UNIVERSIDAD SAN PABLO-CEU

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA SUPERIOR

PROYECTO FINAL DE CARRERA

Implementación de un Módem Digital Basado en el Estándar IEEE 802.15.4

Autor:
Antonio de la Piedra Abenójar

Director:
Gianluca Cornetta

Febrero de 2009

Resumen

El objetivo de este proyecto es la realización de un módem digital de bajo coste basado en el estándar IEEE 802.15.4 siguiendo las pautas descritas en los trabajos de Roger Martinsen Koteng y Gianluca Cornetta. Se trata de un dispositivo diseñado para alcanzar velocidades de hasta 250 kbps con aplicaciones en redes de sensores de bajo consumo y corto alcance que no precisen de ancho de banda elevados. Se ha utilizado el lenguaje de descripción de hardware VHDL con el objetivo de poder generar una solución sintetizable, que pueda simularse en una FPGA o llevar a cabo la construcción de un sistema ASIC. El módem se compone de una unidad moduladora, encargada de preparar los datos para su transmisión utilizando la técnica de espectro expandido por secuencia directa (DSSS) y la modulación OQPSK. También se compone de una unidad demoduladora, capaz de sincronizarse con la fuente de los datos y que implementa dos técnicas combinadas de estimación de frecuencia, una de ellas mediante el estimador de Kay. Puesto que ambas técnicas necesitan manipular valores complejos también se realizó una implementación del CORDIC. Las decisiones de diseño e implementación se basaron en la búsqueda de equilibrio entre en la sencillez de la solución y el área del circuito final.

Abstract

The main objetive of this thesis is the design and construction of a low-cost IEEE 802.15.4 compilant modem, based in the directions of Roger Martinsen Koteng and Gianluca Cornetta works. The device has a transmission rate of 250 kbps which can be used in sensor networks with small bandwidth and short distances. VHDL language was used to describe the modem behaviour thinking in simulating the design in a FPGA or building an ASIC solution. The modem is composed of a modulation and a demodulation unit, controlled by means of a modem general controller. Modulation unit uses Direct Sequence Spread Spectrum technique (DSSS) and OQPSK modulation. Demodulation unit is able of synchronizing with the incoming data and uses two different techniques in the frequency estimation process, including Kay estimator. Because we have to deal with complex values a CORDIC core was implemented. Design and implementation decisions were based in simplicity and circuit area.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Descripción del proyecto

El objetivo de este proyecto es la realización de un módem digital de bajo coste basado en el estándar IEEE 802.15.4 siguiendo las pautas que se describen en [1] y [13]. Un dispositivo diseñado para cumplir el estándar 802.15.4 debe poder alcanzar velocidades de transmsión de hasta 250 kbps, lo que limita su aplicación a redes de sensores de bajo consumo y corto alcance que no precisan de anchos de banda elevados. Este tipo de redes tiene varias aplicaciones especialmente en el ámbito industrial, domótica y telemedicina.

Para la describir el comportamiento del módem se ha utilizado el lenguaje de programación VHDL con el objetivo de poder generar una solución sintetizable, que pueda simularse en una FPGA o llevar a cabo la construcción en un sistema ASIC. En el presente documento se muestra la implementación

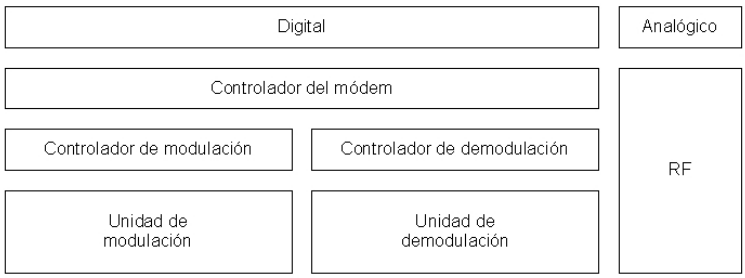

Figura 1.1: Arquitectura general del módem

de la parte moduladora y demoduladora del módem así como las decisiones tomadas y resultados obtenidos. En la Figura 1.1 puede apreciarse el esquema general del módem.

1.1.1. Modulación

En la parte de modulación el módem realiza las consecutivas conversiones de bit a símbolo y de símbolo a chip utilizando la técnica de espectro expandido por secuencia directa (DSSS). Posteriormente los datos son codificados en NRZ, pasados por un proceso de *upsampling* y modulando mediante OQPSK pasando el resultado por un filtro FIR con una respuesta al impulso de tipo medio seno.

1.1.2. Demodulación

La recepción de los datos comienza con la corrección del offset de frecuencia y detección de la fase, que se realiza de dos formas combinadas. Además se implementó un *downsampler*, un codificador RZ y un conversor de símbolo a bit para recuperar la información original antes de ser modulada. Para la detección de los símbolos entrantes se implementó un correlador de recepción. Además, se cuenta con un sistema indicador de nivel de señal en la recepción (Receiver ED), un indicador de la calidad del enlace (LQI, Link quality indicator) y un módulo capaz de hacer medidas de energía sobre el canal, para determinar si está ocupado o no (CCA, *Clear Channel Assessment*).

1.1.3. Controlador

El controlador se encarga de gestionar la transmisión y recepción de los datos que salen y entran del modem y enviar las señales de control necesarias a las partes implicadas. El módem consta de 3 controladores: un controlador general que decide si se puede pasar de un modo de funcionamiento a otro, un controlador de la unidad de modulación y un controlador de la unidad de demodulación.

1.1.4. Técnicas, estructuras y algoritmos

En la implementación del módem se usaron distintos algoritmos y estructuras: el algoritmo de Kay [7], el CORDIC [9], FIR etc. donde las decisiones de diseño se tomaron buscando un equilibrio entre la sencillez de la soluciòn, la reducción de área y las exigencias del éstandar.

1.2. Tecnologías

En esta parte se describen las tecnologías utilizadas, qué definen el éstandar IEEE 802.15.4 y ZigBee, el lenguaje de descripción de hardware VHDL y las FPGAs.

1.2.1. ZigBee y IEEE 802.15.4

El estándar 802.15.4 especifica la capa física (PHY) y de acceso al medio (MAC) para redes inalámbricas personales con tasas bajas de transferencia de datos (Redes LR-WPAN) y sirve de apoyo a ZigBee, que se encarga de definir las capas superiores. Este proyecto está basado en la capa física y se sirve del estándar IEEE 802.15.4.

La capa física puede operar en tres bandas distintas, siendo la de 2.4 GHz (ISM) la elegida en el proyecto. La tecnología nos permite una velocidad de transmisión de 250 kbits/segundo en un rango de 10 a 75 metros. Utiliza la técnica de expansion de espectro por secuencia directa (DSSS) para evitar interferencias y la modulación OQPSK.

Más detalles sobre ZigBee pueden consultarse en el Capítulo 2, en [2] y [3].

1.2.2. FPGA

Uno de los objetivos al utilizar VHDL es la síntesis del código para la simulación en FPGA (Field-Programmable Gate Array) y su posterior implementación en forma de ASIC (Application-Specific Integrated Circuit). Una FPGA consiste en un conjunto de bloques de logica configurable o

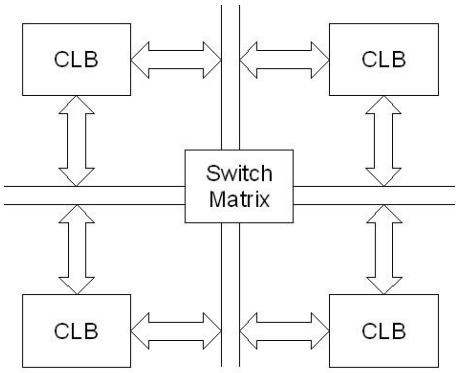

Figura 1.2: Arquitectura de una FPGA

CLBs (Configurable Logic Blocks) interconectados (Figura 1.2). Cada CLB está basado en una LUT (Lookup Table), a diferencia de una PLD, basadas en la implementación de expresiones en suma de productos (SOP) y puertas *and* y *or*. Se puede ver la estructura de un CLB en la Figura 1.3. Muchas FPGA contienen interfaces PCI, serie, USB o bloques dedicados como multiplicadores o procesadores digitales de señal (DSPs), lo que facilita la simulación del diseño.

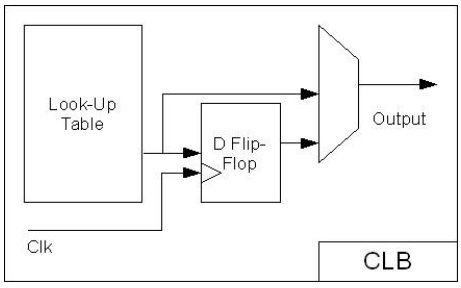

Figura 1.3: Arquitectura de un CLB

La FPGA para la que se ha realizado la síntesis ha sido el modelo Virtex4 (xc4vlx15) de Xilinx. Para la síntesis se utilizó el entorno de desarrollo de Xilinx ISE. Los resultados se han simulado utilizando ModelSIM de Mentor. El resultado de la síntesis ha sido un circuito formado por 728 flip-flops con una frecuencia máxima de funcionamiento de 27.192MHz. (Puede consultarse el informe completo de la síntesis en el Capítulo 10).

Puede encontrarse más información sobre la estructura de una FPGA en [4, apéndice A] y en [5].

1.3. Estructura del trabajo

El Capítulo 2 del documento trata el estándar IEEE 802.15.4, sus aplicaciones y los detalles de la capa física: frecuencias de operación, expansión de espectro, la modulación OQPSK, la forma de pulso y las velocidades de transmisión exigidas. Finalmente se detalla el contenido de la trama de la capa física con sus campos. En la tercera parte, generación numérica de señales, se analizan los algoritmos utilizados: Kay y CORDIC y las decisiones de diseño tomadas de cara a la implementación.

La cuarta parte esta dedicada a la arquitectura general del sistema, donde se describen brevemente los detalles de implementación que se ampliaran en detalle en las partes 5 y 6 correspondientes al subsistema de modulación y demodulación del documento.

Se incluye un capítulo sobre el controlador del módem y el código fuente en VHDL junto a los *testbenches* y los resultados de las simulaciones. En el Capítulo 10 se encuentra el informe de síntesis del código fuente del módem y en el Capítulo 11 las conclusiones y posibles mejoras del proyecto.

Capítulo 2

El estándar IEEE 802.15.4

Hace pocos años que apareció un nuevo estándar dedicado a las áreas inalámbricas personales de baja transferencia de datos (Low-Rate Wireless Personal Area Network, LR-WPAN), redes de bajo coste y bajo rendimiento de procesamiento. Este estándar conocido como ZigBee es fruto del esfuerzo del grupo IEEE 802.15.4 y la ZigBee Alliance. El grupo IEEE 802.15.4 se encarga de definir la capa física (PHY) y de acceso al medio (MAC) mientras que la ZigBee Alliance se ha concentrado en las capas superiores.

La arquitectura de protocolos ZigBee esta basada en el estándar OSI de 7 capas, a saber: aplicación, presentación, sesión, transporte, red, enlace y física. ZigBee trabaja sobre 3 bandas diferentes aunque este proyecto ha sido diseñado para la banda ISM en 2.4 GHz. El objetivo principal de ZigBee son las aplicaciones de ámbito industrial, médico y domótico de bajo consumo de energía y costes, que se consigue con tasas de transferencia de datos bajas aumentando así la duración de la batería.

2.1. La capa física

Este proyecto se basa en la definición de la capa física correspondiente a la banda ISM de 2.4 GHz del estándar. El estándar define 16 símbolos de 4 bits y en cada periodo de símbolo estos cuatro bits son convertidos a una de las 16 secuencias ortogonales pseudo-aleatorias para ser transmitidas. Estas secuencias para cada símbolo se concatenan y modulan mediante OQPSK.

La Figura 2.1 ilustra la secuencia de operaciones realizadas por la capa física durante la fase de transmisión de una secuencia de bits.

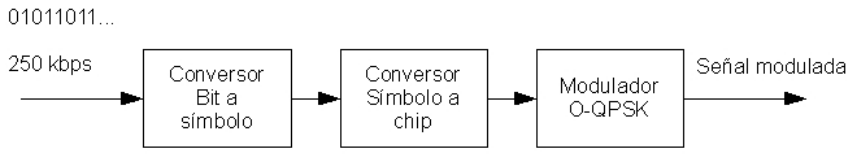

Figura 2.1: Esquema de funcionamiento de la capa física

2.1.1. Bandas de frecuencia

El estándar define 27 canales distribuidos en 3 bandas de frecuencia:

- 1 canal reservado en la banda de 868 MHz
- 10 canales en la banda de 915 MHz
- 16 canales en la banda de 2.4 GHz

Las frecuencias de los distintos canales vienen dados por:

$$F_c(k) = \begin{cases} 868{,}3 & \text{MHz si } k = 0, \\ 906 + 2(k-1) & \text{MHz si } k = 1..,10, \\ 2405 + 5(k-11) & \text{MHz si } k = 11..,26. \end{cases}$$

donde k es el número de canal. Este trabajo se centra únicamente en la implementación en la banda ISM de 2.4 GHz.

2.1.2. DSSS

ZigBee utiliza el método de expansión de espectro por secuencia directa (DSSS). Según el estándar, los símbolos se expanden en secuencias pseudoaleatorias de 32 bits (chips) donde el número de ceros es igual al número de unos. Cada secuencia PN se genera a partir de la anterior mediante una rotación de 4 bits a la derecha en las secuencias de los 8 primeros símbolos. Las secuencias de los últimos 8 símbolos se generan invirtiendo los bits pares de los 8 primeros.

Así, partiendo de la secuencia correspondiente al símbolo cero, pueden generarse todas las demás. En la implementación actual, los *chips* se generan partiendo del primer símbolo mediante un buffer circular para cada uno de los canales, tomando cada secuencia de chip de una posición determinada del buffer.

Información más detallada del generador de secuencias PN puede encontrarse en el Capítulo 6. En el Cuadro 2.1 aparecen los 16 símbolos que define el estándar con su correspondiente secuencia.

Símbolo (b0...b3)	Secuencia PN (c0...c31)
0000	11011001110000110101001000101110
1000	11101101100111000011010100100010
0100	00101110110110011100001101010010
1100	00100010111011011001110000110101
0010	01010010001011101101100111000011
1010	00110101001000101110110110011100
0110	11000011010100100010111011011001
1110	10011100001101010010001011101101
0001	10001100100101100000011101111011
1001	10111000110010010110000001110111
0101	01111011100011001001011000000111
1101	01110111011100011001001011000000
0011	00000111011110111000110010010110
1011	01100000011101111011100011001001
0111	10010110000001110111101110001100
1111	11001001011000000111011110111000

Cuadro 2.1: Secuencias PN para los 16 símbolos del estándar IEEE 802.15.4

2.1.3. Modulaciones QPSK y OQPSK

El estándar IEEE 802.15.4 utiliza la modulación OQPSK, que presenta algunas ventajas sobre QPSK: las variaciones de fase se reducen de 180º a 90º y podemos utilizar en transmisión un amplificador no lineal sin tener excesivo recrecimiento espectral generado por las variaciones de fase de la señal (Se puede encontrar más información sobre este aspecto en [14]). A continuación se presenta la modulación QPSK, de donde se deriva la modulación OQPSK.

En la modulación QPSK un flujo de datos $d_k = d_0, d_1, d_2, ...$ compuesto de pulsos bipolares, +1 o -1, se divide en dos componentes: una componente de fase $d_I(t)$ y una componente en cuadratura, $d_Q(t)$ de la siguiente forma:

$$d_I(t) = d_0, d_2, d_4, ...$$

$$d_Q(t) = d_1, d_3, d_5, ...$$

donde la componente de fase esta compuesta por los bits pares y la componente en cuadratura por los bits impares y tienen un ratio de bit que es la mitad de $d_k(t)$.

Una onda modulada en QPSK se obtiene utilizando las señales de fase y cuadratura para modular en amplitud las funciones de seno y coseno de una onda portadora, dando como resultado:

$$s(t) = \frac{1}{\sqrt{2}} d_I(t) cos(2\pi f_0 t + \frac{\pi}{4}) + \frac{1}{\sqrt{2}} d_Q(t) sin(2\pi f_0 t + \frac{\pi}{4})$$

que puede escribirse como:

$$s(t) = \cos(2\pi f_0 t + \theta(t))$$

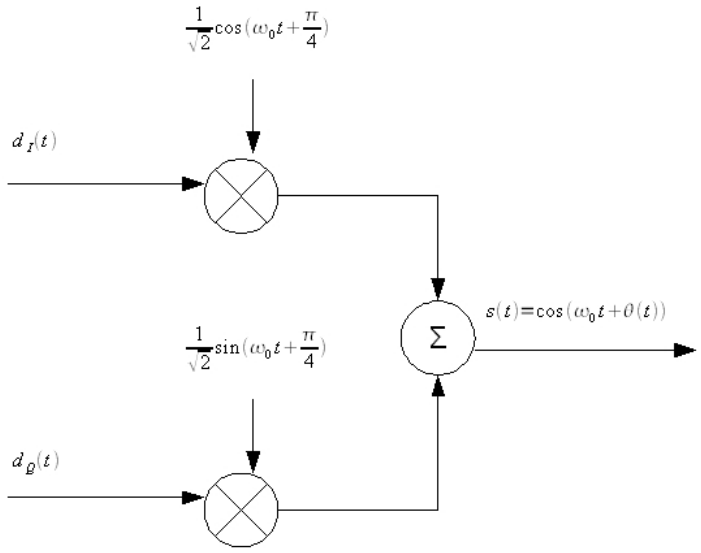

Figura 2.2: Modulador QPSK

La señales d_I y d_Q modulan en amplitud a la función coseno con una amplitud de +1 o -1, lo que es equivalente a un desfase del coseno o seno de 0 o π, siendo ambas ortogonales. La suma de estas dos señales ortogonales dan lugar a la forma de onda típica de QPSK, donde el valor de $\theta(t)$ corresponde a una de las cuatro combinaciones entre d_I y d_Q: 0º, $\pm$90º, 180º. Además, puesto que las dos señales seno y coseno son ortogonales, pueden ser detectadas de forma separada. En las Figuras 2.2 y 2.3 aparecen los diseños de un modulador QPSK Y OQPSK respectivamente.

La técnica de modulación Offset QPSK u OQPSK puede ser representada de la misma manera, la diferencia se encuentra en un desplazamiento de tiempo en la señal d_Q. La duración de cada pulso original en $d_k(t)$ era

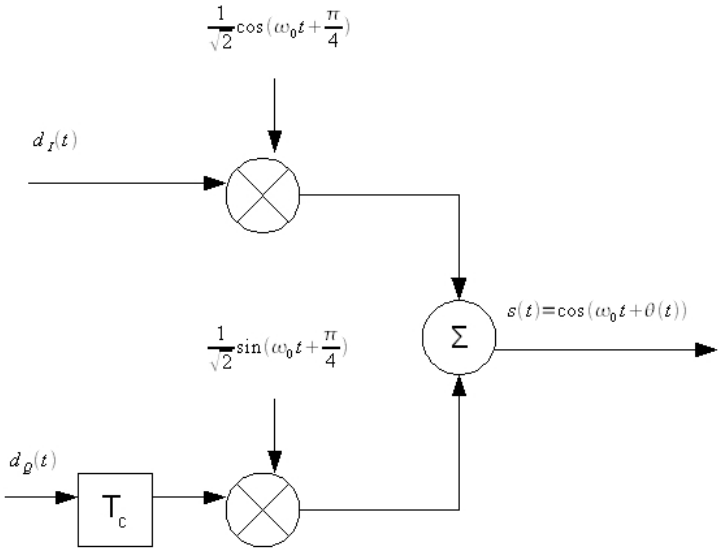

Figura 2.3: Modulador OQPSK

T y en las dos señales separadas 2T. En QPSK los pulsos par e impar son transmitidos a un ratio de $\frac{1}{2T}\ \frac{bits}{s}$, alineados de forma que sus transiciones coincidan. En OQPSK, la señal d_Q se desplaza un periodo de tiempo T.

En QPSK, en cada periodo de tiempo 2T, la fase puede cambiar. Si las dos señales cambian a la vez, obtenemos un desplazamiento de fase de 180º, si una sola cambia, de 90º y si no cambian la fase permanece igual. El problema reside en los cambios de fase de 180º: cuando una señal de este tipo pasa por un filtro paso bajo, estos cambios de fase aparecen como cambios de amplitud muy grandes, no deseables en un sistema de comunicación.

En OQPSK, no es posible que las dos señales cambien a la vez, así los cambios de fase están limitados entre 0 y 90º (Figura 2.4). En la Figura 2.5 aparece el símbolo 0 del estándar modulado en OQPSK. Puede encontrarse más información de las ventajas de OQPSK frente QPSK en [6].

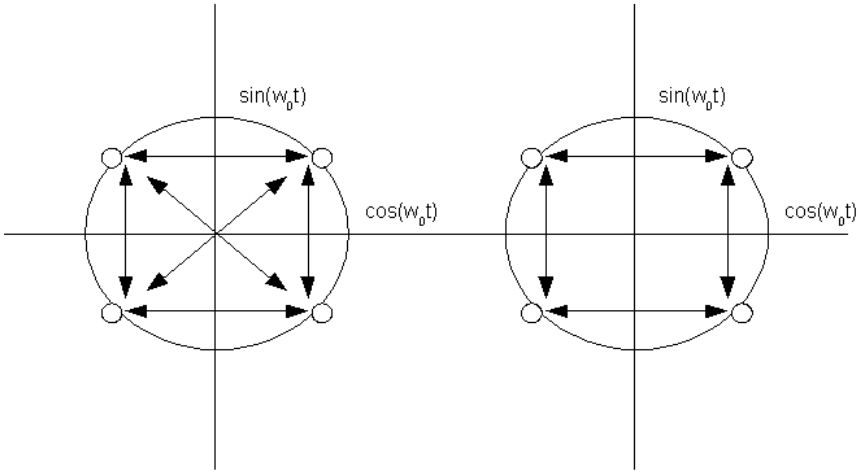

Figura 2.4: Cambios de fase en QPSK Y OQPSK.

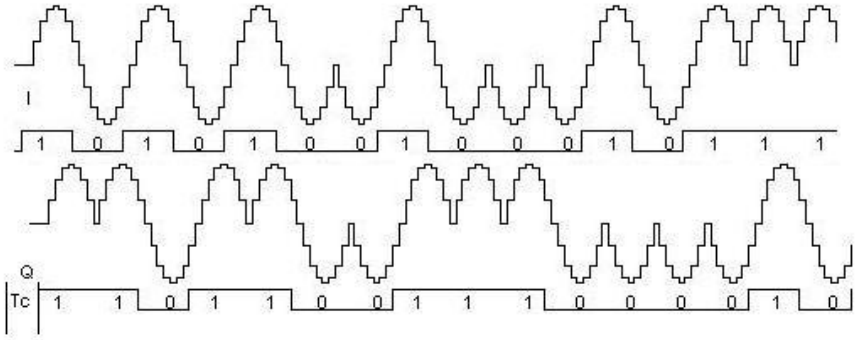

Figura 2.5: Resultado de la modulación del símbolo cero.

2.1.4. Pulse shape

El estándar define una forma de pulso de medio seno para representar los *chips* en banda base dado por:

$$p(t) = \begin{cases} \sin(\pi \frac{t}{2T_c}) & \text{si } 0 \leq t \leq 2T_c \\ 0 & \text{en otro caso} \end{cases}$$

Se pueden encontrar más detalles sobre el diseño e implementación del filtro que implementa la función en los Capítulos 5 y 6.

2.1.5. Velocidades de transmisión

El estándar define una velocidad de símbolo de 62.5 kilosímbolos/segundo, cómo cada símbolo de 4 bit se expande mediante una secuencia de 32 *chips*, el ratio de *chip* será de 2 Mchips/segundo y el ratio de bit de 250 kilobits/segundo.

2.1.6. Orden de transmisión

En cada periodo de símbolo se envía siempre el bit menos significativo y el último, el bit 31, al final.

2.1.7. Sistema indicador de nivel de señal

El sistema indicador de nivel de señal o RSSI (Received Signal Strength Indicator) es una estimación de la potencia de la señal recibida dentro del ancho de banda de un canal definido por el estándar. Podemos calcular este valor midiendo la energía de las muestras entrantes y acumulándola durante un periodo de tiempo determinado. Calcularemos la potencia dividiendo la energía acumulada por el número (N) de muestras analizadas. Así, la potencia de la señal vendrá dada por:

$$RSSI = \frac{1}{N}\sum_{n=0}^{N-1}|m(n)|^2 = \frac{1}{N}\sum_{n=0}^{N-1}s_I^2(n) + s_Q^2(n)$$

donde el módulo de $m(n)$ es el módulo de las muestras complejas entrantes moduladas en cuadratura de fase y N es el número de muestras medidas. En el Capítulo 6 se verá como implementar el RSSI mediante el CORDIC.

2.1.8. Indicador de la calidad del enlace

El indicador de la calidad del enlace o LQI (Link Quality Indicator) es la medida del nivel de calidad de un paquete recibido. Según el estándar IEEE 802.15.4, puede ser implementado a partir del RSSI, estimando el SNR o una combinación de ambos.

2.1.9. CCA

El CCA o *Clear Channel Assessment* permite hacer medidas en el canal para determinar si éste esta libre. El estándar define 3 métodos de determinar el estado del canal:

CCA Modo 1: Energía a partir del umbral CCA determina el estado del canal a partir del nivel de energía en éste sobre un umbral definido.

CCA Modo 2: Detección de portadora CCA determina si el canal esta ocupado o no si es capaz de percibir una señal que cumpla con las características de modulación y expansión de espectro que define el estándar IEEE 802.15.4

CCA Modo 3: Detección de portadora con energía a partir del umbral CCA utiliza la combinación de los dos modos anteriores para determinar el estado del canal.

2.2. PDU

La PDU del estándar IEEE 802.15.4 se compone de las siguientes partes:

SHR Es cabecera de sincronización que permite al dispositivo receptor sincronizarse con el emisor. Está compuesta por el preámbulo y el campo de comienzo de datos o SFD.

PHR Contiene la información sobre la longitud del paquete.

payload Es la información contenida en el paquete, de longitud variable.

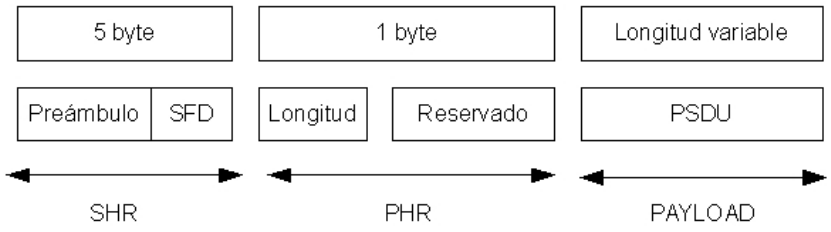

Figura 2.6: PDU de 802.15.4 para 2.4 GHz.

En la Figura 2.6 puede verse la estructura de un paquete IEEE 802.15.4.

El preámbulo es usado por el receptor para lograr la sincronización de símbolo con el mensaje entrante. El estándar define para la capa física que opera sobre la banda ISM de 2.4 GHz una la longitud de preámbulo de 4 octetos u 8 símbolos cero.

El campo SFD indica el final de la cabecera SHR y el comienzo de los datos del paquete. Para la capa física que opera en la banda 2400-2483.5 MHz la longitud del SHR es de 1 octeto o 2 símbolos, y se representa por la secuencia siguiente: 11100101, donde el LSB es el bit más a la izquierda.

Capítulo 3

Generación numérica de señales

3.1. Justificación

A la hora de diseñar un sistema digital hay que buscar un compromiso entre la precisión de la representación digital de las señales y los requerimientos hardware de la implementación. Una precisión elevada es esencial para minimizar los errores de fase de demodulación, sin embargo complican de forma importante la implementación con el consiguiente aumento de los costes de realización y de los consumos de potencia que pueden llegar a ser inaceptables si el objetivo es el diseño de un dispositivo de bajo coste.

En el dominio digital la señal no se representa como una variación continua en el tiempo de una magnitud eléctrica (corriente o tensión) sino como una sucesión de muestras a las que se asocia un valor numérico que representa la intensidad de la señal en el instante de muestreo. Dicho valor se obtiene después de un proceso de conversión del dominio analógico al digital realizado mediante circuitos convertidores. Obviamente, el proceso de conversión no es perfecto y conlleva a un error que depende del número de bits que disponemos para representar la muestra. Por ello, escoger el número de bits adecuado para la representación de las muestras de las señales es crucial para obtener una realización que minimice los errores en demodulación y al mismo tiempo no requiera un hardware excesivamente complicado.

Tal como se mostró en la Sección 2.1.3 una señal $s(t)$ se representa en el dominio complejo como la combinación de dos sinusoides que realizan dos modulaciones binarias ortogonales. Por tanto, la operación de demodulación consiste en reconstruir la señal $s(t)$ a partir de una serie de muestras complejas que, por sencillez, se representan como números en coma fija de 10 bits con 5 bits de parte fraccionaria.

3.2. Algoritmo de Kay

El algorimo de Kay es un estimador de frecuencia para sinusoides complejos que se basa en la técnica de predicción lineal. En la parte demoduladora del módem los datos llegan afectados por ruido de naturaleza Gaussiana lo que se traduce en cambios de fase que pueden llegar a interpretarse como datos erróneos.

El algoritmo de Kay nos permite hacer una corrección "ciega" de los datos entrantes según el valor promedio de diferencia de fases entre muestras medidas. En el proyecto utilizamos la versión sin pesos del algoritmo (para más información, puede consultarse el documento original de Steven Kay en [7]) y un retardo de medición entre muestras D = 1.

El estimador de Kay viene dado por:

$$\hat{\omega} = \frac{1}{D} \sum_{t=0}^{N-2} w_t \angle(x_t * x_{t+D})$$

donde N es el número de muestras usadas en la estimación y D el retardo de medición entre muestras. La función de pesos para la versión generalizada del estimador sin pesos del estimador de Kay es $\frac{1}{N-1}$ de manera que el estimador queda de la siguiente forma:

$$\hat{\omega_0} = \frac{1}{N-1} \sum_{t=0}^{N-2} w_t \angle(x_t * x_{t+D}) =$$

$$\frac{1}{N-1} \sum_{t=0}^{N-2} \angle(x_{t+1}) - \angle(x_t)$$

La idea en la que se basa el estimador es que la frecuencia viene dada por el desplazamiento de fases entre el tiempo, es decir $\omega = \frac{\theta_D - \theta_O}{D}$ donde ω_D es la fase en el instante D y ω_0 en el instante actual y D el retardo.

La elección del parámetro D depende del SNR. A mayor retardo, la estimación será mejor generalmente y el umbral del SNR mayor. En nuestro caso, como queremos optimizar el area y los consumos, hacemos que el estimador trabaje a un SNR bajo a costa de que el estimador dependa de muchas más muestras para obtener una buena estimación. Así, el parámetro D escogido es 1.

