DISEÑO DE LA TANGENTE INVERSA USANDO EL ALGORITMO CORDIC

Ferney Amaya-Fernández, Jaime Velasco-Medina*

Grupo de Bionanoelectrónica, Escuela EIEE, Universidad del Valle, Cali, Colombia
*Grupo de Automática y Robótica, GAR, Universidad Javeriana, Cali, Colombia
E-mail: foamaya@puj.edu.co, jvelasco@univalle.edu.co

ABSTRACT

Este artículo presenta el diseño de la tangente inversa usando el algoritmo CORDIC. El diseño se realizó usando descripción estructural en VHDL y fue sintetizado en el circuito FPGA XC3S200 usando el ambiente Xilinx ISE 7.1. Con el propósito de verificar el funcionamiento de la tangente inversa, se diseñó como vehículo de test un detector de fase, el cual es empleado en la sincronización de portadora de señales moduladas digitalmente. Los resultados de simulación muestran que el diseño de la tangente inversa presenta un buen desempeño usando poca área con respecto a los presentados en la literatura. En este caso, 33.23ns para calcular la tangente inversa usando un total de 198 LUTs de 4 entradas (23 bloques lógicos configurables, CLBs). Adicionalmente, el diseño propuesto para la tangente inversa permite alcanzar una reducción del 34% en área y un aumento del 29% en velocidad con respecto al core de Xilinx.

1. INTRODUCCIÓN

El desarrollo de nuevas técnicas que permitan que un mismo hardware soporte diferentes estándares de comunicación inalámbrica generó la técnica Radio Software (RS), la cual propone el diseño de sistemas inalámbricos configurados y controlados desde software sin cambiar el hardware y con capacidad de operar en un rango de frecuencias amplio [1]. Para llevar a cabo lo anterior considerando la técnica RS, se emplean sofisticadas tareas de procesamiento de señales tales como: compresión de señales, estimación de canal, sincronización de portadora y de bits, control de errores, modulación y dispersión de espectro. En este contexto, uno de los mayores desafíos en un sistema de comunicación es la sincronización de portadora y de bits.

Con el propósito de soportar la flexibilidad requerida por RS para la implementación de radios digitales se emplean circuitos programables como Microprocesadores de Propósito General (General Purpose Processors, GPPs), Procesadores Digitales de Señales (Digital Signal Processors, DSPs), FPGAs (Field Programmable Gate Arrays) y ASICs (Application Specific Integrated Circuits). Teniendo en cuenta que los requerimientos de las aplicaciones actuales de procesamiento digital de señales son tan exigentes en desempeño, la tecnología de FPGAs se ha convertido en la mejor alternativa de implementación debido a su flexibilidad y al alto nivel de paralelismo que puede alcanzarse. Adicionalmente, el tiempo de desarrollo ha disminuido debido a las modernas herramientas de diseño e implementación suministradas por proveedores como Altera y Xilinx.

Considerando lo anterior, en este artículo se presenta el diseño de la tangente inversa usando el algoritmo CORDIC (COrdinate Rotation DIgital Computer), la cual puede ser empleada en un detector de fase para la sincronización de portadora en RS [2], [3]. El diseño se realizó usando descripción estructural en VHDL y fue sintetizado en el circuito FPGA XC3S200 usando el ambiente integrado de Xilinx ISE 7.1.

Este trabajo está organizado de la siguiente forma. Inicialmente la sección 2 presenta los trabajos previos, la sección 3 describe los conceptos básicos del cálculo de la tangente inversa usando el algoritmo CORDIC y las consideraciones de diseño en un FPGA. La sección 4 presenta el diseño de la tangente inversa y en la sección 5 se presentan los resultados de simulación, los cuales se comparan con los presentados en la literatura. Finalmente, la sección 6 presenta las conclusiones y el trabajo futuro.

2. TRABAJOS PREVIOS

En la literatura se encuentran varios trabajos sobre la implementación en hardware del algoritmo CORDIC usando tecnología VLSI. Sin embargo, hace unos años no era posible implementar el algoritmo CORDIC en un FPGA [4], lo cual es posible hoy en día debido a la muy alta densidad de componentes lógicos en los FPGAs.

Tanya Vladimirova y Hans Tiggeler en [5] presentan varios diseños del algoritmo CORCIC en FPGAs de la familia XC4000 de Xilinx. En este caso, para una arquitectura paralela de 12 bits, este diseño presenta un

retardo de 139ns empleando como herramienta de síntesis Xilinx Foundation Series Express 1.5i.

