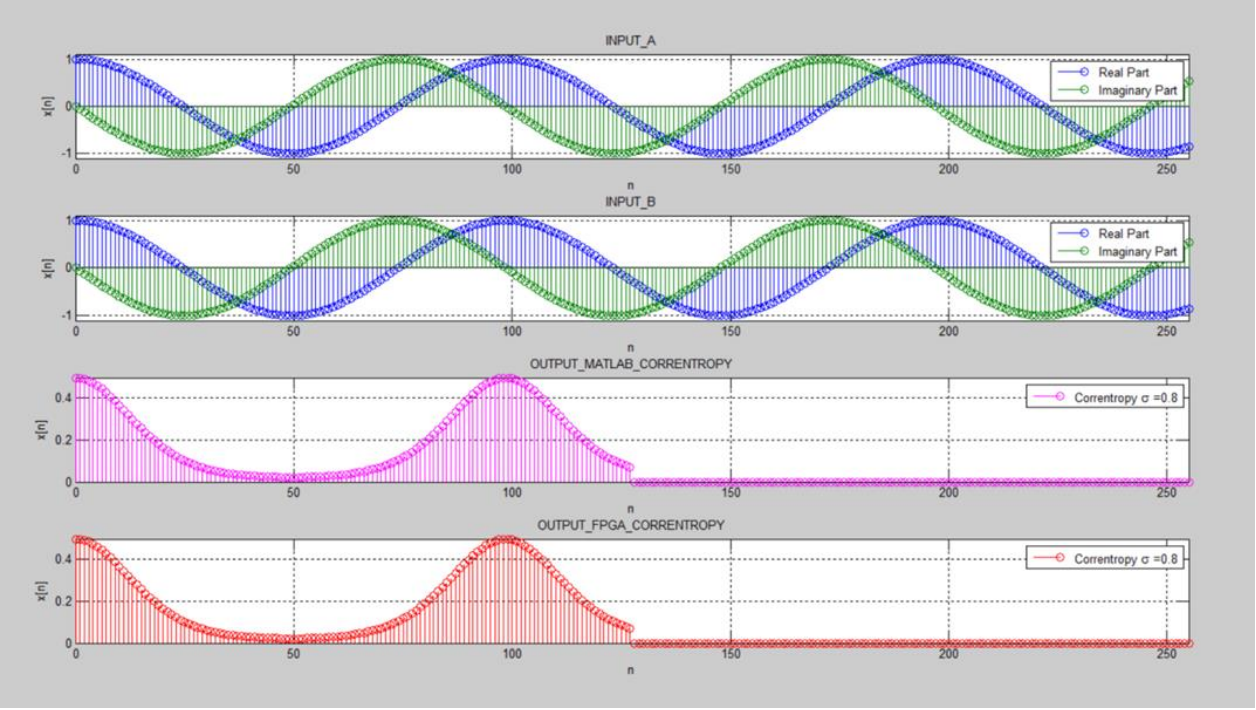

Figura A17: Salida 2 del *CorrentropyTor* según Tabla 3.1

La figura A18 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A17 donde se ve claramente la frecuencia fundamental para esta señal pero también comienzan a aparecer harmónicas.

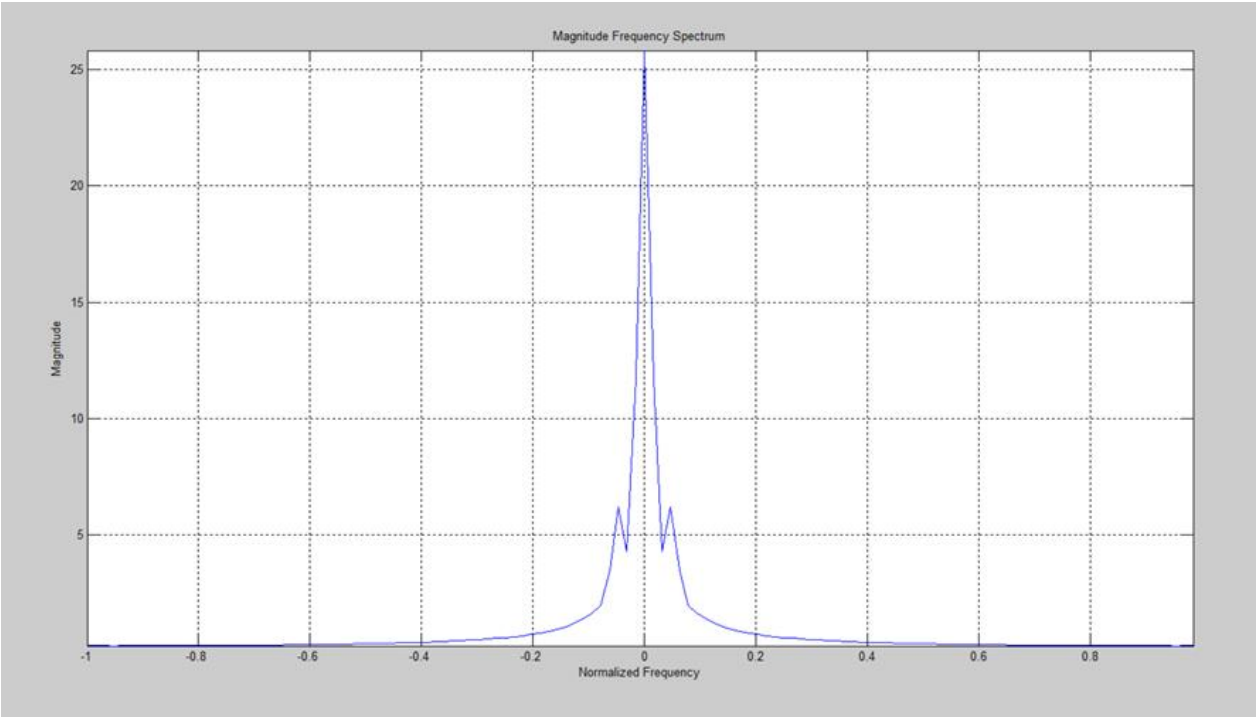

Figura A18: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 2, figura A17)

La figura A19 muestra el resultado obtenido para la Salida 3 de la Tabla 3.1. En este caso lo único que cambia es sigma que toma el valor de 0,4 (una vez Silverman). Igualmente, como en los dos casos anteriores, la salida obtenida con el diseño, corresponde exactamente a lo indicado por MATLAB.

La figura A20 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A19 donde se ve claramente la frecuencia fundamental para esta señal pero también comienzan a aparecer más harmónicas que en la figura A18.

Las salidas 4, 5 y 6 de la Tabla 3.1, se muestran en las figuras 3.9, 3.12 y 3.14, respectivamente, en el apartado 3.3.2.

A continuación se entregan otros ejemplos, las demás salidas de la Tabla 3.1, considerando distintas posibilidades en las señales de entradas y en los valores de sigma.

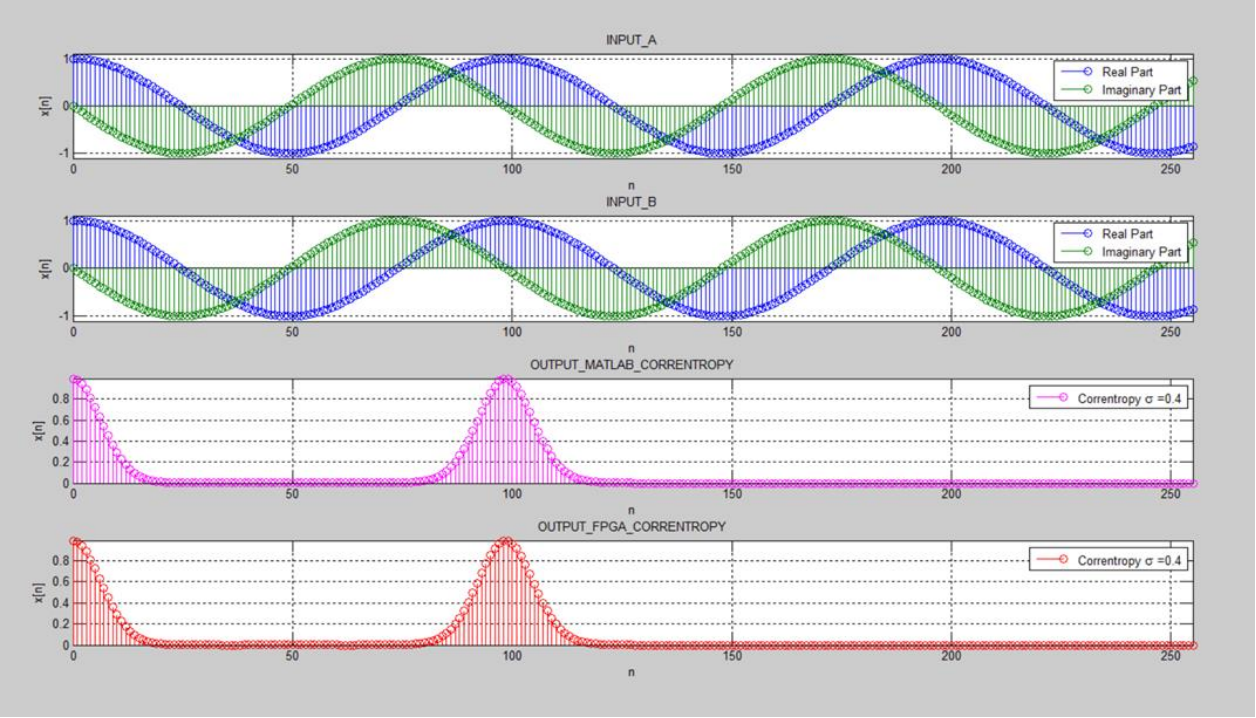

Figura A19: Salida 3 del *CorrentropyTor* según Tabla 3.1

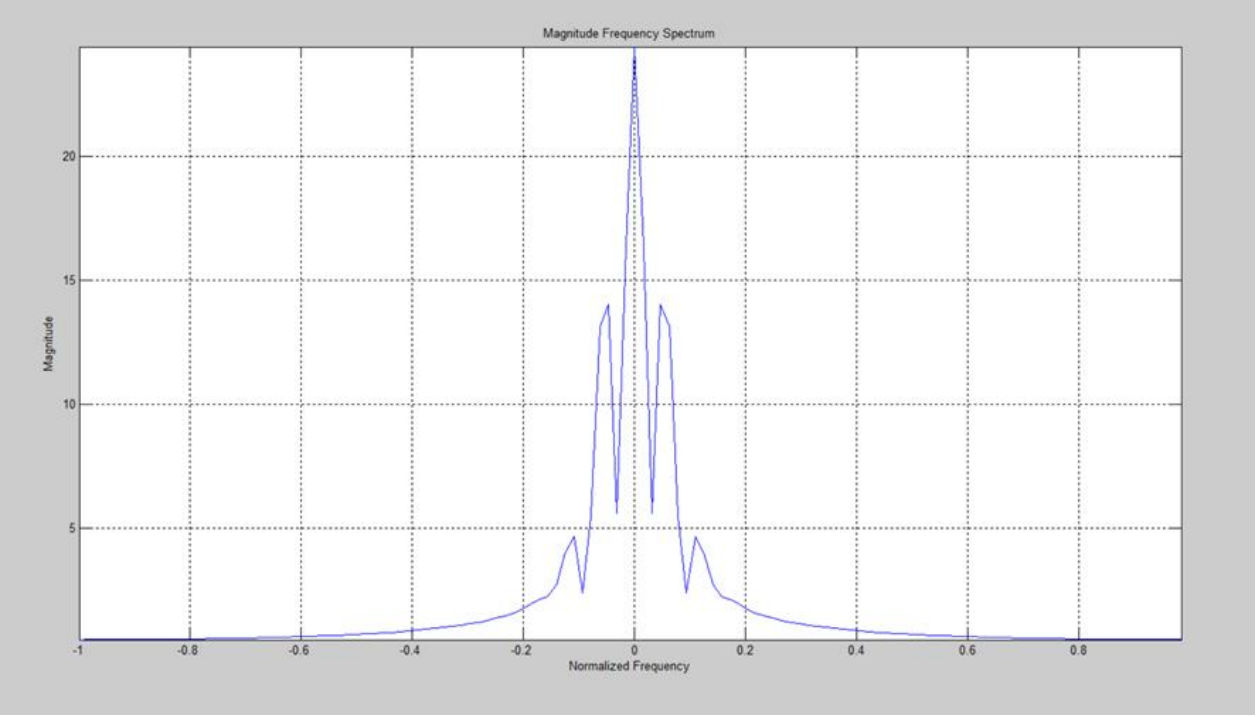

Figura A20: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 3, figura A19)

La figura A21 muestra el resultado correspondiente a la Salida 7 de la Tabla 3.1. En este caso, las entradas son diferentes. La entrada A tiene $FREQ1 = 2,6\,Hz$ y $FREQ2 = 0\,Hz$.

La entrada B tiene $FREQ1 = 23,2\,Hz$ y $FREQ2 = 0\,Hz$. El valor de sigma es de 2,0 (cinco veces Silverman).

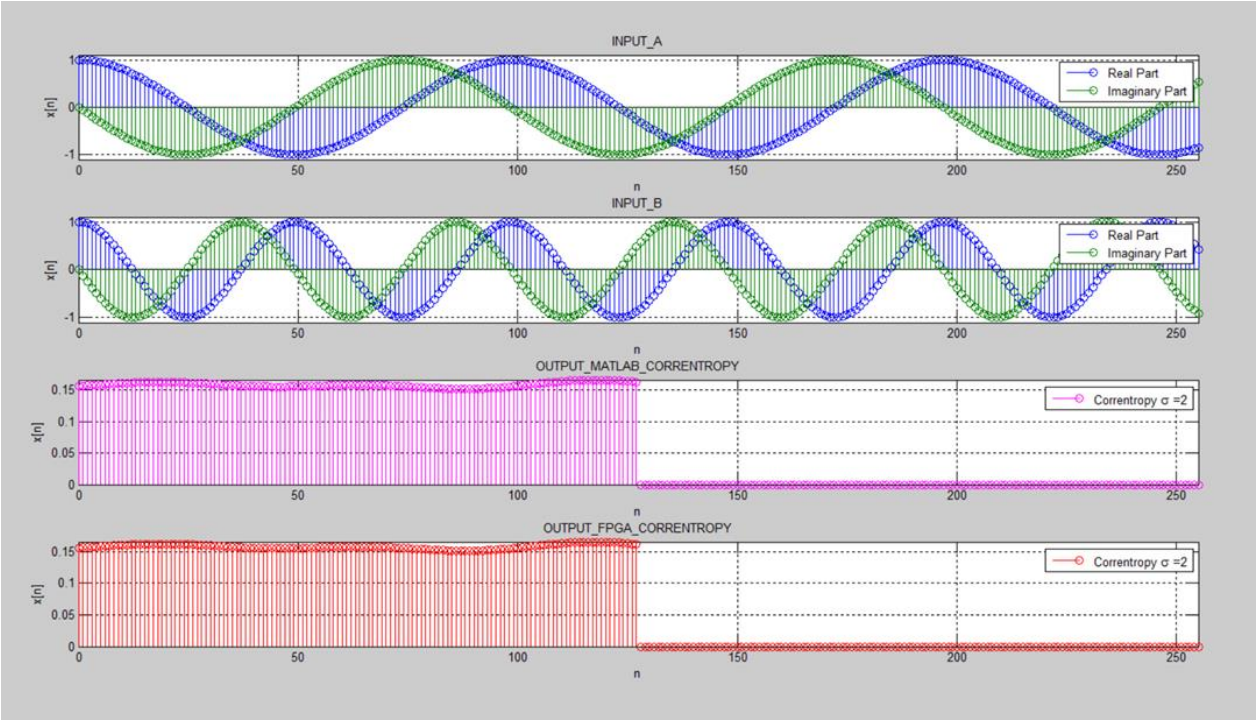

Figura A21: Salida 7 del *CorrentropyTor* según Tabla 3.1

La figura A22 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A21 donde se ve claramente la frecuencia fundamental para esta señal.

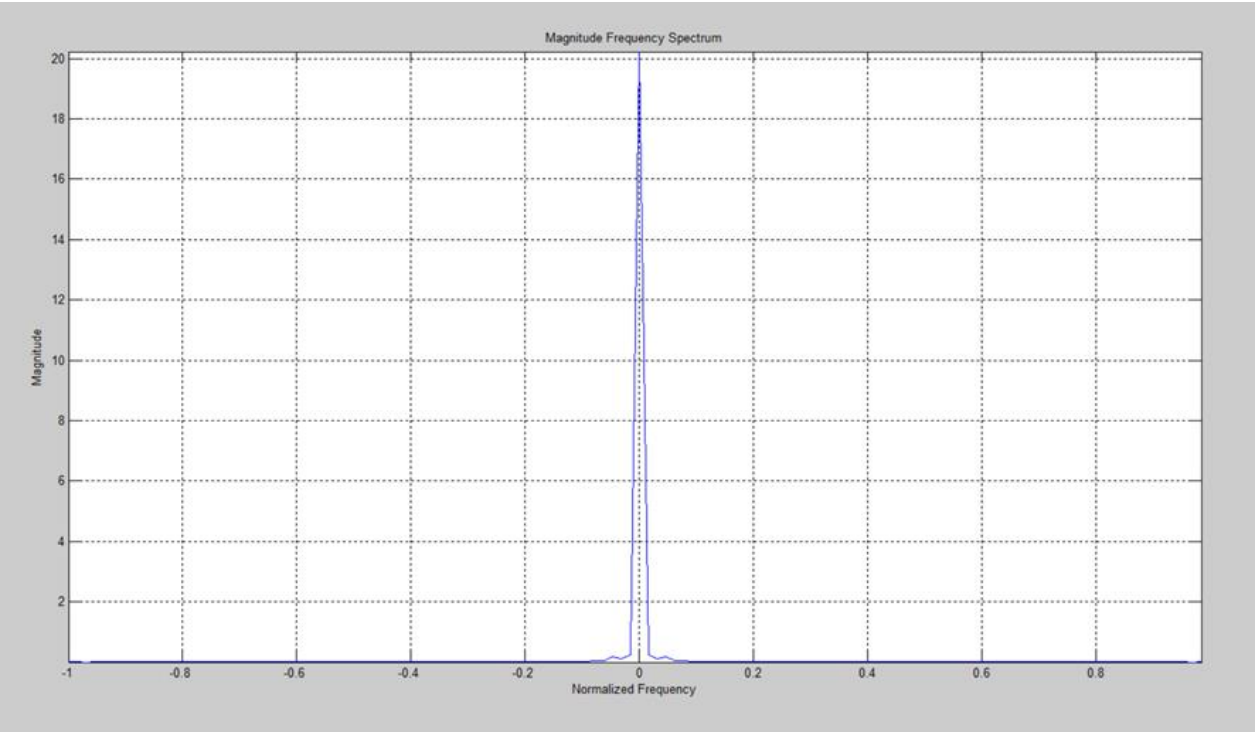

Figura A22: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 7, figura A21)

La figura A23 corresponde a la figura A21 a la cual se le ha restado el valor medio a las salidas de la Correntropía. Dado que estamos trabajando con un valor alto de sigma (2,0 = cinco veces Silverman) podemos comprobar que la señal de la Correntropía (salida FPGA) de la figura A23 sería muy similar a la salida (considerando solamente la parte real) de la Correlación obtenida en la figura A7 del Anexo A.5.