La estructura de bloques del algoritmo puede verse en la Figura 3.1. En el cuadro siguiente se muestra el resultado del algoritmo de Kay sobre 8 muestras complejas, donde la división por el número de muestras se aproxima con un desplazamiento a la derecha (en este caso por 3).

It.	I	Q	θ_{t+1}	θ_t	Σ	$>> 3$
1	0000001100	0000001100	0000011001	0000000000	0000011001	
2	0000010111	0000010111	0000011001	0000011001	0000011001	
3	0000011110	0000011110	0000011001	0000011001	0000011001	
4	0000100000	0000100000	0000011001	0000011001	0000011001	
5	0000011110	0000011110	0000011001	0000011001	0000011001	
6	0000001100	0000001100	0000011001	0000011001	0000011001	
7	0000000000	0000000000	0000000000	0000011001	0000000000	
8	1111110100	0000001100	0001001011	0000000000	0001001011	0000001001

Cuadro 3.1: Ejemplo de estimación con el algoritmo de Kay sobre 8 muestras.

Más información sobre otros estimadores de frecuencia (incluido Kay) y distintas comparativas de rendimiento, pueden consultarse en [8].

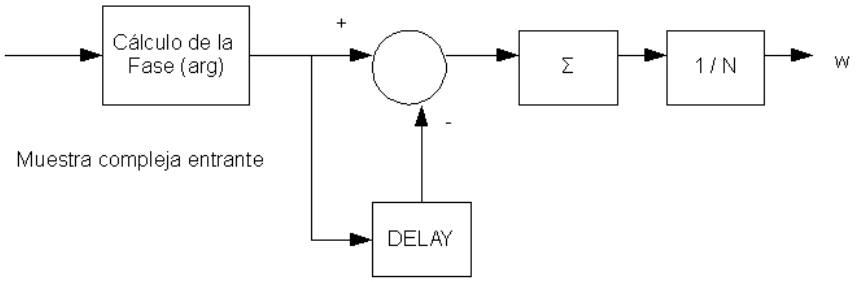

Figura 3.1: Diagrama de bloques del estimador de Kay sin pesos.

3.3. CORDIC

El CORDIC (Coordinate Rotation Digital Computer) permite implementar funciones elementales (trigonómetricas e hiperbólicas) usando una configuración de hardware mínima mediante operaciones sencillas desplazamientos, comparaciones, sumas y restas.

El CORDIC trabaja mediante rotaciones en el campo complejo a través de ángulos constantes hasta que el ángulo es reducido a cero. Estos ángulos se seleccionan de forma que las rotaciones puedan implementarse mediante desplazamientos, sumas y restas.

El articulo original donde apareció descrito el algoritmo puede consultarse en [9] y una generalización del mismo en [10]. Sobre las distintas formas de implementación en hardware puede consultarse [11]. La demostración

matemática que sigue este apartado está basada en [12].

3.3.1. Demostración matemática

El CORDIC rota un valor complejo en el plano, transforma un vector (x_i, y_i) en otro (x_j, y_j) por un ángulo θ. (Figura 3.2) Una rotación en el

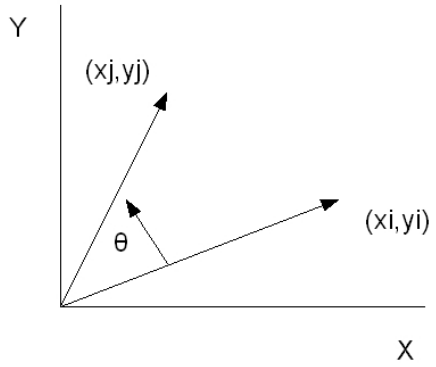

Figura 3.2: Rotación de un valor complejo por θ.

plano complejo puede expresarse en forma matricial como:

$$\begin{bmatrix} X_j \\ Y_j \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \end{bmatrix}$$

El ángulo de rotación θ puede ejecutarse en pequeños pasos de forma iterativa, donde en cada iteración se ejecuta una rotación. Un paso se definirá así:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \\ \sin\theta_n & \cos\theta_n \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix}$$

Podemos pasar de tener cuatro productos a tres eliminando $\cos\theta_n$:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos\theta_n \begin{bmatrix} 1 & -\tan\theta_n \\ \tan\theta_n & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix}$$

El resto de productos pueden eliminarse eligiendo los ángulos de las iteraciones de forma que la tangente del ángulo sea una potencia de dos. (Multiplicar o dividir por una potencia de dos puede implementarse con un desplazamiento).

El ángulo para cada iteración se expresa como:

$$\theta_n = \arctan\frac{1}{2^n}$$

La suma de los ángulos de cada iteración será igual al ángulo θ:

$$\sum_{n=0}^{\infty} S_n \theta_n = \theta$$

donde

$$S_n = \{-1, +1\}$$

y

$$\tan \theta_n = S_n 2^{-n}$$

sustituyendo:

$$\begin{bmatrix} X_{n+1} \\ Y_{n+1} \end{bmatrix} = \cos \theta_n \begin{bmatrix} 1 & -S_n 2^{-n} \\ S_n 2^{-n} & 1 \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \end{bmatrix}$$

Así, el algoritmo ha sido reducido a desplazamientos y sumas, excepto el coeficiente $\cos \theta_n$ que puede aplicarse al final. Podemos aproximar el valor del coeficiente como:

$$\cos \theta_n = \cos \left[\arctan \left(\frac{1}{2^n} \right) \right]$$

Y para todos los valores de n multiplicando los resultados, obtenemos:

$$K = \prod_{n=0}^{\infty} \cos \left[\arctan \left(\frac{1}{2^n} \right) \right] \approx 0{,}607253$$

que tomaremos como constante K.

Ahora, CORDIC puede expresarse como:

$$X_j = K(X_i \cos \theta - Y_i \sin \theta)$$

$$Y_j = K(Y_i \cos \theta + X_i \sin \theta)$$

Introducimos una nueva variable, Z, que representa la parte del ángulo de rotación que aun no ha sido rotada todavía. Si en cada rotación, S_n se calcula de forma que vale -1 si Z_n es menor que cero, o + 1 si es mayor y de forma sucesiva vamos reduciendo el valor de Z a cero mediante valores de arcotangente ya calculados (por ejemplo en una tabla), obtenemos el modo de rotación del CORDIC:

Para $n = 0$ hasta ∞

Si $Z_n \geq 0$

$$X_{n+1} = X_n - \frac{Y_n}{2^n}$$
$$Y_{n+1} = Y_n + \frac{X_n}{2^n}$$
$$Z_{n+1} = Z_n - \arctan\left[\frac{1}{2^n}\right]$$

Si no

$$X_{n+1} = X_n + \frac{Y_n}{2^n}$$
$$Y_{n+1} = Y_n - \frac{X_n}{2^n}$$
$$Z_{n+1} = Z_n + \arctan\left[\frac{1}{2^n}\right]$$

Aplicando al final la constante K.

Si en lugar de reducir Z a cero, reducimos el valor de Y, estamos ante el modo vectorial del CORDIC que nos permite calcular el módulo y fase de un valor complejo: el módulo aparecera en el valor X, puesto que hemos rotado el complejo original sobre el eje de abcisas, la fase aparecerá en Z como acumulado de ángulos sumados o restados según el valor de Y durante el proceso.

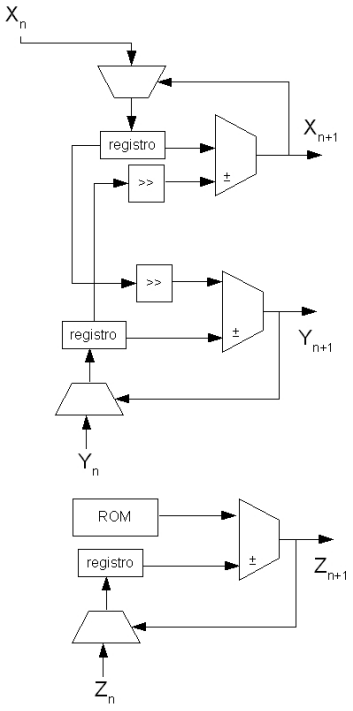

Figura 3.3: Estructura de la implementación del CORDIC iterativa

3.3.2. Consideraciones de implementación

Dado que en el módem necesitamos estimar fases (uso del CORDIC en modo vectorial) y corregirlas (uso de CORIDC en modo rotacional) necesitamos ambas implementaciones del CORDIC. Tenemos tres puntos a tener en cuenta:

1. Como manejar los valores de los ángulos.
2. Como representar los ángulos y que error de aproximación obtenemos.
3. Número de iteraciones.

La implementación, por sencillez, ha sido de forma iterativa (Figura 3.3). Los valores de los ángulos están precalculados en una tabla y representados sobre 10 bits: 5 para la parte decimal y el resto para la parte fraccionaria. Esto nos da los siguientes errores relativos de aproximación:
El número de iteraciones elegido ha sido seis. La constante se implementó co-

Ángulo(grados)	Ángulo (radianes)	Representación	Error
1.78º	0.0273 rad	0000000001	14.65 %
3.57º	0.0585 rad	0000000010	6.4 %
7.12º	0.1210 rad	0000000100	3.30 %
14.03º	0.2421 rad	0000001000	3.16 %
26.56º	0.4609 rad	0000001111	1.71 %
45º	0.7851 rad	0000011001	0.48 %

Cuadro 3.2: Errores de aproximación en la representación de los valores de los ángulos del CORDIC

mo un desplazamiento por dos, aproximando a 0.5 con su determinado error. La forma de probar la implementación del CORDIC del módem ha sido la siguiente:

Lo ideal sería que un circuito pudiera probarse así mismo, generando sus propias entradas, tratando los datos generados y comprobando las salidas respecto a las entradas, en definitiva, que sea éste quien analice su propia precisión. El testbench del CORDIC genera 100 tuplas de 3 elementos de forma pseudo-aleatoria que serían: la parte real de un número complejo, la parte imaginaria de un número complejo y un ángulo de rotación.

Posteriormente cada uno de los 100 números complejos es rotado por cada ángulo y se calcula el error absoluto respecto a la diferencia de fases entre el ángulo rotado y sin rotar contra el ángulo de rotación. De esta forma estaríamos probando: la implementación rotacional del CORDIC (rotaciones) y la implementación vectorial del CORDIC (cálculo de fases).

A continuación se representan los errores para las 100 rotaciones, siendo el máximo error de rotación de 1.563000e-001.

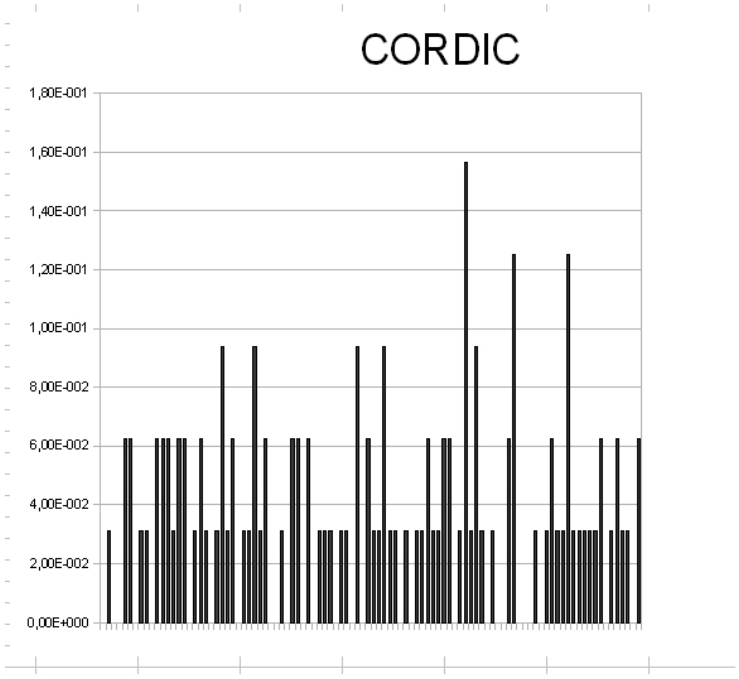

Figura 3.4: Resultados del testbench del CORDIC.

3.3.3. Ejemplo de uso

Con el fin de ilustrar el funcionamiento del CORDIC, a continuación se presenta la rotación del valor complejo 0.5 + 0.5i (fase de 45º o 0.7854 radianes))por un ángulo de 30º (0.5236 radianes) (Cuadro 3.2). El resultado de la rotación tras 6 iteraciones del CORDIC es el complejo 0.1250 + 0.5313i (fase de 76.7607º), donde el ángulo rotado es de 31.7607º, aproximadamente el ángulo elegido: 30º.

Después de las 6 iteraciones, se debe aplicar la constante K, 0.607, que se implementa como un desplazamiento a la derecha (aproximando a 0.5). El resultado final es: X_n = 0000000100 (0.1250) y Y_n = 0000010001 (0.5313).

Iteración	X_n	Y_n	Z_n
1	0000000000 (0)	0000100000 (1)	1111111000 (-0.2500)
2	0000010000 (0.5000)	0000100000 (1)	0000000111 (0.2188)
3	0000001000 (0.2500)	0000100100 (1.1250)	1111111111 (-0.0313)
4	0000001100 (0.3750)	0000100011 (1.0938)	0000000011 (0.0938)
5	0000001010 (0.3125)	0000100011 (1.0938)	0000000001 (0.0313)
6	0000001001 (0.2813)	0000100011 (1.0938)	0000000000 (0)

Cuadro 3.3: Ejemplo de uso de la implementación del CORDIC en modo rotacional

3.4. Estimación de frecuencia y correción de fase en el módem

En un sistema de comunicación inalámbrico las imperfecciones de los osciladores en el transmisor y receptor dan lugar a un *offset* en la portadora al ser detectada en el receptor. Este offset causa una rotación de la constelación de la señal que debe ser corregida para obtener los datos originales en el proceso de demodulación.

El modem utiliza la acción combinada de dos métodos de estimación de frecuencia y corrección de fase. En primer lugar utilizamos el estimador de Kay para hacer una primera estimación y una vez nos hemos sincronizado con los datos recibidos, hacemos una segunda estimación "fina" conociendo los datos que estamos recibiendo. La implementación del estimador de Kay

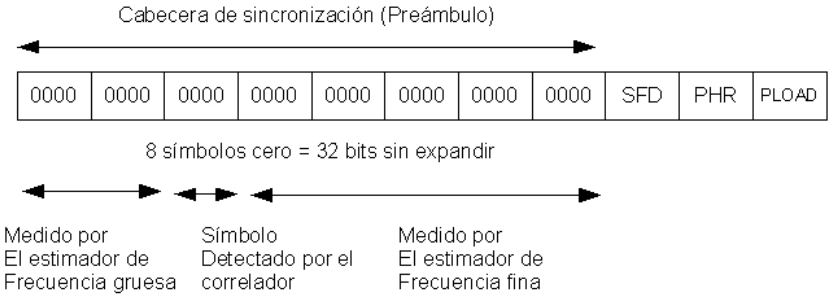

Figura 3.5: Esquema de estimación y correción de fase.

utiliza el CORDIC para estimar las fases de las muestras entrantes durante la medición, y el modo de rotación del CORDIC una vez la medición ha terminado y la corrección esta funcionando. El estimador "fino" utiliza la implementación del CORDIC para medir y rotar las muestras, contrastando las mediciones con una tabla de fases esperadas. (Con los datos que deberían

recibirse). Puede verse un esquema de las distintas partes del preámbulo que trata cada estimador en la Figura 3.5.

La PDU del estándar contiene un preámbulo de 8 símbolos cero que son usados para sincronizarse con los datos recibidos, una vez logremos detectar uno de los símbolos cero, solo nos queda esperar a detectar el campo SFD para estar totalmente seguros de que estamos sincronizados con el emisor.

Hay numerosas formas de combinar los dos estimadores, utilizando más o menos parte del preámbulo para lograr una mejor estimación y sincronizarse antes con los datos recibidos.

En este caso, se ha supuesto que el correlador logra detectar el símbolo cero que se encuentra en la tercera posición del preámbulo, esto es, el tercer símbolo cero de la cabecera de sincronización. Por lo tanto el estimador de frecuencia "gruesa" realiza la estimación durante los dos primeros símbolos cero del preámbulo, esto son, expandidos y pasados por el *downsampler*, 256 *chips* a medir. Después, el estimador pasa a corregir las fases de las siguientes muestras entrantes, confiando en que el correlador sea capaz de detectar después el siguiente símbolo cero.

Una vez el correlador ha detectado un símbolo cero, y sabe que, las siguientes muestras forman parte del preámbulo y son datos conocidos (símbolos cero del estándar), activa el estimador de frecuencia "fina", que estima hasta que termina el preámbulo (esta vez, contrastando las fases con los datos esperados. Cuando el estimador "fino" termina de estimar, comienza a corregir las muestras entrantes. Finalmente, una vez el correlador ha detectado el SFD, estamos totalmente seguros de estar sincronizados con el emisor.

Capítulo 4

Arquitectura del sistema

4.1. Introducción

El módem esta compuesto de un controlador general que maneja las partes de modulación (TX) y demodulación (RX) según el modo de funcionamiento: parada, recepción, transmisión o bucle. Ambas partes son manejadas por otros dos controladores independientes que se encargan de que cada uno de los componentes de transmisión y recepción se activen y desactiven en el orden correcto. Finalmente, un módulo auxiliar permite el paso de los datos de transmisión a recepción si el modo bucle está activado. Puede verse un esquema de la arquitectura general del módem en la Figura 4.1.

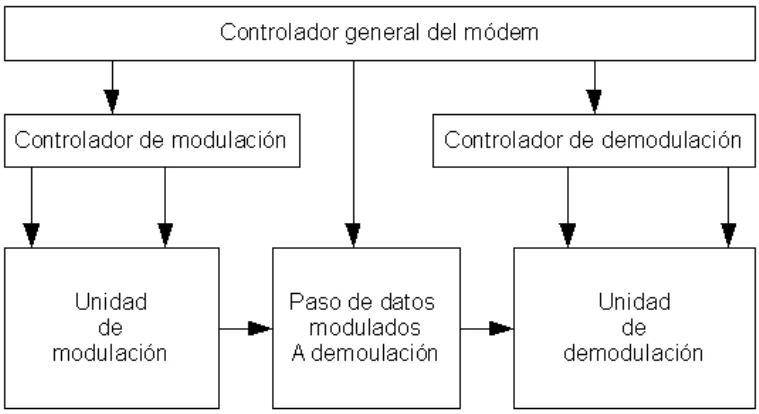

Figura 4.1: Arquitectura del módem.

4.2. Modos de funcionamiento

El módem puede trabajar en los siguientes modos de funcionamiento:

- *TX* Modo de transmisión donde la única salida son los datos modulados a enviar.
- *RX* Modo de recepción donde se demodula la entrada y se obtienen los bits enviados por el emisor.
- *IDLE* Modo de parada, el módem no esta en funcionamiento.
- *LOOP* Modo de bucle, es un modo de test interno donde se conecta la salida de la unidad de modulación a la entrada de la unidad de demodulación para comprobar que la recepción es correcta.

4.3. Puertos

Además de los puertos necesarios para el funcionamiento del módem, se incluyen en la memoria y en el código los usados (debug) para comprobar que todas las unidades de éste funcionan correctamente.

4.3.1. Puertos elementales del módem

modem_clk1 Reloj de entrada a 250 KHz para sincronizar el módem con la entrada binaria a transmitir (250 kbps).

modem_clk2 Reloj de entrada a 1 MHz usado por el generador de *chips* (1 mchips por canal).

modem_clk3 Reloj de entrada a 8 MHz usado por el *upsampler* y el filtro FIR con respuesta al impulso de medio seno.

modem_rst Puerto de *reset* asíncrono.

modem_mode Puerto de entrada para indicar el modo de funcionamiento del módem, codificado sobre 2 bits:

IDLE 00

LOOP 01

TX 10

RX 11

modem_modem_tx_start Puerto de entrada para indicar el inicio de la transmisión.

modem_modem_tx_bit_in Puerto de entrada de la señal binaria a transmitir.

modem_modem_tx_length Entrada del valor de longitud de la PDU de transmisión.

modem_modem_tx_i_out Entrada de los datos modulados para transmitir correspondientes al canal I.

modem_modem_tx_q_out Entrada de los datos modulados para transmitir correspondientes al canal Q.

modem_modem_rx_i_input Entrada de los datos modulados para demodular correspondientes al canal I.

modem_modem_rx_q_input Entrada de los datos modulados para demodular correspondientes al canal Q.

modem_modem_rx_start Puerto de entrada para indicar el inicio de la recepción.

modem_modem_rx_bit_output Puerto de salida con los bits de recepción demodulados.

modem_modem_rx_controller_sfd_detected Puerto de salida que indica que el SFD de la cabecera de la PDU ha sido detectado.

4.3.2. Puertos de *debug* del módem

modem_modem_tx_debug_bit_to_symbol_symbol Salida de *debug* con la salida del conversor bit a símbolo.

modem_modem_tx_debug_bit_to_symbol_valid Salida de *debug* que indica que el conversor de bit a símbolo generó un símbolo correctamente.

modem_modem_tx_debug_chip_gen_i_out Salida del conversor de símbolo a *chip* del canal I.

modem_modem_tx_debug_chip_gen_q_out Salida del conversor de símbolo a *chip* del canal Q.

modem_modem_tx_debug_upsampler_out_i Salida del *upsampler* de transmisión correspondiente al canal I.

modem_modem_tx_debug_upsampler_out_q Salida del *upsampler* de transmisión correspondiente al canal Q.

modem_modem_tx_debug_controller_pre_sent Puerto de salida que indica que el preámbulo fue enviado por la unidad de transmisión.

modem_modem_tx_debug_controller_sfd_sent Puerto de salida que indica que el SFD de la cabecera fue enviado por la unidad de transmisión.

modem_modem_tx_debug_controller_phr_sent Puerto de salida que indica que el PHR del paquete fue enviado por la unidad de transmisión.

modem_modem_rx_debug_symbol Salida de símbolos detectados por el correlador de recepción del módem.

modem_modem_rx_debug_symbol_valid Salida del correlador de recepción para indicar que un nuevo símbolo ha sido detectado correctamente.

modem_modem_rx_debug_coarse_freq_estimator_estim_done Puerto de salida para indicar que la estimación "gruesa" ha terminado de medir la entrada y comienza a corregir las muestras entrantes.

modem_modem_rx_debug_fine_freq_estimator_estim_done Puerto de salida que indica que la estimación "fina" ha terminado de medir la entrada y comienza a corregir las muestras entrantes.

modem_modem_rx_debug_downsampler_output_i Puerto de salida que saca la salida del *downsampler* correspondiente al canal I.

modem_modem_rx_debug_downsampler_output_q Puerto de salida que saca la salida del *downsampler* correspondiente al canal Q.

modem_modem_rx_debug_coarse_freq_estimator_ij Salida del estimador *grueso* correspondiente al canal I.

modem_modem_rx_debug_coarse_freq_estimator_qj Salida del estimador *grueso* correspondiente al canal Q.

modem_modem_rx_debug_fine_freq_estimator_ij Salida del estimador *fino* correspondiente al canal I.

modem_modem_rx_debug_fine_freq_estimator_qj Salida del estimador *fino* correspondiente al canal Q.

modem_modem_rx_debug_rz_encoder_i Salida del codificador RZ de recepción correspondiente al canal I.

modem_modem_rx_debug_rz_encoder_q Salida del codificador RZ de recepción correspondiente al canal Q.

modem_modem_rx_debug_mfilter_rx_i Salida del filtro FIR de recepción correspondiente al canal I.

modem_modem_rx_debug_mfilter_rx_q Salida del fitrlo FIR de recepción corresopndiente al canal Q.

4.4. Controlador del módem

El controlador general del módem comprueba que se puede pasar de un modo a otro sin que se interrumpa el proceso de modulación o demodulación. Información más detallada sobre el controlador se puede encontrar en el Capítulo 7.

4.5. Controlador de transmisión

El controlador de transmisión se ocupa de preparar la cabecera de la PDU y modularla antes de que se se comiencen a modular los datos del emisor y activar todas las partes que forman el proceso de modulación. Información más detallada sobre el controlador de transmisión se puede encontrar en el Capítulo 7.

4.6. Controlador de recepción

El controlador de recepción se ocupa de activar y sincronizar cada una de las partes implicadas en el proceso de demodulación: corrección "gruesa" y "fina" de las muestras entrantes, correlación, *downsampling*, filtrado y conversión a bit. Información más detallada sobre el controlador de recepción se puede encontrar en el Capítulo 7.

4.7. Unidad de modulación

La unidad de modulación (Figura 4.2) se encarga de recibir un flujo binario a modular, con un ratio de 250 kbps. En primer lugar se produce la conversión de bit a símbolo, pasando después por la conversión a secuencia PN. Finalmente los datos son codificados en NRZ, pasados por un *upsampler* de factor 7 y convertidos a una forma de medio seno mediante el *matched filter* definido en el estándar.

Estructura:

Conversor de bit a símbolo Encargado de convertir a símbolo de 4 bits la entrada binaria una velocidad de 250 kbps.

Conversor de símbolo a *chip* Cada uno de los símbolos es transformado en una secuencia PN de 32 *chips* (16 *chips* por canal) a un ratio de 2Mchips por segundo.

***Upsampler* y codificador NRZ** Esta unidad inserta 7 ceros entre cada bit y convierte cada valor en NRZ (valores 1 y -1), codificándolos en complemento a 2 sobre 2 bits.

Matched filter Es el filtro FIR con respuesta al impulso de tipo medio seno que define el estándar IEEE 802.15.4 para la banda ISM.

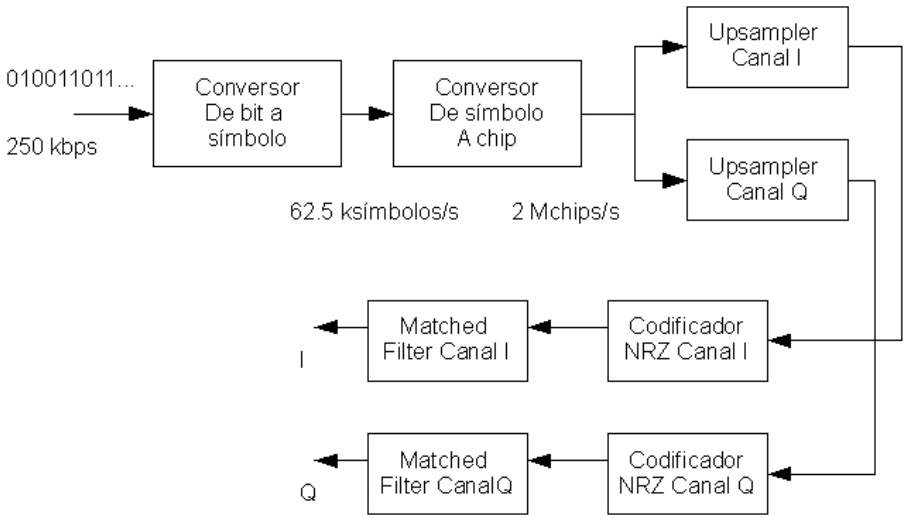

Figura 4.2: Arquitectura de la unidad de modulación del módem.

4.8. Unidad de demodulación

La unidad de demodulación (Figura 4.3) comienza con la estimación de la frecuencia mediante el estimador de Kay, "gruesa", despues los datos son transformados a la forma original, pasándolos por un *downsampler* de factor 7, un filtro de recepción idéntico al de modulación para aumentar el SNR, codificándo los datos sobre RZ y sincronizando la recepción de los datos de forma conjunta mediante el estimador "fino" y el correlador.

Estructura:

Estimador de frecuencia "gruesa" (Kay) Estimador de frecuencia "gruesa" utilizando el estimador de Kay. En primer lugar se hace la estimación durante un periodo determinado de tiempo después del cual, las muestras entrantes son corregidos.

Matched filter Filtro de recepción para aumentar el SNR. Es el mismo filtro utilizado en modulación y el que define el estándar IEEE 802.15.4.

Downsampler *Downsampler* de factor 7 para obtener los datos como originalmente eran.

Estimador de frecuencia "fina" Es el estimador de frecuencia "fina".

Codificador RZ Convierte los datos codificados en NRZ (1, -1) sobre 2 bits en complemento a 2 a formato RZ (0,1).

Correlador de símbolos Es el correlador de recepción capaz de detectar los símbolos.

Conversor de símbolo a bit Finalmente esta unidad convierte los símbolos de 4 bits al flujo binario de ratio 250 kbps original.

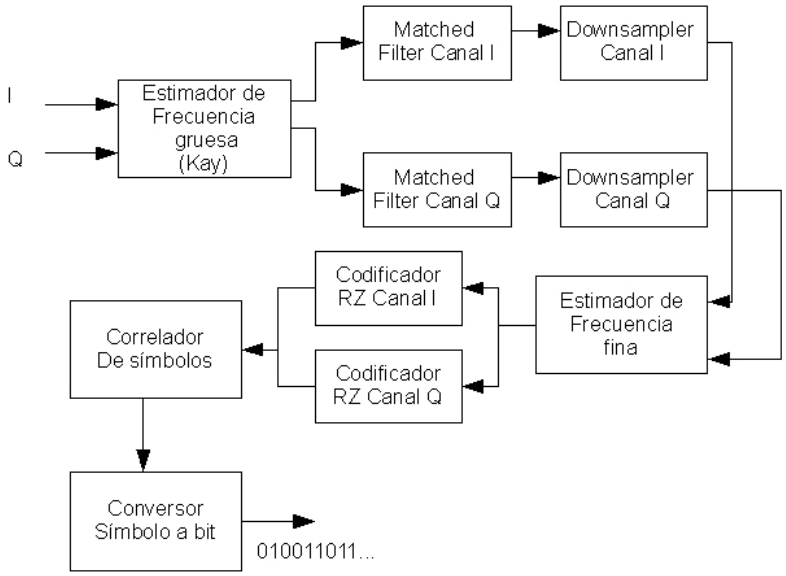

Figura 4.3: Arquitectura de la unidad de demodulación del módem.

Capítulo 5

Subsistema de modulación

5.1. Introducción

La unidad de modulación (Figura 5.1) del módem se encarga de modular los datos antes de proceder a enviarlos por el canal.