Javier Valls y otros en [4] presentan varios diseños del algoritmo CORDIC empleando arquitectura paralela y pipeline implementados en un FPGA XC4000 de Xilinx. En este caso el mejor diseño presenta un retardo de 64.4ns empleando 535.5 CLBs.

En los artículos de Chris Dick, Fred Harris y Michael Rice [2], [3], [6] se presentan diseños del algoritmo CORDIC, el cual es empleado en la sincronización de portadora para QAM (Quadrature Amplitude Modulation) y PSK (Phase Shift Keying). En [2] se presentan dos diseños para sincronizar la portadora en SR para QAM usando un FPGA de Xilinx. Un diseño usa LUTs y el otro el algoritmo CORDIC. En este caso, el algoritmo CORDIC presenta mejor desempeño, obteniéndose un menor tiempo para la adquisición de la portadora.

En [3] se emplea el criterio de máxima probabilidad (maximum-likelihood, ML) para sincronizar la portadora para QPSK (Qadrature Phase Shift Keying) y QAM. En este caso, se usa un PLL (Phase-Looked Loop) para realizar la sincronización, el cual emplea la tangente inversa. El modelo fue verificado empleando Simulink, VHDL y Xilinx Core Generator. En este artículo se presenta el área ocupada pero no la velocidad. El PLL para sincronizar QPSK, usa aproximadamente 1000 elementos lógicos de una FPGA Virtex de Xilinx.

3. TANGENTE INVERSA EMPLEANDO EL ALGORITMO CORDIC

3.1. Algoritmo CORDIC

CORDIC es un algoritmo iterativo que permite calcular funciones trigonométricas, el cual se puede implementar en hardware usando sumadores, registros de desplazamiento y LUTs.

Debido a que CORDIC no requiere de operaciones de multiplicación, este es adecuado para aplicaciones donde no existen los recursos de multiplicadores y se desee economizar recursos de área. Empleando el algoritmo CORDIC se pueden calcular funciones tales como: seno, coseno, tangente inversa y funciones hiperbólicas con la precisión deseada. El algoritmo CORDIC ha sido empleado en aplicaciones de telecomunicaciones para RS en [2], [3], [6].

CORDIC está basado en la rotación de vectores y permite rotar un ángulo ϕ el vector $\boldsymbol{r} = (x,y)$, generando el vector rotado $\boldsymbol{r'} = (x',y')$. En este caso, el algoritmo CORDIC puede ser empleado en dos modos diferentes:

- *Modo rotación*: en este modo las entradas son el vector $\boldsymbol{r} = (x,y)$ y el ángulo ϕ a rotar, y la salida es el vector rotado $\boldsymbol{r'} = (x',y')$.
- *Modo vectorización*: en este modo la entrada es el vector $\boldsymbol{r} = (x,y)$ y la salida es la magnitud R y el ángulo ϕ del vector $\boldsymbol{r} = (x,y)$.

Las ecuaciones que permiten implementar el algoritmo CORDIC son llamadas ecuaciones CORDIC, las cuales son:

$$x_{i+1} = x_i - y_i.d_i.2 \quad (1)$$

$$y_{i+1} = y_i + x_i.d_i.2^{-i} \quad (2)$$

$$z_{i+1} = z_i - d_i.tan^{-1}(2^{-i}) \quad (3)$$

Para cada iteración, usando las ecuaciones CORDIC se obtiene una pseudo-rotación con un ángulo igual a:

$$\alpha_i = tan^{-1}(2^{-i}) \quad (4)$$

Entonces, varias pseudo-rotaciones por α_i permiten rotar el vector $\boldsymbol{r}$ un ángulo próximo a ϕ.

En este caso, las variables x_{i+1}, y_{i+1} tienen una ganancia de procesamiento An, la cual al incrementar el número de iteraciones se aproxima a 1.647. El valor exacto de An es determinado por la ecuación:

$$A_n = \prod \sqrt{1 + 2^{-2i}} \approx 1.647 \quad (5)$$

Los valores del ángulo $\alpha_i = tan^{-1}(2^{-i})$ son constantes, y en la Tabla 1 se presentan los valores para las primeras 7 iteraciones.

i	$\alpha_i = \tan^{-1}(2^{-i})$
0	45.00000
1	26.56505
2	14.03624
3	7.12502
4	3.57633
5	1.78991
6	0.89517

Tabla 1: *Valores de* $\alpha_i = tan^{-1}(2^{-i})$

Usando las ecuaciones CORDIC se obtienen rotaciones para ángulos menores a 90°. Sin embargo, para lograr rotaciones mayores a 90°, se debe realizar primero una rotación por ±90° empleando las siguientes ecuaciones:

$$x_{i+1} = - y_i.d_i \quad (6)$$

$$y_{i+1} = x_i.d_i \quad (7)$$

$$z_{i+1} = d_i.90° \quad (8)$$

donde: d = -1 para rotar 90° y d = 1 para rotar -90°.