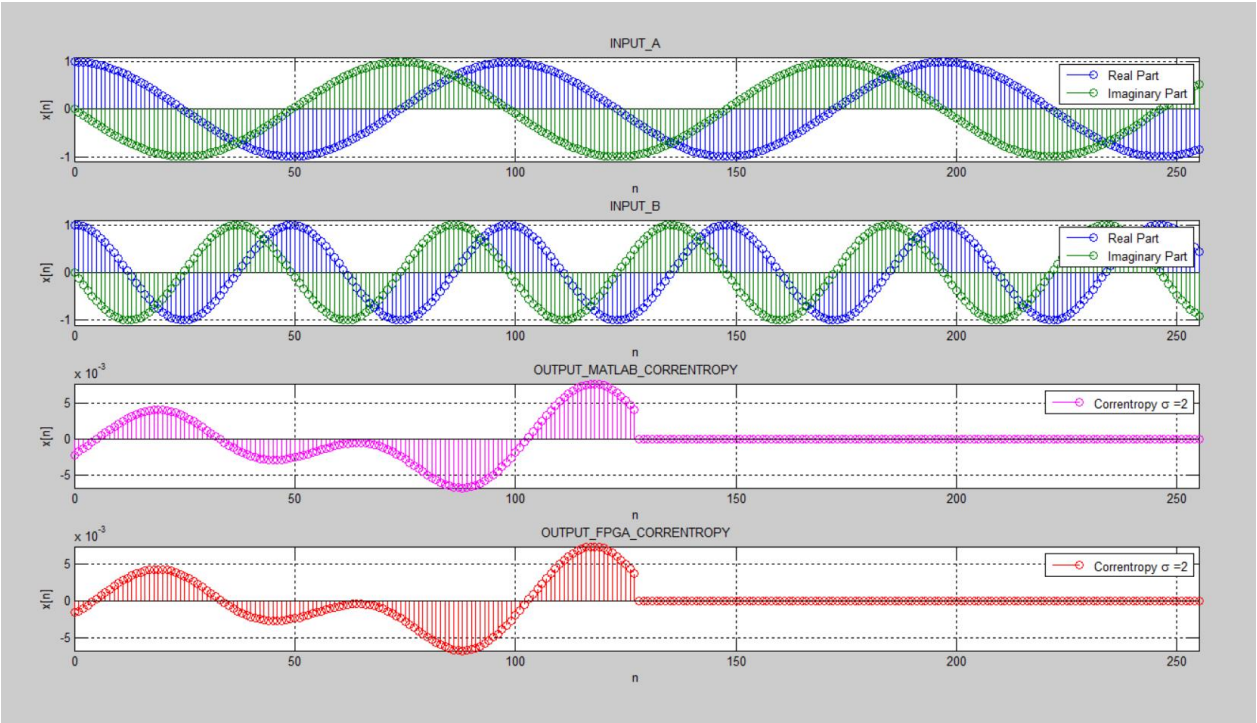

Figura A23: Salida 7 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía

La figura A24 muestra el resultado obtenido para la Salida 8 de la Tabla 3.1. En este caso lo único que cambia es sigma que toma el valor de 0,8 (dos veces Silverman). Igualmente, como en los casos anteriores, la salida obtenida con el diseño, corresponde exactamente a lo indicado por MATLAB.

La figura A25 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A24 donde se ve claramente la frecuencia fundamental para esta señal pero también comienzan a aparecer más harmónicas que en la figura A22.

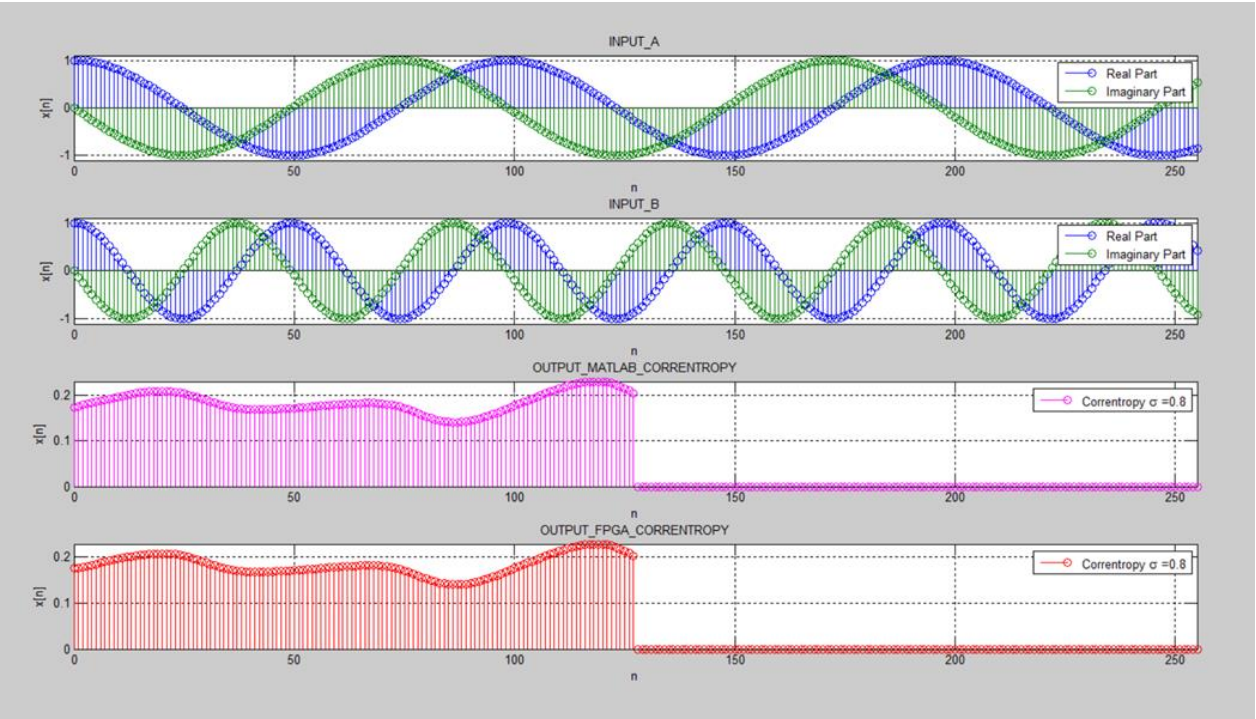

Figura A24: Salida 8 del *CorrentropyTor* según Tabla 3.1

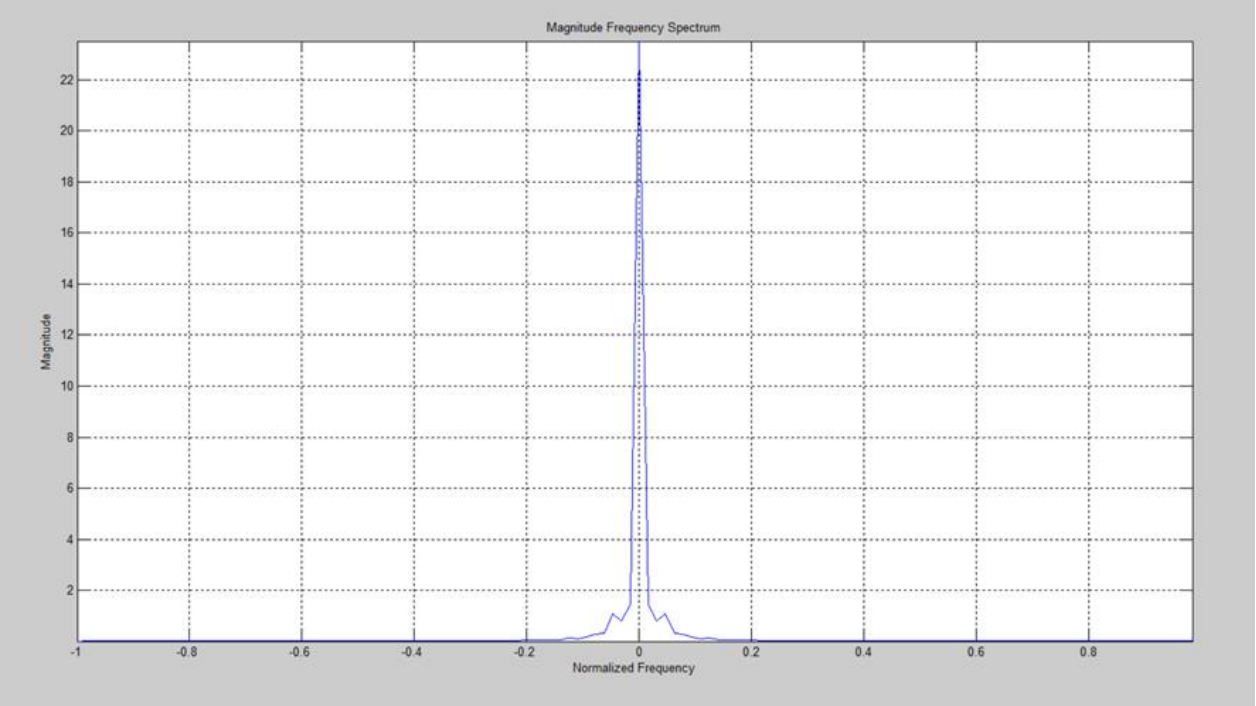

Figura A25: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 8, figura A24)

La figura A26 muestra el resultado obtenido para la Salida 9 de la Tabla 3.1. En este caso lo único que cambia es sigma que toma el valor de 0,4 (una vez Silverman). Igualmente, como en

los casos anteriores, la salida obtenida con el diseño, corresponde exactamente a lo indicado por MATLAB.

La figura A27 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A26 donde se ve claramente la frecuencia fundamental para esta señal.

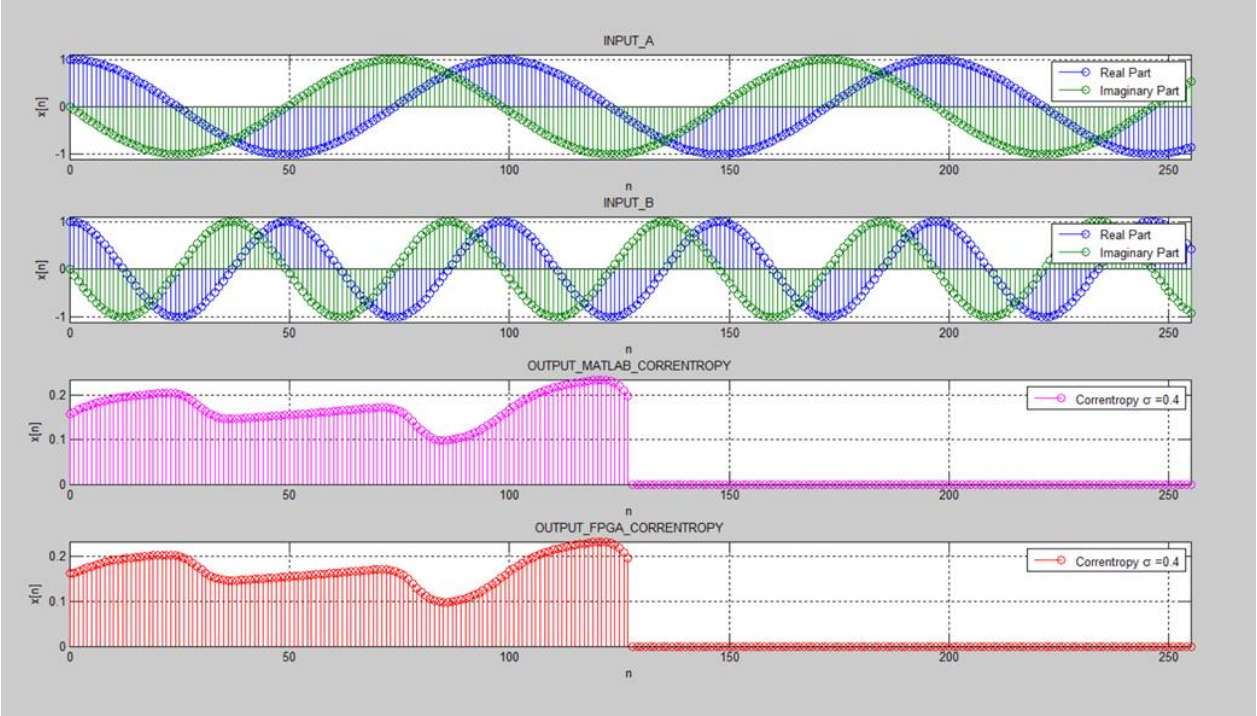

Figura A26: Salida 9 del *CorrentropyTor* según Tabla 3.1

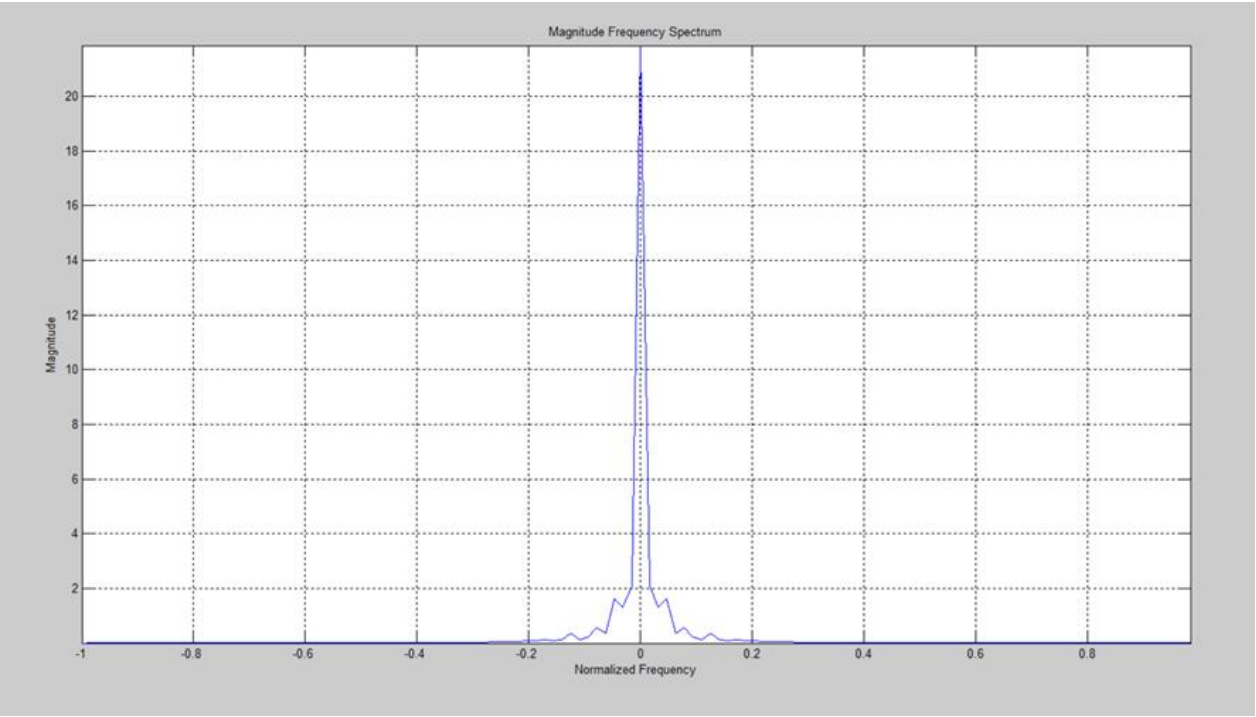

Figura A27: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 9, figura A26)

La figura A28 muestra el resultado correspondiente a la Salida 10 de la Tabla 3.1. En este caso hay más de una frecuencia involucrada. La entrada A tiene $FREQ1 = 2,6\,Hz$ y $FREQ2 = 23,2\,Hz$. La entrada B tiene la misma configuración: $FREQ1 = 2,6\,Hz$ y $FREQ2 = 23,2\,Hz$. El valor de sigma es de 2,0 (cinco veces Silverman) y se mantienen las 256 muestras.