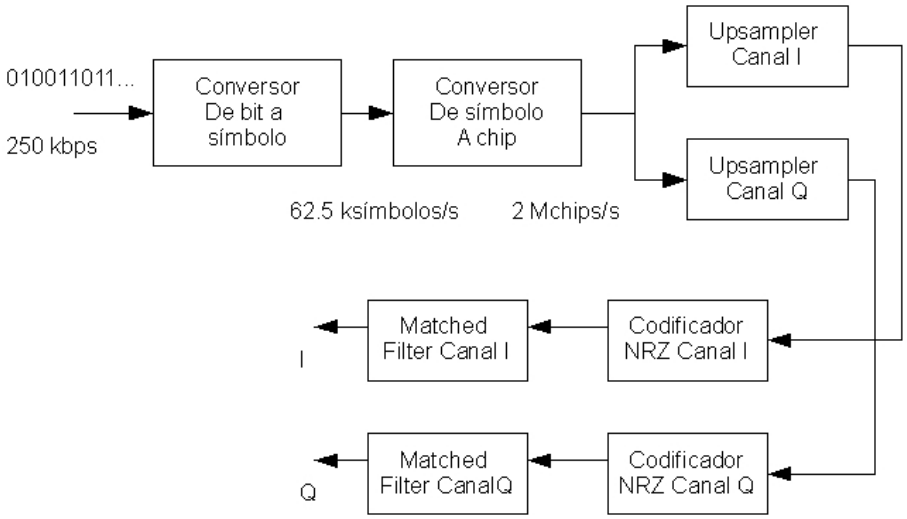

Figura 5.1: Arquitectura de la unidad de modulación del módem.

En este capítulo se describen los puertos de la unidad de modulación del módem y las distintas estructuras que la definen: decisiones de implementación, resultados y alternativas.

5.2. Puertos

5.2.1. Puertos elementales

modem_tx_clk1 Es la entrada de reloj a 250 kbps para sincronizarse con la entrada binaria a modular.

modem_tx_clk2 Reloj a 1 MHz usado por el generador de secuencias PN.

modem_tx_clk3 Reloj a 8 MHz usado por el *upsampler* y el *matched filter*.

modem_tx_start Señal de entrada que indica el inicio de la modulación.

modem_tx_rst Señal de *reset* asíncrona.

modem_tx_bit_in Entrada binaria a modular.

modem_tx_length Valor del campo longitud de la PDU del estándar 802.15.4.

modem_tx_i_out Salida del canal I con los datos modulados.

modem_tx_q_out Salida del canal Q con los datos modulados.

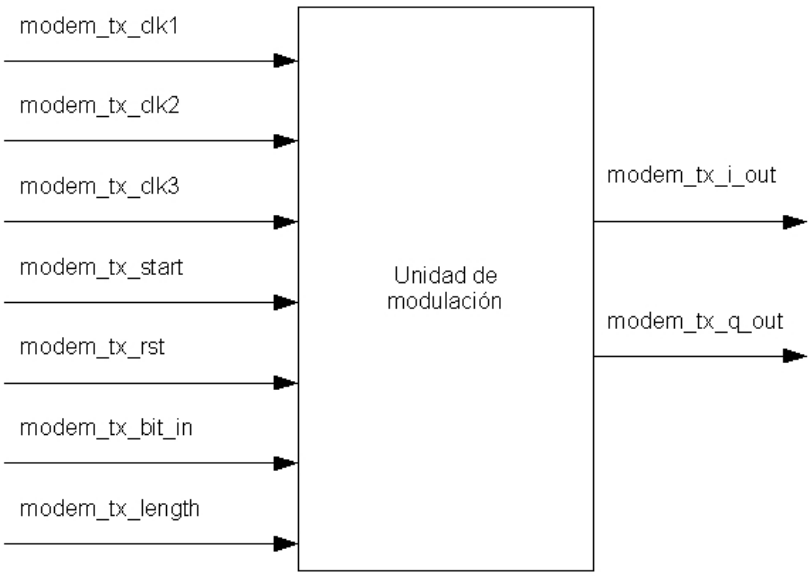

Figura 5.2: Puertos de la unidad de modulación.

5.2.2. Puertos de *debug*

Son los puertos utilizados para comprobar que las estructuras que forman la unidad de modulación funcionan correctamente.

debug_bit_to_symbol_symbol Salida del conversor de bit a símbolo de 4 bits.

debug_bit_to_symbol_valid Salida del conversor de bit a símbolo que indica la salida de un nuevo símbolo.

debug_chip_gen_i_out Salida del conversor símbolo a *chip* del canal I.

debug_chip_gen_q_out Salida del conversor símbolo a *chip* del canal Q.

debug_upsampler_out_i Salida del *upsampler* y codificador NRZ del canal I.

debug_upsampler_out_q Salida del *upsampler* y codificador NRZ del canal Q.

debug_controller_pre_sent Salida que indica que el preámbulo ha sido modulado.

debug_controller_sfd_sent Salida que indica que el campo SFD ha sido modulado.

debug_controller_phr_sent Salida que indica que el campo PHR ha sido modulado.

5.3. Estructura

5.3.1. Conversor de bit a símbolo

Esta parte de la unidad de modulación recibe la entrada binaria a modular de ratio 250 kbps, y la convierte en grupos de 4 bits, símbolos (el estándar define 16 símbolos codificados sobre 4 bits), a un ratio de 62.5 ksímbolos por segundo. Donde cada puerto hace lo siguiente:

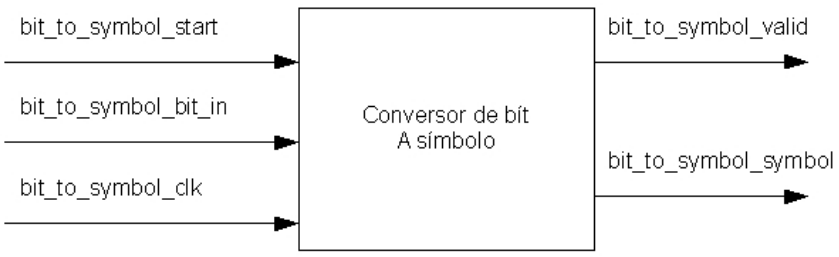

Figura 5.3: Conversor de bit a símbolo.

bit_to_symbol_start Este puerto de entrada indica al conversor que debe empezar a obtener símbolos de la entrada.

bit_to_symbol_bit_in Es la entrada binaria.

bit_to_symbol_clk Reloj a 250 KHz para sincronizarse con la entrada binaria.

bit_to_symbol_symbol_valid Indica que un nuevo símbolo válido ha sido detectado y esta listo en la salida.

bit_to_symbol_symbol Es la salida de los símbolos obtenidos.

5.3.2. Conversor de símbolo a *chip*

Esta parte es la encargada de convertir cada uno de los símbolos en una secuencia PN de *chips*. A partir de aqui, tenemos dos flujos de datos distintos, uno para el canal I y otro para el canal Q. Para cada símbolo, vamos a obtener dos secuencias de 16 bits (32 *chips* en total sin separar en los dos canales con un ratio de 2 Mchips por segundo), fruto de la separación de bits pares e impares (los bits pares pertenecen al canal I y los impares al canal Q). Cada secuencia esta formada por el mismo numero de unos que de ceros, y todas las secuencias pueden generarse a partir de las del símbolo cero mediante rotaciones a la derecha de 4 bits. Las secuencias de los últimos 8 símbolos son generadas invirtiendo los bits pares de las secuencias de los 8 primeros símbolos.

Podemos pensar en dos alternativas a la hora de implementar el generador de secuencias PN, una basándonos en la sencillez y otra buscando reducir el área del circuito:

Generador de secuencias PN basado en ROM Podemos almacenar las 16 secuencias correspondientes a los 16 símbolos en una ROM, y según la entrada (según el símbolo entrante) ir sacando los *chips* a la frecuencia que corresponde. La implementación es sencilla pero también estamos aumentando el área del circuito.

Generador de secuencias PN basado en buffers circulares Aprovechando que todas las secuencias pueden obtenerse mediante rotaciones a partir de la secuencia del primer símbolo (símbolo cero), cargamos en dos buffer circulares las secuencias pertenecientes al canal I y Q del símbolo cero. Según el símbolo entrante tomamos una posición del buffer determinada (correspondiente a ese símbolo) como salida, y vamos rotando los datos, obteniendo la secuencia buscada.

Donde cada uno de los puertos:

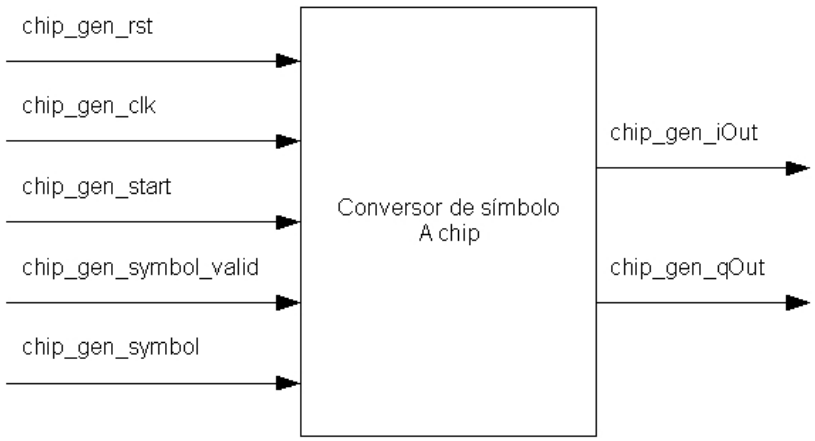

Figura 5.4: Conversor de símbolo a *chip*.

chip_gen_rst Entrada de *reset* asíncrona del circuito.

chip_gen_clk Entrada de reloj a 1 MHz (ratio de 2mchips por segundo entre los dos canales).

chip_gen_start Señal que indica que la entrada de símbolos está activa.

chip_gen_symbol_valid Señal que indica la entrada de un nuevo símbolo.

chip_gen_symbol Entrada de símbolos.

chip_gen_iOut Salida de *chips* correspondientes al canal I.

chip_gen_qOut Salida de *chips* correspondientes al canal Q.

5.3.3. Upsampler

El objetivo del *upsampler* es insertar una serie de ceros (7 en nuestro caso) entre los impulsos entrantes, de manera que las respuestas del filtro FIR no se solapen, evitando así la interferencia de intersímbolo.

La unidad de modulación usa 2 *upsamplers*, uno por canal.
Puertos;

upsampler_output Salida del *upsampler*.

upsampler_input Entrada de los datos al *upsampler*.

upsampler_clk Reloj de 8 MHz.

upsampler_start Señal de entrada que indica que hay nuevos datos a procesar.

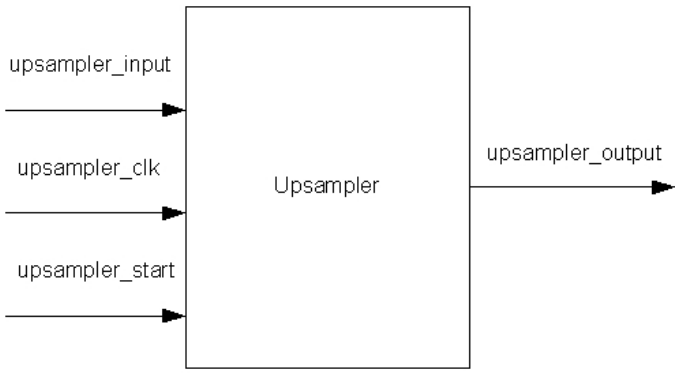

Figura 5.5: *Upsampler.*

5.3.4. Codificador NRZ

El codificador NRZ convierte una entrada RZ en NRZ sobre 2 bits, utilizando complemento a 2. El codificador NRZ se encuentra implementado dentro del *upsampler*.

Entrada	Salida
0	11
1	01

Cuadro 5.1: Conversión RZ a NRZ

5.3.5. Matched filter

El matched filter de la unidad de modulación es filtro FIR con respuesta al impulso de tipo medio seno que define el estándar:

$$p(t) = \begin{cases} \sin(\pi \frac{t}{2T_c}) & \text{si } 0 \leq t \leq 2T_c \\ 0 & \text{en otro caso} \end{cases}$$

Donde los coeficientes del filtro se calculan como:

$$h(nT_s) = h_n \begin{cases} \sin(\pi - \pi \frac{n}{8}) & \text{con n} = 0...7 \\ 0 & \text{en otro caso} \end{cases}$$

Se ha implementado en una estructura de tipo FIR con 8 coeficientes (8 etapas) y un ratio de sampling de 8 MHz. Con 8 etapas hay un buen equilibrio entre retardo de grupo y forma de pulso. A la hora de representar los

coeficientes del filtro se utilizó el formato *fixed-point* con 5 bits dedicados a la parte entera y 5 a la parte fraccionaria, ocasionando los siguientes errores de aproximación:

Coeficiente	Valor	Representación	Aproximación	Error
h0	0	0000000000	0	0 %
h1	0.3826	0000001100	0.3750	2 %
h2	0.7071	0000010111	0.7188	1.65 %
h3	0.9238	0000011110	0.9375	1.48 %
h4	1	0000100000	1	0 %
h5	0.9238	0000011110	0.9375	1.48 %
h6	0.7071	0000010111	0.7188	1.65 %
h7	0.3826	0000001100	0.3750	2 %

Cuadro 5.2: Errores de representación de los coeficientes del filtro

Las operaciones en el filtro de modulación quedan limitadas a sumas, restas y desplazamientos puesto que los datos entrantes son 1, -1 o 0, olvidando multiplicaciones por los coeficientes. La implementación del filtro está basada en el Capítulo 12 de [4]

La implementación del filtro FIR ha sido en forma directa. Para obtener la salida, el filtro sólo se basa en los datos de la entrada actuales y en los anteriores. La salida del filtro puede expresarse simplemente como la convolución de la señal de entrada con la respuesta al impulso del filtro:

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k)$$

Donde M es el orden del filtro, h(k) los coeficientes y x es la entrada.

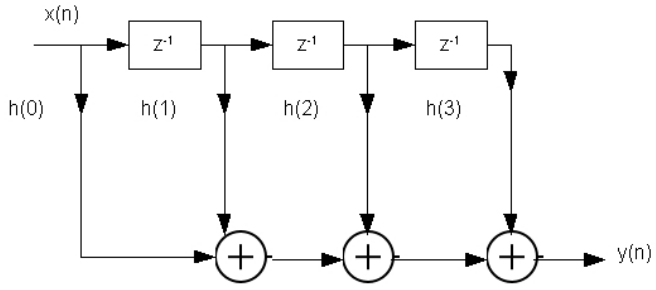

Figura 5.6: Estructura FIR de 4 etapas.

Para cada uno de los canales, I y Q, hay un filtro independiente. En la Figura 5.6 puede verse el diagrama de bloques de un filtro FIR de 4 etapas.

Puertos:

mfilter_input Entrada de los datos al filtro.

mfilter_clk Entrada del reloj de 8 MHz.

mfilter_rst Señal de *reset* asíncrona.

mfilter_output Salida de los datos del filtro.

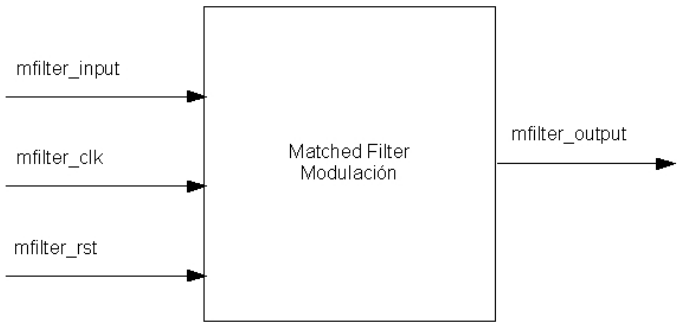

Figura 5.7: Filtro de modulación.

Capítulo 6

Subsistema de demodulación

6.1. Introducción

La unidad de demodulación (Figura 6.1) tiene la labor fundamental de sincronizarse con los datos que envia el emisor.

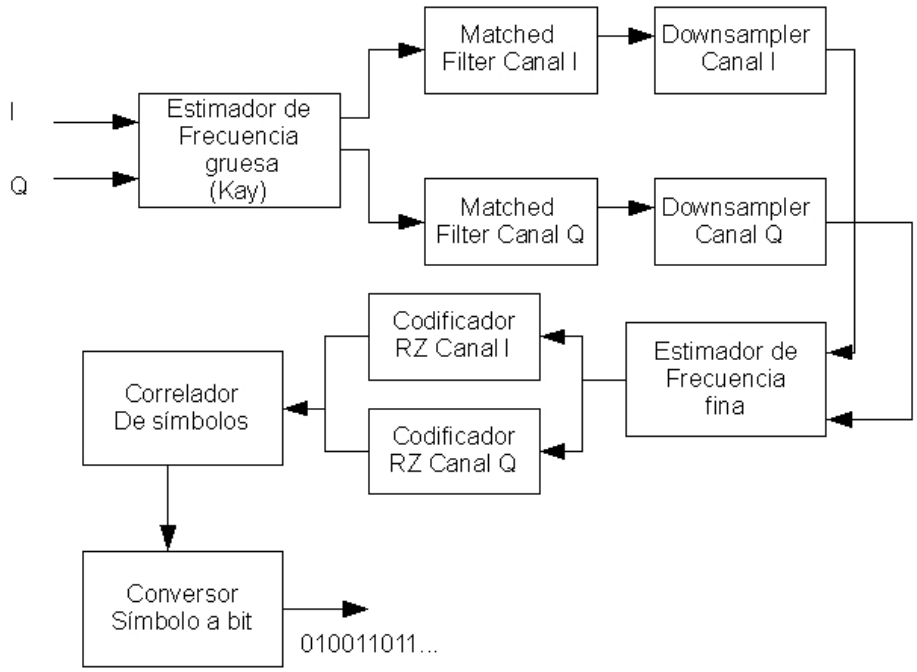

Figura 6.1: Arquitectura de la unidad de demodulación del módem.

Para ello cuenta con dos mecanismos de estimación de frecuencia y corrección de la fase, que actuando de forma conjunta con el correlador de

recepción logra sincronizarse con el emisor, pudiendo obtener los datos originales antes de la modulación.

El filtro de recepción aumenta el SNR, aumentando la probabilidad de éxito en el codificador RZ, que discrimina las muestras entrantes entre 0 o 1.

El *downsampler* se encarga de eliminar los ceros añadidos en la etapa de modulación y el conversor de símbolo a bit en recuperar el flujo binario original que envió el emisor.

6.2. Puertos

6.2.1. Puertos elementales

modem_rx_clk1 Reloj de entrada de 250 kbps.

modem_rx_clk2 Reloj de entrada de 1 MHz.

modem_rx_clk3 Reloj de entrada de 8 MHzz.

modem_rx_start Señal de entrada para indicar el comienzo de la recepción.

modem_rx_rst Señal de *reset* asíncrona.

modem_rx_input_i Entrada de los datos del canal I.

modem_rx_input_q Entrada de los datos del canal Q.

modem_rx_bit_output Salida del flujo binario de datos original.

modem_rx_controller_sfd_detected Señal de salida que indica que el SFD ha sido detectado.

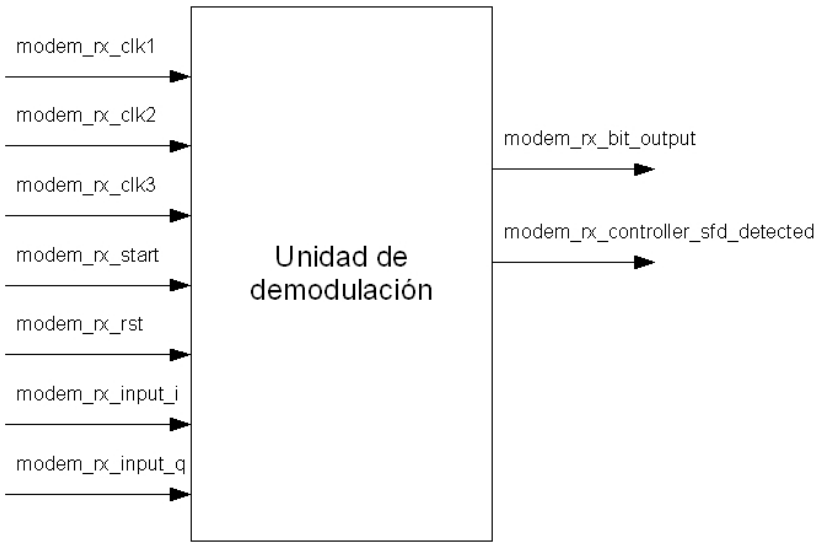

Figura 6.2: Puertos de la unidad de demodulación

6.2.2. Puertos de *debug*

debug_correlator_output Salida de los símbolos detectados por el correlador.

debug_correlator_output_valid Salida que indica que el correlador detectó un símbolo correctamente.

debug_coarse_freq_estimator_estim_done Salida que indica que el estimador de frecuencia "gruesa" ha terminado de estimar.

debug_fine_freq_estimator_estim_done Salida que indica que el estimador de frecuencia "fina" ha terminado de estimar.

debug_downsampler_output_i Salida del *downsampler* para el canal I.

debug_downsampler_output_q Salida del *downsampler* para el canal Q.

debug_coarse_freq_estimator_ij Salida del estimador de frecuencia "gruesa" para el canal I.

debug_coarse_freq_estimator_qj Salida del estimador de frecuencia "gruesa" para el canal Q.

debug_fine_freq_estimator_ij Salida del estimador de frecuencia "fina" para el canal I.

debug_fine_freq_estimator_qj Salida del estimador de frecuencia "fina" para el canal Q.

debug_rz_encoder_i Salida del codificador RZ para el canal I.

debug_rz_encoder_q Salida del codificador Rz para el canal Q.

debug_mfilter_rx_i Salida del filtro de recepción para el canal I.

debug_mfilter_rx_q Salida del filtro de recepción para el canal Q.

6.3. Estructura

6.3.1. Estimador de frecuencia "gruesa"

El estimador de frecuencia "gruesa" está basado en el estimador de Kay. Esta estructura puede configurarse para medir N muestras entrantes antes de comenzar la corrección de fase con las siguientes N + 1 utilizando el parámetro *chips_estim*. Actualmente se encuentra configurado para estimar sobre las primeras 256 muestras entrantes, que corresponden a los dos primeros símbolos cero del preámbulo expandidos y tratados por el *upsampler* de la unidad de modulación. Cuando esta unidad termina de medir, se activa el correlador de recepción en busca de símbolos cero para lograr la sincronización con el emisor. Mientras el estimador esta midiendo ningún dato sale puesto que el resto de estructuras de la unidad de modulación están apagadas hasta que la estimación "gruesa" termina y el correlador es activado para buscar símbolos cero.

El estimador de frecuencia "gruesa" hace uso del módulo del CORDIC para obtener las fases de las muestras complejas entrantes (modo vectorial del CORDIC) y para rotar las muestras (modo rotacional del CORDIC).

Puertos:

coarse_freq_estimator_clk Reloj de entrada que sirve para sincronizarse con los datos entrantes.

coarse_freq_estimator_rst Entrada de *reset* asíncrona.

coarse_freq_estimator_ena Señal de enable para activar la estimación.

coarse_freq_estimator_i Entrada de datos correspondiente al canal I.

coarse_freq_estimator_q Entrada de datos correspondiente al canal Q.

coarse_freq_estimator_estim_done Señal de salida que indica que la estimación ha terminado y comienza la fase de corrección de muestras entrantes.

coarse_freq_estimator_ij Salida de datos correspondiente al canal I.

coarse_freq_estimator_qj Salida de datos coresopndiente al canal Q.

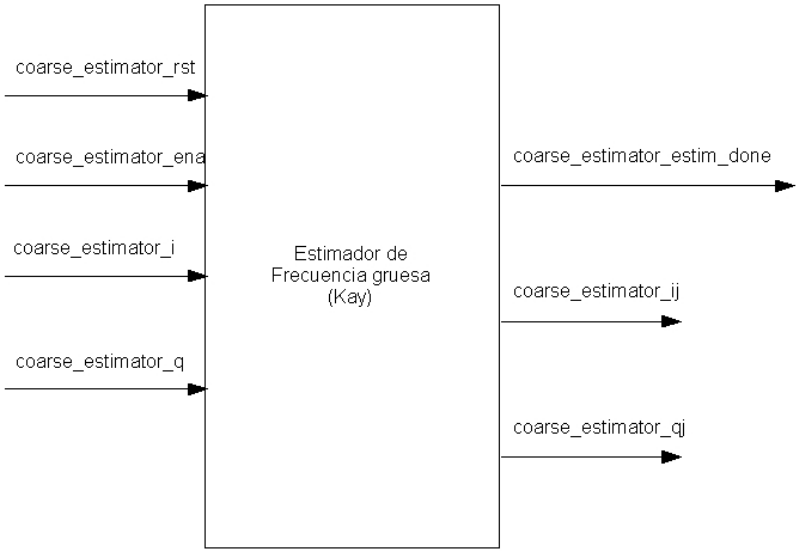

Figura 6.3: Estimador de frecuencia "gruesa"

6.3.2. Filtro de recepción

El filtro de recepción es idéntico al de modulación, permite subir el valor del SNR, lo que es bueno para el codificador RZ al distinguir entre 1 o 0 con mayor probabilidad de acierto.

La estructura e implementación es similar, 8 etapas y 8 MHz de frecuencia de muestreo.

Puertos:

mfilter_input Entrada de datos al filtro.

mfilter_clk Entra del reloj de 8 MHz al filtro.

mfilter_rst Entrada de *reset* asíncrona.

mfilter_output Salida del filtro.

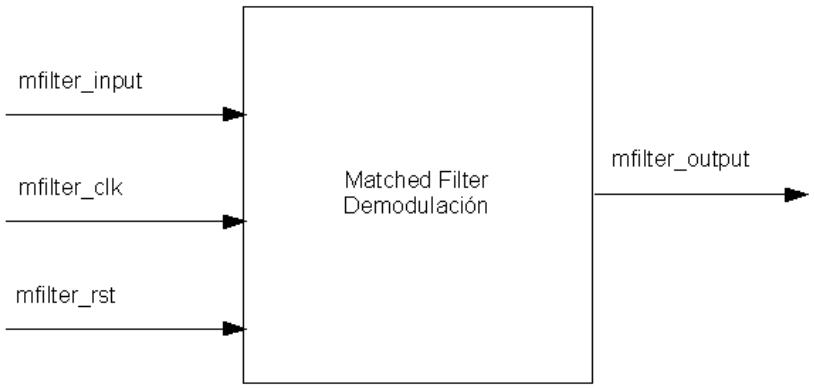

Figura 6.4: Filtro de recepción

6.3.3. Downsampler

El *downsampler* elimina los ceros insertados en el proceso de modulación por el *upsampler*. Hay dos *downsamplers* en la unidad de demodulación, uno para cada canal.

Puertos:

downsampler_clk Entrada del reloj del *downsampler*.

downsampler_start Señal de start que activa el *downsampler*.

downsampler_input Entrada de datos al *downsampler*.

downsampler_output Salida de datos del *downsampler*.

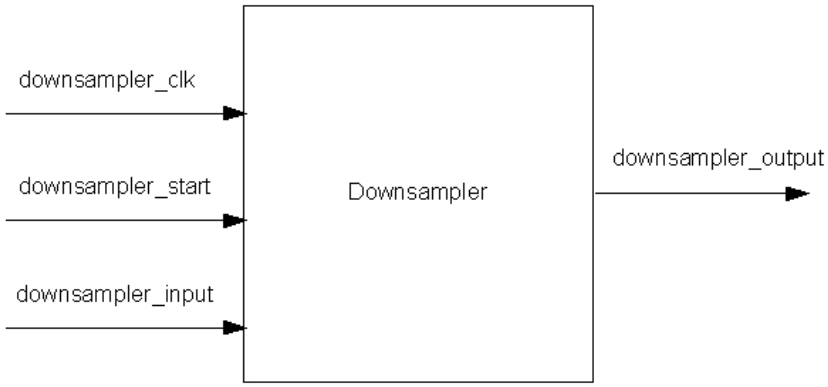

Figura 6.5: Downsampler

6.3.4. Estimador de frecuencia "fina"

El estimador de frecuencia "fina" es activado por el correlador cuando éste detecta un símbolo cero. Es en ese momento cuando se puede hacer una estimación "fina" comparando las fases de las muestras entrantes con las fases esperadas. El estimador de frecuencia "fina" tiene una tabla de fases esperadas con las que comprueba la diferencia existente entre éstas y las de las muestras recibidas, de esta forma la estimación es mucho mas precisa que el caso del estimador de Kay.

El estimador de frecuencia "fina" se configura mediante el parámetro *chips_estim*, que es el número de muestras durante el cual hará la estimación. Después pasa a corregir las fases de los siguientes datos entrantes.

El estimador de frecuencia "fina" hace uso del módulo del CORDIC para estimar las fases de las muestras entrantes y después de hacer la estimación, rotar el resto de muestras entrantes.

Puertos:

fine_freq_estimator_clk Entrada del reloj del estimador.

fine_freq_estimator_rst Entrada de *reset* asíncrona.

fine_freq_estimator_ena Entrada de enable del estimador para que comience la estimación.

fine_freq_estimator_i Entrada de datos del canal I.

fine_freq_estimator_q Entrada de datos del canal Q.

fine_freq_estimator_estim_done Salida que indica que la estimación terminó y comienza la correción de las siguientes muestras entrantes.

fine_freq_estimator_ij Salida de datos del canal I.

fine_freq_estimator_qj Salida de datos del canal Q.

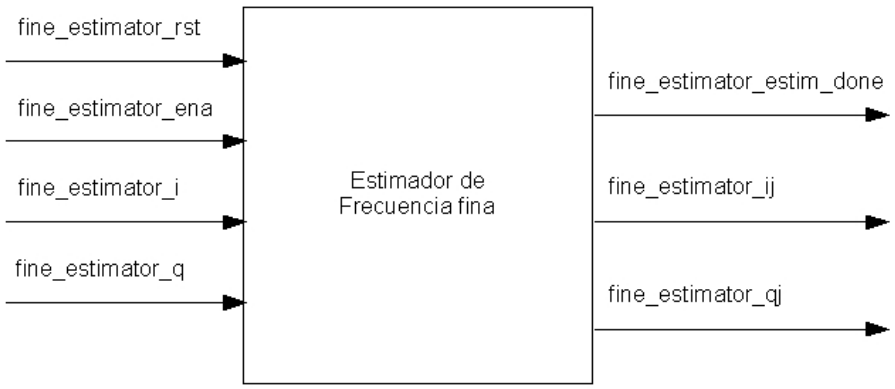

Figura 6.6: Estimador de frecuencia "fina"

6.3.5. Codificador RZ

El codificador RZ tiene que discriminar entre 0 o 1 según la muestra entrante. La unidad de demodulación cuenta con 2 codificadores RZ, uno por cada canal.

Puertos:

rz_encoder_input_i Entrada de datos al codificador RZ del canal I.

rz_encoder_input_j Entrada de datos al codificador Rz del canal Q.

rz_output_i Salida de datos codificados como RZ pertenecientes al canal I.

rz_output_j Salida de datos codificados como RZ pertenecientes al canal Q.

Figura 6.7: Codificador RZ

6.3.6. Correlador de recepción

El correlador de recepción detecta los símbolos de 4 bits del estándar 802.15.4 a partir de la entrada del codificador RZ. Existen múltiples alternativas para hacer la detección de los símbolos, en nuestro caso sólo se usan los datos pertenecientes al canal Q, puesto que solo utilizando los datos del canal I no podríamos distinguir entre la primera mitad de los 16 símbolos y la segunda (Las secuencias PN del canal I para los primeros 8 símbolos son idénticas a las de los 8 últimos).