3.2. Tangente inversa empleando CORDIC

Para calcular la tangente inversa se emplea el algoritmo CORDIC en modo vectorización. En este caso, las

coordenadas del vector $\boldsymbol{r} = (x,y)$ inicializan el algoritmo y al final de las iteraciones se obtiene en la variable z_{i+1} el valor del ángulo ϕ del vector $\boldsymbol{r}$ y en la variable x_{i+1} se obtiene el valor de la magnitud del vector $\boldsymbol{r}$ multiplicada por An. Entonces, para calcular la tangente inversa se requieren dos etapas:

a. Inicialización y rotación por ±90°.
b. Pseudo-rotaciones iterativas que buscan llevar a cero la variable y_i.

Para realizar la etapa de inicialización y rotación por ±90° se utiliza el procedimiento de la Figura 1.

1. Inicialización:
   $x_i = x$; $y_i = y$; valores del vector $\boldsymbol{r} = (x,y)$
   i = 0
2. Si $y_i \geq 0$ entonces $d_i = -1$ en otro caso $d_i = 1$
3. Rotación por ±90°
   $x_{i+1} = - y_i.d_i$
   $y_{i+1} = x_i.d_i$
   $z_{i+1} = d_i.90$

***Figura 1**: Procedimiento de la etapa de inicialización y rotación*

Para realizar la etapa de N pseudo-rotaciones iterativas se utiliza el procedimiento de la Figura 2.

1. Si $y_i \geq 0$ entonces $d_i = -1$ en otro caso $d_i = 1$
2. Pseudorotación
   $x_{i+1} = x_i - y_i.d_i.2^{-i}$
   $y_{i+1} = y_i + x_i.d_i.2^{-i}$
   $z_{i+1} = z_i + d_i.tabla[i]$
3. Incrementar i
4. Si i es menor a N ir al paso 1
5. Salir

***Figura 2**: Procedimiento de la etapa de pseudo-rotaciones.*

3.3 Consideraciones en el diseño de la tangente inversa

Para implementar el algoritmo CORDIC se requieren dos tipos de formato numérico, uno para representar los valores de las variables x_i, y_i y otro para representar el valor del ángulo z_i en grados.

Si el vector de entrada $\boldsymbol{r} = (x,y)$ tiene una magnitud menor a 1, las variables x_i, y_i estarían en el rango de 1 a -1. En este caso para el cálculo de la tangente inversa, la variable y_i tiende a cero mientras la variable x_i tendría un valor máximo igual a la ganancia de procesamiento (An=1.646760). Para emplear el formato en punto fijo fraccional en complemento a dos [7] se debe realizar una división por 2 (desplazamiento a derecha) de los valores iniciales de x_i, y_i, esto es con el propósito de contrarrestar la ganancia de procesamiento. De esta forma el valor máximo que puede alcanzar x_i es de 0.82338 quedando en el rango entre -1 y 1.

CORDIC puede ser implementado en hardware usando una arquitectura secuencial o paralela. En la implementación secuencial se realiza una iteración por cada ciclo de reloj. En este caso, para implementar cada iteración se requieren 3 sumadores/restadores, 2 unidades de desplazamiento, una tabla con los valores de los ángulos α y una máquina de estados finitos.

En la implementación en paralelo se requieren varios sumadores/restadores y una LTU. En este caso, los desplazamientos se hacen en forma alambrada.

4. DISEÑO E IMPLEMENTACIÓN DE LA TANGENTE INVERSA

El diseño de la tangente inversa requiere de una unidad de rotación de ±90° y N unidades de pseudo-rotación, en este caso se usaron 6 y se realizaron dos diseños, uno para calcular la tangente inversa (ángulo) y otro para calcular la tangente inversa y la magnitud. Los diseños realizados son puramente combinatorios y se usó VHDL estructural. Los diagramas de bloques para los diseños se presentan en las Figuras 3 y 4.

Para la implementación del algoritmo CORDIC en el FPGA, se emplearon dos tipos de representación numérica:

- *Formato fraccional en complemento a 2*: empleado para representar los valores de x_i, y_i. En esta representación se emplean 10 bits distribuidos de la siguiente forma: 1 bit para representar el signo y 9 bits para representar la parte fraccional.
- *Formato mixto*: empleado para representar los ángulos. En esta representación se emplean 10 bits distribuidos de la siguiente forma: 1 bit para representar el signo, 8 bits para representar la parte entera y 1 bit para representar la parte fraccional.