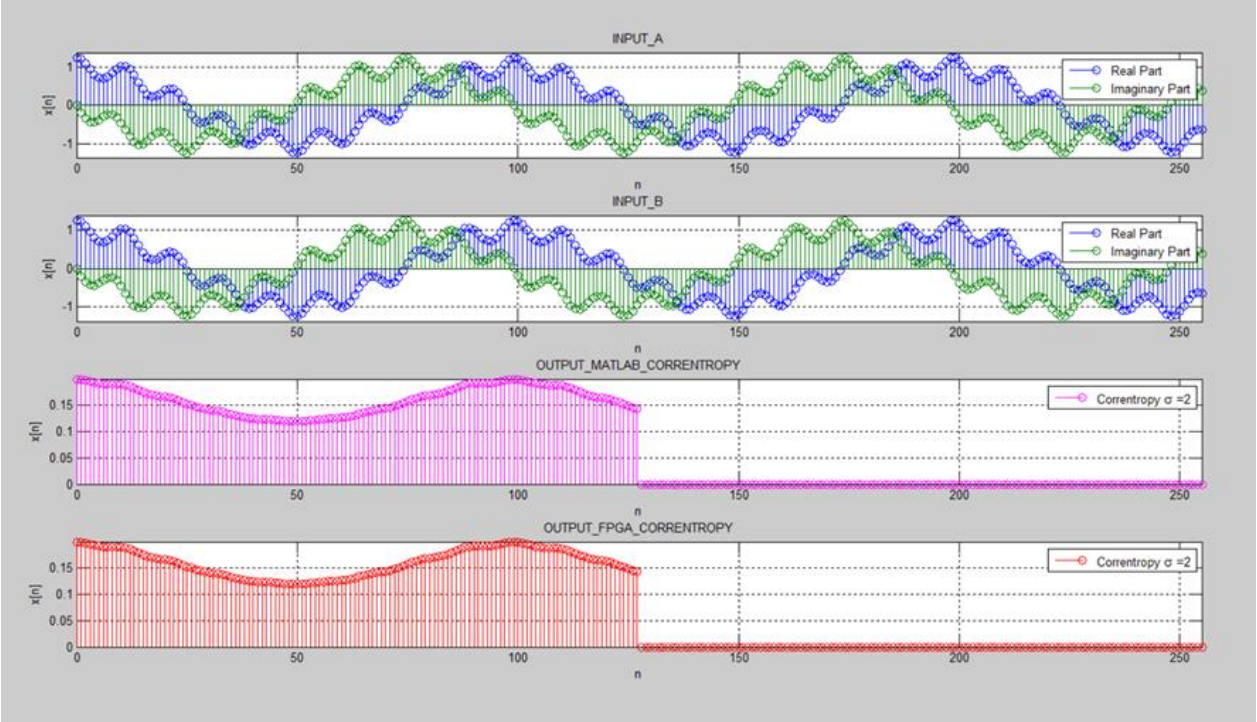

Figura A28: Salida 10 del *CorrentropyTor* según Tabla 3.1

La figura A29 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A28 donde se ve claramente la frecuencia fundamental para esta señal.

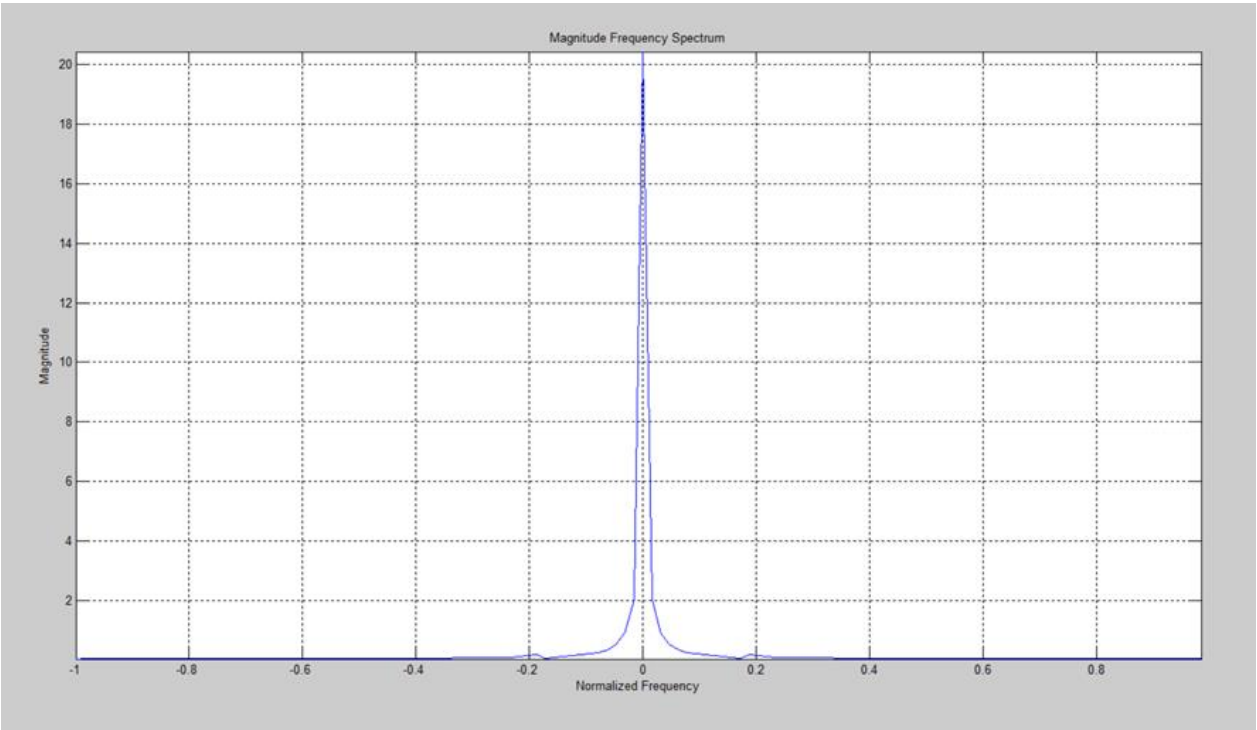

Figura A29: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 10, figura A28)

La figura A30 corresponde a la figura A28 a la cual se le ha restado el valor medio a las salidas de la Correntropía. Dado que estamos trabajando con un valor alto de sigma (2,0 = cinco veces Silverman) podemos comprobar que la señal de la Correntropía (salida FPGA) de la figura A30 sería muy similar a la salida (considerando solamente la parte real) de la Correlación obtenida en la figura A8 del Anexo A.5.

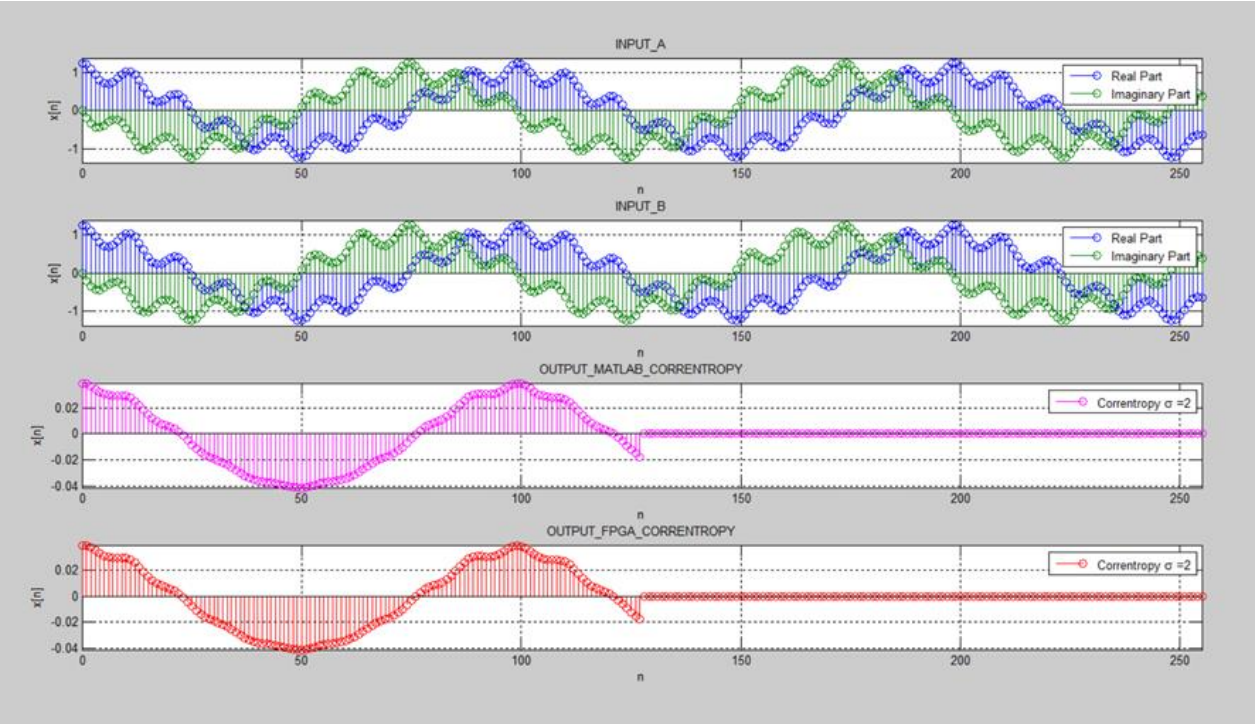

Figura A30: Salida 10 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía

La figura A31 muestra el resultado obtenido para la Salida 11 de la Tabla 3.1. En este caso lo único que cambia es sigma que toma el valor de 0,8 (dos veces Silverman). Igualmente, como en los casos anteriores, la salida obtenida con el diseño, corresponde exactamente a lo indicado por MATLAB.

La figura A32 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A31 donde se ve claramente la frecuencia fundamental para esta señal pero también comienzan a aparecer más harmónicas que en la figura A29.

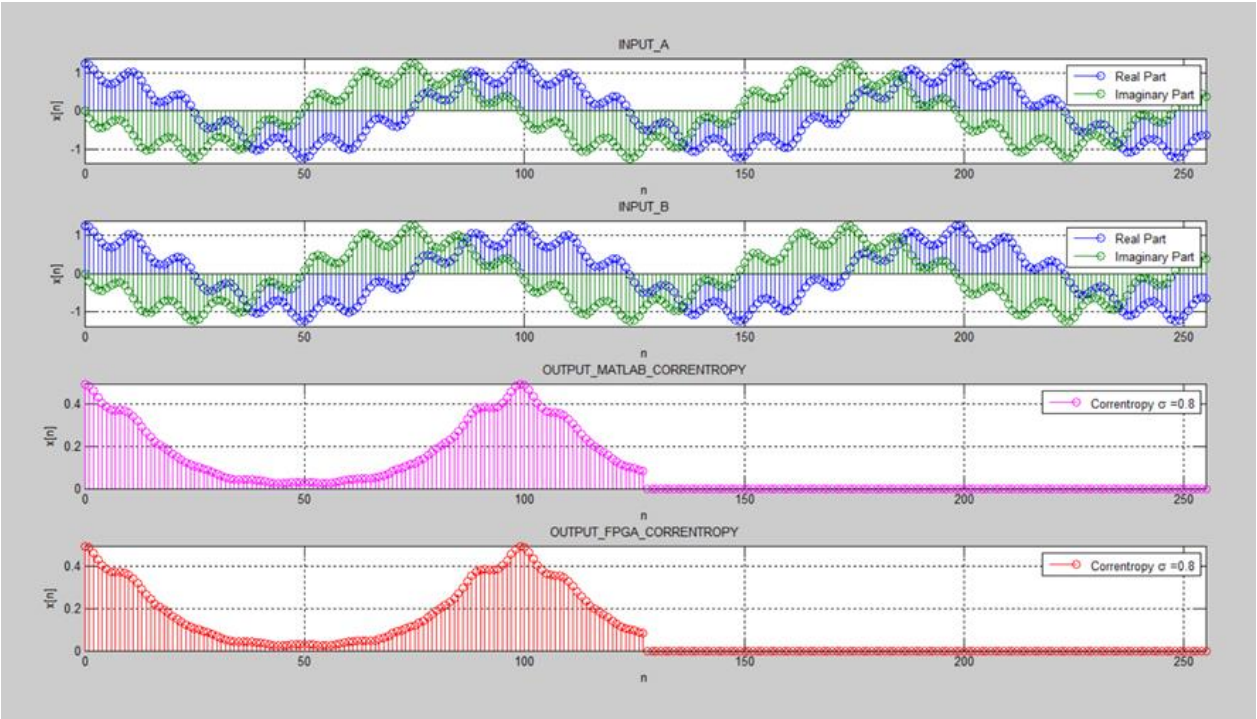

Figura A31: Salida 11 del *CorrentropyTor* según Tabla 3.1

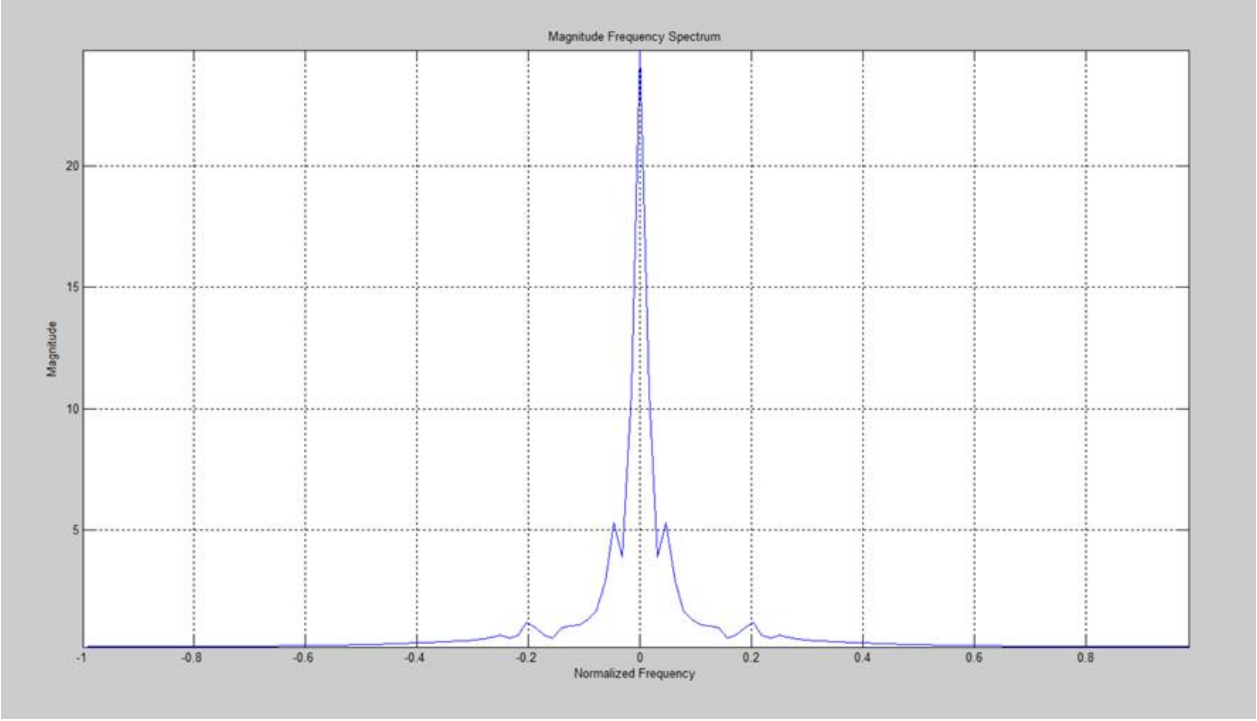

Figura A32: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 11, figura A31)

La figura A33 muestra el resultado obtenido para la Salida 12 de la Tabla 3.1. En este caso lo único que cambia es sigma que toma el valor de 0,4 (una vez Silverman). Igualmente, como en

los casos anteriores, la salida obtenida con el diseño, corresponde exactamente a lo indicado por MATLAB.

La figura A34 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A33 donde se ve claramente la frecuencia fundamental para esta señal.