El correlador implementado está basado en el correlador de recepción del Capítulo 4 de [6] (Figura 6.8). El correlador tiene en un registro almacenado la secuencia PN del símbolo 0 del canal Q. A partir de él puede generar cada una de las secuencias correspondientes a los otros 15 símbolos. (En una manera similar al generador de *chips* de la unidad de modulación).

Una vez el correlador detecta la entrada de datos cada una de las secuencias de referencia que son generadas por el correlador, son multiplicadas por la entrada (al ser datos binarios, esta operación puede implementarse mediante *and*) y el resultado acumulado en 16 registros. Cuando 16 bits han sido analizados, el registro con valor máximo, 8, es el que corresponde al símbolo a detectar. El valor de detección es 8, puesto que cada una de las secuencias contiene el mismo numero de unos que de ceros y estamos analizando secuencias de 16 bits.

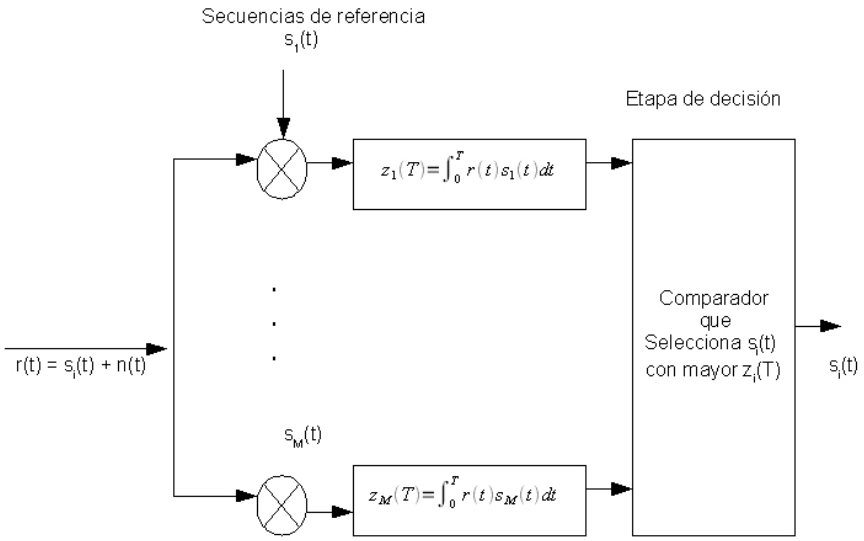

Figura 6.8: Correlador de recepción basado en [6]

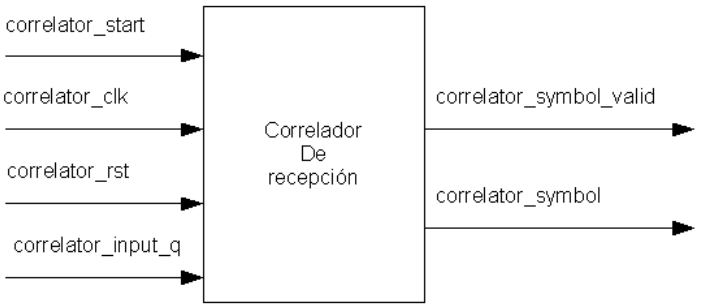

Figura 6.9: Puertos del correlador de recepción

Puertos:

correlator_start Señal de entrada que activa el correlador.

correlator_clk Entra de reloj al correlador.

correlator_rst Señal de entrada de *reset* asíncrona.

correlator_input_q Entrada de datos al correlador del canal Q.

correlator_symbol_valid Señal de salida que indica que un nuevo símbolo ha sido detectado.

correlator_symbol Salida de símbolos detectados por el correlador.

6.3.7. Conversor símbolo a bit

La última estructura de la unidad de demodulación convierte los símbolos de 4 bits al flujo binario original que envió el emisor antes de modular de ratio 250 kbps.

Puertos:

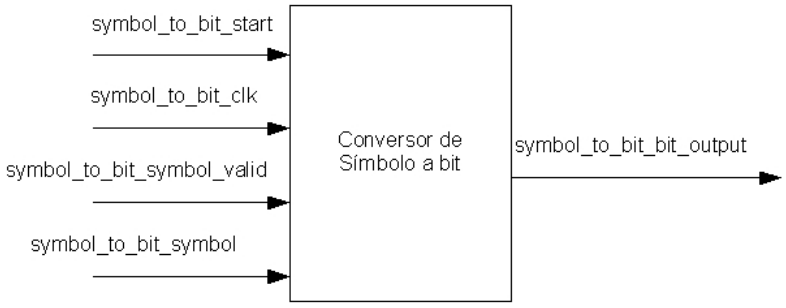

Figura 6.10: Conversor de símbolo a bit

symbol_to_bit_start Señal de entrada que indica el comienzo de la llegada de símbolos.

symbol_to_bit_clk Entrada del reloj al conversor de 250 kbps.

symbol_to_bit_symbol_valid Señal de entrada que le indica al conversor la llegada de un nuevo símbolo válido.

symbol_to_bit_symbol Entrada de símbolos.

symbol_to_bit_bit_output Salida del flujo binario a 250 kbps (Datos demodulados completamente).

6.3.8. RSSI

El sistema indicador de nivel de la señal o RSSI (Received Signal Strength Indicator) puede implementarse mediante el CORDIC. El CORDIC en modo vectorial nos permite convertir un valor complejo de forma cartesiana a forma polar (módulo, fase). Si recordamos, el RSSI venía dado por:

$$RSSI = \frac{1}{N}\sum_{n=0}^{N-1}|m(n)|^2 = \frac{1}{N}\sum_{n=0}^{N-1}s_I^2(n) + s_Q^2(n)$$

Así, solo tenemos que ir acumulando el módulo de las muestras entrantes (calculado con el CORDIC rotacional) durante N muestras y al final, dividir por las N muestras analizadas. Esta división puede implementarse mediante un desplazamiento a la derecha si el número N de muestras es una potencia de 2.

Capítulo 7

Controlador

7.0.9. Controlador general del módem

El controlador general del módem es quien decide si se puede pasar de un modo a otro sin problema de interferir en un proceso que se está llevando a cabo dentro de éste. El controlador recibe tres entradas, una señal que indica el comienzo de una modulación, otra que indica el inicio de la demodulación y otra que es el modo al que cambiar. No se puede cambiar al modo *idle* mientras se esté modulando ni al modo de modulación mientras se esté demodulando. En las Figuras 7.1 y 7.2 pueden apreciarse los diagramas de estados del controlador general.

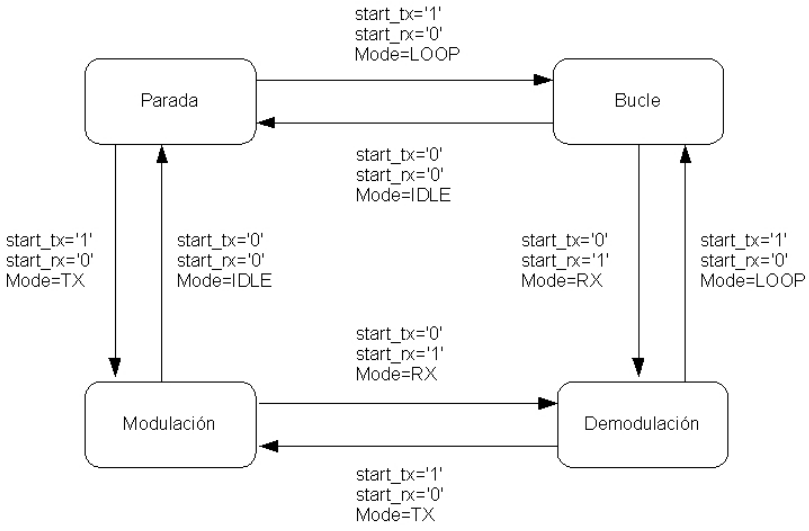

Figura 7.1: Diagrama de estados del controlador general (A)

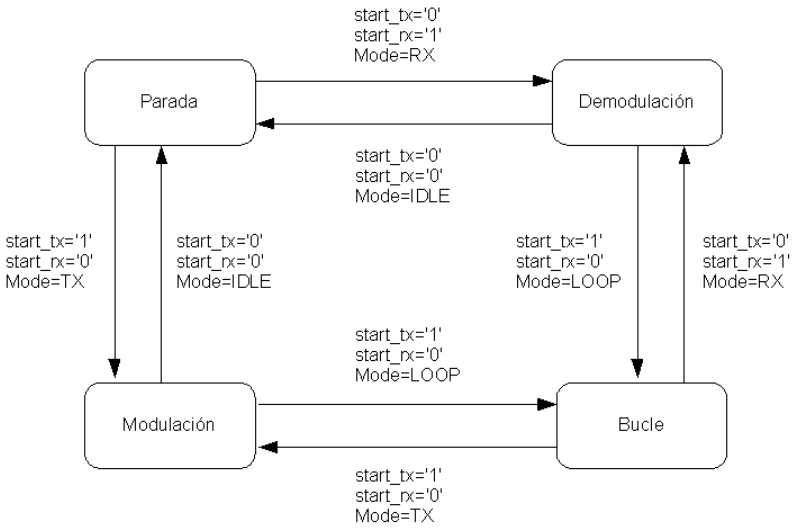

Figura 7.2: Diagrama de estados del controlador general (B)

Puertos del controlador del módem

modem_controller_start_tx_in Señal de entrada que indica que se quiere modular.

modem_controller_start_rx_in Señal de entrada que indica que se quiere demodular.

modem_controller_mode Señal de entrada que indica el modo al que pasar.

modem_controller_start_tx_out Señal de salida hacia el módem, que indica que se va a modular.

modem_controller_start_rx_out Señal de salida hacia el módem, que indica que se va a demodular.

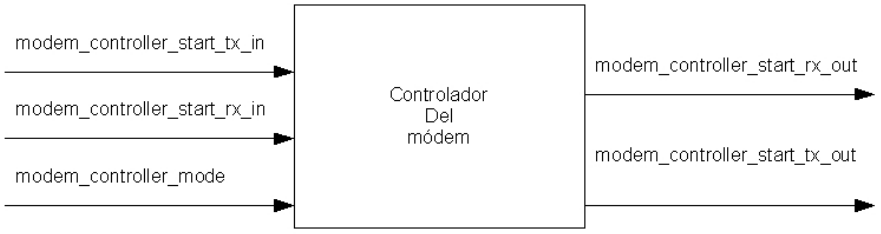

Figura 7.3: Puertos del controlador del módem

Unidad auxiliar *modem_tx_to_rx*

Cuando el modo *bucle* está activo, los datos modulados pasarán a la unidad de demodulación, este paso de datos lo controla el módem mediante la unidad auxiliar *modem_tx_to_rx*:

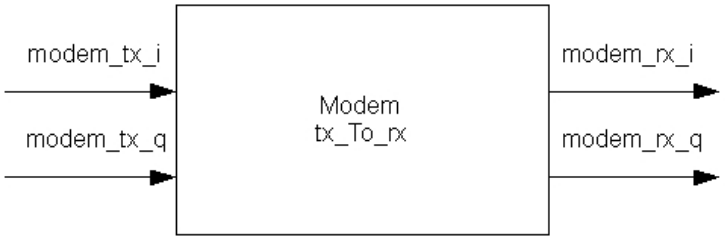

Figura 7.4: Unidad auxiliar de paso de datos

modem_tx_i Entrada de datos modulados del canal I.

modem_tx_q Entrada de datos modulados del canal Q.

modem_rx_i Salida de datos modulados del canal I hacia la unidad de demodulación.

modem_rx_q Salida de datos modulados del canal Q hacia la unidad de demodulación.

7.0.10. Controlador de modulación

El controlador de modulación prepara la cabecera de la PDU cuando recibe la señal de que la modulación debe empezar. En primer lugar envia el preámbulo, 8 símbolos cero. Después envia el SFD y posteriormente la PHR, que incluye el tamaño del PAYLOAD. Finalmente toma la entrada binaria

de datos y comienza su modulación. Durante el proceso, se activan distintas señales de *debug* que indican que el preámbulo, el SFD y la cabecera PHR fueron modulados.

En la Figura 7.5 pueden verse los pasos que sigue el controlador de modulación.

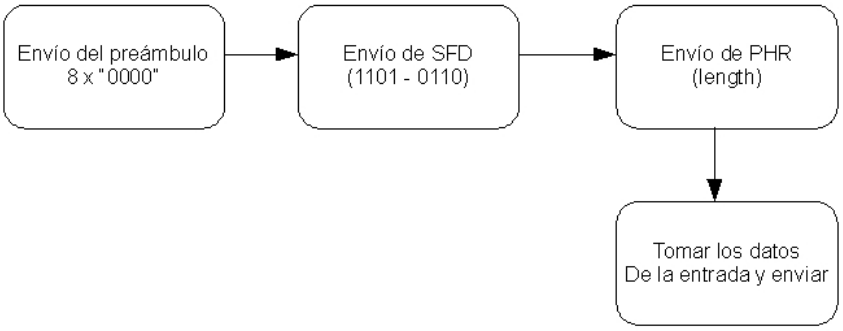

Figura 7.5: Controlador de modulación

Puertos

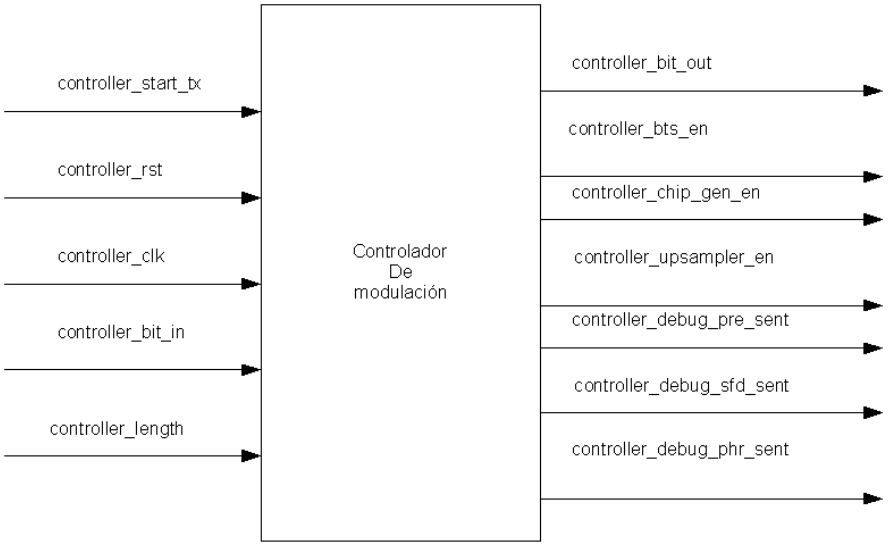

Figura 7.6: Controlador de modulación

controller_start_tx Señal entrada que indica el inicio de la modulación.

controller_rst Señal de *reset* asíncrona.

controller_clk Entrada del reloj al controlador.

controller_bit_in Entrada del flujo binario de datos a modular.

controller_length Entrada del parámetro length de la PDU.

controller_bit_out Salida de datos a modular.

controller_bts_en Señal de activación de conversor bit a símbolo.

controller_chip_gen_en Señal de activación del conversor símbolo a *chip*.

controller_upsampler_en Señal de activación del *upsampler* y codificador NRZ.

controller_debug_pre_sent Señal de salida que indica que el preámbulo fue enviado.

controller_debug_sfd_sent Señal de salida que indica que el campo SFD fue enviado.

controller_debug_phr_sent Señal de salida que indica que el campo PHR fue enviado.

7.0.11. Controlador de demodulación

El controlador de demodulación se activa cuando nuevos datos se van a recibir (señal *start_rx* activa). En primer lugar se activa el estimador de frecuencia "gruesa" (estimador de Kay).

Cuando el estimador de Kay termina, estamos listos para intentar detectar un símbolo cero del preámbulo y sincronizarnos con el emisor. En este paso se activa el correlador y el conversor de símbolo a bit. Cuando el correlador detecta un símbolo cero, avisa al controlador, y éste, una vez sincronizado totalmente con el emisor, activa el estimador de frecuencia "fina". Ahora el estimador de frecuencia "fina" puede hacer una medición precisa, ya que tiene almacenados las fases de los datos que espera recibir y puede contrastarlos con la entrada.

Finalmente, se avisa cuando el SFD es detectado y se continúa demodulando el campo PHR y el PAYLOAD.

En la Figura 7.7 pueden verse los pasos que sigue el controlador de demodulación.

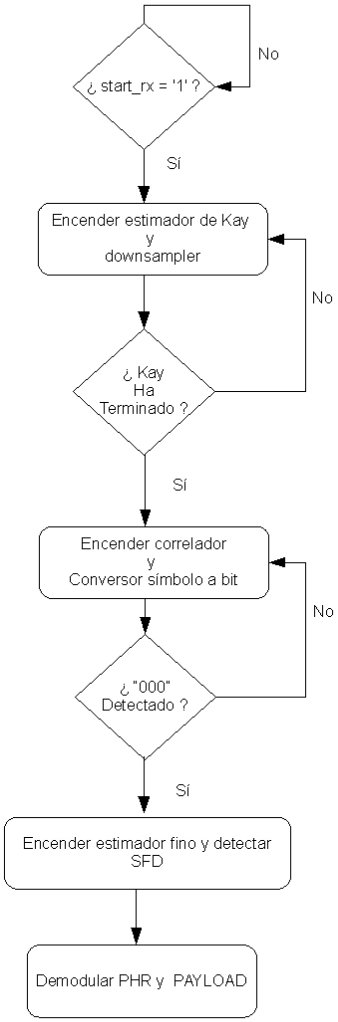

Figura 7.7: Controlador de demodulación

Puertos

controller_start_rx Señal de inicio de demodulación.

controller_symbol_valid El correlador utiliza este puerto para anunciar al controlador que un nuevo símbolo ha sido detectado.

controller_symbol_correlator El correlador utiliza este puerto para anunciar al controlador qué símbolo ha sido detectado (Nosotros estamos buscando el primer símbolo cero del preámbulo detectado para sincronizarnos con el emisor).

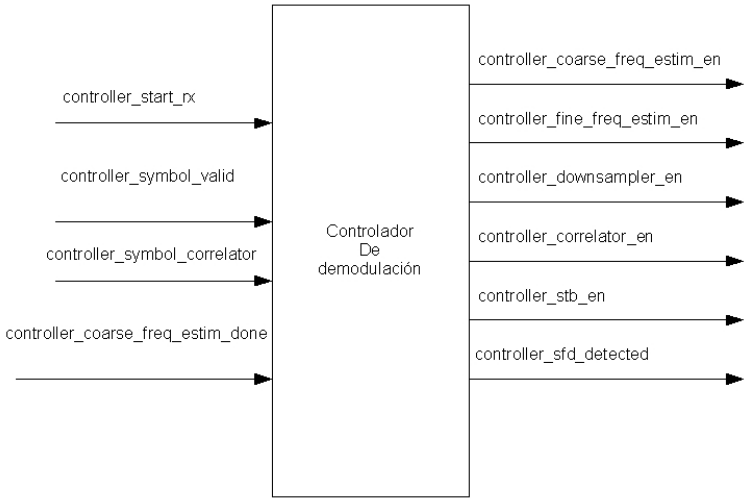

Figura 7.8: Controlador de demodulación

controller_coarse_freq_estim_done El estimador de frecuencia "gruesa" utiliza este puerto para avisar al controlador de que ha terminado de medir y comienza a corregir la fase de las muestras entrantes.

controller_coarse_freq_estim_en Puerto de salida para activar el estimador de frecuencia "gruesa".

controller_fine_freq_estim_en Puerto de salida para activar el estimador de frecuencia "fina".

controller_downsampler_en Puerto de salida para activar el *downsampler*.

controller_correlator_en Puerto de salida para activar el correlador de recepción.

controller_stb_en Puerto de salida para activar el conversor de símbolo a bit.

controller_sfd_detected Puerto de salida para anunciar que el SFD ha sido detectado.

Capítulo 8

Implementación

8.1. modem.vhd

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar.
--
-- Create Date:     12:31:37 09/24/2008
-- Design Name:
-- Module Name:     modem - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- modem.vhd representa el nivel mas alto del módem,
-- por debajo se encuentra el controlador general,
-- las unidades de modulación y demodulación, y la
    estructura auxiliar
-- que controla el paso de los datos de modulación a
    demodulación para el modo bucle.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem is
 port(
```

```
modem_clk1 : in std_logic; -- 250 khz
modem_clk2 : in std_logic; -- 1 mhz
modem_clk3 : in std_logic; -- 8 mhz
modem_rst : in std_logic;
modem_mode : in std_logic_vector(1 downto 0);

-- Modos del Modem
--     IDLE 00
--     LOOP 01
--     TX   10
--     RX   11

modem_modem_tx_start : in std_logic;
modem_modem_tx_bit_in : in std_logic;
modem_modem_tx_length: in std_logic_vector(7 downto 0);
modem_modem_tx_i_out : out std_logic_vector(9 downto 0);
modem_modem_tx_q_out : out std_logic_vector(9 downto 0);

modem_modem_rx_i_input: in std_logic_vector(9 downto 0);
modem_modem_rx_q_input: in std_logic_vector(9 downto 0);
modem_modem_rx_start: in std_logic;
modem_modem_rx_bit_output: out std_logic;
modem_modem_rx_controller_sfd_detected: out std_logic;

modem_modem_tx_debug_bit_to_symbol_symbol: out
    std_logic_vector(3 downto 0);
modem_modem_tx_debug_bit_to_symbol_valid: out std_logic;
modem_modem_tx_debug_chip_gen_i_out: out std_logic;
modem_modem_tx_debug_chip_gen_q_out: out std_logic;
modem_modem_tx_debug_upsampler_out_i: out
    std_logic_vector(1 downto 0);
modem_modem_tx_debug_upsampler_out_q: out
    std_logic_vector(1 downto 0);
modem_modem_tx_debug_controller_pre_sent: out std_logic;
modem_modem_tx_debug_controller_sfd_sent: out std_logic;
modem_modem_tx_debug_controller_phr_sent: out std_logic;

modem_modem_rx_debug_symbol: out std_logic_vector(3
    downto 0);
modem_modem_rx_debug_symbol_valid: out std_logic;
modem_modem_rx_debug_coarse_freq_estimator_estim_done:
    out std_logic;
modem_modem_rx_debug_fine_freq_estimator_estim_done: out
    std_logic;
modem_modem_rx_debug_downsampler_output_i: out
    std_logic_vector(9 downto 0);
modem_modem_rx_debug_downsampler_output_q: out
    std_logic_vector(9 downto 0);
```

```
    modem_modem_rx_debug_coarse_freq_estimator_ij: out
        std_logic_vector(9 downto 0);
    modem_modem_rx_debug_coarse_freq_estimator_qj: out
        std_logic_vector(9 downto 0);
    modem_modem_rx_debug_fine_freq_estimator_ij: out
        std_logic_vector(9 downto 0);
    modem_modem_rx_debug_fine_freq_estimator_qj: out
        std_logic_vector(9 downto 0);
    modem_modem_rx_debug_rz_encoder_i: out std_logic;
    modem_modem_rx_debug_rz_encoder_q: out std_logic;
    modem_modem_rx_debug_mfilter_rx_i: out std_logic_vector(9
         downto 0);
    modem_modem_rx_debug_mfilter_rx_q: out std_logic_vector(9
         downto 0)

);

end modem;

architecture Behavioral of modem is
 constant MODE_IDLE : std_logic_vector(1 downto 0) := "00";
 constant MODE_LOOP  : std_logic_vector(1 downto 0) := "01"
     ;
 constant MODE_TX    : std_logic_vector(1 downto 0) := "10"
     ;
 constant MODE_RX    : std_logic_vector(1 downto 0) := "11"
     ;

 component modem_controller is
  port( modem_controller_start_tx_in : in std_logic;
        modem_controller_start_rx_in : in std_logic;
        modem_controller_mode: in std_logic_vector(1
            downto 0);
        modem_controller_start_tx_out : out std_logic;
        modem_controller_start_rx_out : out std_logic);
 end component;

 component modem_tx is
  port( modem_tx_clk1 : in std_logic;
        modem_tx_clk2 : in std_logic;
        modem_tx_clk3 : in std_logic;
        modem_tx_start : in std_logic;
        modem_tx_rst : in std_logic;
        modem_tx_bit_in : in std_logic;
        modem_tx_length: in std_logic_vector(7 downto 0);
        modem_tx_i_out : out std_logic_vector(9 downto 0);
        modem_tx_q_out : out std_logic_vector(9 downto 0);
        debug_bit_to_symbol_symbol: out std_logic_vector(3
            downto 0);
```

```
        debug_bit_to_symbol_valid: out std_logic;
        debug_chip_gen_i_out: out std_logic;
        debug_chip_gen_q_out: out std_logic;
        debug_upsampler_out_i: out std_logic_vector(1
            downto 0);
        debug_upsampler_out_q: out std_logic_vector(1
            downto 0);
        debug_controller_pre_sent: out std_logic;
        debug_controller_sfd_sent: out std_logic;
        debug_controller_phr_sent: out std_logic);
end component;

component modem_rx is
 port (modem_rx_clk1: in std_logic;
        modem_rx_clk2: in std_logic;
        modem_rx_clk3: in std_logic;
        modem_rx_start: in std_logic;
        modem_rx_rst: in std_logic;
        modem_rx_input_i: in std_logic_vector(9 downto 0);
        modem_rx_input_q: in std_logic_vector(9 downto 0);

        modem_rx_bit_output: out std_logic;
        modem_rx_controller_sfd_detected: out std_logic;

        debug_correlator_output: out std_logic_vector(3
            downto 0);
        debug_correlator_output_valid: out std_logic;
        debug_coarse_freq_estimator_estim_done: out
            std_logic;
        debug_fine_freq_estimator_estim_done: out
            std_logic;
        debug_downsampler_output_i: out std_logic_vector(9
             downto 0);
        debug_downsampler_output_q: out std_logic_vector(9
             downto 0);
        debug_coarse_freq_estimator_ij: out
            std_logic_vector(9 downto 0);
        debug_coarse_freq_estimator_qj: out
            std_logic_vector(9 downto 0);
        debug_fine_freq_estimator_ij: out std_logic_vector
            (9 downto 0);
        debug_fine_freq_estimator_qj: out std_logic_vector
            (9 downto 0);
        debug_rz_encoder_i: out std_logic;
        debug_rz_encoder_q: out std_logic;
        debug_mfilter_rx_i: out std_logic_vector(9 downto
            0);
        debug_mfilter_rx_q: out std_logic_vector(9 downto
            0));
```

```
        end component;

        component modem_tx_to_rx is
         port (modem_tx_i: in std_logic_vector(9 downto 0);
          modem_tx_q: in std_logic_vector(9 downto 0);
                modem_rx_i: out std_logic_vector(9 downto
                    0);
                modem_rx_q: out std_logic_vector(9 downto
                    0));
        end component;

        signal modem_tx_output_i_temp,
            modem_tx_output_q_temp,
         modem_tx_output_i_temp_pipe,
             modem_tx_output_q_temp_pipe, to_rx_i, to_rx_q:
             std_logic_vector(9 downto 0);

        signal start_tx_temp, start_rx_temp: std_logic;

begin

  M_CTX: modem_controller port map (modem_modem_tx_start,
   modem_modem_rx_start,
   modem_mode,
   start_tx_temp,
   start_rx_temp);

  M_TX: modem_tx port map (modem_clk1,
   modem_clk2,
   modem_clk3,
   start_tx_temp,
   modem_rst,
   modem_modem_tx_bit_in,
   modem_modem_tx_length,
   modem_tx_output_i_temp,
   modem_tx_output_q_temp,
   modem_modem_tx_debug_bit_to_symbol_symbol,
   modem_modem_tx_debug_bit_to_symbol_valid,
   modem_modem_tx_debug_chip_gen_i_out,
   modem_modem_tx_debug_chip_gen_q_out,
   modem_modem_tx_debug_upsampler_out_i,
   modem_modem_tx_debug_upsampler_out_q,
   modem_modem_tx_debug_controller_pre_sent,
   modem_modem_tx_debug_controller_sfd_sent,
   modem_modem_tx_debug_controller_phr_sent);

  -- Las señales modem_tx_output_i_temp_pipe y
      modem_tx_output_q_temp_pipe controlan
```

```
-- el paso de los datos hacia la parte de recepción del
   modem.

 modem_tx_output_i_temp_pipe <=  modem_tx_output_i_temp
    when modem_mode = MODE_LOOP else
         modem_modem_rx_i_input when modem_mode = MODE_RX
   else (others => '0');

modem_tx_output_q_temp_pipe <=  modem_tx_output_q_temp
   when modem_mode = MODE_LOOP else
  modem_modem_rx_q_input when modem_mode = MODE_RX
  else (others => '0');

modem_modem_tx_i_out <= modem_tx_output_i_temp when
   modem_mode = MODE_TX
  or modem_mode = MODE_LOOP else (others => '0');

modem_modem_tx_q_out <= modem_tx_output_q_temp when
   modem_mode = MODE_TX
  or modem_mode = MODE_LOOP else (others => '0');


M_TX_TO_M_RX: modem_tx_to_rx port map (
   modem_tx_output_i_temp_pipe,
  modem_tx_output_q_temp_pipe,
  to_rx_i,
  to_rx_q);


M_RX: modem_rx port map (modem_clk1,
 modem_clk2,
 modem_clk3,
 start_rx_temp,
 modem_rst,
 to_rx_i,
 to_rx_q,
 modem_modem_rx_bit_output,
 modem_modem_rx_controller_sfd_detected,

 modem_modem_rx_debug_symbol,
 modem_modem_rx_debug_symbol_valid,
 modem_modem_rx_debug_coarse_freq_estimator_estim_done,
 modem_modem_rx_debug_fine_freq_estimator_estim_done,
 modem_modem_rx_debug_downsampler_output_i,
 modem_modem_rx_debug_downsampler_output_q,
 modem_modem_rx_debug_coarse_freq_estimator_ij,
 modem_modem_rx_debug_coarse_freq_estimator_qj,
 modem_modem_rx_debug_fine_freq_estimator_ij,
 modem_modem_rx_debug_fine_freq_estimator_qj,
```