4.1. Diseño de la unidad de rotación de ±90°

La unidad de rotación de ±90° se encarga de implementar la primera etapa para calcular la tangente inversa. El diagrama funcional de esta unidad se presenta en la Figura 5. En este caso se requieren 3 sumadores/restadores, y el sumador/restador empleado fue generado usando la herramienta Xilinx Core Generator.

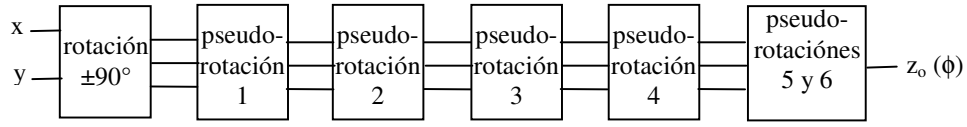


***Figura 3**: Diagrama de bloques de la tangente inversa (ángulo ϕ del vector **r**)*

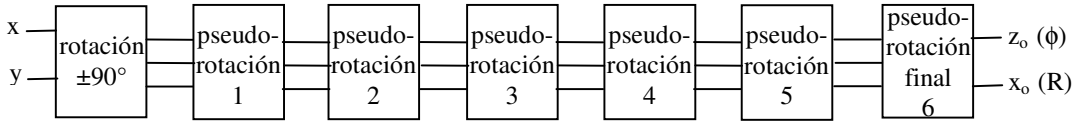


Figura 4:** Diagrama de bloques de la tangente inversa (ángulo ϕ) y la magnitud R del vector **r

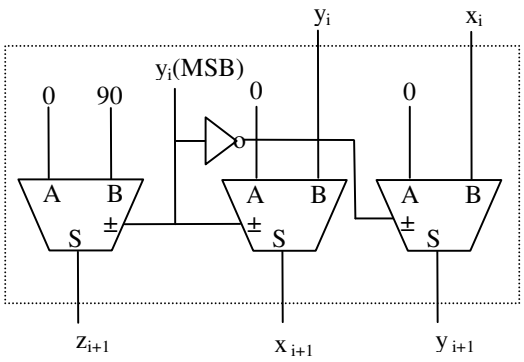


***Figura 5:** Diagrama funcional de la unidad de rotación de ±90°.*

4.2. Diseño de la unidad de pseudo-rotación

La unidad de pseudo-rotación se encarga de implementar la segunda etapa para calcular la tangente inversa, cada unidad realiza una iteración. El diagrama funcional de esta unidad se presenta en la Figura 6.

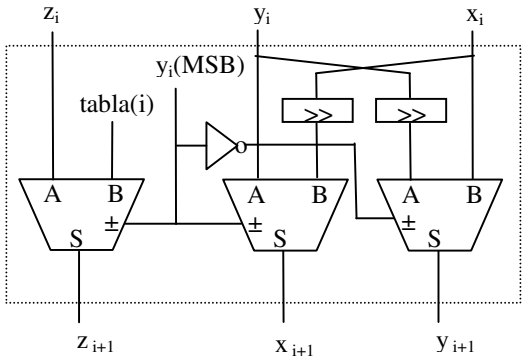


***Figura 6:** Diagrama funcional de la unidad de pseudo-rotación.*

En este caso se requieren 3 sumadores/restadores y 2 unidades de desplazamiento, las cuales se realizaron en forma alambrada. Para determinar la operación de suma o resta se emplea el bit y_i*(MSB)*, el cual es el bit más significativo de la variable y_i, y representa el signo de y_i. Los valores *tabla(i)* son los ángulos $\alpha_i = tan^{-1}(2^{-i})$ y se presentan en formato decimal y en formato binario mixto en la Tabla 2.

i	tabla(i) decimal	tabla(i) formato mixto	error (grados)
0	45.0000	0001011010	0
1	26.5650	0000110101	0.06505
2	14.0362	0000011100	0.03624
3	7.12502	0000001110	0.12502
4	3.57633	0000000111	0.07633
5	1.78991	0000000011	0.28991

***Tabla 2:** Valores de tabla(i) en formato decimal y formato binario mixto.*

El error en grados que aparece en la Tabla 2 es debido a la pérdida de resolución al emplear el formato binario mixto.

4.3. Diseño de la unidad de rotación final

Para calcular el ángulo y la magnitud no se requiere usar todos los sumadores durante la última iteración. En este caso no es necesario calcular, en la última iteración, el valor de y_{i+1}, lo cual permite ahorrar un sumador.