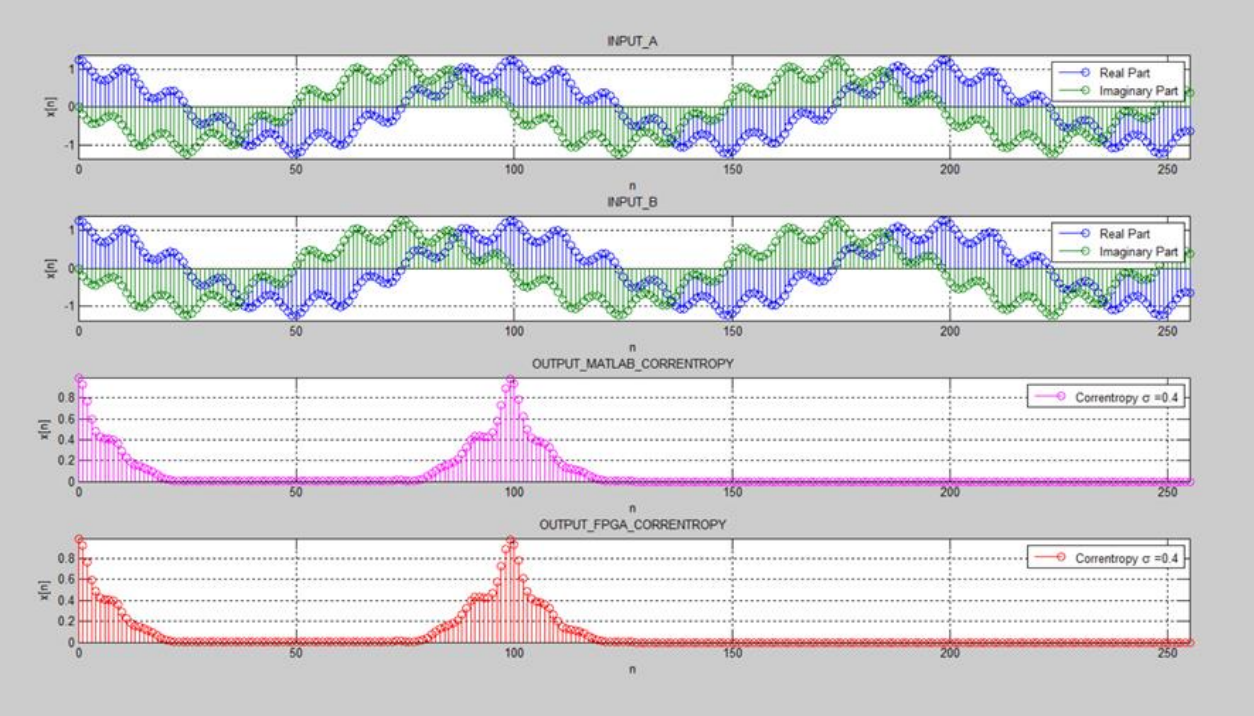

Figura A33: Salida 12 del *CorrentropyTor* según Tabla 3.1

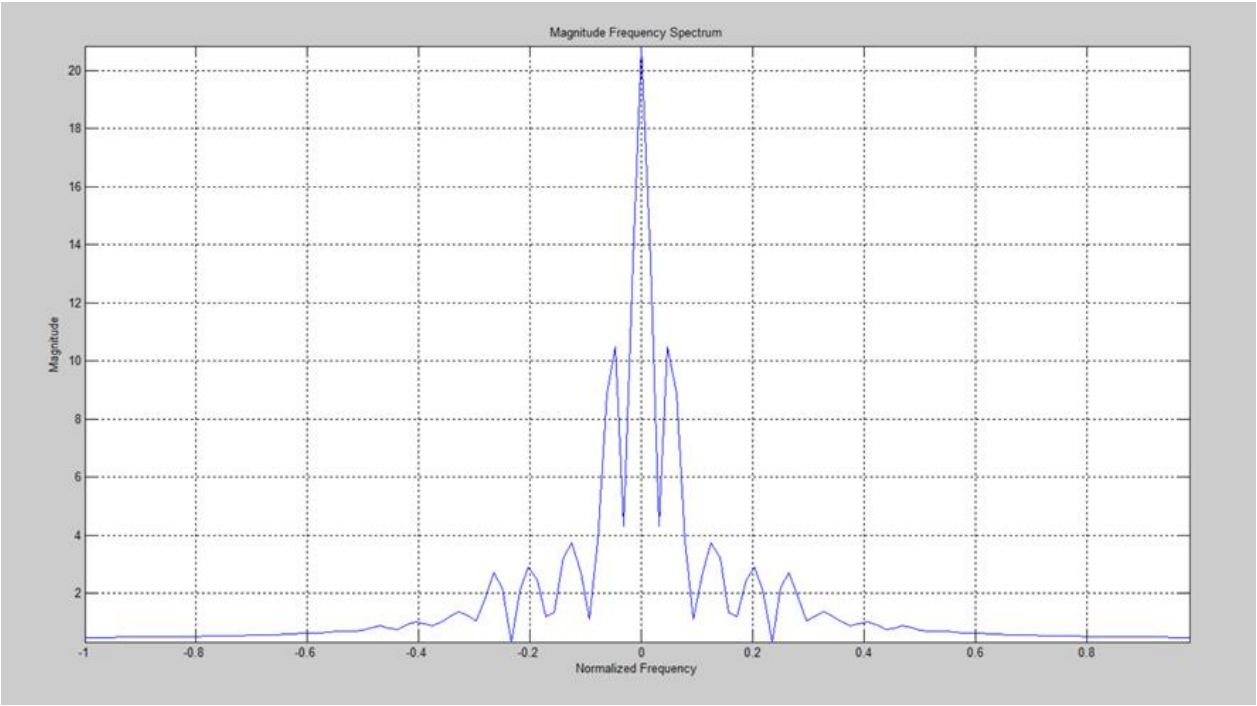

Figura A34: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 12, figura A33)

La figura A35 muestra el resultado correspondiente a la Salida 13 de la Tabla 3.1. En este caso hay más de una frecuencia involucrada y las entradas son diferentes. La entrada A tiene: $FREQ1 = 2,6\,Hz$ y $FREQ2 = 23,2\,Hz$. La entrada B tiene: $FREQ1 = 5,2\,Hz$ y $FREQ2 = 46,4\,Hz$. El valor de sigma es de 2,0 (cinco veces Silverman) y se mantienen las 256 muestras.

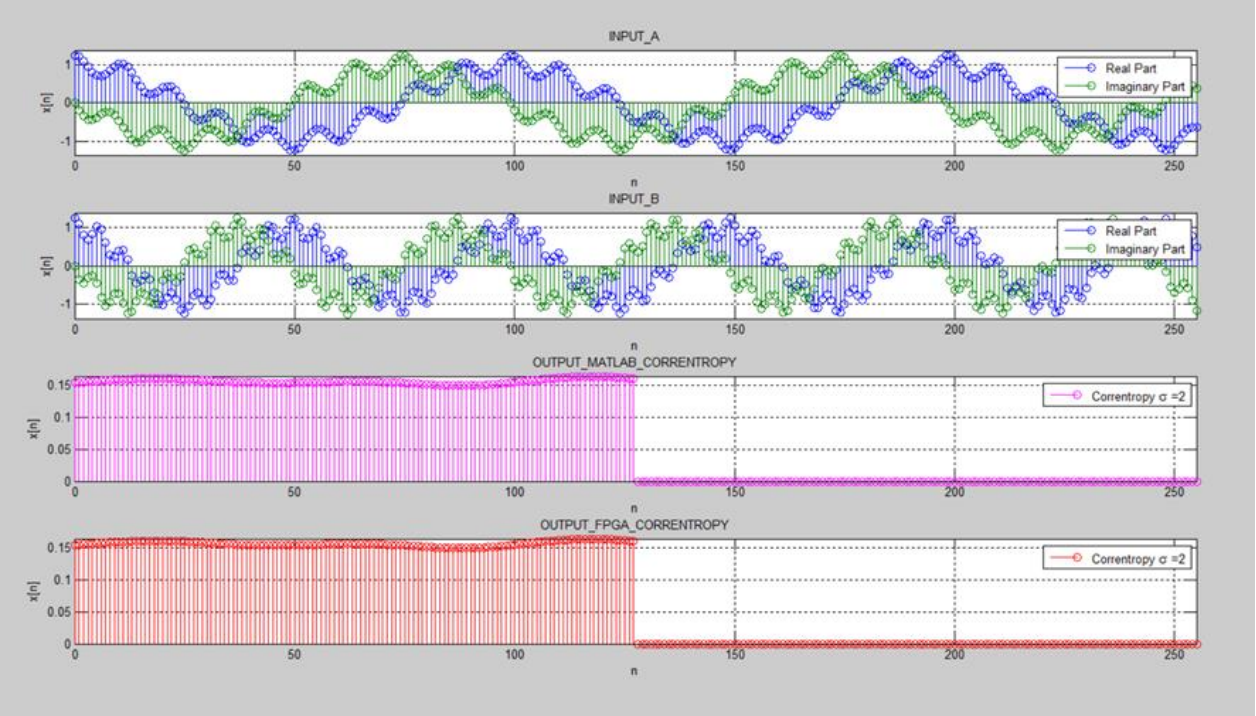

Figura A35: Salida 13 del *CorrentropyTor* según Tabla 3.1

La figura A36 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A35 donde se ve claramente la frecuencia fundamental para esta señal.

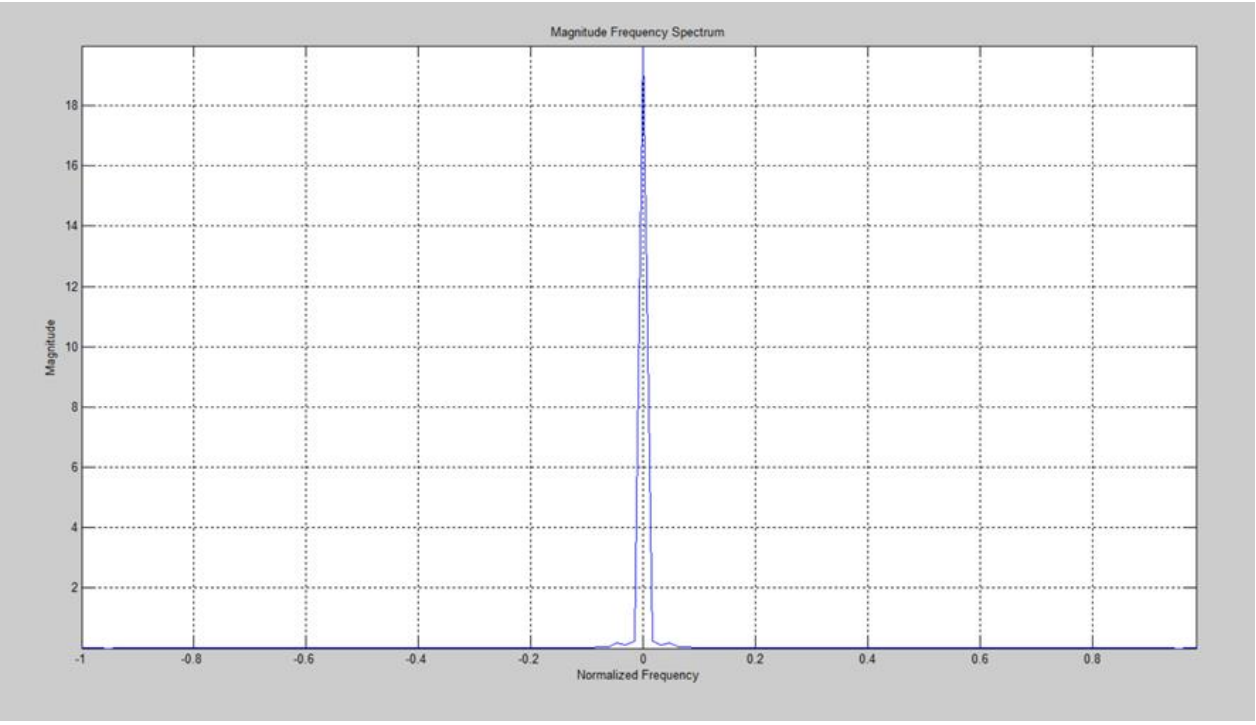

Figura A36: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 13, figura A35)

La figura A37 corresponde a la figura A35 a la cual se le ha restado el valor medio a las salidas de la Correntropía. Dado que estamos trabajando con un valor alto de sigma (2,0 = cinco veces Silverman) podemos comprobar que la señal de la Correntropía (salida FPGA) de la figura A37 sería muy similar a la salida (considerando solamente la parte real) de la Correlación obtenida en la figura A9 del Anexo A.5.

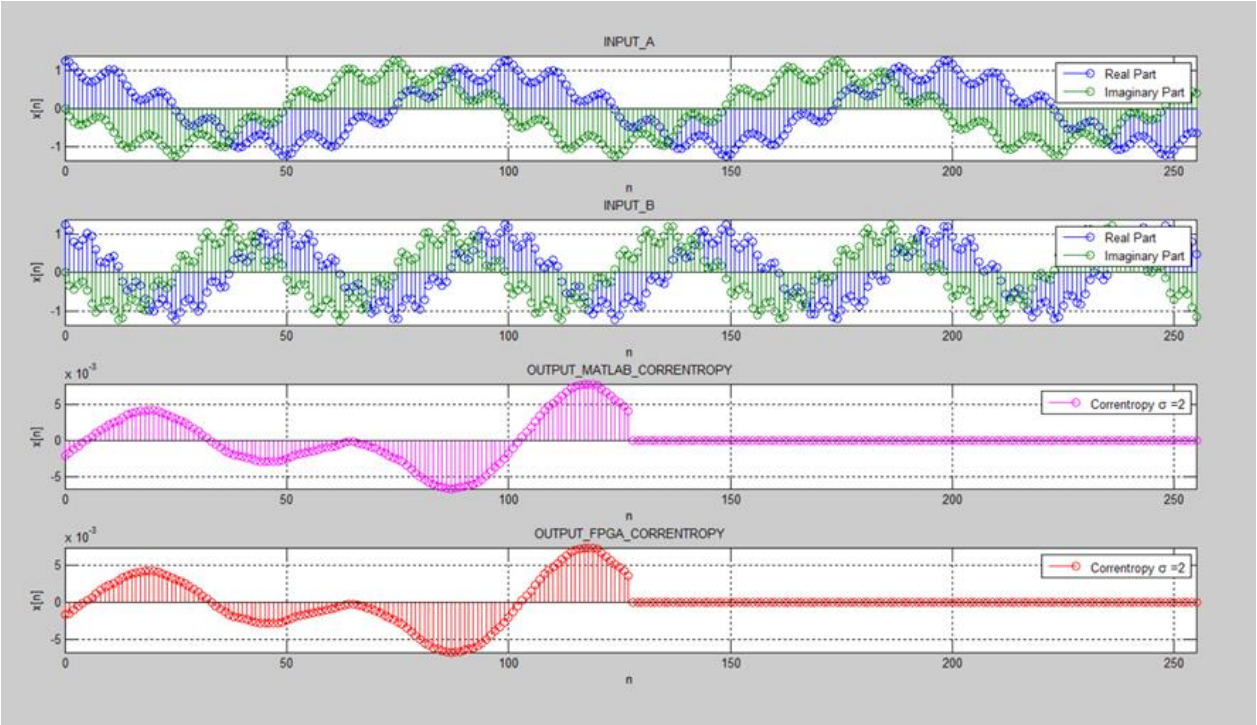

Figura A37: Salida 13 del *CorrentropyTor* según Tabla 3.1 con la media descontada para salida FPGA de la Correntropía

La figura A38 muestra el resultado obtenido para la Salida 14 de la Tabla 3.1. En este caso lo único que cambia es sigma que toma el valor de 0,8 (dos veces Silverman). Igualmente, como en los casos anteriores, la salida obtenida con el diseño, corresponde exactamente a lo indicado por MATLAB.

La figura A39 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A38 donde se ve claramente la frecuencia fundamental para esta señal.

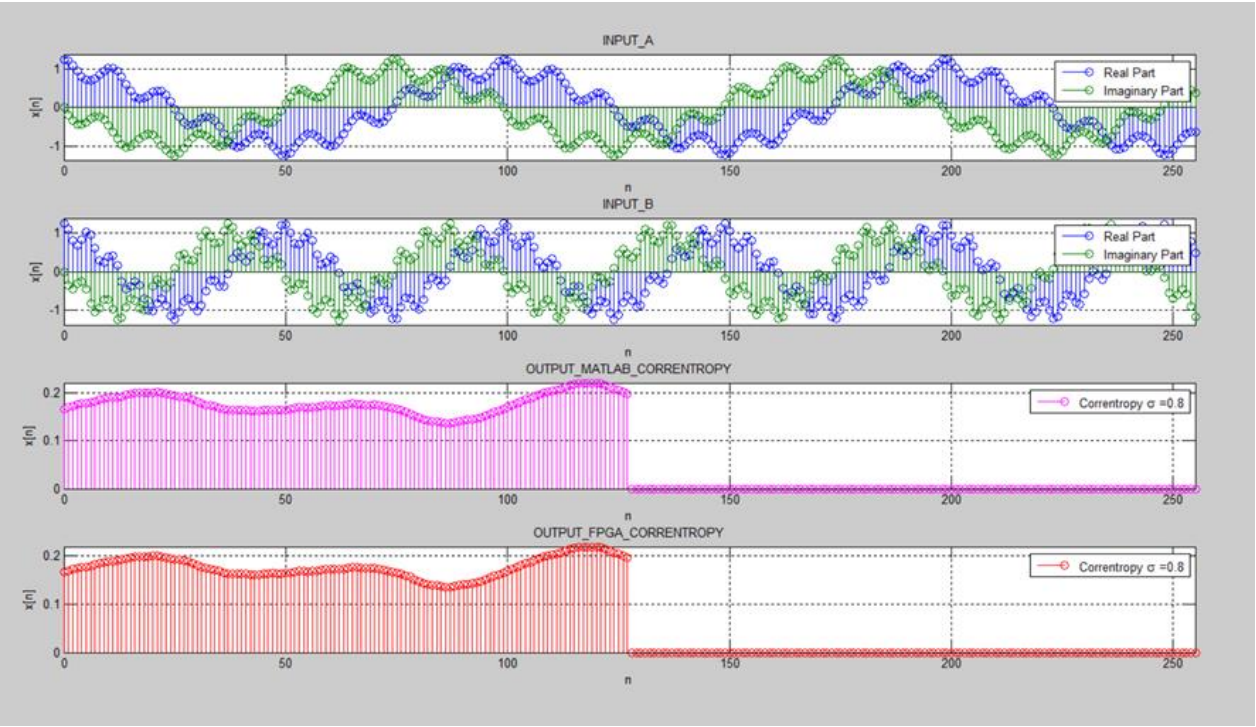

Figura A38: Salida 14 del *CorrentropyTor* según Tabla 3.1

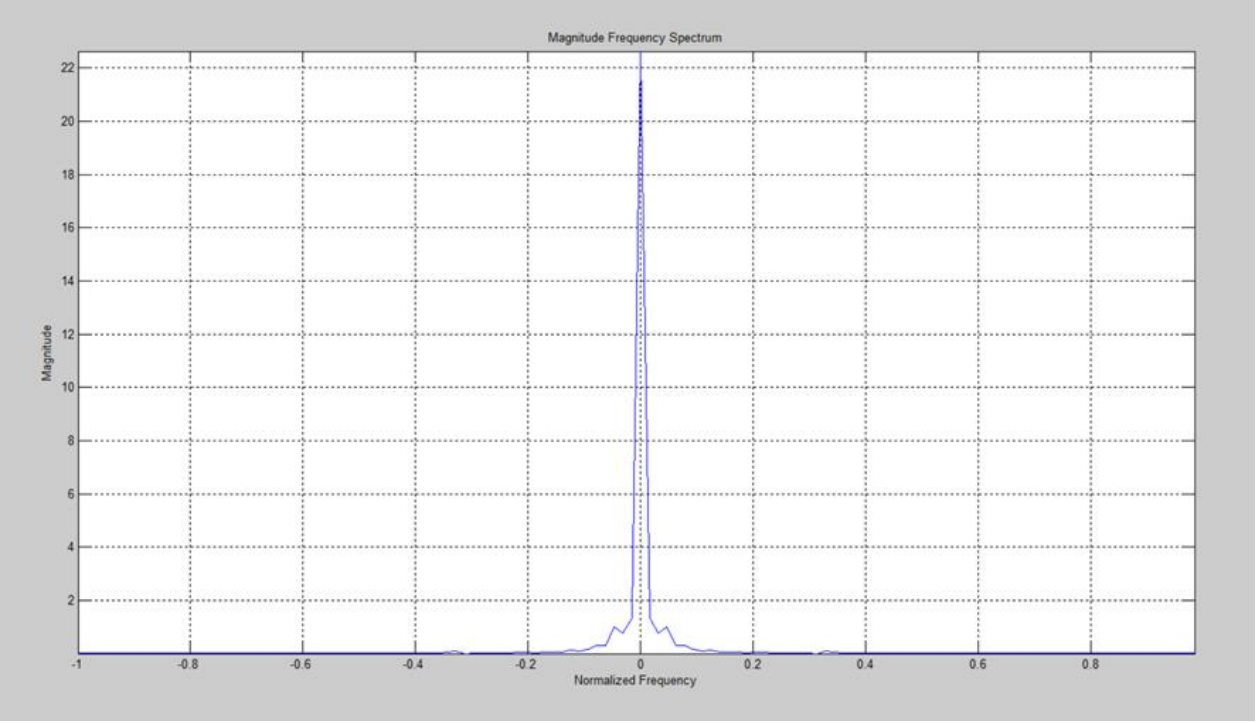

Figura A39: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 14, figura A38)

La figura A40 muestra el resultado obtenido para la Salida 15 de la Tabla 3.1. En este caso lo único que cambia es sigma que toma el valor de 0,4 (una vez Silverman). Igualmente, como en

los casos anteriores, la salida obtenida con el diseño, corresponde exactamente a lo indicado por MATLAB.