```
    modem_modem_rx_debug_rz_encoder_i,
    modem_modem_rx_debug_rz_encoder_q,
    modem_modem_rx_debug_mfilter_rx_i,
    modem_modem_rx_debug_mfilter_rx_q);

end Behavioral;
```

8.2. modem_controller.vhd

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     15:49:27 12/15/2008
-- Design Name:
-- Module Name:     modem_controller - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_controller is
 port(  modem_controller_start_tx_in : in std_logic;
   modem_controller_start_rx_in : in std_logic;
   modem_controller_mode: in std_logic_vector(1 downto 0);
   modem_controller_start_tx_out : out std_logic;
   modem_controller_start_rx_out : out std_logic);
end modem_controller;

architecture Behavioral of modem_controller is

 constant MODE_IDLE : std_logic_vector(1 downto 0) := "00";
 constant MODE_LOOP  : std_logic_vector(1 downto 0) := "01"
     ;
```

```
 constant MODE_TX     : std_logic_vector(1 downto 0) := "10"
    ;
 constant MODE_RX     : std_logic_vector(1 downto 0) := "11"
    ;

begin

 modem_controller_start_rx_out <= '1' when
    modem_controller_start_rx_in = '1'
  and modem_controller_start_tx_in = '0'
  and modem_controller_mode = MODE_RX else
  '1' after 18.625 us when modem_controller_mode =
     MODE_LOOP else
  '0';

 modem_controller_start_tx_out <= '1' when
    modem_controller_start_tx_in = '1'
  and modem_controller_mode = MODE_LOOP else
  '1' when modem_controller_start_tx_in = '1' and
     modem_controller_start_rx_in = '0'
  and modem_controller_mode = MODE_TX else
  '0';

end Behavioral;
```

8.3. modem_tx.vhd

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     12:52:42 08/28/2008
-- Design Name:
-- Module Name:     modem_tx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_tx is

 Port(  modem_tx_clk1 : in std_logic; -- 250 khz
  modem_tx_clk2 : in std_logic; -- 1 mhz
  modem_tx_clk3 : in std_logic; -- 8 mhz
  modem_tx_start : in std_logic;
  modem_tx_rst : in std_logic;
  modem_tx_bit_in: in std_logic;
  modem_tx_length: in std_logic_vector(7 downto 0);

  -- salidas de los filtros (sistema)

  modem_tx_i_out : out std_logic_vector(9 downto 0);
  modem_tx_q_out : out std_logic_vector(9 downto 0);

  -- salidas de debug

  debug_bit_to_symbol_symbol: out std_logic_vector(3 downto
      0);
  debug_bit_to_symbol_valid: out std_logic;
  debug_chip_gen_i_out: out std_logic;
  debug_chip_gen_q_out: out std_logic;
  debug_upsampler_out_i: out std_logic_vector(1 downto 0);
  debug_upsampler_out_q: out std_logic_vector(1 downto 0);
  debug_controller_pre_sent: out std_logic;
  debug_controller_sfd_sent: out std_logic;
  debug_controller_phr_sent: out std_logic);

end modem_tx;

architecture Behavioral of modem_tx is

 component modem_tx_controller is
  port (controller_start_tx: in std_logic;
   controller_rst:   in std_logic;
   controller_clk:   in std_logic;
   controller_bit_in: in std_logic;
   controller_length: in std_logic_vector(7 downto 0);

   controller_bit_out: out std_logic;
   controller_bts_en: out std_logic;
   controller_chip_gen_en: out std_logic;
   controller_upsampler_en: out std_logic;

   controller_debug_pre_sent: out std_logic;
```

```
    controller_debug_sfd_sent: out std_logic;
    controller_debug_phr_sent: out std_logic);

 end component;

 component modem_tx_bit_to_symbol is
  port(bit_to_symbol_start: in std_logic;
    bit_to_symbol_bit_in: in std_logic;
    bit_to_symbol_clk: in std_logic;
    bit_to_symbol_symbol_valid: out std_logic;
    bit_to_symbol_symbol: out std_logic_vector(3 downto 0));
 end component;

 component modem_tx_chip_gen is
  Port( chip_gen_rst : in std_logic;
    chip_gen_clk : in std_logic;
    chip_gen_start : in std_logic;
    chip_gen_symbol_valid: in std_logic;
    chip_gen_symbol: in std_logic_vector(3 downto 0);
    chip_gen_iOut: out std_logic;
    chip_gen_qOut: out std_logic);
 end component;

 component modem_tx_upsampler is
  Port( upsampler_output: out std_logic_vector(1 downto 0);
    upsampler_input: in std_logic;
    upsampler_clk: in std_logic;
    upsampler_start: in std_logic);
  end component;

        component modem_tx_mfilter is

                Port( mfilter_input: in std_logic_vector(1
                    downto 0);
                                mfilter_clk: in std_logic;
                                mfilter_rst: in std_logic;
                                mfilter_output: out
                                    std_logic_vector(9
                                    downto 0));
 end component;

signal bit_temp, bts_en_temp, chip_gen_en_temp,
    upsampler_en_temp, chip_gen_iOut_temp,
    chip_gen_qOut_temp : std_logic;
signal upsampler_out_i_temp, upsampler_out_q_temp :
    std_logic_vector(1 DOWNTO 0);
signal symbol_v_temp: std_logic;
signal symbol_bts_temp: std_logic_vector(3 downto 0);
```

```
begin
 CTX: modem_tx_controller port map (modem_tx_start,
  modem_tx_rst,
  modem_tx_clk1,
  modem_tx_bit_in,
  modem_tx_length,
  bit_temp,
  bts_en_temp,
  chip_gen_en_temp,
  upsampler_en_temp,
  debug_controller_pre_sent,
  debug_controller_sfd_sent,
  debug_controller_phr_sent);

 Bts:  modem_tx_bit_to_symbol port map (bts_en_temp,
  bit_temp,
  modem_tx_clk1,
  symbol_v_temp,
  symbol_bts_temp);

        debug_bit_to_symbol_symbol <= symbol_bts_temp;
        debug_bit_to_symbol_valid <= symbol_v_temp;


 CGen: modem_tx_chip_gen port map (modem_tx_rst,
         modem_tx_clk2,
    chip_gen_en_temp,
    symbol_v_temp,
    symbol_bts_temp,
    chip_gen_iOut_temp,
    chip_gen_qOut_temp);

 UPS_i: modem_tx_upsampler port map (upsampler_out_i_temp,
  chip_gen_iOut_temp,
  modem_tx_clk3,
  upsampler_en_temp);

 UPS_q: modem_tx_upsampler port map (upsampler_out_q_temp,
  chip_gen_qOut_temp,
  modem_tx_clk3,
  upsampler_en_temp);

 MF_i: modem_tx_mfilter port map (upsampler_out_i_temp,
  modem_tx_clk3,
  modem_tx_rst,
  modem_tx_i_out);

 MF_q: modem_tx_mfilter port map (upsampler_out_q_temp,
  modem_tx_clk3,
```

```
  modem_tx_rst,
  modem_tx_q_out);

 debug_chip_gen_i_out <= chip_gen_iOut_temp;
 debug_chip_gen_q_out  <= chip_gen_qOut_temp;
 debug_upsampler_out_i <= upsampler_out_i_temp;
 debug_upsampler_out_q <= upsampler_out_q_temp;

end Behavioral;
```

8.4. modem_tx_controller

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:    10:03:19 10/28/2008
-- Design Name:
-- Module Name:    modem_tx_controller - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_tx_controller is

 port (controller_start_tx: in std_logic;
  controller_rst:   in std_logic;
  controller_clk:   in std_logic;
  controller_bit_in: in std_logic;
  controller_length: in std_logic_vector(7 downto 0);

  controller_bit_out: out std_logic;
  controller_bts_en: out std_logic;
  controller_chip_gen_en: out std_logic;
```

```
  controller_upsampler_en: out std_logic;

  controller_debug_pre_sent: out std_logic;
  controller_debug_sfd_sent: out std_logic;
  controller_debug_phr_sent: out std_logic);
end modem_tx_controller;

architecture Behavioral of modem_tx_controller is
  signal count_temp: integer range 0 to 1024;
begin

  controller_counter: process(controller_clk,
     controller_rst, controller_start_tx)
   variable count: integer range 0 to 1024 := 0;
  begin
   if controller_rst = '1' then
    count := 0;
   elsif rising_edge(controller_clk) and
      controller_start_tx = '1' then
    if (count < 1024 ) then
     count := count + 1;
    end if;
   end if;

   count_temp <= count;
 end process;


 controller_bts_en <= '1' when controller_start_tx = '1'
    else '0';

 controller_bit_out <=  '0' when (count_temp >= 1 and
    count_temp < 33) else -- PRE 8x"000"

 '1' when count_temp = 33 else -- SFD
 '1' when count_temp = 34 else
 '1' when count_temp = 35 else
 '0' when count_temp = 36 else

 '0' when count_temp = 37 else
 '1' when count_temp = 38 else
 '0' when count_temp = 39 else
 '1' when count_temp = 40 else

 controller_length(0) when count_temp = 41 else -- PHR
 controller_length(1) when count_temp = 42 else
 controller_length(2) when count_temp = 43 else
 controller_length(3) when count_temp = 44 else
 controller_length(4) when count_temp = 45 else
```

```
 controller_length(5) when count_temp = 46 else
 controller_length(6) when count_temp = 47 else
 controller_length(7) when count_temp = 48 else
 controller_bit_in when (count_temp > 48 and
    controller_start_tx = '1') else
 '0';

 controller_debug_pre_sent <= '1' when count_temp = 32 else
     '0';
 controller_debug_sfd_sent <= '1' when count_temp = 40 else
     '0';
 controller_debug_phr_sent <= '1' when count_temp = 48 else
     '0';

-- Tiempos para LOOP
-- Requeridos para que la parte demoduladora tome los datos
     de la parte moduladora
-- correctamente.

 controller_chip_gen_en <= '1' after 16 us when
    controller_start_tx = '1' else '0';
 controller_upsampler_en <= '1' after 16 us when
    controller_start_tx = '1' else '0';

-- Tiempos para RX

 -- controller_chip_gen_en <= '1' when count_temp = 5 and
     controller_start_tx = '1' ; -- 5: Retraso de 1 Tb
     entre bts y chip_gen.
 -- controller_upsampler_en <= '1' when count_temp = 5 and
     controller_start_tx = '1' ; -- 5: Retraso de 1 Tb entre
      bts y upsampler.

end Behavioral;
```

8.5. modem_tx_bit_to_symbol

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     20:39:56 10/26/2008
-- Design Name:
-- Module Name:     modem_tx_bit_to_symbol - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
```

```
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_tx_bit_to_symbol is
 port(bit_to_symbol_start: in std_logic;
  bit_to_symbol_bit_in: in std_logic;
  bit_to_symbol_clk: in std_logic;
  bit_to_symbol_symbol_valid: out std_logic;
  bit_to_symbol_symbol: out std_logic_vector(3 downto 0));
end modem_tx_bit_to_symbol;

architecture Behavioral of modem_tx_bit_to_symbol is
 signal symbol_reg_temp: std_logic_vector(3 downto 0);
 signal count_temp: integer range 0 to 4 ;
begin
 bts_shr_symbol: process(bit_to_symbol_clk,
    bit_to_symbol_start)

  variable symbol_reg: std_logic_vector(3 downto 0):= (
     others=>'0');
  variable count: integer range 0 to 4 := 0;

  begin
   if rising_edge(bit_to_symbol_clk) and
      bit_to_symbol_start = '1' then

    symbol_reg := symbol_reg(2 downto 0) &
       bit_to_symbol_bit_in;

    if count = 4 then
     count := 0;
    end if;

    count := count + 1;

   end if;

  symbol_reg_temp <= symbol_reg;
```

```
  count_temp <= count;

 end process;

 bit_to_symbol_symbol_valid <= '1' when count_temp = 4 and
    bit_to_symbol_start = '1' else '0';
 bit_to_symbol_symbol <= symbol_reg_temp when count_temp =
    4 and bit_to_symbol_start = '1' else "0000" ;

end Behavioral;
```

8.6. modem_tx_chip_gen

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     13:03:38 08/28/2008
-- Design Name:
-- Module Name:     modem_tx_chip_gen - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.ALL;

entity modem_tx_chip_gen is

 Port( chip_gen_rst : IN STD_LOGIC;
  chip_gen_clk : IN STD_LOGIC;
  chip_gen_start : IN STD_LOGIC;
  chip_gen_symbol_valid: in std_logic;
  chip_gen_symbol: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
  chip_gen_iOut: OUT STD_LOGIC;
  chip_gen_qOut: OUT STD_LOGIC);
```

```
end modem_tx_chip_gen;

architecture Behavioral of modem_tx_chip_gen is
  TYPE symbol_array is ARRAY (7 DOWNTO 0) OF
      STD_LOGIC_VECTOR (3 DOWNTO 0);

  CONSTANT symbols: symbol_array := ( "0111",
   "0110",
   "0101",
   "0100",
   "0011",
   "0010",
   "0001",
   "0000");

  CONSTANT symbol_zero_i : STD_LOGIC_VECTOR (15 DOWNTO 0)
     := "1010100100010111";
        CONSTANT symbol_zero_q : STD_LOGIC_VECTOR (15
           DOWNTO 0) := "1101100111000010";

  SIGNAL muxIout, muxQout: STD_LOGIC;
  SIGNAL sr1, sr2 : STD_LOGIC_VECTOR(15 DOWNTO 0);

BEGIN

   shr: PROCESS(chip_gen_clk, chip_gen_start, chip_gen_rst)

  BEGIN

   IF (chip_gen_rst = '1') THEN
    sr1 <= symbol_zero_i;
    sr2 <= symbol_zero_q;

   ELSIF (rising_edge(chip_gen_clk) AND chip_gen_start =
       '1') THEN

    sr1 <= std_logic_vector(unsigned(sr1) rol 1);
    sr2 <= std_logic_vector(unsigned(sr2) rol 1);

   END IF;

  END PROCESS;

   muxI: PROCESS(chip_gen_symbol, sr1, chip_gen_symbol_valid
       )
    variable symbol_reg_i : std_logic_vector(3 downto 0) :=
       "0000";
  BEGIN
```

```
    if chip_gen_symbol_valid = '1' then
     symbol_reg_i := chip_gen_symbol;
    end if;

    IF (symbol_reg_i(2 DOWNTO 0) = symbols(0)(2 DOWNTO 0))
       THEN
     muxIout <= sr1(0);
    ELSIF (symbol_reg_i(2 DOWNTO 0) = symbols(1)(2 DOWNTO 0)
       ) THEN
     muxIout <= sr1(2);
    ELSIF (symbol_reg_i(2 DOWNTO 0) = symbols(2)(2 DOWNTO 0)
       ) THEN
     muxIout <= sr1(4);
    ELSIF (symbol_reg_i(2 DOWNTO 0) = symbols(3)(2 DOWNTO 0)
       ) THEN
     muxIout <= sr1(6);
    ELSIF (symbol_reg_i(2 DOWNTO 0) = symbols(4)(2 DOWNTO 0)
       ) THEN
     muxIout <= sr1(8);
    ELSIF (symbol_reg_i(2 DOWNTO 0) = symbols(5)(2 DOWNTO 0)
       ) THEN
     muxIout <= sr1(10);
    ELSIF (symbol_reg_i(2 DOWNTO 0) = symbols(6)(2 DOWNTO 0)
       ) THEN
     muxIout <= sr1(12);
    ELSIF (symbol_reg_i(2 DOWNTO 0) = symbols(7)(2 DOWNTO 0)
       ) THEN
     muxIout <= sr1(14);
   END IF;
END PROCESS;

muxQ: PROCESS(chip_gen_symbol, sr2, chip_gen_symbol_valid)
  variable symbol_reg_q : std_logic_vector(3 downto 0) := "
     0000";
BEGIN

  if chip_gen_symbol_valid = '1' then
   symbol_reg_q := chip_gen_symbol;
  end if;

  IF (symbol_reg_q(2 DOWNTO 0) = symbols(0)(2 DOWNTO 0))
     THEN
   muxQout <= sr2(0) xor symbol_reg_q(3);
  ELSIF (symbol_reg_q(2 DOWNTO 0) = symbols(1)(2 DOWNTO 0))
      THEN
   muxQout <= sr2(2) xor symbol_reg_q(3);
  ELSIF (symbol_reg_q(2 DOWNTO 0) = symbols(2)(2 DOWNTO 0))
      THEN
   muxQout <= sr2(4) xor symbol_reg_q(3);
```

```
  ELSIF (symbol_reg_q(2 DOWNTO 0) = symbols(3)(2 DOWNTO 0))
      THEN
    muxQout <= sr2(6) xor symbol_reg_q(3);
  ELSIF (symbol_reg_q(2 DOWNTO 0) = symbols(4)(2 DOWNTO 0))
      THEN
    muxQout <= sr2(8) xor symbol_reg_q(3);
  ELSIF (symbol_reg_q(2 DOWNTO 0) = symbols(5)(2 DOWNTO 0))
      THEN
    muxQout <= sr2(10) xor symbol_reg_q(3);
  ELSIF (symbol_reg_q(2 DOWNTO 0) = symbols(6)(2 DOWNTO 0))
      THEN
    muxQout <= sr2(12) xor symbol_reg_q(3);
  ELSIF (symbol_reg_q(2 DOWNTO 0) = symbols(7)(2 DOWNTO 0))
      THEN
    muxQout <= sr2(14) xor symbol_reg_q(3);
  END IF;
 END PROCESS;

 chip_gen_iOut <= muxIout WHEN chip_gen_start = '1' ELSE
     '0';
 chip_gen_qOut <= muxQout  WHEN chip_gen_start = '1' ELSE
     '0';

end Behavioral;
```

8.7. modem_tx_upsampler

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     17:31:46 08/28/2008
-- Design Name:
-- Module Name:     modem_tx_upsampler - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_tx_upsampler is

 generic (factor : integer := 7);
 Port( upsampler_output: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
  upsampler_input: IN  STD_LOGIC;
  upsampler_clk: IN STD_LOGIC;
  upsampler_start: IN STD_LOGIC);

end modem_tx_upsampler;

architecture Behavioral of modem_tx_upsampler is
 signal count_temp: integer range 0 to factor + 1;
begin

 counter: process(upsampler_clk, upsampler_start)
  variable count: integer range 0 to factor + 1;
  begin
   if (upsampler_start = '1' and rising_edge(upsampler_clk)
      ) then
    if (count = factor) then
     count := 0;
    else
     count := count + 1;
    end if;
   end if;

   count_temp <= count;

 end process;

 upsampler_output <= (not upsampler_input) & '1' when (
    upsampler_start = '1' and count_temp = 1)
  else "00";
end Behavioral;
```

8.8. modem_tx_mfilter

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     10:48:30 08/29/2008
-- Design Name:
```

```
-- Module Name:     modem_tx_mfilter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--


 --
 -- Coeficientes del filtro representados sobre 10 bits en
    C-2,
 -- 5 para la parte entera y 5 para la parte fraccionaria.
 --
 --
 --      h0 = 0          -> 0000000000
 -- h1 = 0.3826    -> 0000001100
 -- h2 = 0.7071    -> 0000010111
 -- h3 = 0.9238    -> 0000011110
 -- h4 = 1         -> 0000100000
 -- h5 = 0.9238    -> 0000011110
 -- h6 = 0.7071    -> 0000010111
 -- h7 = 0.3826    -> 0000001100
 --


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity modem_tx_mfilter is

 port (mfilter_input: in std_logic_vector(1 downto 0);
  mfilter_clk:   in std_logic;
  mfilter_rst:   in std_logic;
  mfilter_output: out std_logic_vector(9 downto 0));

end modem_tx_mfilter;

architecture Behavioral of modem_tx_mfilter is
 type registers is array (6 downto 0) of signed(1 downto 0)
     ;
```

```
type coefficients is array (7 downto 0) of signed(9 downto
    0);

signal reg: registers;
signal output_temp: std_logic_vector(9 downto 0);

constant coef: coefficients := ("0000001100",
 "0000010111",
 "0000011110",
 "0000100000",
 "0000011110",
 "0000010111",
 "0000001100",
 "0000000000");

 begin
  process (mfilter_clk, mfilter_rst)
   variable acc: signed(9 downto 0) := (others => '0');
  begin

   if (mfilter_rst = '1') then
    for i in 6 downto 0 loop
     reg(i) <= (others => '0');
    end loop;

   elsif rising_edge(mfilter_clk) then

     if (mfilter_input = "01") then
      acc := resize(coef(0), acc'length);
     elsif (mfilter_input = "11") then
      acc := resize(-coef(0), acc'length);
     else
      acc := (others => '0');
     end if;

     for i in 1 to 7 loop
      if (reg(7-i) = "01") then
       acc := resize(acc + coef(i), acc'length);
      elsif (reg(7-i)= "11") then
       acc := resize(acc - coef(i), acc'length);
      end if;
     end loop;

    reg <= signed(mfilter_input) & reg(6 DOWNTO 1);

  end if;
              output_temp <= std_logic_vector(acc);

 end process;
```

```
  mfilter_output <= output_temp;

end Behavioral;
```

8.9. modem_tx_to_rx

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     09:46:03 10/08/2008
-- Design Name:
-- Module Name:     modem_tx_to_rx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_tx_to_rx is
 port (modem_tx_i: in std_logic_vector(9 downto 0);
   modem_tx_q: in std_logic_vector(9 downto 0);
   modem_rx_i: out std_logic_vector(9 downto 0);
   modem_rx_q: out std_logic_vector(9 downto 0));
end modem_tx_to_rx;

architecture Behavioral of modem_tx_to_rx is

begin

   modem_rx_i <= modem_tx_i;
   modem_rx_q <= modem_tx_q;

end Behavioral;
```

8.10. modem_rx

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     12:20:48 09/24/2008
-- Design Name:
-- Module Name:     modem_rx - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.ALL;

entity modem_rx is
 port (modem_rx_clk1: in std_logic;
  modem_rx_clk2: in std_logic;
  modem_rx_clk3: in std_logic;
  modem_rx_start: in std_logic;
  modem_rx_rst: in std_logic;
  modem_rx_input_i: in std_logic_vector(9 downto 0);
  modem_rx_input_q: in std_logic_vector(9 downto 0);

  modem_rx_bit_output: out std_logic;
  modem_rx_controller_sfd_detected: out std_logic;

  debug_correlator_output: out std_logic_vector(3 downto 0)
      ;
  debug_correlator_output_valid: out std_logic;
  debug_coarse_freq_estimator_estim_done: out std_logic;
  debug_fine_freq_estimator_estim_done: out std_logic;
  debug_downsampler_output_i: out std_logic_vector(9 downto
       0);
  debug_downsampler_output_q: out std_logic_vector(9 downto
       0);
  debug_coarse_freq_estimator_ij: out std_logic_vector(9
     downto 0);
```

```
    debug_coarse_freq_estimator_qj: out std_logic_vector(9
        downto 0);
    debug_fine_freq_estimator_ij: out std_logic_vector(9
        downto 0);
    debug_fine_freq_estimator_qj: out std_logic_vector(9
        downto 0);
    debug_rz_encoder_i: out std_logic;
    debug_rz_encoder_q: out std_logic;
    debug_mfilter_rx_i: out std_logic_vector(9 downto 0);
    debug_mfilter_rx_q: out std_logic_vector(9 downto 0));

end modem_rx;

architecture Behavioral of modem_rx is

 component modem_rx_symbol_to_bit is
  port (symbol_to_bit_start: in std_logic;
   symbol_to_bit_clk: in std_logic;
   symbol_to_bit_symbol_valid: in std_logic;
   symbol_to_bit_symbol: in std_logic_vector(3 downto 0);
   symbol_to_bit_bit_output: out std_logic);
  end component;

 component modem_rx_controller is
  port (controller_start_rx: in std_logic;
   controller_symbol_valid: in std_logic;
    controller_symbol_correlator: in std_logic_vector(3
        downto 0);
    controller_coarse_freq_estim_done: in std_logic
    controller_coarse_freq_estim_en: out std_logic;
    controller_fine_freq_estim_en: out std_logic;
    controller_downsampler_en: out std_logic;
    controller_correlator_en: out std_logic;
    controller_stb_en: out std_logic;
    controller_sfd_detected: out std_logic);
 end component;

 component modem_rx_downsampler is
  port (downsampler_clk: in std_logic;
   downsampler_start: in std_logic;
   downsampler_input: in std_logic_vector(9 downto 0);
   downsampler_output: out std_logic_vector(9 downto 0));
 end component;

 component modem_rx_coarse_freq_estimator is
  port( coarse_freq_estimator_clk: in std_logic;
   coarse_freq_estimator_rst: in std_logic;
   coarse_freq_estimator_ena: in std_logic;
```

```
    coarse_freq_estimator_i: in std_logic_vector(9 downto 0)
        ;
    coarse_freq_estimator_q: in std_logic_vector(9 downto 0)
        ;
    coarse_freq_estimator_estim_done: out std_logic;
    coarse_freq_estimator_ij: out std_logic_vector(9 downto
        0);
    coarse_freq_estimator_qj: out std_logic_vector(9 downto
        0));
end component;

component modem_rx_fine_freq_estimator is
 port( fine_freq_estimator_clk: in std_logic;
    fine_freq_estimator_rst: in std_logic;
    fine_freq_estimator_ena: in std_logic;
    fine_freq_estimator_i: in std_logic_vector(9 downto 0);
    fine_freq_estimator_q: in std_logic_vector(9 downto 0);
    fine_freq_estimator_estim_done: out std_logic;
    fine_freq_estimator_ij: out std_logic_vector(9 downto 0)
        ;
    fine_freq_estimator_qj: out std_logic_vector(9 downto 0)
        );
end component;

component modem_rx_mfilter is
 port (mfilter_input: in std_logic_vector(9 downto 0);
    mfilter_clk:    in std_logic;
    mfilter_rst:    in std_logic;
    mfilter_output: out std_logic_vector(9 downto 0));
end component;

component modem_rx_rz_encoder is
 port (rz_encoder_input_i: in std_logic_vector(9 downto 0)
     ;
    rz_encoder_input_j: in std_logic_vector(9 downto 0);
    rz_output_i: out std_logic;
    rz_output_j: out std_logic);
end component;

component modem_rx_correlator is
 port( correlator_start :in std_logic;
    correlator_clk: in std_logic;
    correlator_rst: in std_logic;
    correlator_input_q: in std_logic;
    correlator_symbol_valid: out std_logic;
    correlator_symbol: out std_logic_vector(3 downto 0));
end component;
```

```
signal fine_output_i_temp, fine_output_q_temp,
    downsampler_output_i_temp, downsampler_output_q_temp:
    std_logic_vector(9 downto 0);
signal mfilter_rx_i_temp, mfilter_rx_q_temp,
    coarse_freq_estimator_ij_temp,
    coarse_freq_estimator_qj_temp: std_logic_vector(9
    downto 0);
signal symbol_valid_temp, coarse_done_temp, coarse_temp_en
    , fine_temp_en, downsampler_temp_en, correlator_temp_en
    , rz_encoder_i_temp, rz_encoder_q_temp: std_logic;
signal symbol_temp: std_logic_vector(3 downto 0);
signal stb_en_temp, stb_output_temp: std_logic;

begin


CRX: modem_rx_controller port map (modem_rx_start,
 symbol_valid_temp,
 symbol_temp,
 coarse_done_temp,
 coarse_temp_en,
 fine_temp_en,
 downsampler_temp_en,
 correlator_temp_en,
 stb_en_temp,
 modem_rx_controller_sfd_detected);


FR_COAR: modem_rx_coarse_freq_estimator port map (
 modem_rx_clk3,
 modem_rx_rst,
 coarse_temp_en,
 modem_rx_input_i,
 modem_rx_input_q,
 coarse_done_temp,
 coarse_freq_estimator_ij_temp,
 coarse_freq_estimator_qj_temp);

debug_coarse_freq_estimator_estim_done <= coarse_done_temp
    ;

debug_coarse_freq_estimator_ij <=
    coarse_freq_estimator_ij_temp;
debug_coarse_freq_estimator_qj <=
    coarse_freq_estimator_qj_temp;

MF_RX_i: modem_rx_mfilter port map (
    coarse_freq_estimator_ij_temp,
 modem_rx_clk3,
```

```
 modem_rx_rst,
 mfilter_rx_i_temp);

MF_RX_q: modem_rx_mfilter port map (
    coarse_freq_estimator_qj_temp,
 modem_rx_clk3,
 modem_rx_rst,
 mfilter_rx_q_temp);

debug_mfilter_rx_i <= mfilter_rx_i_temp;
debug_mfilter_rx_q <= mfilter_rx_q_temp;

DOW_i: modem_rx_downsampler port map (modem_rx_clk2,
 downsampler_temp_en,
 mfilter_rx_i_temp,
 downsampler_output_i_temp);

DOW_q: modem_rx_downsampler port map (modem_rx_clk2,
 downsampler_temp_en,
 mfilter_rx_q_temp,
 downsampler_output_q_temp);

debug_downsampler_output_i <= downsampler_output_i_temp;
debug_downsampler_output_q     <=
    downsampler_output_q_temp;

FR_FINE: modem_rx_fine_freq_estimator port map (
    modem_rx_clk2,
 modem_rx_rst,
 fine_temp_en,
 downsampler_output_i_temp,
 downsampler_output_q_temp,
 debug_fine_freq_estimator_estim_done,
 fine_output_i_temp,
 fine_output_q_temp);

debug_fine_freq_estimator_ij <= fine_output_i_temp;
debug_fine_freq_estimator_qj <= fine_output_q_temp;

RZ_EN: modem_rx_rz_encoder port map (fine_output_i_temp,
 fine_output_q_temp,
 rz_encoder_i_temp,
 rz_encoder_q_temp);

debug_rz_encoder_q <= rz_encoder_q_temp;
debug_rz_encoder_i <= rz_encoder_i_temp;

CORR: modem_rx_correlator port map (correlator_temp_en,
 modem_rx_clk2,
```

```
   modem_rx_rst,
   rz_encoder_q_temp,
   symbol_valid_temp,
   symbol_temp);

 debug_correlator_output_valid <= symbol_valid_temp;
 debug_correlator_output <= symbol_temp;

 STB: modem_rx_symbol_to_bit port map (stb_en_temp,
   modem_rx_clk1,
   symbol_valid_temp,
   symbol_temp,
   stb_output_temp);

 modem_rx_bit_output <= stb_output_temp;

end Behavioral;
```

8.11. modem_rx_controller

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     11:52:24 10/29/2008
-- Design Name:
-- Module Name:     modem_rx_controller - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_rx_controller is

 port (controller_start_rx: in std_logic;
```

```
    controller_symbol_valid: in std_logic;
    controller_symbol_correlator: in std_logic_vector(3
        downto 0);
    controller_coarse_freq_estim_done: in std_logic;
    controller_coarse_freq_estim_en: out std_logic;
    controller_fine_freq_estim_en: out std_logic;
    controller_downsampler_en: out std_logic;
    controller_correlator_en: out std_logic;
    controller_stb_en: out std_logic;
    controller_sfd_detected: out std_logic);

end modem_rx_controller;

architecture Behavioral of modem_rx_controller is

begin

    -- Tiempos para el modo LOOP

    -- El estimador de frecuencia gruesa (kay) y el
        downsampler comienzan
    -- nada mas la recepción está en high.