Si solo se requiere el cálculo de la tangente inversa, puede obtenerse una reducción adicional de área eliminando los sumadores que generan x_{i+2}, y_{i+2} en la última iteración y el sumador que genera x_{i+1} en la penúltima iteración. En la Figura 7 se presenta el diagrama funcional de la unidad que realiza las dos últimas pseudo-rotaciones para el cálculo del ángulo.

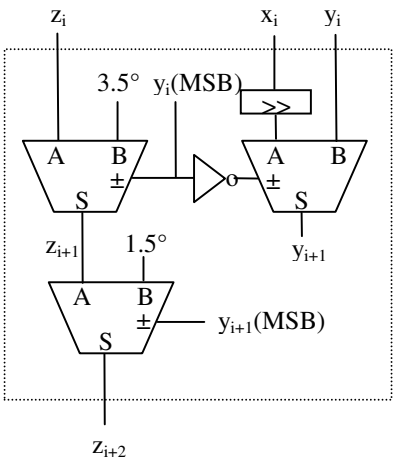


Figura 7: *Diagrama funcional de la unidad que realiza las dos últimas pseudo-rotaciones para el cálculo del ángulo.*

4.4. Resultados de simulación para la tangente inversa

Con el propósito de verificar el desempeño de la tangente inversa se realizó una comparación con el core de Xilinx, en este caso se consideraron los siguientes parámetros de diseño:

Función: arctan
Arquitectura: paralela, sin pileline
Datos de entrada: 10 bits
Número de iteraciones: 6

En la Tabla 3 se presentan los resultados de simulación para los parámetros de diseño de área y velocidad. En este caso los diseños fueron sintetizados en una Spartan-3 XC3S200.

	Tangente inversa	Tangente inversa y magnitud	Core Xilinx
Total de LUTs	178	198	271
LUTs usados como lógica	164	173	234
LUTs usados para enrutar	14	25	37
Número de Slices ocupados	90	100	147
Retardo	34.702ns	33.235ns	49.11ns

Tabla 3: *Resultados de simulación considerando los parámetros de área y velocidad*

Como se puede observar desde la Tabla 3, los diseños propuestos en este trabajo presentan menor retardo y ocupan menor área que el core de Xilinx. En este caso, el diseño para calcular la tangente inversa permite alcanzar una reducción del 34% en área y un aumento del 29% en velocidad con respecto al core de Xilinx.

5. DISEÑO DEL DETECTOR DE FASE

Con el propósito de verificar el funcionamiento del diseño de la tangente inversa usando el algoritmo CORDIC, se utilizó como vehículo de test el diseñó de un detector de fase, el cual se puede usar para la sincronización de portadora y demodulación de señales PSK y QAM.

5.1 Detector de fase

Asumiendo que se transmite una portadora sin modular y considerando un canal AWGN (Additive White Gaussian Noise), la señal recibida por el receptor es:

$$r(t) = A\cos(2\pi f_c + \phi) + n(t) \qquad (9)$$

donde *A* es la amplitud, f_c es la frecuencia de la portadora, ϕ es la fase desconocida de la portadora y *n(t)* es el ruido gausiano.

Uno de los criterios para estimar el valor de la fase desconocida ϕ es el criterio de máxima probabilidad (maximum-likelihood, ML), el cual permite encontrar la mejor estimación de ϕ al maximizar la función de probabilidad $\Lambda(\phi)$. El valor que más se acerca a la fase desconocida está dado por la siguiente ecuación [8]:

$$\phi_{ML} = -\tan^{-1}\left\{\frac{\int_{To} r(t)\sin\omega_c t\, dt}{\int_{To} r(t)\cos\omega_c t\, dt}\right\} \qquad (10)$$

donde *To* es el tiempo de símbolo.

A partir de la ecuación anterior, se puede generar el diagrama de bloques de la Figura 8 [8].

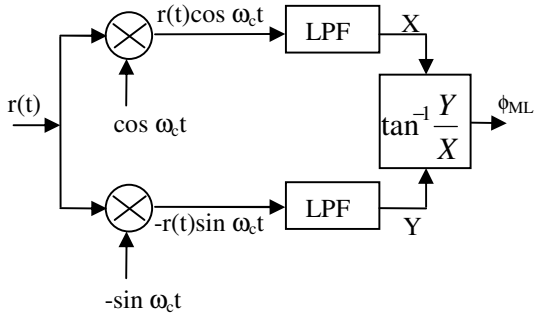


Figura 8: *Diagrama de bloques del detector de fase.*

Este diagrama es empleado para estimar la fase de la portadora no modulada en [2] y [3] para la detección y sincronización de PSK y QAM. Una de las características

de este detector de fase, es que puede ser implementado digitalmente, proporcionando la flexibilidad requerida en RS al permitir la detección y sincronización de varias técnicas de demodulación como QAM y PSK.