La figura A41 muestra la magnitud del espectro de frecuencia correspondiente a la salida FPGA de la figura A40 donde se ve claramente la frecuencia fundamental para esta señal.

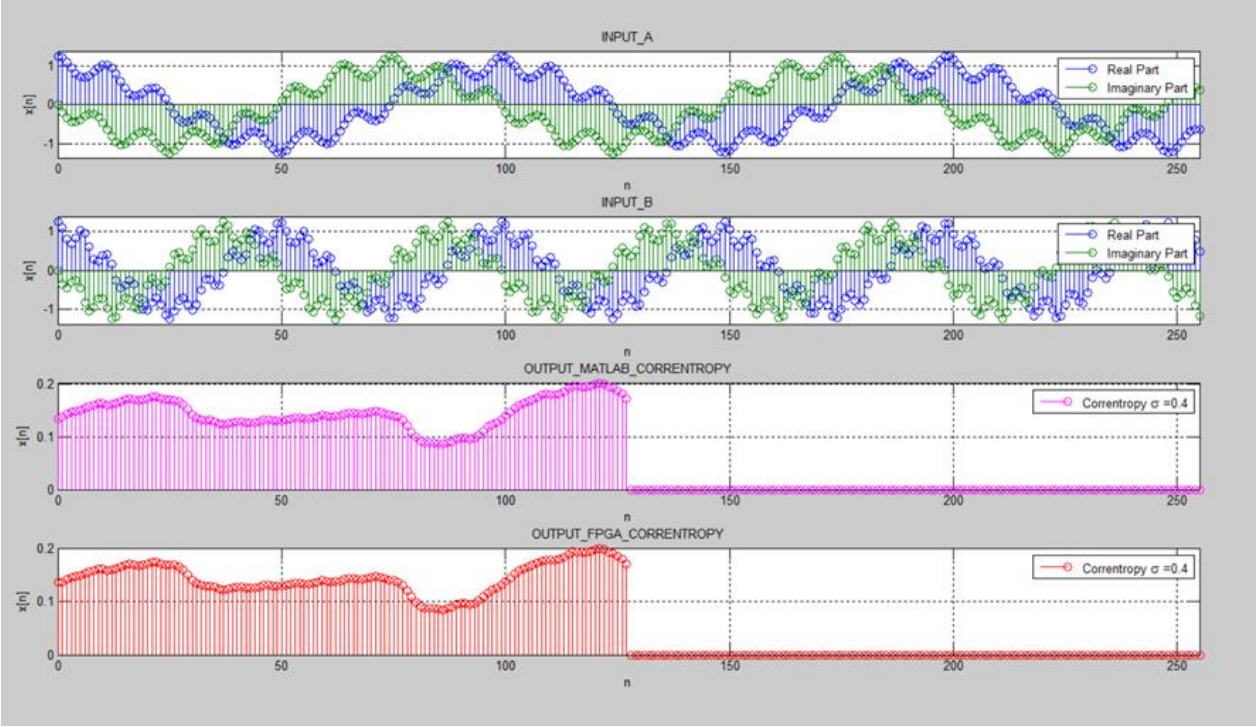

Figura A40: Salida 15 del *CorrentropyTor* según Tabla 3.1

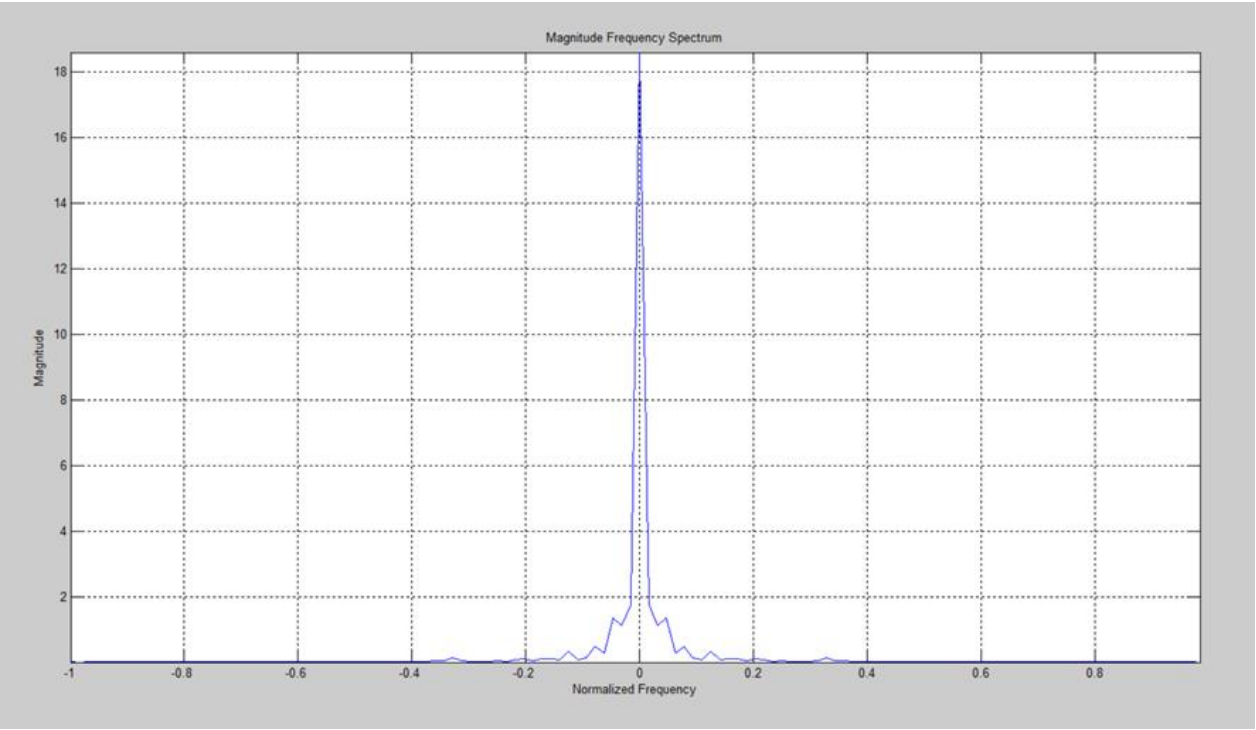

Figura A41: Magnitud del Espectro de Frecuencia de la Correntropía (salida FPGA de la Salida 15, figura A38)

Las figuras A42, A43 y A44 muestran las salidas 16, 17 y 18 de la Tabla 3.1, respectivamente, para 1024 muestras. La figura A42 corresponde a un sigma de 1,6 (cinco veces Silverman), la figura A43 corresponde a un sigma de 0,64 (dos veces Silverman) y la figura A44 corresponde a un sigma de 0,32 (una vez Silverman). Las simulaciones para obtener cada una de estas figuras tardaron más de 7 horas. Nótese que aunque las entradas de las salidas 16, 17 y 18 de la Tabla 3.1 son las mismas de las salidas 13, 14 y 15 de la Tabla 3.1, respectivamente, el sigma de Silverman obtenido es diferente; esto se debe a que la cantidad de muestras son diferentes en cada caso.

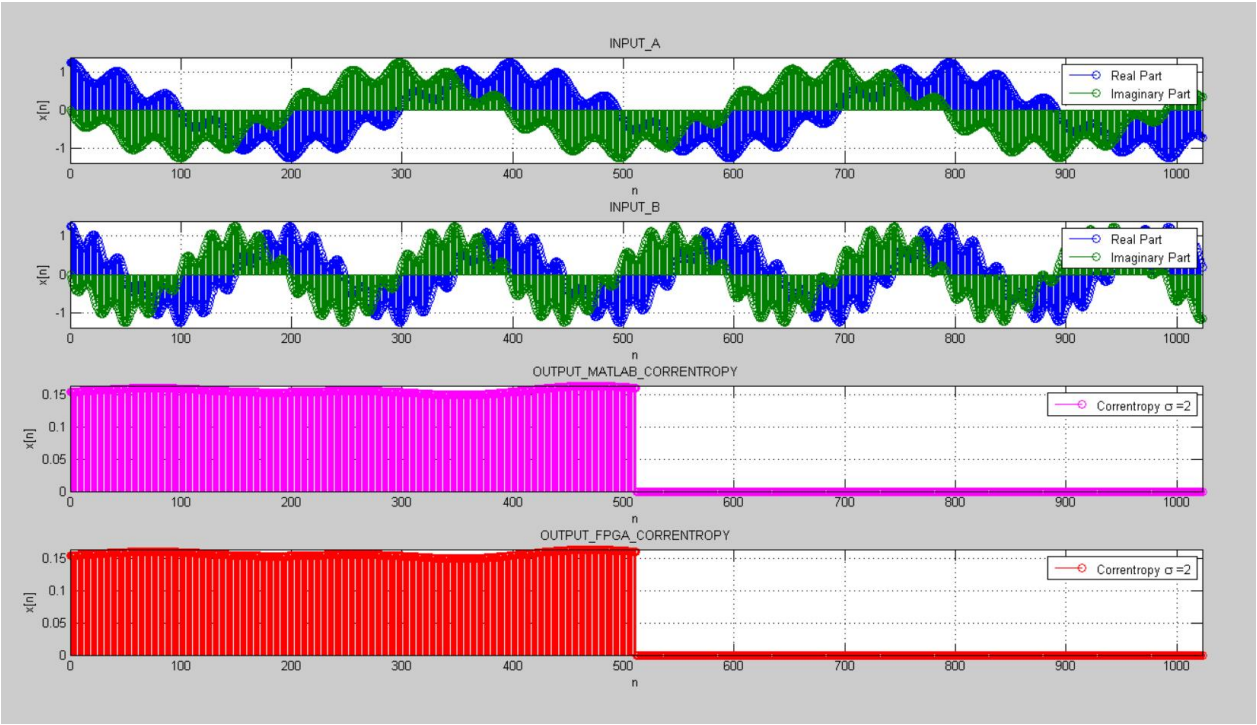

Figura A42: Salida 16 del *CorrentropyTor* según Tabla 3.1

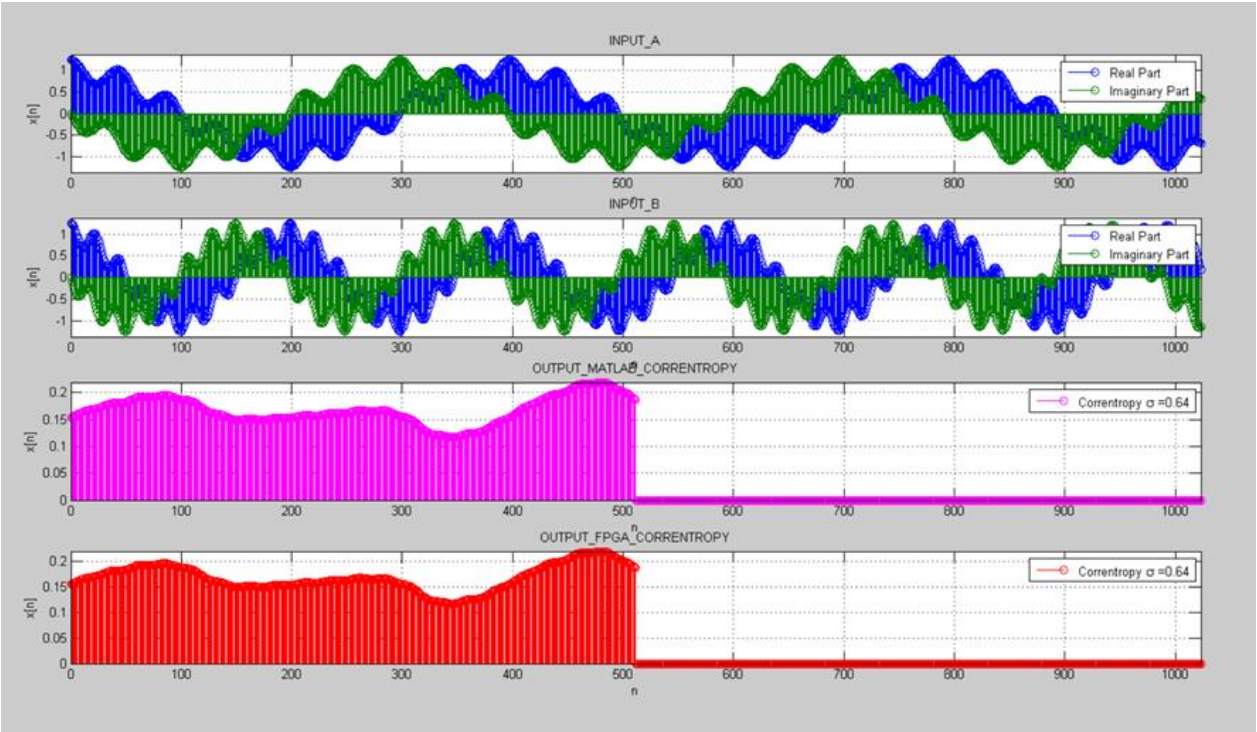

Figura A43: Salida 17 del *CorrentropyTor* según Tabla 3.1

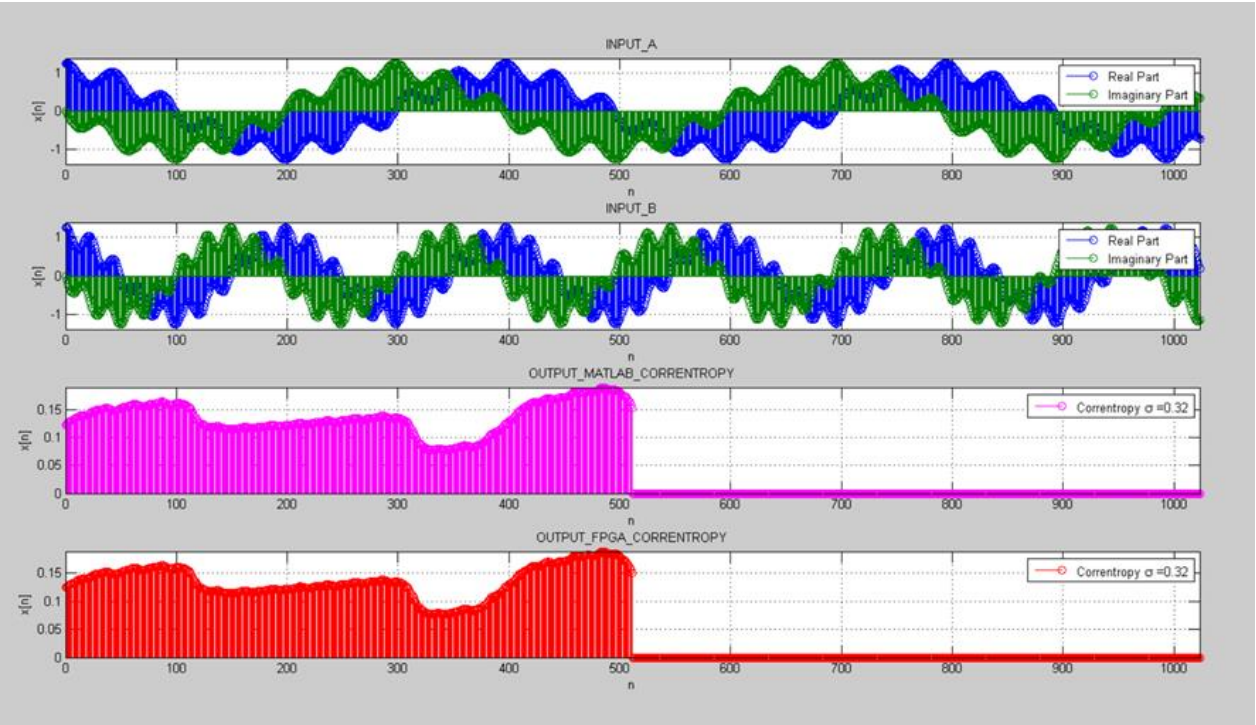

Figura A44: Salida 18 del *CorrentropyTor* según Tabla 3.1

Anexo A.9: Implementación de la Función Correntropía en MATLAB

A continuación se entrega la implementación en MATLAB de la función Correntropía en base a las ecuaciones 1 y 2 dadas en el punto 2 de la Introducción.

```
% Correntropy
sgk = input('Ingrese SIGMA del Kernel: ');    % SIGMA KERNEL
%% GENERA FUNCION CORRENTROPIA
%tic
A_ret_z = zeros(3*L,1);
A_ret = complex(A_ret_z,0);
InA_ret_z = zeros(L,1);
InA_ret = complex(InA_ret_z,0);
InA = complex(B(:,6),B(:,7));
[n,m] = size(InA);
InB = complex(B(:,8),B(:,9));
A_ret(L+1:2*L,1) = InA;
sg = sgk; % SIGMA KERNEL
P = [InA InB];
sigma_silv = 1*silverman(P); % SIGMA KERNEL SILVERMAN
rtemp = zeros(L,1); cross_corr = zeros(L,1);
q = 1;
for p = 0:L-1
    InA_ret = A_ret(L+1+p:2*L+p,1);
    e = (InA_ret - InB);
    ne = real(e).*real(e) + imag(e).*imag(e); %e.*conj(e);
    narg_exp = ne/(2*sg*sg);
    rtemp = exp(-(ne/(2*sg*sg)));   %exp(-((ne)*ones(m,1))/(2*sg*sg));
    cross_corr(q,1) = sum(rtemp(1:L-p,1),'double')/((2*pi*sg*sg)^(m/2));  %Correntropia Cruzada
    q = q + 1;
end
for r = 1:round(0.5*L)    % L/2
    B(r,14) = cross_corr(r,1)/(L - r + 1);
end
```

```
function s = silverman(P)
[N,dim] = size(P);
mtxCov = sqrt(sum(diag(cov(P)))/dim);
s = (mtxCov*(4/((dim+2)*N))^(1/(dim+4)));
fprintf('     Sigma de Silverman:      %f\n',s);
end
```

Anexo A.10: Representación de Números Binarios en Formato Punto-Fijo

Para la representación de números en formato Punto-Fijo, existen varias definiciones de formatos [2][35]. Uno de estos formatos es el XQN que representa un número binario en complemento de 2 de $1+X+N$ bits. Esto quiere decir: un bit de signo seguido de X bits, representando un número entero, y N bits que representan la parte fraccionaria (mantisa). Este formato, XQN, puede ser utilizado para expresar números en el rango $[-2^X]$ a $[2^X - 2^{-N}]$.