    -- Cuando el estimador de kay termina, se activa el
        correlador, y cuando éste
    -- detecta un símbolo cero, comienza a funcionar el
        estimador "fino".

  sfd_monitor: process(controller_symbol_correlator)
   variable part1_detected: std_logic;
  begin
   if controller_symbol_correlator = "1110" then
    part1_detected := '1';
   end if;

   if part1_detected = '1' and controller_symbol_correlator
      = "0101" then
    controller_sfd_detected <= '1';
    part1_detected := '0';
   else
    controller_sfd_detected <= '0';
   end if;

  end process;

  controller_coarse_freq_estim_en <= '1' when
      controller_start_rx = '1' else '0';
  controller_downsampler_en       <= '1' when
      controller_start_rx = '1' else '0';
```

```
 controller_stb_en
    <= '1'  after 0.875 us when
     controller_coarse_freq_estim_done = '1' else '0';
 controller_correlator_en        <= '1'  after 0.875 us
    when controller_coarse_freq_estim_done = '1' else '0';
 controller_fine_freq_estim_en   <= '1' when
     controller_symbol_valid = '1' and
     controller_symbol_correlator =    "0000";

end Behavioral;
```

8.12. modem_rx_coarse_freq_estimator

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     12:01:39 09/22/2008
-- Design Name:
-- Module Name:     modem_rx_coarse_freq_estimator -
    Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.ALL;

use work.modem_cordic.ALL;

entity modem_rx_coarse_freq_estimator is

 generic(chips_estim : integer := 256;  -- chips durante
     los que hacer la estimacion.
  n: integer := 8);                                    --
      log2(chip_estim).
```

```
 Port( coarse_freq_estimator_clk: in std_logic;
  coarse_freq_estimator_rst: in std_logic;
  coarse_freq_estimator_ena: in std_logic;
  coarse_freq_estimator_i: in std_logic_vector(9 downto 0);
  coarse_freq_estimator_q: in std_logic_vector(9 downto 0);
  coarse_freq_estimator_estim_done: out std_logic;
  coarse_freq_estimator_ij: out std_logic_vector(9 downto
     0);
  coarse_freq_estimator_qj: out std_logic_vector(9 downto
     0));

end modem_rx_coarse_freq_estimator;

architecture Behavioral of modem_rx_coarse_freq_estimator
    is
 signal output_i, output_q: std_logic_vector(9 downto 0);
 signal count_temp: integer range 0 to chips_estim + 1;
begin

 kay_chip_counter: process(coarse_freq_estimator_ena,
     coarse_freq_estimator_clk, coarse_freq_estimator_rst)
  variable count: integer range 0 to chips_estim  + 1 := 0;
 begin
  if coarse_freq_estimator_rst = '1' then
   count := 0;
  elsif rising_edge(coarse_freq_estimator_clk) and
      coarse_freq_estimator_ena = '1' then
   if (count < chips_estim ) then
    count := count + 1;
   end if;
  end if;

  count_temp <= count;
 end process;

 kay_sum_proc: process(coarse_freq_estimator_clk,
     coarse_freq_estimator_ena, coarse_freq_estimator_rst)
  variable phase_acc, phase_temp, kay_sum, last_phase:
      signed(9 downto 0) := (others => '0');
  variable cordic_temp_vec, cordic_temp_rot: cordic_output2
      := ((others=>'0'),(others=>'0'));
 begin
  if coarse_freq_estimator_rst = '1'then
   phase_acc := (others => '0');
   phase_temp := (others => '0');
   kay_sum := (others => '0');
   last_phase := (others => '0');
   cordic_temp_vec := ((others=>'0'),(others=>'0'));
   cordic_temp_rot := ((others=>'0'),(others=>'0'));
```

```
  elsif coarse_freq_estimator_ena = '1' then
   if rising_edge(coarse_freq_estimator_clk) then
    if (count_temp < chips_estim) then
     cordic_temp_vec := cordic_vec2(coarse_freq_estimator_i
        ,
      coarse_freq_estimator_q);
     phase_temp := signed(cordic_temp_vec(0));
     kay_sum := kay_sum + (phase_temp - last_phase);
     last_phase := phase_temp;
    else
     phase_acc := signed(right_shift(n, kay_sum));
     cordic_temp_rot := cordic_rot2(coarse_freq_estimator_i
        ,
      coarse_freq_estimator_q,
      std_logic_vector(phase_acc));
   end if;
  end if;
 end if;

 output_i <= cordic_temp_rot(1);
 output_q <= cordic_temp_rot(0);

 end process;

 coarse_freq_estimator_ij <= output_i when (count_temp =
    chips_estim) and (coarse_freq_estimator_ena = '1')
  else (others => '0');

 coarse_freq_estimator_qj <= output_q when (count_temp =
    chips_estim) and (coarse_freq_estimator_ena = '1')
  else (others => '0');

 coarse_freq_estimator_estim_done <= '1' when (count_temp =
     chips_estim) and (coarse_freq_estimator_ena = '1')
  else '0';

end Behavioral;
```

8.13. modem_rx_mfilter

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     10:48:30 08/29/2008
-- Design Name:
-- Module Name:     modem_rx_mfilter - Behavioral
```

```
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity modem_rx_mfilter is

 port (mfilter_input: in std_logic_vector(9 downto 0);
  mfilter_clk:   in std_logic;
  mfilter_rst:   in std_logic;
  mfilter_output: out std_logic_vector(9 downto 0));
end modem_rx_mfilter;

architecture Behavioral of modem_rx_mfilter is
 type registers is array (6 downto 0) of signed(9 downto 0)
    ;
 type coefficients is array (7 downto 0) of signed(9 downto
     0);

 signal reg: registers;
 signal output_temp: std_logic_vector(9 downto 0);

 constant coef: coefficients := ("0000001100",
  "0000010111",
  "0000011110",
  "0000100000",
  "0000011110",
  "0000010111",
  "0000001100",
  "0000000000");

 begin
  process (mfilter_clk, mfilter_rst)
   variable acc, prod: signed(19 downto 0) := (others =>
       '0');
    begin
```

```
      if (mfilter_rst = '1') then
       for i in 6 downto 0 loop
        reg(i) <= (others => '0');
       end loop;

      elsif rising_edge(mfilter_clk) then
       acc := coef(0)*signed(mfilter_input);

       for i in 1 to 7 loop
        prod := coef(i)*reg(7-i);
        acc := acc + prod;
       end loop;

       reg <= signed(mfilter_input) & reg(6 DOWNTO 1);
                  end if;

      output_temp <= std_logic_vector(resize(acc,
         mfilter_output'length));

 end process;

 mfilter_output <= output_temp;

end Behavioral;
```

8.14. modem_rx_downsampler

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     12:02:50 09/24/2008
-- Design Name:
-- Module Name:     modem_rx_downsampler - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_rx_downsampler is
 port (downsampler_clk: in std_logic;
  downsampler_start: in std_logic;
  downsampler_input: in std_logic_vector(9 downto 0);
  downsampler_output: out std_logic_vector(9 downto 0));
end modem_rx_downsampler;

architecture Behavioral of modem_rx_downsampler is
begin
 process(downsampler_clk, downsampler_start,
    downsampler_input)
  begin
   if falling_edge(downsampler_clk) and downsampler_start =
       '1' then
    downsampler_output <= downsampler_input;
   end if;
 end process;

end Behavioral;
```

8.15. modem_rx_fine_freq_estimator

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     13:15:28 09/28/2008
-- Design Name:
-- Module Name:     modem_rx_fine_freq_estimator -
    Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.ALL;

use work.modem_cordic.ALL;

entity modem_rx_fine_freq_estimator is

 generic(chips_estim : integer := 80;   -- chips durante
    los que hacer la estimacion.
  n: integer := 6);                                    --
     log2(chip_estim).

 Port( fine_freq_estimator_clk: in std_logic;
  fine_freq_estimator_rst: in std_logic;
  fine_freq_estimator_ena: in std_logic;
  fine_freq_estimator_i: in std_logic_vector(9 downto 0);
  fine_freq_estimator_q: in std_logic_vector(9 downto 0);
  fine_freq_estimator_estim_done: out std_logic;
  fine_freq_estimator_ij: out std_logic_vector(9 downto 0);
  fine_freq_estimator_qj: out std_logic_vector(9 downto 0))
     ;

end modem_rx_fine_freq_estimator;

architecture Behavioral of modem_rx_fine_freq_estimator is

 type phase_table is array (15 downto 0) of
    std_logic_vector(9 downto 0);

 constant phase_table_sym_0 : phase_table :=    ("
    1111100111",
  "0000011001",
  "1111100111",
  "1110110101",
  "1111100111",
  "1110110101",
  "0001001011",
  "0001001011",
  "0000011001",
  "1110110101",
  "1110110101",
  "0000011001",
  "0001001011",
  "1111100111",
  "0001001011",
  "0000011001");

 signal output_i, output_q, actual_phase: std_logic_vector
    (9 downto 0);
```

```
 signal count_temp: integer range 0 to chips_estim  := 0;

begin

 fine_phase_shr: process(fine_freq_estimator_clk,
    fine_freq_estimator_rst, fine_freq_estimator_ena)
  variable phase_table_0: phase_table;
 begin

  if (fine_freq_estimator_rst = '1') then
   phase_table_0 := phase_table_sym_0;
  elsif rising_edge(fine_freq_estimator_clk) and (
     fine_freq_estimator_ena = '1') then
   phase_table_0 := phase_table_0(0) & phase_table_0(15
      downto 1);
 end if;

  actual_phase <= phase_table_0(0);

 end process;

 fine_chip_counter: process(fine_freq_estimator_ena,
    fine_freq_estimator_clk, fine_freq_estimator_rst)
  variable count: integer range 0 to chips_estim  := 0;
 begin
  if fine_freq_estimator_rst = '1' then
   count := 0;
  elsif rising_edge(fine_freq_estimator_clk) and
     fine_freq_estimator_ena = '1' then
   if (count < chips_estim ) then
    count := count + 1;
   end if;
  end if;

  count_temp <= count;
 end process;

 fine_sum: process(fine_freq_estimator_ena,
    fine_freq_estimator_clk, fine_freq_estimator_i,
    fine_freq_estimator_q, fine_freq_estimator_rst)
  variable phase_acc, phase_temp1, phase_temp2, fine_sum :
     signed(9 downto 0) := (others => '0');
  variable cordic_temp_vec, cordic_temp_rot: cordic_output2
      := ((others=>'0'),(others=>'0'));
 begin
  if fine_freq_estimator_rst = '1' then
   phase_acc := (others => '0');
   phase_temp1 := (others => '0');
   phase_temp2 := (others => '0');
```

```
    fine_sum := (others => '0');
    cordic_temp_vec := ((others=>'0'),(others=>'0'));
    cordic_temp_rot := ((others=>'0'),(others=>'0'));
     elsif rising_edge(fine_freq_estimator_clk) then
      if fine_freq_estimator_ena = '1' then
       if (count_temp < chips_estim ) then
        phase_temp1 := signed(actual_phase);
        cordic_temp_vec := cordic_vec2(fine_freq_estimator_i
           ,
         fine_freq_estimator_q);
        phase_temp2 := signed(cordic_temp_vec(0));
        fine_sum := fine_sum + (phase_temp1 - phase_temp2);

        output_i <= fine_freq_estimator_i;
        output_q <= fine_freq_estimator_q;

                          else

        phase_acc := signed(right_shift(n, fine_sum));
        cordic_temp_rot := cordic_rot2(fine_freq_estimator_i
           ,
         fine_freq_estimator_q,
         std_logic_vector(phase_acc));
        output_i <= cordic_temp_rot(1);
        output_q <= cordic_temp_rot(0);

       end if;
      else
       output_i <= fine_freq_estimator_i;
       output_q <= fine_freq_estimator_q;
      end if;
  end if;

 end process;

 fine_freq_estimator_ij <= output_i;
 fine_freq_estimator_qj <= output_q;
 fine_freq_estimator_estim_done <= '1' when
     fine_freq_estimator_ena = '1' and count_temp =
     chips_estim
 else '0';

end Behavioral;
```

8.16. modem_rx_rz_encoder

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     12:38:31 09/25/2008
-- Design Name:
-- Module Name:     modem_rx_rz_encoder - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_rx_rz_encoder is
 Port( rz_encoder_input_i: in std_logic_vector(9 downto 0);
  rz_encoder_input_j: in std_logic_vector(9 downto 0);
  rz_output_i: out std_logic;
  rz_output_j: out std_logic);
end modem_rx_rz_encoder;

architecture Behavioral of modem_rx_rz_encoder is

begin
 rz_output_i <= '1' when signed(rz_encoder_input_i) > 0
    else '0';
 rz_output_j <= '1' when signed(rz_encoder_input_j) > 0
    else '0';
end Behavioral;
```

8.17. modem_rx_correlator

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     12:51:36 09/26/2008
-- Design Name:
-- Module Name:     modem_rx_correlator - Behavioral
```

```
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.ALL;


entity modem_rx_correlator is
 generic (t_symbol: integer := 16);
 port( correlator_start :in std_logic;
  correlator_clk: in std_logic;
  correlator_rst: in std_logic;
  correlator_input_q: in std_logic;
  correlator_symbol_valid: out std_logic;
  correlator_symbol: out std_logic_vector(3 downto 0));

end modem_rx_correlator;

architecture Behavioral of modem_rx_correlator is
 signal count_temp: integer range 0 to 17;
 signal chip_shr: std_logic_vector(15 downto 0);

 type sym_t is array (15 downto 0) of std_logic_vector(15
    downto 0);
 type symbol_table is array (15 downto 0) of
     std_logic_vector(3 downto 0);
 type corr_value_t is array (15 downto 0) of integer;
 type max_fun_output is array (1 downto 0) of integer range
      0 to 16;

 constant symbol_zero_q : std_logic_vector(15 downto 0) :=
    "1101100111000010";

 constant symbol_table_q: symbol_table := ("1111",
  "1110",
  "1101",
  "1100",
  "1011",
  "1010",
```

```
  "1001",
  "1000",
  "0111",
  "0110",
  "0101",
  "0100",
  "0011",
  "0010",
  "0001",
  "0000");


 function bit_to_integer(input_value: std_logic) return
    integer is
  begin
   if (input_value = '0' ) then
    return 0;
   else
    return 1;
   end if;
 end function bit_to_integer;

 function max_vector(vector: corr_value_t ) return
    max_fun_output is
  variable temp: integer range 0 to 8;
  variable pos: integer range 0 to 16;
  begin
   temp := 0;

    for i in vector'range loop
     if vector(i) > temp then
      temp := vector(i);
      pos  := i;
     end if;
    end loop;

   return max_fun_output'(temp, pos);
 end function max_vector;

begin

 corr_shr: process(correlator_start, correlator_clk,
    correlator_rst)
  variable chip_reg: std_logic_vector(15 downto 0);
 begin
  if (correlator_rst = '1') then
   chip_reg := symbol_zero_q;
  elsif (rising_edge(correlator_clk) and correlator_start =
      '1') then
```

```
   chip_reg := std_logic_vector(unsigned(chip_shr) rol 1);
  end if;

  chip_shr <= chip_reg;
 end process;

 corr_counter: process(correlator_start, correlator_clk)
  variable count: integer range 0 to t_symbol + 1:= 0;
 begin
  if (rising_edge(correlator_clk) and correlator_start =
      '1') then
   count := count + 1;
   if (count = t_symbol + 1) then
    count := 1;
   end if;
  end if;

        count_temp <= count;
 end process;

 corr: process(correlator_start, correlator_clk,
    correlator_rst)
  variable reg: std_logic_vector(15 downto 0) := (others =>
      '0');
  variable sym: sym_t;
  variable corr_value: corr_value_t;
  variable sym_pos: max_fun_output;
  variable entero: integer := 0;
  begin
   if (correlator_rst = '1') then
    for i in 0 to 15 loop
     sym(i):= (others =>'0');
     corr_value(i) := 0;
                end loop;

   reg := (others => '0');
   elsif (rising_edge(correlator_clk) and correlator_start
      = '1') then
    reg := reg(14 downto 0) & correlator_input_q;

    sym(0):= sym(0)(14 downto 0) & chip_shr(0);
    sym(1):= sym(1)(14 downto 0) & chip_shr(2);
    sym(2):= sym(2)(14 downto 0) & chip_shr(4);
    sym(3):= sym(3)(14 downto 0) & chip_shr(6);
    sym(4):= sym(4)(14 downto 0) & chip_shr(8);
    sym(5):= sym(5)(14 downto 0) & chip_shr(10);
    sym(6):= sym(6)(14 downto 0) & chip_shr(12);
    sym(7):= sym(7)(14 downto 0) & chip_shr(14);
    sym(8):= sym(8)(14 downto 0) & not chip_shr(0);
```

```
    sym(9):= sym(9)(14 downto 0) & not chip_shr(2);
    sym(10):= sym(10)(14 downto 0) & not chip_shr(4);
    sym(11):= sym(11)(14 downto 0) & not chip_shr(6);
    sym(12):= sym(12)(14 downto 0) & not chip_shr(8);
    sym(13):= sym(13)(14 downto 0) & not chip_shr(10);
    sym(14):= sym(14)(14 downto 0) & not chip_shr(12);
    sym(15):= sym(15)(14 downto 0) & not chip_shr(14);

-- Tenemos un nuevo simbolo.

                if (count_temp = t_symbol) then
      for i in 0 to 15 loop
       for j in 0 to 15 loop
        corr_value(i) := corr_value(i) + bit_to_integer(reg
            (j) and sym(i)(j));
       end loop;
      end loop;

      sym_pos := max_vector(corr_value);

      if sym_pos(1) = 8 then
       correlator_symbol_valid <= '1';
      else
       correlator_symbol_valid <= '0';
      end if;

      correlator_symbol <= symbol_table_q(sym_pos(0));

      for i in 0 to 15 loop
       corr_value(i) := 0;
      end loop;
               else
     correlator_symbol_valid <= '0';
   end if;

  end if;
 end process;

end Behavioral;
```

8.18. modem_rx_symbol_to_bit

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:     11:30:10 10/30/2008
```

```
-- Design Name:
-- Module Name:     modem_rx_symbol_to_bit - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modem_rx_symbol_to_bit is

 port (symbol_to_bit_start: in std_logic;
  symbol_to_bit_clk: in std_logic;
  symbol_to_bit_symbol_valid: in std_logic;
  symbol_to_bit_symbol: in std_logic_vector(3 downto 0);
  symbol_to_bit_bit_output: out std_logic);

end modem_rx_symbol_to_bit;

architecture Behavioral of modem_rx_symbol_to_bit is
 signal bit_temp: std_logic;
begin

 process(symbol_to_bit_start, symbol_to_bit_clk,
    symbol_to_bit_symbol_valid, symbol_to_bit_symbol)
  variable symbol_reg: std_logic_vector(3 downto 0);
 begin
  if symbol_to_bit_start = '1' then
   if symbol_to_bit_symbol_valid = '1' then
    symbol_reg := symbol_to_bit_symbol;
   elsif rising_edge(symbol_to_bit_clk) then
    bit_temp <= symbol_reg(3);
    symbol_reg(3 downto 1) := symbol_reg(2 downto 0);
   end if;
  end if;

 end process;

 symbol_to_bit_bit_output <= bit_temp;
```

```
end Behavioral;
```

8.19. modem_cordic.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.ALL;

package modem_cordic is
 type atan_table2 is array (5 downto 0) of signed(9 downto
    0);
 type cordic_output2 is array (1 downto 0) of
    std_logic_vector(9 downto 0);

 -- Valores de arctan(k) representadas en C-2, 5 bits parte
     entera, 5 bits parte fraccionaria.


 constant atan_t2: atan_table2 := (
  "0000000001", -- 1.78º / 0.0273 rad
  "0000000010", -- 3.57º / 0.0585 rad
  "0000000100", -- 7.12º / 0.1210 rad
  "0000001000", -- 14.03º / 0.2421 rad
  "0000001111", -- 26.56º / 0.4609 rad
  "0000011001"); -- 45º / 0.7851 rad


 constant pi_22: signed(9 downto 0) := "0000110010";

 -- right_shift realiza divisiones por cnt mediante srl
 -- independientemente de si el numero es positivo o
    negativo (c2).

 function right_shift(cnt: integer; input_value: signed)
    return signed;

 -- cordic_rot2 rota el complejo i + qi en i' + q'i por los
     radianes indicados
 -- en cordic_rot_angle.

 function cordic_rot2 ( cordic_rot_i : std_logic_vector(9
    downto 0);
  cordic_rot_q : std_logic_vector(9 downto 0);
```

```
  cordic_rot_angle : std_logic_vector(9 downto 0)
  ) return cordic_output2;

 -- cordic_vec2 convierte a forma polar (modulo, fase) el
    complejo i + qi.

 function cordic_vec2 ( cordic_vec_i : std_logic_vector(9
    downto 0);
  cordic_vec_q : std_logic_vector(9 downto 0)
  ) return cordic_output2;


end modem_cordic;

package body modem_cordic is

 function right_shift(cnt: integer; input_value: signed)
    return signed is
  variable temp: signed(input_value'range);
 begin
  temp := abs(input_value) srl cnt;
   if (input_value < 0) then
    return -temp;
   else
    return temp;
   end if;
 end function right_shift;

 function cordic_rot2 ( cordic_rot_i : std_logic_vector(9
    downto 0);
  cordic_rot_q : std_logic_vector(9 downto 0);
  cordic_rot_angle : std_logic_vector(9 downto 0)
  ) return cordic_output2 is

  variable i_signed, q_signed, temp_i: signed(9 downto 0)
     := (others => '0');
  variable angle_temp: signed(9 downto 0) := (others =>
     '0');

  begin

   i_signed := resize(signed(cordic_rot_i), i_signed'length
      );
   q_signed := resize(signed(cordic_rot_q), q_signed'length
      );
   angle_temp := resize(signed(cordic_rot_angle),
      angle_temp'length);

   if (cordic_rot_angle = "0000000000") then
```

```
    return cordic_output2'(std_logic_vector(i_signed),
        std_logic_vector(q_signed));
   else
    for c in 0 to 5 loop
     temp_i := i_signed;
     if (angle_temp < 0) then
      i_signed := resize(i_signed + right_shift(c, q_signed
          ), i_signed'length);
      q_signed := resize(q_signed - right_shift(c, temp_i),
           q_signed'length);
      angle_temp := resize(angle_temp + atan_t2(c),
          angle_temp'length);
     else
      i_signed := resize(i_signed - right_shift(c, q_signed
          ), i_signed'length);
      q_signed := resize(q_signed + right_shift(c, temp_i),
           q_signed'length);
      angle_temp := resize(angle_temp - atan_t2(c),
          angle_temp'length);
     end if;
    end loop;

   -- Aproximación de k = 0.607 como 0.5

   return cordic_output2'(std_logic_vector(right_shift(1,
       i_signed)),
           std_logic_vector(right_shift(1, q_signed)));
 end if;
end cordic_rot2;

function cordic_vec2 ( cordic_vec_i : std_logic_vector(9
    downto 0);
 cordic_vec_q : std_logic_vector(9 downto 0)
 ) return cordic_output2 is -- (modulo, fase)

 variable i_signed, q_signed, temp_i, acc_phase_rads:
     signed(9 downto 0) := (others => '0');

 begin

  -- Hacemos la fase original inferior a 90º.
  -- Solo para valores de i < 0, para el resto
  -- la fase siempre es menor que 90º.
  -- arctan(q/0 ~ inf) = 90º.

   i_signed := resize(signed(cordic_vec_i), i_signed'
       length);
   q_signed := resize(signed(cordic_vec_q), q_signed'
       length);
```

```
                if (cordic_vec_i = "0000000000" and
                    cordic_vec_q = "0000000000") then
     return cordic_output2'("0000000000", "0000000000");
    else
     if (i_signed < 0) then
      temp_i := i_signed;
      if (q_signed > 0) then
       i_signed := q_signed;
       q_signed := -temp_i;
       acc_phase_rads := resize(-pi_22, acc_phase_rads'
           length);
      else
       i_signed := -q_signed;
       q_signed := temp_i;
       acc_phase_rads := resize(pi_22, acc_phase_rads'
           length);
      end if;
     else
       acc_phase_rads := (others => '0');
     end if;

    for c in 0 to 5 loop
     temp_i := i_signed;
     if (q_signed >= 0) then
      i_signed := resize(i_signed + right_shift(c, q_signed
          ), i_signed'length);
      q_signed := resize(q_signed - right_shift(c, temp_i),
           q_signed'length);
      acc_phase_rads := resize(acc_phase_rads - atan_t2(c),
           acc_phase_rads'length);
     else
      i_signed := resize(i_signed - right_shift(c, q_signed
          ), i_signed'length);
      q_signed := resize(q_signed + right_shift(c, temp_i),
           q_signed'length);
      acc_phase_rads := resize(acc_phase_rads + atan_t2(c),
           acc_phase_rads'length);
     end if;
    end loop;

    -- Aproximación de k = 0.607 como 0.5

    return cordic_output2'(std_logic_vector(right_shift(1,
        i_signed)),
     std_logic_vector(-acc_phase_rads));
  end if;
 end cordic_vec2;
end modem_cordic;
```

Capítulo 9

Resultados de los testbenches

9.1. Test del modo LOOP

Una de las mejores formas de comprobar que la modulación y demodulación se han implementado correctamente, es conectar la unidad de modulación a la unidad de demodulación mediante el modo *loop*, observando que los datos demodulados son similares a los originales antes de la modulación.

9.1.1. modem_test.vhd

El test *modem_test.vhd* activa el modo *loop* del módem, y da las instrucciones para modular la secuencia binaria 10010011.