5.2 Diseño del detector de fase

De acuerdo a la Figura 8, el detector de fase está conformado por un NCO (Numerically Controlled Oscillator), un filtro pasa bajas (Low Pass Filter, LPF) y un bloque que permite calcular la tangente inversa.

El diagrama de bloques del NCO diseñado se presenta en la Figura 9.

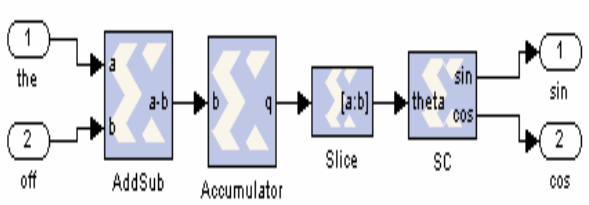


Figura 9: *Diagrama de bloques del NCO.*

La entrada *the*, es una constante que especifica la frecuencia del NCO y la entrada *off*, permite variar la fase y/o frecuencia del NCO. Tanto la señal *the* y *off* tienen 10 bits para representar la parte entera y 2 bits para la parte fraccional. El bloque *AddSub* realiza la suma de las entradas *the* y *off*, el bloque *Accumulator* acumula el valor de la salida del sumador. El bloque *SC* es una ROM, la cual contiene los valores de seno y coseno. La función del bloque *Slice* es seleccionar los 10 bits de la parte entera de la salida del acumulador, los cuales se usan para direccionar la ROM. Las salidas *cos* y *sin* son los valores de coseno y seno del ángulo θ de acuerdo a la siguiente ecuación:

$$\theta = (the + off)\frac{2\pi}{2^{10}} \text{ radianes} \qquad (11)$$

Para el diseño del detector de fase se emplearon los siguientes parámetros:

- Muestras por ciclo de portadora: 8
- Muestras por símbolo: 90
- Filtro FIR pasa bajas de 51 coeficientes

El filtro pasa bajas del detector de fase es un filtro FIR de 51 coeficientes diseñado por el método de Ventana de Hamming. La respuesta en frecuencia (magnitud) del filtro se presenta en la Figura 10.

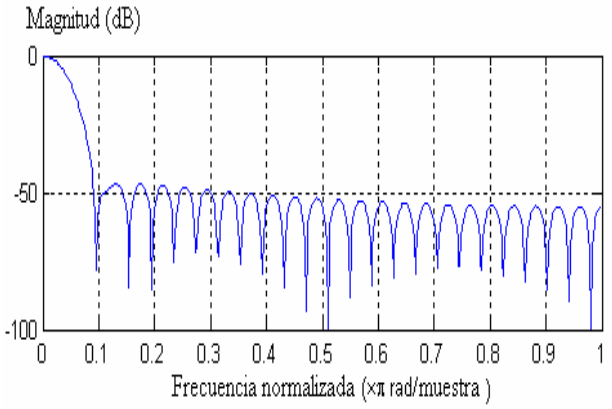


Figura 10: *Respuesta en frecuencia del filtro FIR de 51coeficientes.*

El diagrama de bloques del detector de fase incluyendo el bloque para calcular la tangente inversa se presenta en la Figura 11.

La entrada *mod*, es la señal modulada y las entradas *the* y *off* van directamente al NCO. Las salidas del detector de fase son:

- *Ang*: valor del ángulo (fase) de la señal modulada
- *Mag*: magnitud escalada de la señal modulada

En este caso, el retardo del NCO es de 2 ciclos de reloj y el retardo total del detector de fase es de 13 ciclos de reloj.

5.3. Resultados de simulación para el detector de fase

Con el propósito de verificar el funcionamiento del detector de fase se realizaron varias simulaciones