Existe también una notación equivalente llamada formato del tipo "System Generator Fix" y definida como "Fixword_length_fractional_length". Se expresa normalmente como $fix_{(1+X+N)\_N}$.

Un número utilizando el formato $Q15$ es equivalente a un número utilizando la representación fix_{16_15} y un número en el formato $1Q15$ es equivalente a un número utilizando la representación fix_{17_15}. En este documento utilizaremos este último formato.

Las Tablas A1 y A2 muestran ejemplos de números representados en formato XQN o $fix_{(1+X+N)\_N}$.

	(signo) Bit 8	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
$+1$	0	1	0	0	0	0	0	0	0
-1	1	1	0	0	0	0	0	0	0
$+\frac{\pi}{4}$	0	0	1	1	0	0	1	0	0
$-\frac{\pi}{4}$	1	1	0	0	1	1	0	1	1
			Bits Parte Fraccionaria						

Tabla A1: Ejemplo de Números Binarios con Formato $1Q7$ o fix_{9_7}

	(signo) Bit 8	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
$+1$	0	0	1	0	0	0	0	0	0
-1	1	1	1	0	0	0	0	0	0
$+\pi$	0	1	1	0	0	1	0	0	1
$-\pi$	1	1	0	1	1	0	1	1	1
				Bits Parte Fraccionaria					

Tabla A2: Ejemplo de Números Binarios con Formato $2Q6$ o fix_{9_6}

Anexo A.11: Conversión de un Diagrama de Flujo a un Diagrama MDS

El Diagrama MDS (*Mnemonic Documented State diagram*) [11] es un diagrama de estado documentado con nemónicos, sin elementos polarizantes, los cuales solamente simbolizan condiciones ACTIVAS (*ASSERTED*) o NO-ACTIVAS (*NOT-ASSERTED*), independiente de los niveles de voltaje. Una vez en la etapa de implementación en hardware, se pueden agregar los elementos polarizantes con sus niveles de voltaje correspondientes.

El Diagrama MDS es equivalente a lo que se conoce en los textos como diagrama ASM [4] (*Algorithmic State Machine*). Entre las características y ventajas de utilizar un Diagrama MDS, se destacan:

- Es una extensión del Diagrama de Estado utilizado en diseño Sistemas Digitales basado en "Flip-Flops" y compuertas [28].
- Generalmente es menos voluminoso que un Diagrama de Flujo.
- Es un diagrama más simple y más fácil de entender que un diagrama ASM.
- Corresponde al paso anterior a la fase inicial de la implementación de hardware.
- Los arcos en un Diagrama MDS, en general, corresponde a expresiones booleanas en vez de alfabetos de entrada.
- Una implementación para un Controlador en un Lenguaje de Descripción de Hardware (HDL), se obtiene directamente del Diagrama MDS.

Las figuras A45 y A46 muestran ejemplos de la simbología utilizada en un Diagrama MDS. En la figura A45 si la expresión Booleana: $\overline{START} \cdot \overline{READY}$ es verdadera entonces el Controlador permanece en el estado 0. Si, en cambio, la expresión $START \cdot \overline{READY}$ es verdadera entonces el Controlador realiza una transición desde el estado 0 al estado X.

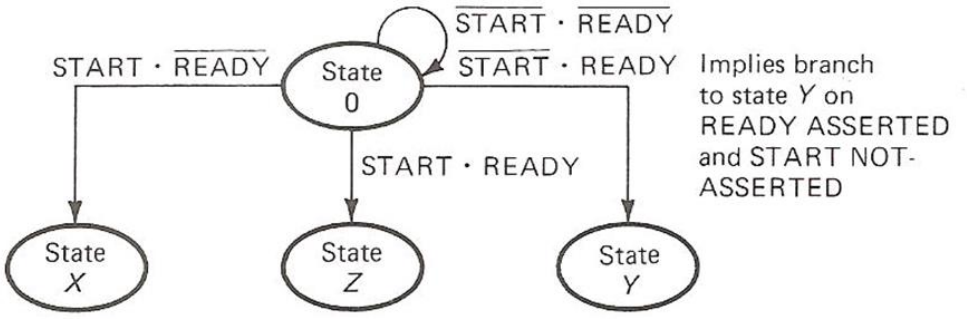

Figura A45: Ejemplo Simbología utilizada en el Diagrama MDS

En la figura A46 se muestran varios nemónicos y símbolos que se describen a continuación:

1. **GATCMD** ↑: señal asociada al estado X y significa que la salida $GATCMD$ se activa (*ASSERTED*) cuando el Controlador entra al estado X. De acuerdo a la figura A46, esta señal es desactivada (*DE-ASSERTED*) cuando se entra al estado S.
2. **RUN** ↑↓: simboliza que la salida RUN es activada (↑) cuando el Controlador entra al estado R y se desactiva (↓) cuando el Controlador deja el estado R.

3. *****: el asterisco $(*)$ mostrado en el estado X es un descriptor especial que indica que la transición desde el estado X es controlada por una variable ($STAK$ en la figura A46) que no está implícita o explícitamente sincronizada con un reloj. Por lo tanto, es definida como una variable asíncrona. En ese caso, se deben considerar asignaciones de estados especiales para los estados siguientes de los estados cuyas transiciones están controladas por entradas asíncronas.
4. **SOC** $\uparrow\downarrow=$ $(State\, S \cdot EOC \cdot \overline{(CLK)}$: este tipo de simbología es utilizada para expresar salidas condicionales. En este caso, la salida SOC es activada condicionalmente en el estado S. Las condiciones en este caso son: la entrada EOC debe estar activada y el reloj CLK debe estar desactivado.

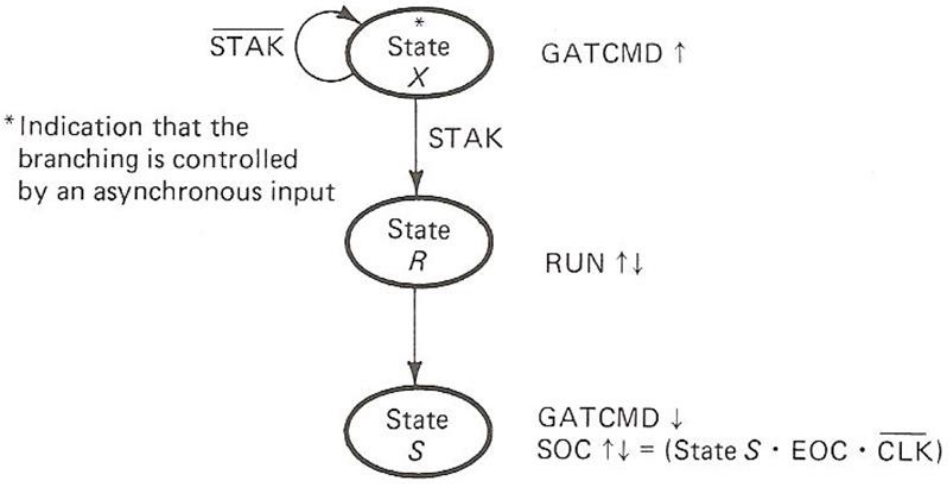

Figura A46: Ejemplo Simbología utilizada en el Diagrama MDS

Conceptos vinculados a la Construcción de un Diagrama MDS desde un Diagrama de Flujo

Concepto 1 : un bloque de proceso de un Diagrama de Flujo, corresponde a un estado en un Diagrama MDS. Esto se muestra en las figuras A47 y A48.

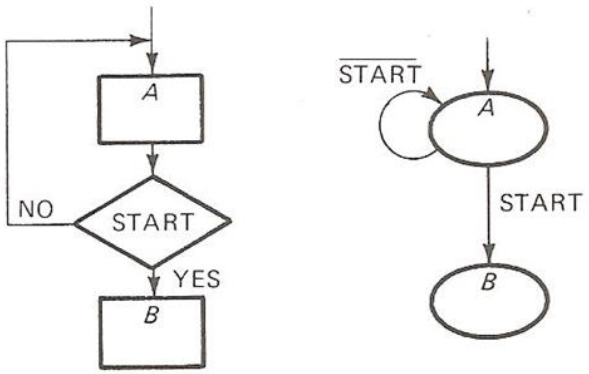

Figura A47: Ejemplo 1 de Conversión Bloques de Proceso a Diagrama MDS

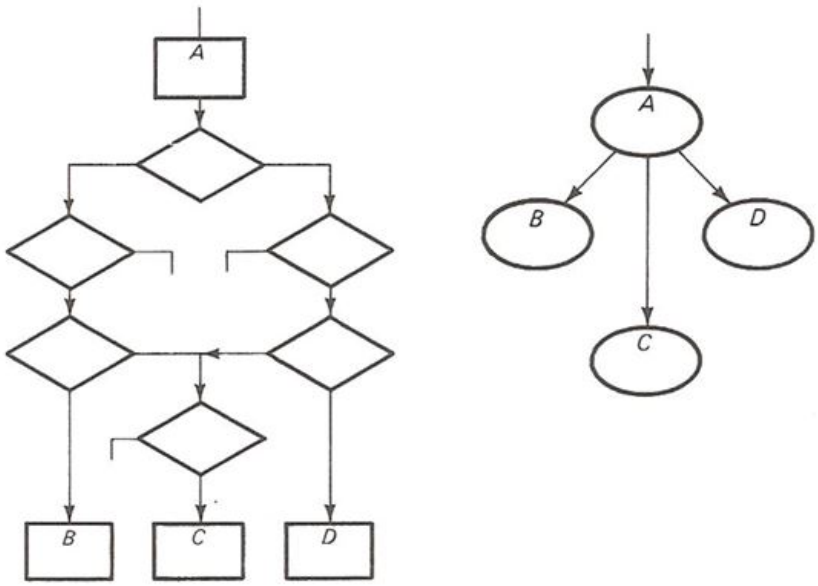

Figura A48: Ejemplo 2 de Conversión Bloques de Proceso a Diagrama MDS

Concepto 2 : las expresiones para las ramas del Diagrama MDS, resultan de los caminos de decisión del Diagrama de Flujo. Esto se muestra en las figuras A49 y A50.

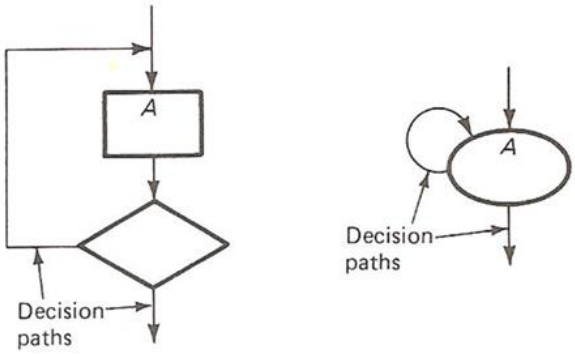

Figura A49: Caminos de Decisión

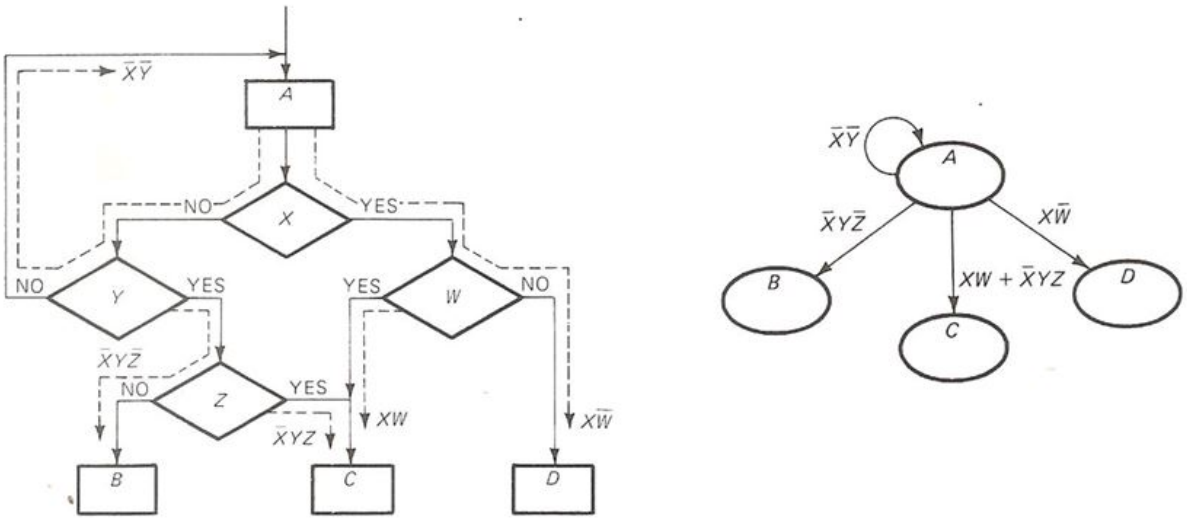

Figura A50: Múltiples Caminos de Decisión

Concepto 3 : se deben evitar caminos de decisión que involucren a más de una variable de decisión asíncrona. Por ejemplo, en el Diagrama de Flujo y en el Diagrama MDS que se muestran en la figura A51, hay dos variables asíncronas ($DATA$ y $TERM$) marcadas con un asterisco ($*$). Para evitar problemas (evitar las llamadas *carreras críticas*), se define un nuevo estado (D) como lo muestra la figura A52.