```
-- Company:
-- Engineer: Antonio de la Piedra Abenójar
--
-- Create Date:    12:21:20 10/30/2008
-- Design Name:
-- Module Name:    C:/PFC/modem_test.vhd
-- Project Name:   PFC
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: modem
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

```
-- Notes:
-- This testbench has been automatically generated using
    types std_logic and
-- std_logic_vector for the ports of the unit under test.
    Xilinx recommends
-- that these types always be used for the top-level I/O of
     a design in order
-- to guarantee that the testbench will bind correctly to
    the post-implementation
-- simulation model.
--
    ------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY modem_test IS
END modem_test;

ARCHITECTURE behavior OF modem_test IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT modem
    PORT(
        modem_clk1 : IN  std_logic;
        modem_clk2 : IN  std_logic;
        modem_clk3 : IN  std_logic;
        modem_rst : IN  std_logic;
        modem_mode : IN  std_logic_vector(1 downto 0);
        modem_modem_tx_start : IN  std_logic;
        modem_modem_tx_bit_in : IN  std_logic;
        modem_modem_tx_length : IN  std_logic_vector(7
            downto 0);
        modem_modem_tx_i_out : OUT  std_logic_vector(9
            downto 0);
        modem_modem_tx_q_out : OUT  std_logic_vector(9
            downto 0);
        modem_modem_rx_i_input : IN  std_logic_vector(9
            downto 0);
        modem_modem_rx_q_input : IN  std_logic_vector(9
            downto 0);
        modem_modem_rx_start : IN  std_logic;
        modem_modem_rx_bit_output : OUT  std_logic;
        modem_modem_rx_controller_sfd_detected : OUT
            std_logic;
        modem_modem_tx_debug_bit_to_symbol_symbol : OUT
```

```
            std_logic_vector(3 downto 0);
        modem_modem_tx_debug_bit_to_symbol_valid : OUT
            std_logic;
        modem_modem_tx_debug_chip_gen_i_out : OUT
            std_logic;
        modem_modem_tx_debug_chip_gen_q_out : OUT
            std_logic;
        modem_modem_tx_debug_upsampler_out_i : OUT
            std_logic_vector(1 downto 0);
        modem_modem_tx_debug_upsampler_out_q : OUT
            std_logic_vector(1 downto 0);
        modem_modem_tx_debug_controller_pre_sent : OUT
            std_logic;
        modem_modem_tx_debug_controller_sfd_sent : OUT
            std_logic;
        modem_modem_tx_debug_controller_phr_sent : OUT
            std_logic;
        modem_modem_rx_debug_symbol : OUT
            std_logic_vector(3 downto 0);
        modem_modem_rx_debug_symbol_valid : OUT  std_logic
            ;
        modem_modem_rx_debug_coarse_freq_estimator_estim_done
             : OUT  std_logic;
        modem_modem_rx_debug_fine_freq_estimator_estim_done
             : OUT  std_logic;
        modem_modem_rx_debug_downsampler_output_i : OUT
            std_logic_vector(9 downto 0);
        modem_modem_rx_debug_downsampler_output_q : OUT
            std_logic_vector(9 downto 0);
        modem_modem_rx_debug_coarse_freq_estimator_ij :
           OUT  std_logic_vector(9 downto 0);
        modem_modem_rx_debug_coarse_freq_estimator_qj :
           OUT  std_logic_vector(9 downto 0);
        modem_modem_rx_debug_fine_freq_estimator_ij : OUT
             std_logic_vector(9 downto 0);
        modem_modem_rx_debug_fine_freq_estimator_qj : OUT
             std_logic_vector(9 downto 0);
        modem_modem_rx_debug_rz_encoder_i : OUT  std_logic
            ;
        modem_modem_rx_debug_rz_encoder_q : OUT  std_logic
            ;
        modem_modem_rx_debug_mfilter_rx_i : OUT
            std_logic_vector(9 downto 0);
        modem_modem_rx_debug_mfilter_rx_q : OUT
            std_logic_vector(9 downto 0)
    );
END COMPONENT;
```

```
--Inputs
signal modem_clk1 : std_logic := '0';
signal modem_clk2 : std_logic := '0';
signal modem_clk3 : std_logic := '0';
signal modem_rst : std_logic := '0';
signal modem_mode : std_logic_vector(1 downto 0) := (
   others => '0');
signal modem_modem_tx_start : std_logic := '0';
signal modem_modem_tx_bit_in : std_logic := '0';
signal modem_modem_tx_length : std_logic_vector(7 downto
    0) := (others => '0');
signal modem_modem_rx_i_input : std_logic_vector(9
   downto 0) := (others => '0');
signal modem_modem_rx_q_input : std_logic_vector(9
   downto 0) := (others => '0');
signal modem_modem_rx_start : std_logic := '0';

    --Outputs
signal modem_modem_tx_i_out : std_logic_vector(9 downto
   0);
signal modem_modem_tx_q_out : std_logic_vector(9 downto
   0);
signal modem_modem_rx_bit_output : std_logic;
signal modem_modem_rx_controller_sfd_detected :
   std_logic;
signal modem_modem_tx_debug_bit_to_symbol_symbol :
   std_logic_vector(3 downto 0);
signal modem_modem_tx_debug_bit_to_symbol_valid :
   std_logic;
signal modem_modem_tx_debug_chip_gen_i_out : std_logic;
signal modem_modem_tx_debug_chip_gen_q_out : std_logic;
signal modem_modem_tx_debug_upsampler_out_i :
   std_logic_vector(1 downto 0);
signal modem_modem_tx_debug_upsampler_out_q :
   std_logic_vector(1 downto 0);
signal modem_modem_tx_debug_controller_pre_sent :
   std_logic;
signal modem_modem_tx_debug_controller_sfd_sent :
   std_logic;
signal modem_modem_tx_debug_controller_phr_sent :
   std_logic;
signal modem_modem_rx_debug_symbol : std_logic_vector(3
   downto 0);
signal modem_modem_rx_debug_symbol_valid : std_logic;
signal
   modem_modem_rx_debug_coarse_freq_estimator_estim_done
    : std_logic;
signal
   modem_modem_rx_debug_fine_freq_estimator_estim_done :
```

```
        std_logic;
    signal modem_modem_rx_debug_downsampler_output_i :
        std_logic_vector(9 downto 0);
    signal modem_modem_rx_debug_downsampler_output_q :
        std_logic_vector(9 downto 0);
    signal modem_modem_rx_debug_coarse_freq_estimator_ij :
        std_logic_vector(9 downto 0);
    signal modem_modem_rx_debug_coarse_freq_estimator_qj :
        std_logic_vector(9 downto 0);
    signal modem_modem_rx_debug_fine_freq_estimator_ij :
        std_logic_vector(9 downto 0);
    signal modem_modem_rx_debug_fine_freq_estimator_qj :
        std_logic_vector(9 downto 0);
    signal modem_modem_rx_debug_rz_encoder_i : std_logic;
    signal modem_modem_rx_debug_rz_encoder_q : std_logic;
    signal modem_modem_rx_debug_mfilter_rx_i :
        std_logic_vector(9 downto 0);
    signal modem_modem_rx_debug_mfilter_rx_q :
        std_logic_vector(9 downto 0);

-- Constantes del reloj de 250 KHz.

   constant PERIOD_clk1 : time := 4 us;
   constant DUTY_CYCLE_clk1 : real := 0.5;
   constant OFFSET_clk1 : time := 0 us;

-- Constantes del reloj de 1 MHz.

   constant PERIOD_clk2 : time := 1 us;
   constant DUTY_CYCLE_clk2 : real := 0.5;
   constant OFFSET_clk2 : time := 1.5 us;

-- Constantes del reloj de 8 MHz.

   constant PERIOD_clk3 : time := 0.125 us;
   constant DUTY_CYCLE_clk3 : real := 0.5;
   constant OFFSET_clk3 : time := 1.9375 us;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
    uut: modem PORT MAP (
           modem_clk1 => modem_clk1,
           modem_clk2 => modem_clk2,
           modem_clk3 => modem_clk3,
           modem_rst => modem_rst,
           modem_mode => modem_mode,
           modem_modem_tx_start => modem_modem_tx_start,
           modem_modem_tx_bit_in => modem_modem_tx_bit_in,
```

```
modem_modem_tx_length => modem_modem_tx_length,
modem_modem_tx_i_out => modem_modem_tx_i_out,
modem_modem_tx_q_out => modem_modem_tx_q_out,
modem_modem_rx_i_input => modem_modem_rx_i_input,
modem_modem_rx_q_input => modem_modem_rx_q_input,
modem_modem_rx_start => modem_modem_rx_start,
modem_modem_rx_bit_output =>
    modem_modem_rx_bit_output,
modem_modem_rx_controller_sfd_detected =>
    modem_modem_rx_controller_sfd_detected,
modem_modem_tx_debug_bit_to_symbol_symbol =>
    modem_modem_tx_debug_bit_to_symbol_symbol,
modem_modem_tx_debug_bit_to_symbol_valid =>
    modem_modem_tx_debug_bit_to_symbol_valid,
modem_modem_tx_debug_chip_gen_i_out =>
    modem_modem_tx_debug_chip_gen_i_out,
modem_modem_tx_debug_chip_gen_q_out =>
    modem_modem_tx_debug_chip_gen_q_out,
modem_modem_tx_debug_upsampler_out_i =>
    modem_modem_tx_debug_upsampler_out_i,
modem_modem_tx_debug_upsampler_out_q =>
    modem_modem_tx_debug_upsampler_out_q,
modem_modem_tx_debug_controller_pre_sent =>
    modem_modem_tx_debug_controller_pre_sent,
modem_modem_tx_debug_controller_sfd_sent =>
    modem_modem_tx_debug_controller_sfd_sent,
modem_modem_tx_debug_controller_phr_sent =>
    modem_modem_tx_debug_controller_phr_sent,
modem_modem_rx_debug_symbol =>
    modem_modem_rx_debug_symbol,
modem_modem_rx_debug_symbol_valid =>
    modem_modem_rx_debug_symbol_valid,
modem_modem_rx_debug_coarse_freq_estimator_estim_done
    =>
    modem_modem_rx_debug_coarse_freq_estimator_estim_done
    ,
modem_modem_rx_debug_fine_freq_estimator_estim_done
    =>
    modem_modem_rx_debug_fine_freq_estimator_estim_done
    ,
modem_modem_rx_debug_downsampler_output_i =>
    modem_modem_rx_debug_downsampler_output_i,
modem_modem_rx_debug_downsampler_output_q =>
    modem_modem_rx_debug_downsampler_output_q,
modem_modem_rx_debug_coarse_freq_estimator_ij =>
    modem_modem_rx_debug_coarse_freq_estimator_ij,
modem_modem_rx_debug_coarse_freq_estimator_qj =>
    modem_modem_rx_debug_coarse_freq_estimator_qj,
modem_modem_rx_debug_fine_freq_estimator_ij =>
```

```
            modem_modem_rx_debug_fine_freq_estimator_ij,
        modem_modem_rx_debug_fine_freq_estimator_qj =>
            modem_modem_rx_debug_fine_freq_estimator_qj,
        modem_modem_rx_debug_rz_encoder_i =>
            modem_modem_rx_debug_rz_encoder_i,
        modem_modem_rx_debug_rz_encoder_q =>
            modem_modem_rx_debug_rz_encoder_q,
        modem_modem_rx_debug_mfilter_rx_i =>
            modem_modem_rx_debug_mfilter_rx_i,
        modem_modem_rx_debug_mfilter_rx_q =>
            modem_modem_rx_debug_mfilter_rx_q
    );

-- Proceso del reloj de 250 KHz.

    clk1: PROCESS
        BEGIN
            WAIT for OFFSET_clk1;
            CLOCK_LOOP : LOOP
                modem_clk1 <= '0';
                WAIT FOR (PERIOD_clk1 - (PERIOD_clk1 *
                    DUTY_CYCLE_clk1));
                modem_clk1 <= '1';
                WAIT FOR (PERIOD_clk1 * DUTY_CYCLE_clk1);
            END LOOP CLOCK_LOOP;
        END PROCESS;

-- Proceso del reloj de 1 MHz.

        clk2: PROCESS
        BEGIN
            WAIT for OFFSET_clk2;
            CLOCK_LOOP : LOOP
                                        modem_clk2 <= '0';
                WAIT FOR (PERIOD_clk2 - (PERIOD_clk2 *
                    DUTY_CYCLE_clk2));
                                        modem_clk2 <=
                                            '1';
                WAIT FOR (PERIOD_clk2 * DUTY_CYCLE_clk2);
            END LOOP CLOCK_LOOP;
        END PROCESS;

-- Proceso del reloj de 8 MHz.

        clk3: PROCESS
        BEGIN
            WAIT for OFFSET_clk3;
            CLOCK_LOOP : LOOP
                                        modem_clk3 <= '0';
```

```
                WAIT FOR (PERIOD_clk3 - (PERIOD_clk3 *
                    DUTY_CYCLE_clk3));
                                        modem_clk3 <=
                                            '1';
                WAIT FOR (PERIOD_clk3 * DUTY_CYCLE_clk3);
            END LOOP CLOCK_LOOP;
        END PROCESS;


  stim_proc: process
   begin
          modem_mode <= "01";
    modem_rst <= '1';
                wait for 2 us;
                modem_rst <= '0';
                modem_modem_tx_length <= "11111110";
                modem_modem_tx_start <= '1';
                wait for 194 us;
                modem_modem_tx_bit_in <= '1';
                wait for PERIOD_clk1;
                modem_modem_tx_bit_in <= '0';
                wait for PERIOD_clk1;
                modem_modem_tx_bit_in <= '0';
                wait for PERIOD_clk1;
                modem_modem_tx_bit_in <= '1';
                wait for PERIOD_clk1;

                modem_modem_tx_bit_in <= '0';
                wait for PERIOD_clk1;
                modem_modem_tx_bit_in <= '0';
                wait for PERIOD_clk1;
                modem_modem_tx_bit_in <= '1';
                wait for PERIOD_clk1;
                modem_modem_tx_bit_in <= '1';
                wait for 18 us;
                modem_modem_tx_start <= '0';

                wait;
   end process;


END;
```

9.1.2. Resultados de la unidad de modulación

En la primera imágen (9.1) aparece la modulación tal cual, con todas las salidas de la unidad de modulación.

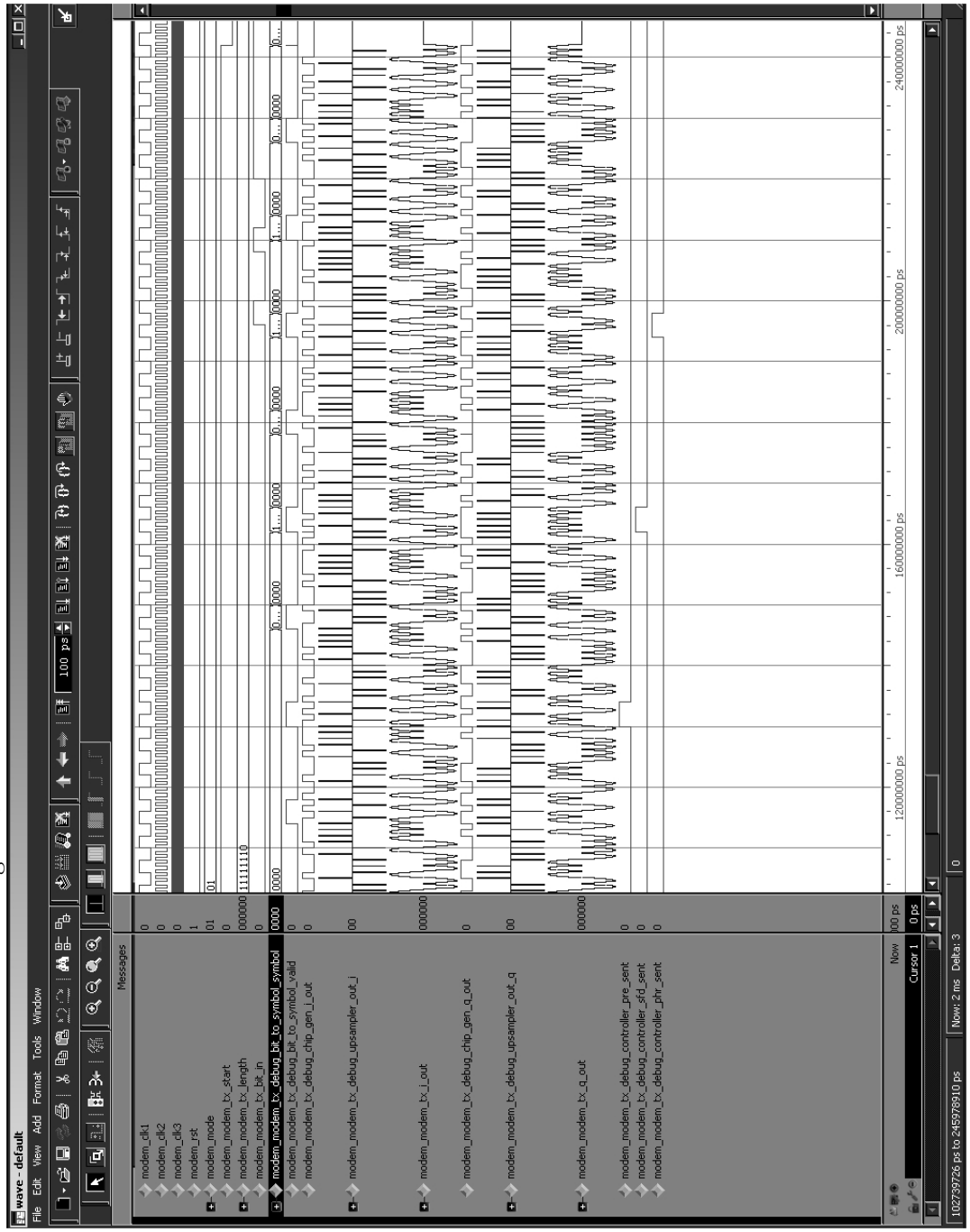

Figura 9.1: Resultado de la unidad de modulación

En la segunda imágen (9.2) aparecen las tres etapas por las que pasan los datos de los canales I y Q: generación de *chips*, *upsampling* y filtrado.

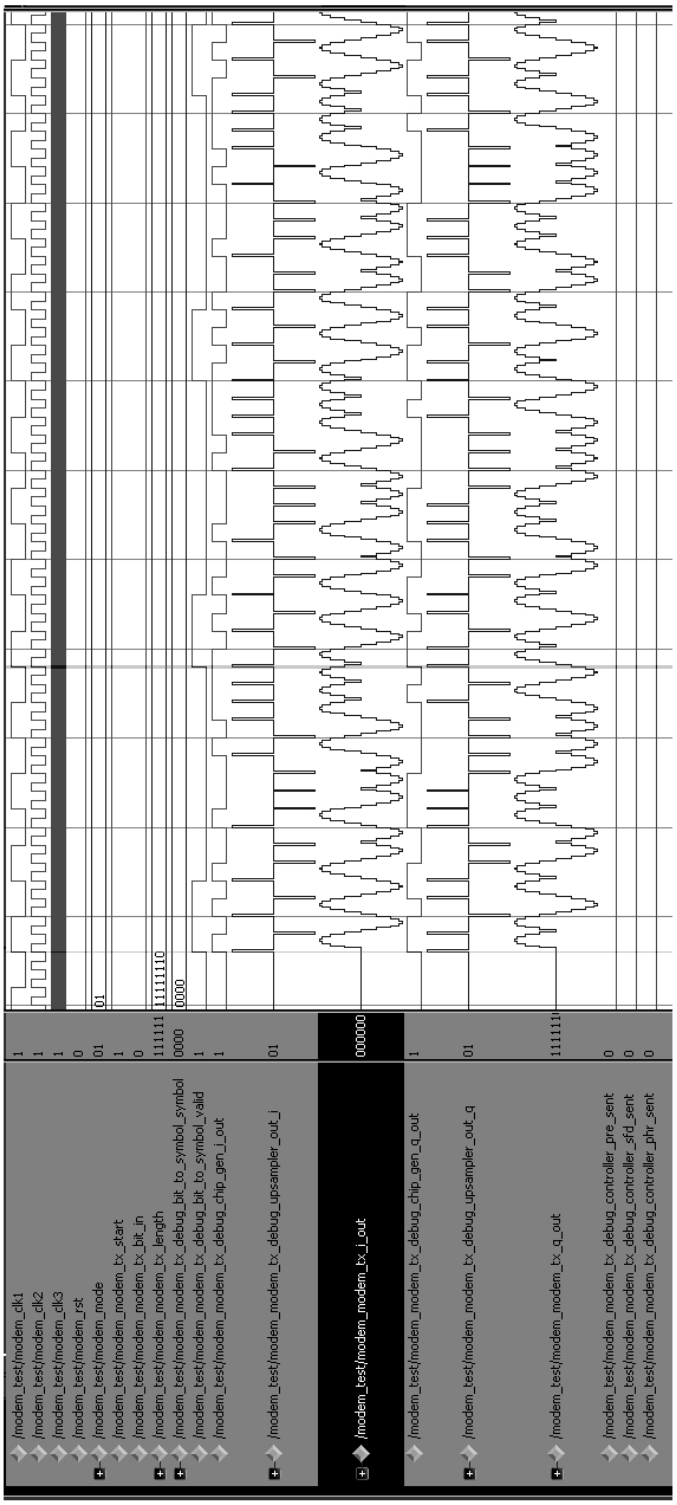

Figura 9.2: Resultado del generador de *chips*, *upsampler* y filtro

9.1.3. Resultados de la unidad de demodulación

En la imágen puede verse como son detectados los símbolos por el correlador: se aprecian los símbolos que componen el SFD (11100101) seguidos de los valores del campo *length*. También puede apreciarse el efecto que tiene el filtro de recepción sobre la señal entrante, aumentando el nivel de la señal. Esto proporciona una ventaja para el codificador RZ, que tiene que decidir si las muestras entrantes son valores 0 o 1. Cuando el SFD es detectado, el controlador de demodulación pone activa la señal de debug *sfd_detected*.

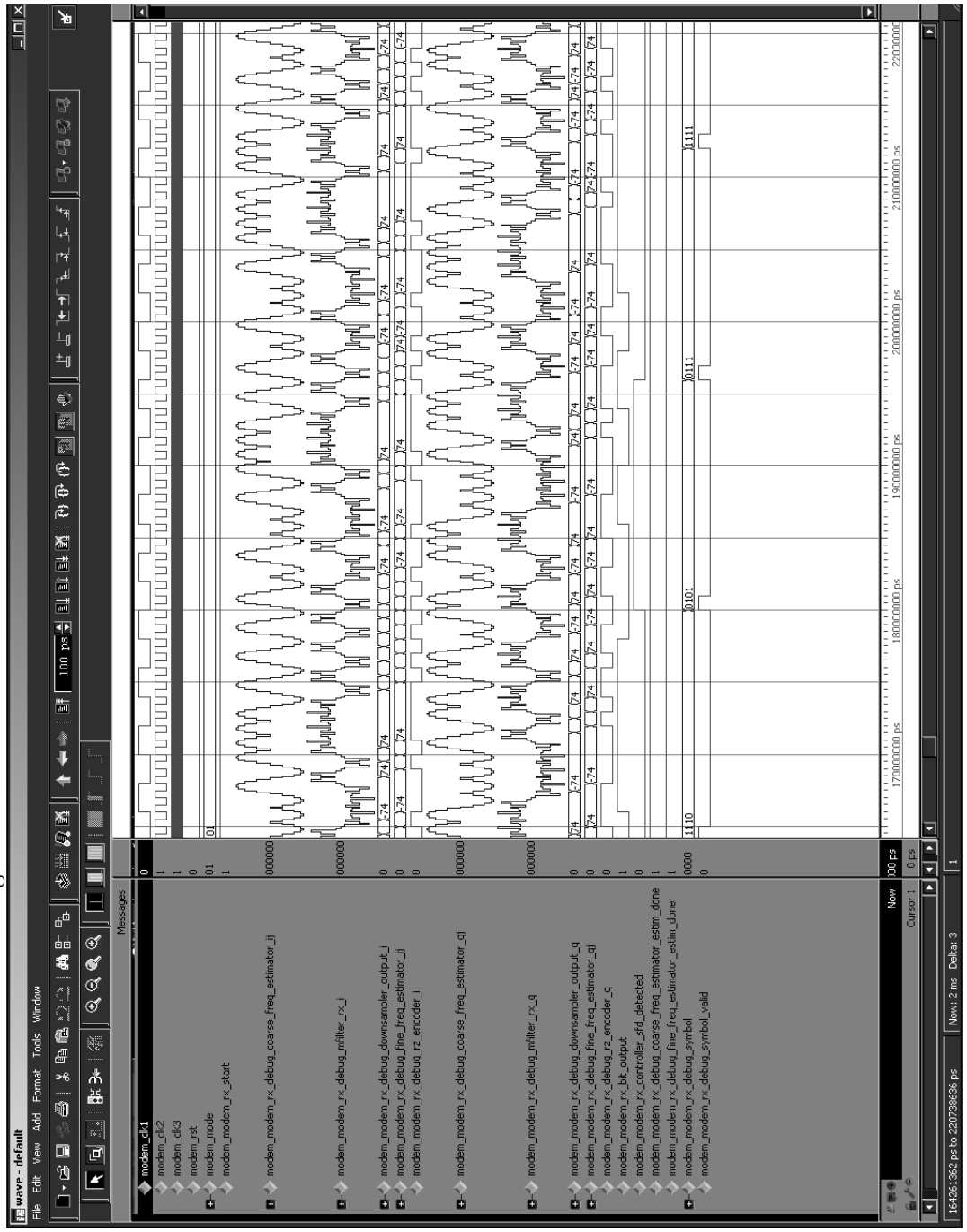

Figura 9.3: Resultado de la unidad de demodulación

9.2. Test del módulo CORDIC

Como se explicaba en la Sección 3.3.3, el módulo del CORDIC ha sido testado de forma que pueda él mismo medir su propia precisión, generando 100 tuplas de muestras complejas a rotar, y comprobando tras la rotación el error rotacion respecto a los valores sin rotar. Así, estamos probando a la vez la implementación del modo rotacional del CORDIC y la del modo vectorial.

9.2.1. test_modem_cordic.vhd

```
-- Company:
-- Engineer: Antonio de la Piedra Abenojar
--
-- Create Date:    14:20:45 10/12/2008
-- Design Name:
-- Module Name:    C:/PFC/test_modem_cordic.vhd
-- Project Name:   PFC
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module:
    modem_tx_mfilter
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using
    types std_logic and
-- std_logic_vector for the ports of the unit under test.
    Xilinx recommends
-- that these types always be used for the top-level I/O of
     a design in order
-- to guarantee that the testbench will bind correctly to
    the post-implementation
-- simulation model.

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
use ieee.math_real.all;
```

```
use work.modem_cordic.all;

ENTITY test_modem_cordic IS
END test_modem_cordic;

ARCHITECTURE behavior OF test_modem_cordic IS
        signal clk1: std_logic;

 constant PERIOD_clk1 : time := 1 us;
 constant DUTY_CYCLE_clk1 : real := 0.5;
 constant OFFSET_clk1 : time := 0 us;

        function bit_to_real(input_value: std_logic) return
            real is
        begin
                if (input_value = '0' ) then
                        return 0.0;
                else
                        return 1.0;
                end if;
        end function bit_to_real;

        function std_logic_vector_to_real55_abs (vec:
           std_logic_vector(9 downto 0))return real is
                variable temp: real := 0.0;
        begin
                temp :=    0.0313*bit_to_real(vec(0))
                             + 0.0625*bit_to_real(vec(1))
                                                 + 0.1250*
                                                    bit_to_real
                                                    (vec(2))
                                                 + 0.2500*
                                                    bit_to_real
                                                    (vec(3))
                                                 + 0.5*
                                                    bit_to_real
                                                    (vec(4))
                                                 + 1.0*
                                                    bit_to_real
                                                    (vec(5))
                                                 + 2.0*
                                                    bit_to_real
                                                    (vec(6))
                                                 + 4.0*
                                                    bit_to_real
                                                    (vec(7))
                                                 + 8.0*
                                                    bit_to_real
                                                    (vec(8))
```

```
                                                    + 16.0*
                                                      bit_to_real
                                                      (vec(9))
                                                      ;
            return temp;
        end function;

BEGIN

        clock: PROCESS
        BEGIN
            WAIT for OFFSET_clk1;
            CLOCK_LOOP : LOOP
                clk1 <= '0';
                WAIT FOR (PERIOD_clk1 - (PERIOD_clk1 *
                    DUTY_CYCLE_clk1));
                clk1 <= '1';
                WAIT FOR (PERIOD_clk1 * DUTY_CYCLE_clk1);
            END LOOP CLOCK_LOOP;
        END PROCESS;

        -- Para comprobar el funcionamiento de la
            implementacion vectorial y de rotacion de
        -- CORDIC, generamos de forma aleatoria la tupla (i
            , q, angulo de rotacion) y rotamos por
        -- dicho angulo el complejo i + qi. Luego
            calculamos el error absoluto entre el angulo
        -- de rotacion y la diferencia de fase entre el
            complejo antes de ser rotado y rotado.

        cordic_test: process
                variable resultado_rotacion,
                    resultado_mod_fase1, resultado_mod_fase2
                    : cordic_output2;
                variable err_fase, input_i, input_q,
                    input_phase, resta_fase:
                    std_logic_vector(9 downto 0);
                variable seed1 : integer := 13830;
                variable seed2 : integer := 12282;
                variable rndreal : real;
                variable int_rand, err_fase_i: integer;
    begin

        for i in 0 to 99 loop
          uniform(seed1, seed2, rndreal);
          int_rand := integer(trunc(rndreal*128.0));
          input_i := std_logic_vector(to_unsigned(int_rand,
              input_i'LENGTH));
```

```
            uniform(seed1, seed2, rndreal);
            int_rand := integer(trunc(rndreal*128.0));
            input_q := std_logic_vector(to_unsigned(int_rand,
                input_q'LENGTH));

        uniform(seed1, seed2, rndreal);
            int_rand := integer(trunc(rndreal*48.0));
            input_phase := std_logic_vector(to_unsigned(
                int_rand, input_phase'LENGTH));

            resultado_mod_fase1 := cordic_vec2(input_i,
                input_q);
        resultado_rotacion  := cordic_rot2(input_i, input_q,
            input_phase);
            resultado_mod_fase2 := cordic_vec2(
                resultado_rotacion(1), resultado_rotacion(0));

            resta_fase := std_logic_vector(signed(
                resultado_mod_fase2(0)) - signed(
                resultado_mod_fase1(0)));
            err_fase := std_logic_vector(abs(signed(
                resta_fase) - signed(input_phase)));

report "Complejo(" & integer'image(to_integer(signed(
    input_i))) & ","
 & integer'image(to_integer(signed(input_q))) & ") "
 & "rotado a ("
 & integer'image(to_integer(signed(resultado_rotacion(1))))
     & ","
 & integer'image(to_integer(signed(resultado_rotacion(0))))
     & ") "
 & "ERROR ABSOLUTO DE ROTACION = " & real'image(
     std_logic_vector_to_real55_abs(err_fase));

            wait for 1 us;
        end loop;

        wait;
  end process;


END;
```