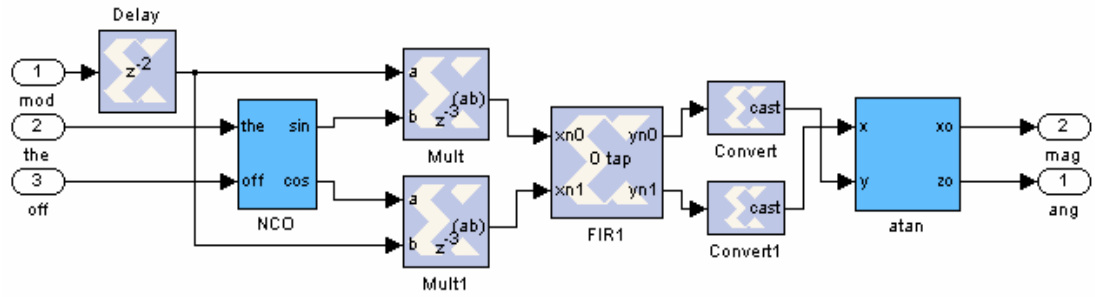


Figura 11: *Diagrama de bloques del detector de fase*

empleando System Generator. La primera simulación tiene como entrada una portadora con valores de fase entre 0° y 180° con pasos de 20° y con 90 muestras por valor de fase. La señal de salida del detector de fase se presenta en la Figura 12. En este caso, se obtiene un valor de 1.5° como máximo error de fase. La parte derecha de la Figura 12, presenta una ampliación de una sección de la señal de salida del detector de fase, la cual permite apreciar la respuesta cuando ocurre un cambio de fase de 20°. En este caso, se observa que el detector de fase responde a los cambios de fase en aproximadamente 25 muestras.

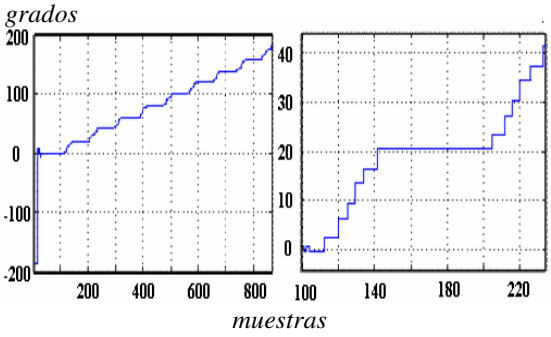
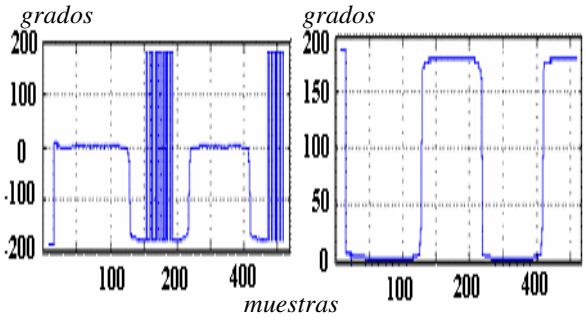


Figura 12: *Señal de salida del detector de fase para una entrada con desfases en pasos de 20º.*

Para la segunda simulación se tiene como entrada una señal BPSK (Binary Phase Shift Keying). En este caso se calculó el valor absoluto a la salida del detector de fase, lo cual permite evitar que desfases mayores a 180° generen oscilaciones entre 0° y 180°. En la Figura 13 se observa la salida del detector de fase presentando los dos casos anteriores: la salida sin el valor absoluto (Figura 13.a) y la salida con el valor absoluto (Figura 13.b).



a. sin valor absoluto b. con valor absoluto

Figura 13: *Señal de salida del detector de fase para una señal BPSK de entrada.*

Desde la Figura 13.b se puede observar que la señal de salida del detector de fase puede emplearse en la detección de los bits recibidos en BPSK.

La simulación final consistió en generar una señal QAM empleando los símbolos:

$$s_i(t) = A_i \cos(2\pi f_c + \phi_i) \qquad (12)$$

En este caso, se usan dos valores para la amplitud: $A_0 = 0.5$ y $A_1 = 1.0$ y 4 valores de fase: $\phi_1 = 45°$, $\phi_2 = -45°$, $\phi_3 = 135°$ y $\phi_4 = -135°$. Las salidas del detector de fase, tangente inversa y magnitud, se presentan en las Figuras 14 y 15 respectivamente.

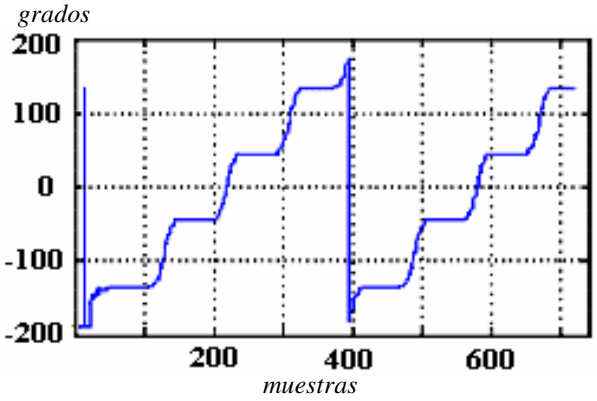


Figura 14: *Valores del ángulo entregados por el detector de fase para una señal QAM de entrada.*