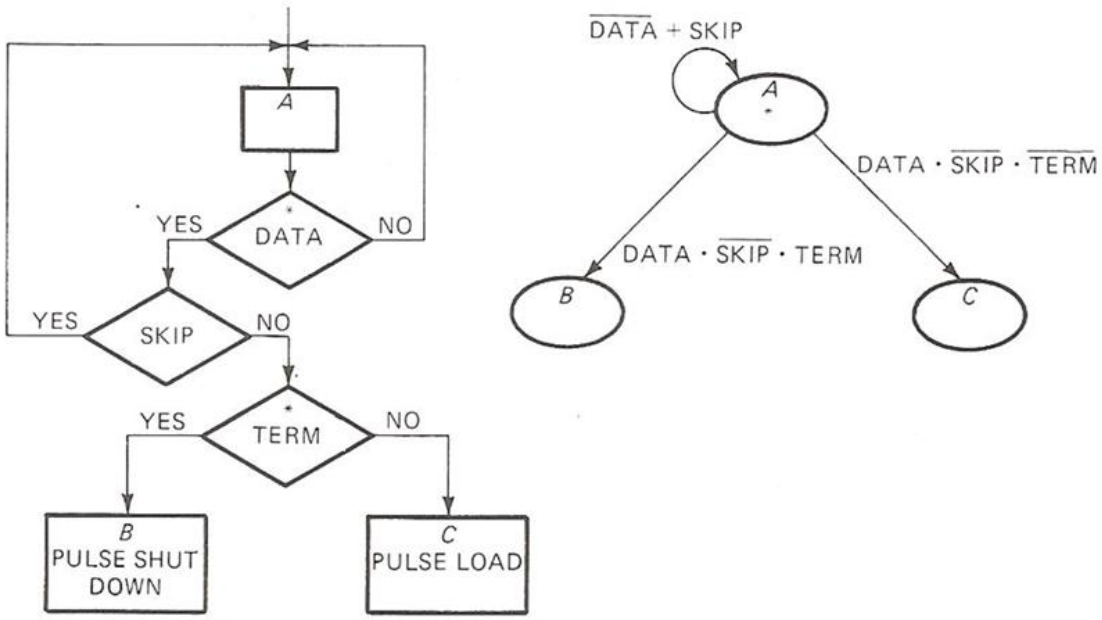

Figura A51: Diagramas de Flujo y MDS con dos Entradas Asíncronas

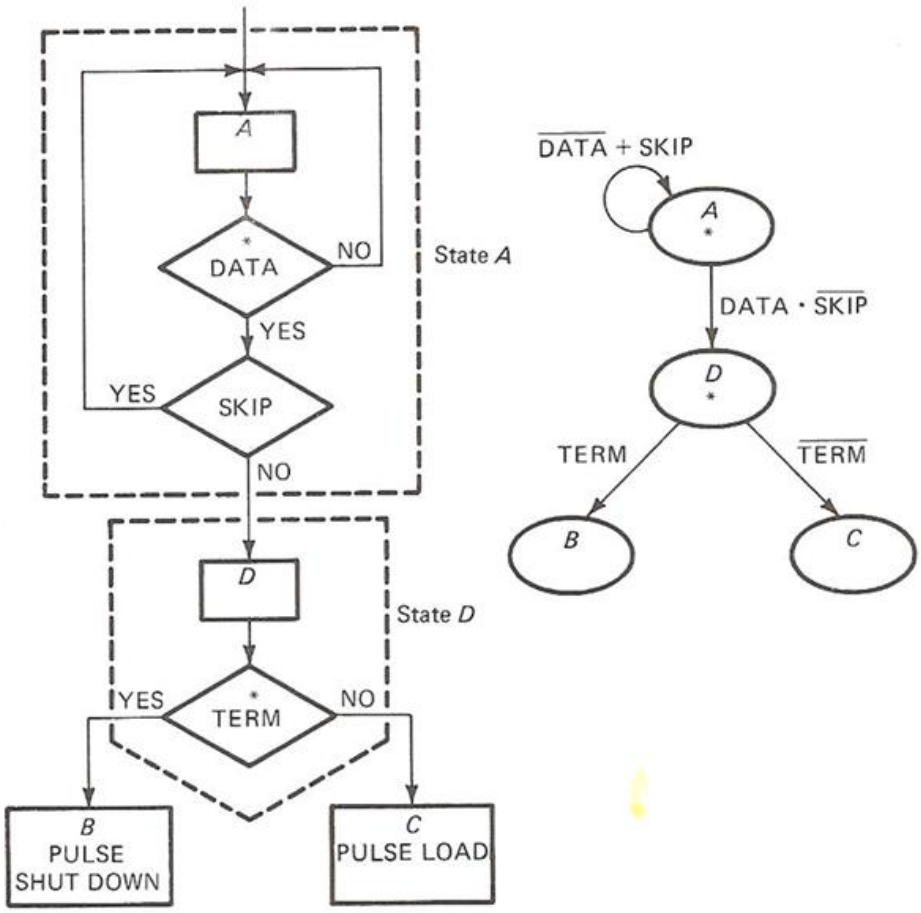

Figura A52: Diagrama de Flujo y MDS con dos Entradas Asíncronas sin *Carreras Críticas*

Concepto 4 : la generación de salidas deben ser esquematizadas de acuerdo a la acción a realizar:

- Salidas Incondicionales (inmediatas). Se muestra en la figura A53.
- Salidas Condicionales (dependiente de las entradas). Se muestra en la figura A54.

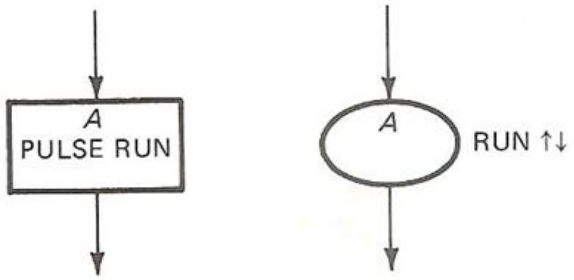

Figura A53: Especificación de una Salida Incondicional

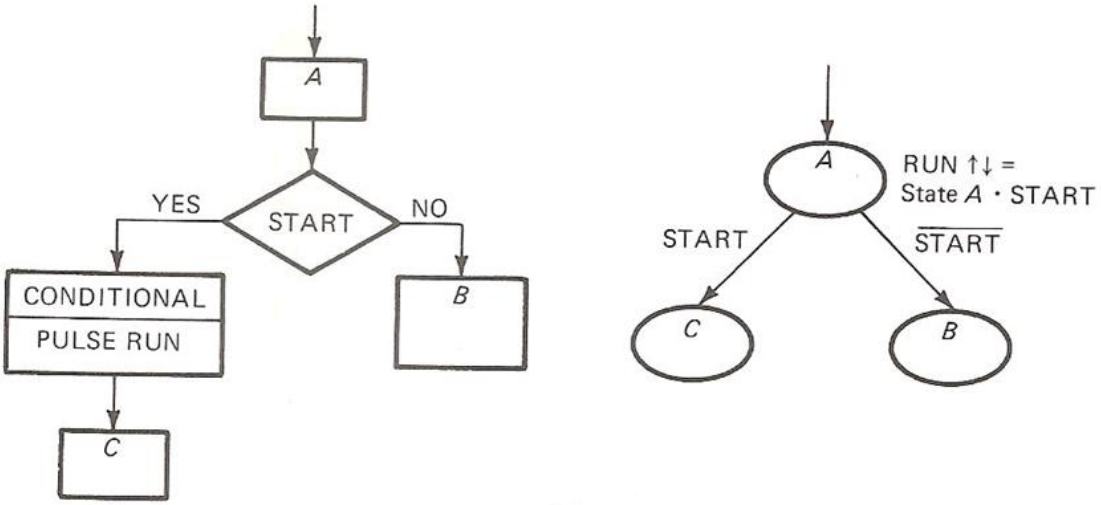

Figura A54: Especificación de una Salida Condicional

Concepto 5 : las salidas incondicionales y condicionales pueden ser especificadas dependiendo de un tiempo de duración en una variable de decisión, que se ACTIVA (*ASSERTED*) y "espera" por una respuesta. Esto se muestra en las figuras A55 y A56.

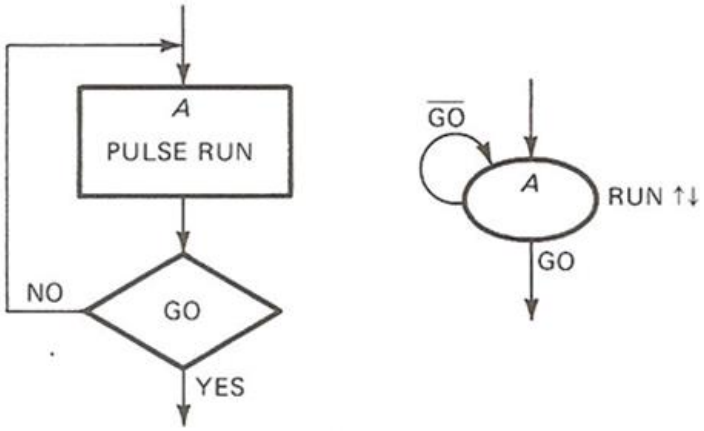

Figura A55: Salida Incondicional con una Dependencia del Tiempo de Duración de una Entrada

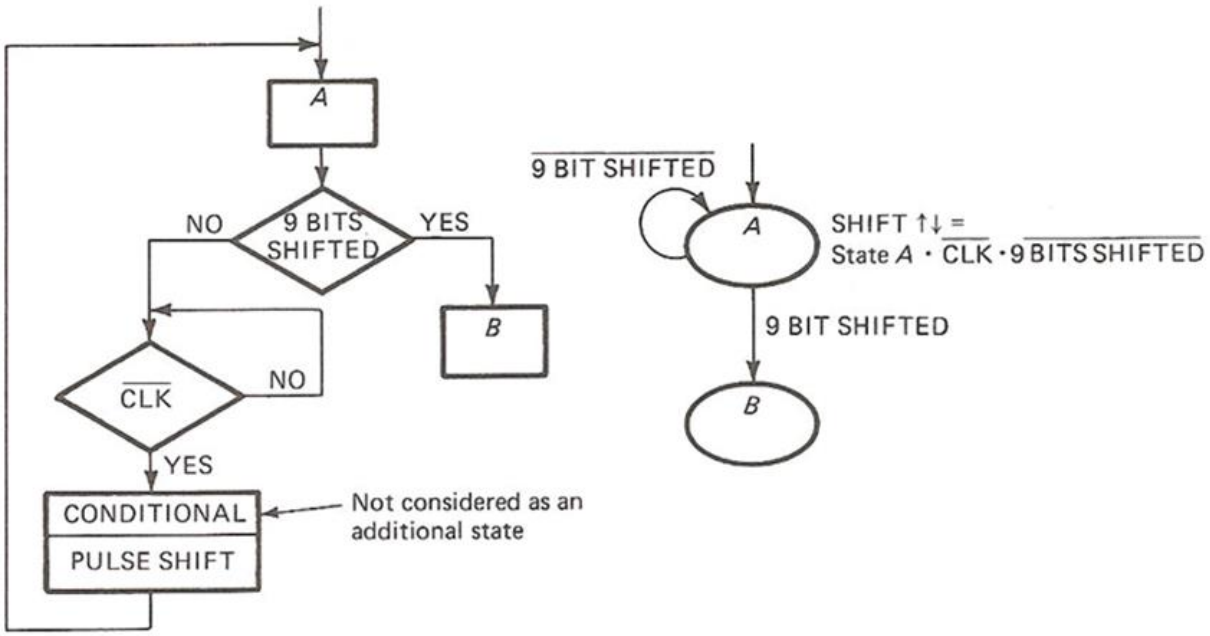

Figura A56: Salida Condicional con una Dependencia del Tiempo de Duración de una Entrada

Anexo A.12: Programa en SystemVerilog del CONTROLLER del *CorrentropyTor*

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Universidad de Chile
// Engineer: Francisco Rivera
//
// Create Date: 24.07.2016 21:29:07
// Design Name: CorrentropyTor
// Module Name: Controller
// Project Name: Tesis Magister
// Target Devices:
// Tool Versions: 2017.1
// Description: CONTROLLER module of CorrentropyTor
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module Controller #( parameter ADDRESS_WIDTH = 8, // addresses lines determining number of words in ram
                               MANT_WIDTH    = 13,
                               DATA_WIDTH    = 16,
                               WORD_WIDTH    = 32,  // number of bits in word
                               MAX_SAMPLES   = 2**ADDRESS_WIDTH,
                               real PI       = 3.14159265358979323846264338327950288419716939937511,
                               real PI_4     = PI/4,
                               real SIGMA    = 0.75,
                               real RC2PI    = 2.5066282731003,
                               real MAX_DIV  = (8-2**(-18))/(2*SIGMA*SIGMA),
                               real FACTOR   = 2*PI/(MAX_DIV)
                   )
        // IO ports
                   (input logic  aclk, reset, start, empty, full,
                                 s_axis_phase_tready, m_axis_dout_tvalid,
                                 o_complete_arg_re, o_complete_arg_im,
                                 o_complete_narg, o_complete_sigma_f2,
                                 o_complete_rc2pi, o_complete_corr,
                                 o_complete_facpi, o_complete_rc2pixsigxsmpl,
 (* dont_touch = "yes"*) input logic  ovr_arg_re, ovr_arg_im, ovr_sig2, ovr_rc2pi, o_overflow_narg,
                                 o_overflow_corr, ovr_facpi, ovr_rc2pixsigxsmpl,
                    input logic [2*WORD_WIDTH-1:0] read_data,
                    input logic signed [2*WORD_WIDTH-1:0] m_axis_dout_tdata,
```

```
                    input logic signed [DATA_WIDTH-1:0] arg_exp_re_fix,
                    input logic signed [DATA_WIDTH-1:0] arg_exp_im_fix,
                    input logic signed [WORD_WIDTH-1:0] arg_exp_fix2,
                    //input logic signed [WORD_WIDTH-1:0] narg_exp_div,
 (* dont_touch = "yes"*) input logic signed [WORD_WIDTH-1:0] narg_exp_facpi,
                    input logic signed [WORD_WIDTH-1:0] cosh_sinh_fix,
                    input logic signed [WORD_WIDTH-1:0] suma_rtemp_fix,
                    input logic signed [WORD_WIDTH-1:0] out_corrpy_fix,
                    input logic signed [DATA_WIDTH-1:0] A_data_table_re [0:2**ADDRESS_WIDTH-1],
                    input logic signed [DATA_WIDTH-1:0] A_data_table_im [0:2**ADDRESS_WIDTH-1],
                    input logic signed [DATA_WIDTH-1:0] B_data_table_re [0:2**ADDRESS_WIDTH-1],
                    input logic signed [DATA_WIDTH-1:0] B_data_table_im [0:2**ADDRESS_WIDTH-1],
                    input logic [WORD_WIDTH-1:0] SIGMA_F2,
                    output logic aresetn, enable, read, write, s_axis_phase_tvalid,
                                 m_axis_dout_tready, corr_out, i_start_arg_re, i_start_arg_im,
                                 i_start_narg, i_start_sigma_f2, i_start_rc2pi, i_start_corr,
                                 i_start_facpi, i_start_rc2pixsigxsmpl,
                    output logic [WORD_WIDTH-1:0] SIGMA_F, dosxSIGMA_F2, FACPI_32_18,
                    output logic [WORD_WIDTH-1:0] SIGMA_F16, RC2PI_F, M_SMPLS_F16,
                    output logic signed [WORD_WIDTH-1:0] s_axis_phase_tdata,
                    output logic [2*WORD_WIDTH-1:0] write_data,
```

```
66                  output logic signed [DATA_WIDTH-1:0] InA_ret_re_fix,
67                  output logic signed [DATA_WIDTH-1:0] InA_ret_im_fix,
68                  output logic signed [DATA_WIDTH-1:0] InB_re_fix,
69                  output logic signed [DATA_WIDTH-1:0] InB_im_fix,
70                  output logic signed [DATA_WIDTH-1:0] InA_re [0:2**ADDRESS_WIDTH-1],
71                  output logic signed [DATA_WIDTH-1:0] InA_im [0:2**ADDRESS_WIDTH-1],
72                  output logic signed [DATA_WIDTH-1:0] InB_re [0:2**ADDRESS_WIDTH-1],
73                  output logic signed [DATA_WIDTH-1:0] InB_im [0:2**ADDRESS_WIDTH-1],
74                  output logic signed [WORD_WIDTH-1:0] arg_exp_re_fix_32_18,
75                  output logic signed [WORD_WIDTH-1:0] arg_exp_im_fix_32_18,
76                  output logic signed [WORD_WIDTH-1:0] narg_exp_fix32,
77                  output logic signed [WORD_WIDTH-1:0] suma_rtemp,
78                  output logic signed [WORD_WIDTH-1:0] rtemp_fix,
79                  output logic signed [WORD_WIDTH-1:0] rtemp_acum_fix,
80                  output logic signed [WORD_WIDTH-1:0] cosh_cordic_fix,
81                  output logic signed [WORD_WIDTH-1:0] sinh_cordic_fix,
82                  output logic signed [WORD_WIDTH-1:0] out_corrpy_scalar [0:2**ADDRESS_WIDTH-1]
83                  );
84
85                  logic [11:0] total_samples, sample_r, sample_w, smpl_cordic, smpl_corr;
86                  logic signed [WORD_WIDTH-1:0] sinh_cordic [0:2**ADDRESS_WIDTH-1];
87                  logic signed [WORD_WIDTH-1:0] cosh_cordic [0:2**ADDRESS_WIDTH-1];
88                  logic signed [WORD_WIDTH-1:0] rtemp [0:2**ADDRESS_WIDTH-1];
89                  logic signed [WORD_WIDTH-1:0] rtemp_acum [0:2**ADDRESS_WIDTH-1];
90                  logic signed [DATA_WIDTH-1:0] InA_ret_re [0:2**ADDRESS_WIDTH-1];
91                  logic signed [DATA_WIDTH-1:0] InA_ret_im [0:2**ADDRESS_WIDTH-1];
92                  logic signed [WORD_WIDTH-1:0] arg_exp_re_fix_32_29;
93                  logic signed [WORD_WIDTH-1:0] arg_exp_im_fix_32_29;
94                  logic signed [WORD_WIDTH-1:0] narg_exp [0:2**ADDRESS_WIDTH-1];
95                  logic signed [WORD_WIDTH-1:0] narg_exp_facpi2;
96                  logic [WORD_WIDTH-1:0] UNO_32_18, PI_32_18, PI_4_32_18, MAX_DIV_32_18;
97                  logic [WORD_WIDTH-1:0] CUATRO_32_18, narg_exp_fix32_29;
98                  logic [7:0] status_reg;
99
100 // The following contains SystemVerilog constructs for the MDS (FSM)
101     enum reg [5:0]   {S0  = 6'b000000,
102                       S1  = 6'b000001,
103                       S2  = 6'b000010,
104                       S3  = 6'b000011,
105                       S4  = 6'b000100,
106                       S5  = 6'b000101,
107                       S6  = 6'b000110,
108                       S7  = 6'b000111,
109                       S8  = 6'b001000,
110                       S9  = 6'b001001,
111                       S10 = 6'b001010,
112                       S100= 6'b101100,
113                       S101= 6'b101101,
114                       S11 = 6'b001011,
115                       S12 = 6'b001100,
116                       S13 = 6'b001101,
117                       S14 = 6'b001110,
118                       S15 = 6'b001111,
119                       S16 = 6'b010000,
120                       S17 = 6'b010001,
121                       S18 = 6'b010010,
122                       S19 = 6'b010011,
123                       S20 = 6'b010100,
124                       S21 = 6'b010101,
125                       S210= 6'b101110,
126                       S22 = 6'b010110,
127                       S23 = 6'b010111,
128                       S24 = 6'b011000,
129                       S25 = 6'b011001,
130                       S26 = 6'b011010,
131                       S80 = 6'b011011,
132                       S81 = 6'b011100,
133                       S82 = 6'b011101,
134                       S83 = 6'b011110,
135                       S84 = 6'b011111,
136                       S85 = 6'b100000,
137                       S86 = 6'b100001,
138                       S87 = 6'b100010,
139                       S88 = 6'b100011,
```