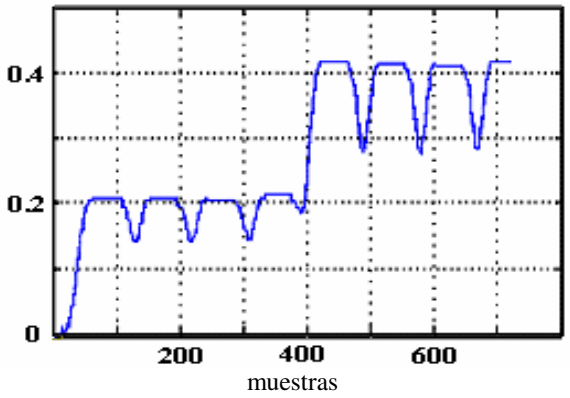


Figura 15: *Valores de amplitud entregados por el detector de fase para una señal QAM de entrada.*

En este caso, se puede observar que la tangente inversa y la magnitud permiten diferenciar los 2 valores de amplitud y los 4 valores de fase de la señal QAM. Lo cual permite la detección de los bits recibidos.

6. CONCLUSIONES Y TRABAJO FUTURO

En este artículo se presenta el diseño de un bloque funcional que permite calcular la tangente inversa empleando el algoritmo CORDIC. La verificación del diseño propuesto se llevó a cabo usando un detector de fase, el cual puede ser empleado en la detección y sincronización para PSK y QAM. En este caso, el diseño

se basa en una arquitectura paralela, la cual se describe usando una descripción estructural en VHDL y es sintetizada en una Spartan-3 XC3S200. El bloque funcional diseñado presenta un buen desempeño y ocupa poca área, lo cual es adecuado para aplicaciones de RS donde el algoritmo CORDIC es ampliamente utilizado. Los resultados de simulación muestran que el diseño para calcular la tangente inversa permite alcanzar una reducción del 34% en área y un aumento del 29% en velocidad con respecto al core de Xilinx.

El trabajo futuro será orientado hacia el diseño de la transformada rápida de Fourier (FFT: Fast Fourier Transform) usando el algoritmo CORDIC y hacia el diseño de "cores" para Radio Software con el propósito de diseñar un procesador de banda base y frecuencia intermedia con características de muy alto desempeño.

7. REFERENCIAS

[1] J. Mitola, *Software Radio Architecture*,.John Wiley & Sons, Inc, 2000.

[2] Chris Dick, Fred Harris y Michael Rice, "FPGA Implementation of carrier Synchronization for QAM receivers," *Journal of VLSI Signal Processing Systems,* Volumen 36, pp. 57-71, Ene. 2004.

[3] Michael Rice, Chris Dick, y fred harris, "Maximum Likelihood Carrier Phase Synchronization in FPGA-Based Software Defined Radios," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, May. 2001.

[4] Javier Valls, M. Kuhlmann, y K.K. Parhi, "Evaluation of CORDIC Algorithms for FPGA design," *Journal of VLSI Signal Processing*, pp. 207-222, Nov. 2002.

[5] Tanya Vladimirova y Hans Tiggeler, "FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm," *annual Military and Aerospace Applications of Programmable Devices and Technologies Conference, MAPLD-1999.*

[6] F.J. Harris y C.H. Dick, "On Structure and Implementation of Algorithms for Carrier and Symbol Synchronization in Software Defined Radios," *EUSIPCO-2000, "Efficient Algorithms for Hardware Implementation of DSP Systems," Tempere, Finland,* Sept. 2000.

[7] Douglas Jones, "Fixed-Point Number Representation," Universidad de Rice, Dic. 2004. Disponible en http://cnx.rice.edu/content/m11930/1.2/.

[8] John Proakis, *Digital Communicactions 4ed*, Mc Graw Hill. 2000.

[9] Mel Tsai, Jeff Bier, y Jennifer Eyre, "Evaluating FPGAs for communication infrastructure applications," *Berkeley Design Technology, Inc*.

[10] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, V. EC-8, No. 3, pp. 330-334. 1959.

[11] R. Andraka, "A Survey of CORDIC Algorithms for FPGAs," *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays (FPGA '98)* , Monterey, CA, pp. 191–200. Feb. 1998.

[12] Marvin Frerking, *Digital Signal Processing in Communication Systems*, Kluwer Academic Publishers. 2003.

[13] Xilinx Inc., System Generator for DSP, http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=dr_dt_system_generator

[14] Xilinx Core Generator, http://www.xilinx.com/products/logicore/coregen/index.htm