```
                    S110= 6'b100100,
                    S120= 6'b100101,
                    S121= 6'b100110,
                    S122= 6'b100111,
                    S150= 6'b101000,
                    S151= 6'b101001,
                    S1510=6'b101010,
                    S1511=6'b101011,
                    S152= 6'b101111,
                    S153= 6'b110000,
                    S154= 6'b110001,
                    S155= 6'b110010,
                    S201= 6'b110011,
                    S220= 6'b110100,
                    S230= 6'b110101,
                    S231= 6'b110110,
                    XX  = 'x       } state, next;

always @(posedge aclk)
   if (!reset) state <= S0;
   else        state <= next;

always @* begin
      next = XX;
      case (state)
         S0 : // Reset for IP modules (aresetn) during two clock cycles
                                    next    = S1;
         S1 : if (start)            next    = S2;
              else                  next    = S0;
         S2 : //  Data to FIFO Buffer
              if (empty)
                                    next    = S3;
              else                  next    = S0;
         S3 : //  Data to CORDIC module
              if (empty && !full)
                                    next   = S4;
              else                  next   = S3;
         S4 : //  Write to FIFO Buffer
              if (full)
                                  next   = S6;
                else              next   = S5;
          S5 : //  Increment Write Pointer
                                  next   = S4;
          S6 : // Stop Writting
                                  next   = S7;
          S7 : //  Read Data from FIFO Buffer
                                  next   = S8;
          S8 : //  Increment Read Pointer
               if (sample_r > total_samples)
                                  next   = S80;
               else if ((sample_r <= total_samples) && !empty)
                                  next   = S7;
                    else          next   = S4;
          S80:                    next   = S81;
          S81:                    next   = S82;
          S82: // Waiting for qmults module output
               if (o_complete_sigma_f2)
                                  next   = S83;
               else               next   = S82;
          S83:                    next   = S84;
          S84:                    next   = S85;
          S85: // Waiting for qmults module output
               if (o_complete_rc2pi)
                                  next   = S86;
               else               next   = S85;
          S86:                    next   = S87;
          S87:                    next   = S88;
```

```
207                 S88: // Waiting for qmults module output
208                      if (o_complete_rc2pixsigxsmpl)
209                                                 next   = S9;
210                      else                       next   = S88;
211                 S9:                             next   = S10;
212                 S10: //  Input Data Available
213                      if (s_axis_phase_tready)
214                                                 next   = S100;
215                      else                       next   = S10;
216                 S100:                           next   = S101;
217                 S101:                           next   = S11;
218                 S11:                            next   = S110;
219                 S110:                           next   = S12;
220                 S12: // Waiting for qmults module output
221                      if (o_complete_arg_re)
222                                                 next   = S120;
223                      else                       next   = S12;
224                 S120:                           next   = S121;
225                 S121:                           next   = S122;
226                 S122: // Waiting for qmults module output
227                      if (o_complete_arg_im)
228                                                 next   = S13;
229                      else                       next   = S122;
230                 S13:                            next   = S14;
231                 S14:                            next   = S15;
232                 S15:                            next   = S150;
233                 S150:                           next   = S151;
234                 S151: // Waiting for qdiv module output
235                      if (o_complete_narg)
236                                                 next   = S1510;
237                      else                       next   = S151;
238                 S1510: // Factor caculation
239                      if (MAX_DIV_32_18 > PI_32_18)
240                                                 next   = S1511;
241                      else                       next   = S152;
242                 S1511:                          next   = S152;
243                 S152:                           next   = S153;
244                 S153:                           next   = S154;
245                 S154: // Waiting for qmult module output
246                      if (o_complete_facpi)
247                                                 next   = S155;
248                      else                       next   = S154;
249                 S155:                           next   = S16;
250                 S16:                            next   = S17;
251                 S17:                            next   = S18;
252                 S18:                            next   = S19;
253                 S19: //  Wait CORDIC Output
254                      if (m_axis_dout_tvalid)
255                                                 next   = S20;
256                      else                       next   = S19;
257                 S20:                            next   = S201;
258                 S201:                           next   = S21;
259                 S21:                            next   = S210;
260                 S210:                           next   = S22;
261                 S22: // Correntropy Exponential calculation
262                      if (smpl_cordic >= total_samples)
263                                                 next   = S220;
264                      else                       next   = S10;
265                 S220:                           next   = S23;
266                 S23:                            next   = S230;
267                 S230:                           next   = S231;
268                 S231: // Waiting for qdiv module output
269                      if (o_complete_corr)
270                                                 next   = S24;
271                      else                       next   = S231;
272                 S24: // Correntropy Scalar Data available
273                                                 next   = S25;
```

```
            S25: // Update Correntropy Pointers
                if (smpl_corr > total_samples)
                                        next   = S26;
                else                    next   = S9;
            S26: if (start || !corr_out)
                                        next   = S26;
                else                    next   = S0;
            default : // Fault Recovery
                                        next   = S0;
        endcase
     end

always @(posedge aclk)
    if (!reset) begin
        aresetn <= 1'b0; enable <= 1'b0; //enable_kernel <= 1'b0;
        read <= 1'b0; write <= 1'b0; smpl_corr <= 0;
        write_data <= 0;
        s_axis_phase_tvalid <= 0;
        m_axis_dout_tready <= 0;
        s_axis_phase_tdata <= 0;
        InA_ret_re_fix <= 0;
        InA_ret_im_fix <= 0;
        InB_re_fix <= 0;
        InB_im_fix <= 0;
        InA_re <= '{default: 16'h0};
        InA_im <= '{default: 16'h0};
        InB_re <= '{default: 16'h0};
        InB_im <= '{default: 16'h0};
        InA_ret_re <= '{default: 16'h0};
        InA_ret_im <= '{default: 16'h0};
        //arg_exp_re <= '{default: 16'h0};
        //arg_exp_im <= '{default: 16'h0};
        narg_exp <= '{default: 32'h0};
        out_corrpy_scalar <= '{default: 32'h0};
        sinh_cordic <= '{default: 32'h0};
        cosh_cordic <= '{default: 32'h0};
        cosh_cordic_fix <= 0;
        sinh_cordic_fix <= 0;
        rtemp <= '{default: 32'h0};
        rtemp_acum <= '{default: 32'h0};
        suma_rtemp <= 0;
        corr_out <= 0;
        i_start_sigma_f2 <= 1'b0;
        i_start_rc2pi <= 1'b0;
        i_start_arg_re <= 1'b0;
        i_start_arg_im <= 1'b0;
        i_start_narg <= 1'b0;
        i_start_corr <= 1'b0;
        i_start_rc2pixsigxsmpl <= 1'b0;
        i_start_facpi <= 1'b0;
        status_reg <= 0;
    end
    else begin
        aresetn <= 1'b1;
        read <= 1'b0; enable <= 1'b0;
        case (next)
          S0:       aresetn <= 0;
          S1:       aresetn <= 0;
          S2:   begin
                    aresetn <= 1;
                    enable  <= 1;
                end
```

```
            S3:   begin
                      total_samples <= MAX_SAMPLES-1;
                      sample_r <= 0;
                      sample_w <= 0;
                      smpl_cordic <= 0;
                      smpl_corr <= 0;
                  end
            S4:   begin
                      write <= 1;
                      write_data[31:16] <= A_data_table_im[sample_w][DATA_WIDTH-1:0];
                      write_data[15:0] <= A_data_table_re[sample_w][DATA_WIDTH-1:0];
                      write_data[63:48] <= B_data_table_im[sample_w][DATA_WIDTH-1:0];
                      write_data[47:32] <= B_data_table_re[sample_w][DATA_WIDTH-1:0];
                  end
            S5:   begin
                      write <= 0;
                      sample_w ++;
                  end
            S6:   begin
                      write <= 0;
                      write_data <= 64'h0;
                  end
            S7:   begin
                      read <= 1;
                      InA_re[sample_r][DATA_WIDTH-1:0] <= read_data[15:0];
                      InA_im[sample_r][DATA_WIDTH-1:0] <= read_data[31:16];
                      InB_re[sample_r][DATA_WIDTH-1:0] <= read_data[47:32];
                      InB_im[sample_r][DATA_WIDTH-1:0] <= read_data[63:48];
                  end
            S8:   begin
                      sample_r ++;
                      SIGMA_F  <= integer'(real'(SIGMA) * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                      SIGMA_F16  <= integer'(real'(SIGMA) * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                      RC2PI_F  <= integer'(real'(RC2PI) * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                      UNO_32_18 <= integer'(1.0 * real'((2**(2*MANT_WIDTH-8)-1)));   // fix32_18
                      CUATRO_32_18 <= integer'((4.0-2**(-18)) * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                      PI_32_18 <= integer'(PI * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                      PI_4_32_18 <= integer'(PI_4 * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                      MAX_DIV_32_18 <= integer'(MAX_DIV * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                      M_SMPLS_F16 <= integer'(real'(MAX_SAMPLES) * real'((2**(2*MANT_WIDTH-8)-1)));// fix32_18
                  end
            S80:  begin
                      i_start_sigma_f2 <= 1;
                  end
            S81:  begin
                      i_start_sigma_f2 <= 1;
                  end
            S82:  begin   //  Waiting for qmults module output (SIGMA_F * SIGMA_F = SIGMA_F2)
                      i_start_sigma_f2 <= 0;
                  end
            S83:  begin
                      if (ovr_sig2) $display ("%0dns ovr_sig2 = %0d", $time, ovr_sig2);
                      i_start_rc2pi <= 1;
                      status_reg[0] <= ovr_sig2;
                  end
            S84:  begin
                      i_start_rc2pi <= 1;
                  end
            S85:  begin   //  Waiting for qmults module output (RC2PI_F * SIGMA_F16 = RC2PI_FxSIGMA_F16)
                      i_start_rc2pi <= 0;
                  end
```

```
S86:  begin
          if (ovr_rc2pi) $display ("%0dns ovr_rc2pi = %0d", $time, ovr_rc2pi);
          i_start_rc2pixsigxsmpl <= 1;
          status_reg[1] <= ovr_rc2pi;
      end
S87:  begin
          i_start_rc2pixsigxsmpl <= 1;
      end
S88:  begin   //  Waiting for qmults module output
              //(RC2PI_FxSIGMA_F16 * M_SMPLS_F16 = RC2PI_FxSIGMA_F16xM_SMPLS)
          i_start_rc2pixsigxsmpl <= 0;
          if (ovr_rc2pixsigxsmpl) $display ("%0dns ovr_rc2pixsigxsmpl = %0d",
                                  $time, ovr_rc2pixsigxsmpl);
          status_reg[2] <= ovr_rc2pixsigxsmpl;
      end
S9:   begin   // Start 'smpl_corr' loop (external loop)
          if (smpl_corr == 0) begin
                InA_ret_re <= InA_re;
                InA_ret_im <= InA_im;
          end
          else if (smpl_corr != 0) begin
                    for (int i = 0; i < MAX_SAMPLES-1; i++) begin
                        InA_ret_re[i] <= InA_ret_re[i+1];
                        InA_ret_im[i] <= InA_ret_im[i+1];
                    end
                    InA_ret_re[MAX_SAMPLES-1] <= 0;
                    InA_ret_im[MAX_SAMPLES-1] <= 0;
          end
      end
S10:  begin   // Start 'smpl_cordic' loop (internal loop)
          InA_ret_re_fix <= InA_ret_re[smpl_cordic];
          InB_re_fix <= -InB_re[smpl_cordic];
          InA_ret_im_fix <= InA_ret_im[smpl_cordic];
          InB_im_fix <= -InB_im[smpl_cordic];
      end
S100:  begin
          arg_exp_re_fix_32_29 <= {arg_exp_re_fix, 16'h0};// fix16_13 -> fix32_29
          arg_exp_im_fix_32_29 <= {arg_exp_im_fix, 16'h0};// fix16_13 -> fix32_29
      end
S101: begin
          arg_exp_re_fix_32_18 <= arg_exp_re_fix_32_29/(2**11);// fix32_29 -> fix32_18
          arg_exp_im_fix_32_18 <= arg_exp_im_fix_32_29/(2**11);// fix32_29 -> fix32_18
      end
S11:  begin
          i_start_arg_re <= 1;
      end
S110: begin
          i_start_arg_re <= 1;
      end
S12:  begin   //  Waiting for qmults module output (arg_exp_re_fix2)
          i_start_arg_re <= 0;
      end
S120: begin
          if (ovr_arg_re) $display ("%0dns smpl_corr %0d smpl_cordic %0d ovr_arg_re = %0d",
                          $time, smpl_corr, smpl_cordic, ovr_arg_re);
          i_start_arg_im <= 1;
          status_reg[3] <= ovr_arg_re;
      end
S121: begin
          i_start_arg_im <= 1;
      end
S122: begin   //  Waiting for qmults module output (arg_exp_im_fix2)
          i_start_arg_im <= 0;
      end
```

```
S13:  begin   // arg_exp_fix2 = arg_exp_re_fix2 + arg_exp_im_fix2, is obtained
          if (ovr_arg_im) $display ("%0dns smpl_corr %0d smpl_cordic %0d ovr_arg_im = %0d",
                          $time, smpl_corr, smpl_cordic, ovr_arg_im);
          dosxSIGMA_F2 <= SIGMA_F2 << 1;// (2*SIGMA_F*SIGMA_F); fix32_18
          narg_exp_fix32 <= arg_exp_fix2;  // fix32_18
          status_reg[4] <= ovr_arg_im;
      end
S14:  begin   // This state is needed for displaying
          if (narg_exp_fix32 > integer'(4.0 * real'((2**(2*MANT_WIDTH-8)-1))))
              $display ("%0dns smpl_corr %0d smpl_cordic %0d narg_exp_fix32 = %0d",
              $time, smpl_corr, smpl_cordic, narg_exp_fix32);
      end
S15:  begin
          i_start_narg <= 1;
      end
S150:  begin
          i_start_narg <= 1;
      end
S151: begin // Waiting for qdiv module output
        //(narg_exp_div = narg_exp_fix32/(2*SIGMA_F*SIGMA_F)
          i_start_narg <= 0;
      end
S1510: begin
          if (o_overflow_narg) $display ("%0dns smpl_corr %0d smpl_cordic %0d o_overflow_narg = %0d",
                               $time, smpl_corr, smpl_cordic, o_overflow_narg);
          FACPI_32_18 <= UNO_32_18;
          status_reg[5] <= o_overflow_narg;
       end
S1511: begin
          FACPI_32_18 <= integer'(FACTOR * real'((2**(2*MANT_WIDTH-8)-1)));
       end
S152: begin
          i_start_facpi <= 1;
      end
S153: begin
          i_start_facpi <= 1;
      end
S154: begin // Waiting for qmults module output
         //(narg_exp_facpi = narg_exp_div * FACPI_32_18)
          i_start_facpi <= 0;
      end
S155: begin
          if (ovr_facpi) $display ("%0dns smpl_corr %0d smpl_cordic %0d ovr_facpi = %0d",
                         $time, smpl_corr, smpl_cordic, ovr_facpi);
          narg_exp_facpi2 <=  narg_exp_facpi*(2**11);// fix32_18 -> fix32_29
                                                     //(positive result must be assured)
          status_reg[6] <= ovr_facpi;
      end
S16:  begin
          narg_exp[smpl_cordic] <= narg_exp_facpi2;// o_complete_narg of qdiv module activated
      end
S17:  begin
          s_axis_phase_tdata <= narg_exp[smpl_cordic][WORD_WIDTH-1:0];
          s_axis_phase_tvalid <= 1;
      end
S18:  begin
          s_axis_phase_tvalid <= 0;
      end
S19:  begin   //  Wait CORDIC Output
          m_axis_dout_tready <= 1;
      end
S20:  begin
          sinh_cordic[smpl_cordic][WORD_WIDTH-1:0] <= m_axis_dout_tdata[63:32];// fix32_30
          cosh_cordic[smpl_cordic][WORD_WIDTH-1:0] <= m_axis_dout_tdata[31:0];// fix32_30
          sinh_cordic_fix <= -m_axis_dout_tdata[63:32]; //  fix32_30
          cosh_cordic_fix <= m_axis_dout_tdata[31:0];   //  fix32_30
      end
S201: begin
          rtemp[smpl_cordic][WORD_WIDTH-1:0] <= cosh_sinh_fix/(2**12);// fix32_30 -> fix32_18
      end
```

```
            S21:  begin
                      m_axis_dout_tready <= 0;
                      rtemp_fix <= rtemp[smpl_cordic];// call to qadd for suma_rtemp = suma_rtemp + rtemp[i]
                      suma_rtemp <= rtemp_acum[smpl_corr];
                  end
            S22:  begin //  End 'smpl_cordic' loop (internal loop)
                      rtemp_acum[smpl_corr] <= suma_rtemp_fix;
                      smpl_cordic <= smpl_cordic + 1;
                  end
            S220: begin
                      rtemp_acum_fix <= rtemp_acum[smpl_corr];
                  end
            S23:  begin
                      i_start_corr <= 1;
                  end
            S230:  begin
                      i_start_corr <= 1;
                  end
            S231: begin // Waiting for qdiv module output
                       //(out_corrpy_fix = rtemp_acum_fix / RC2PI_FxSIGMA_F16xM_SMPLS)
                      i_start_corr <= 0;
                  end
            S24:  begin
                      if (o_overflow_corr) $display ("%0dns smpl_corr %0d o_overflow_corr = %0d",
                                                     $time, smpl_corr, o_overflow_corr);
                      out_corrpy_scalar[smpl_corr][WORD_WIDTH-1:0] <= out_corrpy_fix;// Correntropy output value
                      status_reg[7] <= o_overflow_corr;
                  end
            S25:  begin   // End 'smpl_corr' loop (external loop)
                      smpl_corr <= smpl_corr + 1;
                      smpl_cordic <= 0;
                      suma_rtemp <= 0;
                  end
            S26:  corr_out <= 1;
            default:  ;
          endcase
        end
endmodule
```