9.2.2. Resultado del test

```
# Reading C:/Modeltech_pe_edu_6.4a/tcl/vsim/pref.tcl
```

```
# do {test_modem_cordic.fdo}
# ** Warning: (vlib-34) Library already exists at "work".
# Model Technology ModelSim PE Student Edition vcom 6.4a
    Compiler 2008.08 Aug 28 2008
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Loading package std_logic_arith
# -- Loading package std_logic_unsigned
# -- Loading package numeric_std
# -- Compiling package modem_cordic
# -- Compiling package body modem_cordic
# -- Loading package modem_cordic
# Model Technology ModelSim PE Student Edition vcom 6.4a
    Compiler 2008.08 Aug 28 2008
# -- Loading package standard
# -- Loading package std_logic_1164
# -- Loading package std_logic_arith
# -- Loading package std_logic_unsigned
# -- Loading package numeric_std
# -- Loading package math_real
# -- Loading package modem_cordic
# -- Compiling entity test_modem_cordic
# -- Compiling architecture behavior of test_modem_cordic
# //  ModelSim PE Student Edition 6.4a Aug 28 2008
# //
# //  Copyright 1991-2008 Mentor Graphics Corporation
# //                All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
#
# // NOT FOR CORPORATE OR PRODUCTION USE.
# // THE ModelSim PE Student Edition IS NOT A SUPPORTED
    PRODUCT.
# // FOR HIGHER EDUCATION PURPOSES ONLY
# //
# vsim -lib work -t 1ps test_modem_cordic
# Loading std.standard
# Loading ieee.std_logic_1164(body)
# Loading ieee.std_logic_arith(body)
# Loading ieee.std_logic_unsigned(body)
# Loading ieee.numeric_std(body)
# Loading ieee.math_real(body)
# Loading work.modem_cordic(body)
# Loading work.test_modem_cordic(behavior)
```

```
# ** Warning: (vsim-WLF-5000) WLF file currently in use:
    vsim.wlf
#           File in use by: vmr  Hostname: BBOPPOOM-D441FF
     ProcessID: 1084
#           Attempting to use alternate WLF file "./
    wlftnk4zcx".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim
    .wlf
#           Using alternate file: ./wlftnk4zcx
# .main_pane.mdi.interior.cs.vm.paneset.cli_0.wf.clip.cs.pw
    .wf
# .main_pane.mdi.interior.cs.vm.paneset.cli_1.wf.clip.cs.
    editor
# .main_pane.mdi.interior.cs.vm.paneset.cli_2.wf.clip.cs
# .main_pane.workspace.interior.cs.nb.canvas.notebook.cs.
    page2.cs
# .main_pane.signals.interior.cs
# .main_pane.variables.interior.cs
# .main_pane.activeproc.interior.cs
# ** Note: Complejo(3,26) rotado a (-20,6) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 0 ps  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(5,75) rotado a (-55,28) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 1 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(121,101) rotado a (-21,127) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 2 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(35,36) rotado a (-8,41) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 3 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(93,64) rotado a (33,87) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 4 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(126,87) rotado a (38,120) ERROR
    ABSOLUTO DE ROTACION = 6.250000e-002
#    Time: 5 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(30,48) rotado a (30,48) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 6 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(127,0) rotado a (36,97) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 7 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(76,99) rotado a (-38,95) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 8 us  Iteration: 0  Instance: /test_modem_cordic
# ** Note: Complejo(25,96) rotado a (25,96) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 9 us  Iteration: 0  Instance: /test_modem_cordic
```

```
# ** Note: Complejo(55,57) rotado a (11,64) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 10 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(37,42) rotado a (3,45) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 11 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(107,127) rotado a (-67,118) ERROR
    ABSOLUTO DE ROTACION = 6.250000e-002
#    Time: 12 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(110,14) rotado a (84,37) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 13 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(26,59) rotado a (-23,48) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 14 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(99,89) rotado a (20,107) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 15 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(11,21) rotado a (4,18) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 16 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(42,89) rotado a (-3,80) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 17 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(26,45) rotado a (12,40) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 18 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(15,96) rotado a (-38,70) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 19 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(22,70) rotado a (-24,56) ERROR ABSOLUTO
     DE ROTACION = 0.000000e+000
#    Time: 20 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(118,42) rotado a (1,102) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 21 us  Iteration: 0  Instance: /
    test_modem_cordic
```

```
# ** Note: Complejo(40,3) rotado a (7,32) ERROR ABSOLUTO DE
     ROTACION = 9.380000e-002
#    Time: 22 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(54,68) rotado a (-29,65) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 23 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(98,19) rotado a (41,71) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 24 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(32,69) rotado a (-35,51) ERROR ABSOLUTO
     DE ROTACION = 0.000000e+000
#    Time: 25 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(11,39) rotado a (-4,33) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 26 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(66,112) rotado a (32,102) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 27 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(24,37) rotado a (5,35) ERROR ABSOLUTO
    DE ROTACION = 9.380000e-002
#    Time: 28 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(45,91) rotado a (-45,69) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 29 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(59,21) rotado a (30,41) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 30 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(70,21) rotado a (-7,59) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 31 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(18,96) rotado a (-32,73) ERROR ABSOLUTO
     DE ROTACION = 0.000000e+000
#    Time: 32 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(10,51) rotado a (-26,33) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 33 us  Iteration: 0  Instance: /
    test_modem_cordic
```

```
# ** Note: Complejo(1,3) rotado a (0,2) ERROR ABSOLUTO DE
    ROTACION = 0.000000e+000
#    Time: 34 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(115,20) rotado a (29,91) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 35 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(63,85) rotado a (-38,78) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 36 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(95,62) rotado a (64,68) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 37 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(46,93) rotado a (-41,74) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 38 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(111,108) rotado a (62,111) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 39 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(95,110) rotado a (-51,108) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 40 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(45,21) rotado a (7,40) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 41 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(40,96) rotado a (-7,85) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 42 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(10,89) rotado a (-4,72) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 43 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(12,101) rotado a (-72,42) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 44 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(95,115) rotado a (67,102) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 45 us  Iteration: 0  Instance: /
    test_modem_cordic
```

```
# ** Note: Complejo(37,89) rotado a (-10,78) ERROR ABSOLUTO
     DE ROTACION = 0.000000e+000
#    Time: 46 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(22,21) rotado a (15,20) ERROR ABSOLUTO
    DE ROTACION = 9.380000e-002
#    Time: 47 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(121,121) rotado a (-26,138) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 48 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(13,58) rotado a (-33,36) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 49 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(42,47) rotado a (1,51) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 50 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(12,112) rotado a (-23,89) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 51 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(2,89) rotado a (-59,43) ERROR ABSOLUTO
    DE ROTACION = 9.380000e-002
#    Time: 52 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(98,6) rotado a (61,53) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 53 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(12,39) rotado a (-8,33) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 54 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(124,81) rotado a (53,109) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 55 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(72,21) rotado a (54,30) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 56 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(21,80) rotado a (-33,59) ERROR ABSOLUTO
     DE ROTACION = 0.000000e+000
#    Time: 57 us  Iteration: 0  Instance: /
    test_modem_cordic
```

```
# ** Note: Complejo(62,72) rotado a (0,77) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 58 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(24,113) rotado a (-61,72) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 59 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(50,75) rotado a (21,70) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 60 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(66,0) rotado a (15,52) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 61 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(46,65) rotado a (-48,44) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 62 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(13,41) rotado a (-23,26) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 63 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(38,62) rotado a (4,59) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 64 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(54,126) rotado a (-41,105) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 65 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(127,101) rotado a (63,118) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 66 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(8,17) rotado a (-12,9) ERROR ABSOLUTO
    DE ROTACION = 1.563000e-001
#    Time: 67 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(84,81) rotado a (47,83) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 68 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(11,29) rotado a (-4,25) ERROR ABSOLUTO
    DE ROTACION = 9.380000e-002
#    Time: 69 us  Iteration: 0  Instance: /
    test_modem_cordic
```

```
# ** Note: Complejo(111,33) rotado a (69,65) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 70 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(102,44) rotado a (-15,90) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 71 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(30,102) rotado a (-67,56) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 72 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(62,108) rotado a (-82,60) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 73 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(85,48) rotado a (-30,74) ERROR ABSOLUTO
     DE ROTACION = 0.000000e+000
#    Time: 74 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(122,60) rotado a (8,111) ERROR ABSOLUTO
     DE ROTACION = 6.250000e-002
#    Time: 75 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(87,1) rotado a (27,66) ERROR ABSOLUTO
    DE ROTACION = 1.250000e-001
#    Time: 76 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(121,70) rotado a (-46,105) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 77 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(48,56) rotado a (-3,60) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 78 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(26,44) rotado a (-9,41) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 79 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(34,1) rotado a (12,24) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 80 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(86,121) rotado a (-91,82) ERROR
    ABSOLUTO DE ROTACION = 0.000000e+000
#    Time: 81 us  Iteration: 0  Instance: /
    test_modem_cordic
```

```
# ** Note: Complejo(123,117) rotado a (-84,111) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 82 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(110,46) rotado a (2,97) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 83 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(74,115) rotado a (-73,86) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 84 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(21,68) rotado a (-34,47) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 85 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(29,9) rotado a (5,24) ERROR ABSOLUTO DE
     ROTACION = 1.250000e-001
#    Time: 86 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(27,44) rotado a (-2,41) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 87 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(57,73) rotado a (-51,56) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 88 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(59,22) rotado a (0,50) ERROR ABSOLUTO
    DE ROTACION = 3.130000e-002
#    Time: 89 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(24,48) rotado a (-28,34) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 90 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(107,78) rotado a (36,102) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 91 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(70,7) rotado a (50,28) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 92 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(54,30) rotado a (28,42) ERROR ABSOLUTO
    DE ROTACION = 0.000000e+000
#    Time: 93 us  Iteration: 0  Instance: /
    test_modem_cordic
```

```
# ** Note: Complejo(12,127) rotado a (-101,27) ERROR
    ABSOLUTO DE ROTACION = 3.130000e-002
#    Time: 94 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(67,89) rotado a (10,91) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 95 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(8,31) rotado a (3,26) ERROR ABSOLUTO DE
     ROTACION = 3.130000e-002
#    Time: 96 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(65,41) rotado a (-24,58) ERROR ABSOLUTO
     DE ROTACION = 3.130000e-002
#    Time: 97 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(93,86) rotado a (-56,87) ERROR ABSOLUTO
     DE ROTACION = 0.000000e+000
#    Time: 98 us  Iteration: 0  Instance: /
    test_modem_cordic
# ** Note: Complejo(73,51) rotado a (41,60) ERROR ABSOLUTO
    DE ROTACION = 6.250000e-002
#    Time: 99 us  Iteration: 0  Instance: /
    test_modem_cordic
```

Representación de los errores para los 100 valores

En la siguiente gráfica, como se explicó en la Sección 3.3.3, aparecen representados los errores absolutos de rotación para las 100 tuplas, siendo el máximo error de 1.563000e-001 y único.

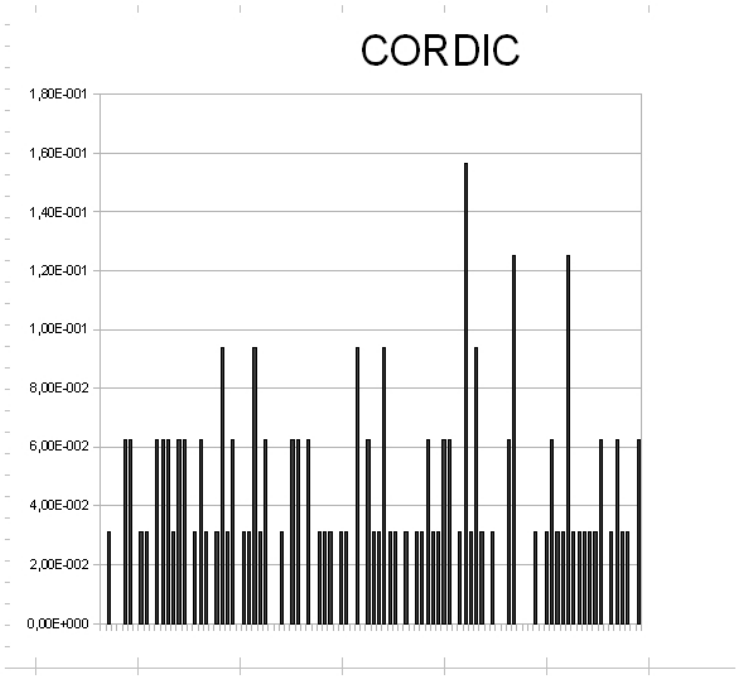

Figura 9.4: Resultados del testbench del CORDIC.

Capítulo 10

Resultados de la síntesis

10.1. Introducción

En este capítulo se muestran los informes de síntesis del código del módem producido por el entorno Xilinx ISE.

10.2. Design Summary

Figura 10.1: Design Summary

PFC Project Status (12/25/2008 - 16:19:47)			
Project File:	PFC.ise	**Current State:**	Programming File Generated
Module Name:	modem	**• Errors:**	No Errors
Target Device:	xc4vlx15-12sf363	**• Warnings:**	81 Warnings
Product Version:	ISE 10.1.03 - WebPACK	**• Routing Results:**	All Signals Completely Routed
Design Goal:	Balanced	**• Timing Constraints:**	All Constraints Met
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	0 (Timing Report)

PFC Partition Summary [-]
No partition information was found.

Device Utilization Summary [-]				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	747	12,288	6%	
Number used as Flip Flops	741			
Number used as Latches	6			
Number of 4 input LUTs	3,178	12,288	25%	
Logic Distribution				
Number of occupied Slices	2,030	6,144	33%	
Number of Slices containing only related logic	2,030	2,030	100%	
Number of Slices containing unrelated logic	0	2,030	0%	
Total Number of 4 input LUTs	3,307	12,288	26%	
Number used as logic	3,170			
Number used as a route-thru	129			
Number used as Shift registers	8			
Number of bonded IOBs				
Number of bonded	162	240	67%	
IOB Flip Flops	20			
Number of BUFG/BUFGCTRLs	3	32	9%	
Number used as BUFGs	3			
Number of DSP48s	12	32	37%	

Performance Summary [+]

Detailed Reports [+]

Date Generated: 12/25/2008 - 16:39:56

10.3. Synthesis Report

10.3.1. Synthesis Options Summary

```
=========================  Source Parameters
Input File Name                    : "modem.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

=========================  Target Parameters
Output File Name                   : "modem"
Output Format                      : NGC
Target Device                      : xc4vlx15-12-sf363

=========================  Source Options
Top Module Name                    : modem
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
ROM Style                          : Auto
Mux Extraction                     : YES
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Use DSP Block                      : auto
Automatic Register Balancing       : No

=========================  Target Options
Add IO Buffers                     : YES
Global Maximum Fanout              : 500
Add Generic Clock Buffer(BUFG)     : 32
Number of Regional Clock Buffers   : 16
Register Duplication               : YES
Slice Packing                      : YES
Optimize Instantiated Primitives   : NO
Use Clock Enable                   : Auto
Use Synchronous Set                : Auto
Use Synchronous Reset              : Auto
Pack IO Registers into IOBs        : auto
```

```
Equivalent register Removal          : YES

---- General Options
Optimization Goal                    : Speed
Optimization Effort                  : 1
Power Reduction                      : NO
Library Search Order                 : modem.lso
Keep Hierarchy                       : NO
Netlist Hierarchy                    : as_optimized
RTL Output                           : Yes
Global Optimization                  : AllClockNets
Read Cores                           : YES
Write Timing Constraints             : NO
Cross Clock Analysis                 : NO
Hierarchy Separator                  : /
Bus Delimiter                        : <>
Case Specifier                       : maintain
Slice Utilization Ratio              : 100
BRAM Utilization Ratio               : 100
DSP48 Utilization Ratio              : 100
Verilog 2001                         : YES
Auto BRAM Packing                    : NO
Slice Utilization Ratio Delta        : 5
```

10.3.2. HDL Compilation

```
Compiling vhdl file "C:/PFC/modem_cordic.vhd" in Library
    work.
Architecture modem_cordic of Entity modem_cordic is up to
    date.
Compiling vhdl file "C:/PFC/modem_rx_controller.vhd" in
    Library work.
Architecture behavioral of Entity modem_rx_controller is up
     to date.
Compiling vhdl file "C:/PFC/modem_rx_coarse_freq_estimator.
    vhd" in Library work.
Architecture behavioral of Entity
    modem_rx_coarse_freq_estimator is up to date.
Compiling vhdl file "C:/PFC/modem_rx_mfilter.vhd" in
    Library work.
Architecture behavioral of Entity modem_rx_mfilter is up to
     date.
Compiling vhdl file "C:/PFC/modem_rx_downsampler.vhd" in
    Library work.
Architecture behavioral of Entity modem_rx_downsampler is
    up to date.
```

```
Compiling vhdl file "C:/PFC/modem_rx_fine_freq_estimator.
    vhd" in Library work.
Architecture behavioral of Entity
    modem_rx_fine_freq_estimator is up to date.
Compiling vhdl file "C:/PFC/modem_rx_rz_encoder.vhd" in
    Library work.
Architecture behavioral of Entity modem_rx_rz_encoder is up
     to date.
Compiling vhdl file "C:/PFC/modem_rx_correlator.vhd" in
    Library work.
Architecture behavioral of Entity modem_rx_correlator is up
     to date.
Compiling vhdl file "C:/PFC/modem_rx_symbol_to_bit.vhd" in
    Library work.
Architecture behavioral of Entity modem_rx_symbol_to_bit is
     up to date.
Compiling vhdl file "C:/PFC/modem_tx_controller.vhd" in
    Library work.
Architecture behavioral of Entity modem_tx_controller is up
     to date.
Compiling vhdl file "C:/PFC/modem_tx_bit_to_symbol.vhd" in
    Library work.
Architecture behavioral of Entity modem_tx_bit_to_symbol is
     up to date.
Compiling vhdl file "C:/PFC/modem_tx_chip_gen.vhd" in
    Library work.
Architecture behavioral of Entity modem_tx_chip_gen is up
    to date.
Compiling vhdl file "C:/PFC/modem_tx_upsampler.vhd" in
    Library work.
Architecture behavioral of Entity modem_tx_upsampler is up
    to date.
Compiling vhdl file "C:/PFC/modem_tx_mfilter.vhd" in
    Library work.
Architecture behavioral of Entity modem_tx_mfilter is up to
     date.
Compiling vhdl file "C:/PFC/modem_controller.vhd" in
    Library work.
Architecture behavioral of Entity modem_controller is up to
     date.
Compiling vhdl file "C:/PFC/modem_tx.vhd" in Library work.
Architecture behavioral of Entity modem_tx is up to date.
Compiling vhdl file "C:/PFC/modem_tx_to_rx.vhd" in Library
    work.
Architecture behavioral of Entity modem_tx_to_rx is up to
    date.
Compiling vhdl file "C:/PFC/modem_rx.vhd" in Library work.
Architecture behavioral of Entity modem_rx is up to date.
Compiling vhdl file "C:/PFC/modem.vhd" in Library work.
```

```
Architecture behavioral of Entity modem is up to date.
```

10.3.3. Design Hierarchy Analysis

```
Analyzing hierarchy for entity <modem> in library <work> (
    architecture <behavioral>).

Analyzing hierarchy for entity <modem_controller> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <modem_tx> in library <work>
     (architecture <behavioral>).

Analyzing hierarchy for entity <modem_tx_to_rx> in library
    <work> (architecture <behavioral>).

Analyzing hierarchy for entity <modem_rx> in library <work>
     (architecture <behavioral>).

Analyzing hierarchy for entity <modem_tx_controller> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <modem_tx_bit_to_symbol> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <modem_tx_chip_gen> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <modem_tx_upsampler> in
    library <work> (architecture <behavioral>) with generics
    .
        factor = 7

Analyzing hierarchy for entity <modem_tx_mfilter> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <modem_rx_controller> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <
    modem_rx_coarse_freq_estimator> in library <work> (
    architecture <behavioral>) with generics.
        chips_estim = 256
        n = 8

Analyzing hierarchy for entity <modem_rx_mfilter> in
    library <work> (architecture <behavioral>).
```

```
Analyzing hierarchy for entity <modem_rx_downsampler> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <
    modem_rx_fine_freq_estimator> in library <work> (
    architecture <behavioral>) with generics.
        chips_estim = 80
        n = 6

Analyzing hierarchy for entity <modem_rx_rz_encoder> in
    library <work> (architecture <behavioral>).

Analyzing hierarchy for entity <modem_rx_correlator> in
    library <work> (architecture <behavioral>) with generics
    .
        t_symbol = 16

Analyzing hierarchy for entity <modem_rx_symbol_to_bit> in
    library <work> (architecture <behavioral>).
```

10.3.4. HDL Analysis

```
Analyzing Entity <modem> in library <work> (Architecture <
    behavioral>).
Entity <modem> analyzed. Unit <modem> generated.

Analyzing Entity <modem_controller> in library <work> (
    Architecture <behavioral>).
Entity <modem_controller> analyzed. Unit <modem_controller>
     generated.

Analyzing Entity <modem_tx> in library <work> (Architecture
     <behavioral>).
Entity <modem_tx> analyzed. Unit <modem_tx> generated.

Analyzing Entity <modem_tx_controller> in library <work> (
    Architecture <behavioral>).
Entity <modem_tx_controller> analyzed. Unit <
    modem_tx_controller> generated.

Analyzing Entity <modem_tx_bit_to_symbol> in library <work>
     (Architecture <behavioral>).
Entity <modem_tx_bit_to_symbol> analyzed. Unit <
    modem_tx_bit_to_symbol> generated.
```

```
Analyzing Entity <modem_tx_chip_gen> in library <work> (
    Architecture <behavioral>).
Entity <modem_tx_chip_gen> analyzed. Unit <
    modem_tx_chip_gen> generated.

Analyzing generic Entity <modem_tx_upsampler> in library <
    work> (Architecture <behavioral>).
        factor = 7
Entity <modem_tx_upsampler> analyzed. Unit <
    modem_tx_upsampler> generated.

Analyzing Entity <modem_tx_mfilter> in library <work> (
    Architecture <behavioral>).
Entity <modem_tx_mfilter> analyzed. Unit <modem_tx_mfilter>
     generated.

Analyzing Entity <modem_tx_to_rx> in library <work> (
    Architecture <behavioral>).
Entity <modem_tx_to_rx> analyzed. Unit <modem_tx_to_rx>
    generated.

Analyzing Entity <modem_rx> in library <work> (Architecture
     <behavioral>).
Entity <modem_rx> analyzed. Unit <modem_rx> generated.

Analyzing Entity <modem_rx_controller> in library <work> (
    Architecture <behavioral>).
INFO:Xst:2679 - Register <controller_fine_freq_estim_en> in
     unit <modem_rx_controller> has a constant value of 1
    during circuit operation. The register is replaced by
    logic.
Entity <modem_rx_controller> analyzed. Unit <
    modem_rx_controller> generated.

Analyzing generic Entity <modem_rx_coarse_freq_estimator>
    in library <work> (Architecture <behavioral>).
        chips_estim = 256
        n = 8
Entity <modem_rx_coarse_freq_estimator> analyzed. Unit <
    modem_rx_coarse_freq_estimator> generated.

Analyzing Entity <modem_rx_mfilter> in library <work> (
    Architecture <behavioral>).
Entity <modem_rx_mfilter> analyzed. Unit <modem_rx_mfilter>
     generated.

Analyzing Entity <modem_rx_downsampler> in library <work> (
    Architecture <behavioral>).
```

```
Entity <modem_rx_downsampler> analyzed. Unit <
    modem_rx_downsampler> generated.

Analyzing generic Entity <modem_rx_fine_freq_estimator> in
    library <work> (Architecture <behavioral>).
        chips_estim = 80
        n = 6
Entity <modem_rx_fine_freq_estimator> analyzed. Unit <
    modem_rx_fine_freq_estimator> generated.

Analyzing Entity <modem_rx_rz_encoder> in library <work> (
    Architecture <behavioral>).
Entity <modem_rx_rz_encoder> analyzed. Unit <
    modem_rx_rz_encoder> generated.

Analyzing generic Entity <modem_rx_correlator> in library <
    work> (Architecture <behavioral>).
        t_symbol = 16
INFO:Xst:2679 - Register <corr_value<15>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<14>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<13>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<12>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<11>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<10>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<9>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<8>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
```

```
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<7>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<6>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<5>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<4>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<3>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<2>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<1>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
INFO:Xst:2679 - Register <corr_value<0>> in unit <
    modem_rx_correlator> has a constant value of
    00000000000000000000000000000000 during circuit
    operation. The register is replaced by logic.
Entity <modem_rx_correlator> analyzed. Unit <
    modem_rx_correlator> generated.

Analyzing Entity <modem_rx_symbol_to_bit> in library <work>
    (Architecture <behavioral>).
Entity <modem_rx_symbol_to_bit> analyzed. Unit <
    modem_rx_symbol_to_bit> generated.
```

10.3.5. HDL Synthesis

```
Performing bidirectional port resolution...

Synthesizing Unit <modem_controller>.
    Related source file is "C:/PFC/modem_controller.vhd".
```

```
Unit <modem_controller> synthesized.


Synthesizing Unit <modem_tx_to_rx>.
    Related source file is "C:/PFC/modem_tx_to_rx.vhd".
Unit <modem_tx_to_rx> synthesized.


Synthesizing Unit <modem_tx_controller>.
    Related source file is "C:/PFC/modem_tx_controller.vhd"
        .
    Found 11-bit comparator greatequal for signal <
        controller_bit_out$cmp_ge0000> created at line 64.
    Found 11-bit comparator greater for signal <
        controller_bit_out$cmp_gt0000> created at line 64.
    Found 11-bit comparator less for signal <
        controller_bit_out$cmp_lt0000> created at line 64.
    Found 11-bit up counter for signal <count>.
    Found 11-bit comparator greatequal for signal <
        count$cmp_ge0000> created at line 53.
    Summary:
        inferred   1 Counter(s).
        inferred   4 Comparator(s).
Unit <modem_tx_controller> synthesized.


Synthesizing Unit <modem_tx_bit_to_symbol>.
    Related source file is "C:/PFC/modem_tx_bit_to_symbol.
        vhd".
    Found 3-bit register for signal <count>.
    Found 3-bit adder for signal <count$add0000> created at
         line 49.
    Found 4-bit register for signal <symbol_reg>.
    Summary:
        inferred   7 D-type flip-flop(s).
        inferred   1 Adder/Subtractor(s).
Unit <modem_tx_bit_to_symbol> synthesized.


Synthesizing Unit <modem_tx_chip_gen>.
    Related source file is "C:/PFC/modem_tx_chip_gen.vhd".
    Register <symbol_reg_q$mux0000> equivalent to <
        symbol_reg_i$mux0000> has been removed
WARNING:Xst:737 - Found 4-bit latch for signal <
    symbol_reg_i$mux0000>. Latches may be generated from
    incomplete case or if statements. We do not recommend
    the use of latches in FPGA/CPLD designs, as they may
    lead to timing problems.
    Found 1-bit 8-to-1 multiplexer for signal <muxIout>.
```

```
    Found 1-bit 8-to-1 multiplexer for signal <muxQout>.
    Found 1-bit xor2 for signal <muxQout$xor0000> created
       at line 126.
    Found 1-bit xor2 for signal <muxQout$xor0001> created
       at line 124.
    Found 1-bit xor2 for signal <muxQout$xor0002> created
       at line 122.
    Found 1-bit xor2 for signal <muxQout$xor0003> created
       at line 120.
    Found 1-bit xor2 for signal <muxQout$xor0004> created
       at line 118.
    Found 1-bit xor2 for signal <muxQout$xor0005> created
       at line 116.
    Found 1-bit xor2 for signal <muxQout$xor0006> created
       at line 114.
    Found 1-bit xor2 for signal <muxQout$xor0007> created
       at line 112.
    Found 16-bit register for signal <sr1>.
    Found 16-bit register for signal <sr2>.
    Summary:
        inferred   2 Multiplexer(s).
Unit <modem_tx_chip_gen> synthesized.


Synthesizing Unit <modem_tx_upsampler>.
    Related source file is "C:/PFC/modem_tx_upsampler.vhd".
    Found 4-bit up counter for signal <count>.
    Summary:
        inferred   1 Counter(s).
Unit <modem_tx_upsampler> synthesized.


Synthesizing Unit <modem_tx_mfilter>.
    Related source file is "C:/PFC/modem_tx_mfilter.vhd".
    Found 10-bit register for signal <acc>.
    Found 10-bit addsub for signal <acc$share0000> created
       at line 89.
    Found 10-bit addsub for signal <acc$share0001> created
       at line 89.
    Found 10-bit addsub for signal <acc$share0002> created
       at line 89.
    Found 10-bit addsub for signal <acc$share0003> created
       at line 89.
    Found 10-bit addsub for signal <acc$share0004> created
       at line 89.
    Found 10-bit addsub for signal <acc$share0005>.
    Found 14-bit register for signal <reg>.
    Summary:
        inferred  24 D-type flip-flop(s).
```

```
        inferred    6 Adder/Subtractor(s).
Unit <modem_tx_mfilter> synthesized.


Synthesizing Unit <modem_rx_controller>.
    Related source file is "C:/PFC/modem_rx_controller.vhd"
        .
WARNING:Xst:647 - Input <controller_symbol_valid> is never
    used. This port will be preserved and left unconnected
    if it belongs to a top-level block or it belongs to a
    sub-block and the hierarchy of this sub-block is
    preserved.
WARNING:Xst:737 - Found 1-bit latch for signal <
    part1_detected>. Latches may be generated from
    incomplete case or if statements. We do not recommend
    the use of latches in FPGA/CPLD designs, as they may
    lead to timing problems.
Unit <modem_rx_controller> synthesized.


Synthesizing Unit <modem_rx_coarse_freq_estimator>.
    Related source file is "C:/PFC/
        modem_rx_coarse_freq_estimator.vhd".
    Found 10-bit comparator greatequal for signal <
        acc_phase_rads$cmp_ge0000> created at line 141.
    Found 10-bit comparator greatequal for signal <
        acc_phase_rads$cmp_ge0001> created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0002>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0003>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0004>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0005>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0006>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0007>
         created at line 141.
    Found 10-bit addsub for signal <angle_temp$mux0000>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0001>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0002>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0003>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0004>
        created at line 89.
```

```
Found 20-bit register for signal <cordic_temp_rot>.
Found 10-bit adder for signal <
    cordic_temp_rot$addsub0000> created at line 61.
Found 10-bit adder for signal <
    cordic_temp_rot$addsub0001> created at line 61.
Found 10-bit comparator less for signal <
    cordic_temp_rot$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    cordic_temp_rot$cmp_lt0001> created at line 60.
Found 10-bit adder for signal <
    cordic_temp_vec$addsub0000> created at line 152.
Found 9-bit up counter for signal <count>.
Found 9-bit comparator greatequal for signal <
    count$cmp_ge0000> created at line 54.
Found 10-bit adder for signal <i_signed$addsub0000>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0001>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0002>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0003>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0004>
    created at line 61.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0000> created at line 141.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0001> created at line 141.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0002> created at line 141.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0003> created at line 141.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0003> created at line 60.
Found 10-bit addsub for signal <i_signed$mux0004>
    created at line 141.
Found 10-bit addsub for signal <i_signed$mux0006>
    created at line 141.
Found 10-bit addsub for signal <i_signed$mux0008>
    created at line 141.
Found 10-bit addsub for signal <i_signed$mux0010>
    created at line 141.
Found 10-bit adder for signal <i_signed0$add0000>.
```

```
Found 10-bit adder for signal <i_signed0$addsub0000>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0001>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0002>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0003>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0004>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0005>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0006>
    created at line 61.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0003> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0004> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0005> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0006> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0007> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0008> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0009> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0010> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0011> created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0002>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0004>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0006>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0008>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0010>
    created at line 89.
```

```
Found 10-bit addsub for signal <i_signed0$mux0012>
    created at line 89.
Found 10-bit up accumulator for signal <kay_sum>.
Found 9-bit comparator less for signal <
    kay_sum$cmp_lt0000> created at line 75.
Found 10-bit subtractor for signal <kay_sum$sub0000>
    created at line 80.
Found 10-bit register for signal <last_phase>.
Found 10-bit adder for signal <phase_acc$addsub0000>
    created at line 61.
Found 10-bit comparator less for signal <
    phase_acc$cmp_lt0000> created at line 60.
Found 10-bit adder for signal <q_signed$addsub0000>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0001>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0002>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0003>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0004>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0005>
    created at line 61.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0003> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0004> created at line 60.
Found 10-bit addsub for signal <q_signed$mux0004>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0006>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0008>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0010>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0012>
    created at line 141.
Found 10-bit adder for signal <q_signed0$add0000>.
Found 10-bit adder for signal <q_signed0$addsub0000>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0001>
    created at line 61.
```

```
Found 10-bit adder for signal <q_signed0$addsub0002>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0003>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0004>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0005>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0006>
    created at line 61.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0003> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0004> created at line 60.
Found 10-bit addsub for signal <q_signed0$mux0002>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0004>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0006>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0008>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0010>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0012>
    created at line 89.
Found 10-bit adder for signal <temp0$add0000>.
Found 10-bit adder for signal <temp0$sub0000> created
    at line 131.
Found 10-bit adder for signal <temp1$add0000>.
Found 10-bit comparator greater for signal <
    temp1$cmp_gt0000> created at line 126.
Found 10-bit comparator less for signal <
    temp1$cmp_lt0000> created at line 124.
Found 10-bit adder for signal <temp1$sub0000> created
    at line 128.
Found 10-bit adder for signal <temp11$addsub0000>.
Found 10-bit adder for signal <temp12$addsub0000>.
Found 10-bit adder for signal <temp16$addsub0000>.
Found 10-bit adder for signal <temp24$addsub0000>.
Found 10-bit adder for signal <temp29$addsub0000>.
Found 10-bit adder for signal <temp3$addsub0000>.
Found 10-bit adder for signal <temp30$addsub0000>.
```

```
    Found 10-bit adder for signal <temp33$addsub0000>.
    Found 10-bit adder for signal <temp34$addsub0000>.
    Found 10-bit adder for signal <temp37$addsub0000>.
    Found 10-bit adder for signal <temp38$addsub0000>.
    Found 10-bit adder for signal <temp4$addsub0000>.
    Found 10-bit adder for signal <temp41$addsub0000>.
    Found 10-bit adder for signal <temp42$addsub0000>.
    Found 10-bit adder for signal <temp45$addsub0000>.
    Found 10-bit adder for signal <temp46$addsub0000>.
    Found 10-bit adder for signal <temp49$addsub0000>.
    Found 10-bit adder for signal <temp50$addsub0000>.
    Found 10-bit adder for signal <temp7$addsub0000>.
    Found 10-bit adder for signal <temp8$addsub0000>.
    Summary:
        inferred   1 Counter(s).
        inferred   1 Accumulator(s).
        inferred  30 D-type flip-flop(s).
        inferred  88 Adder/Subtractor(s).
        inferred  39 Comparator(s).
Unit <modem_rx_coarse_freq_estimator> synthesized.


Synthesizing Unit <modem_rx_mfilter>.
    Related source file is "C:/PFC/modem_rx_mfilter.vhd".
    Found 20-bit adder for signal <acc$add0000> created at
       line 82.
    Found 20-bit adder for signal <acc$add0001> created at
       line 82.
    Found 20-bit adder for signal <acc$addsub0000> created
       at line 82.
    Found 20-bit adder for signal <acc$addsub0001> created
       at line 82.
    Found 20-bit adder for signal <acc$addsub0002> created
       at line 82.
    Found 20-bit adder for signal <acc$addsub0003> created
       at line 82.
    Found 10x10-bit multiplier for signal <acc$mult0000>
       created at line 82.
    Found 10x10-bit multiplier for signal <acc$mult0001>
       created at line 81.
    Found 10x10-bit multiplier for signal <acc$mult0002>
       created at line 81.
    Found 10x10-bit multiplier for signal <acc$mult0003>
       created at line 81.
    Found 10x10-bit multiplier for signal <acc$mult0004>
       created at line 81.
    Found 10x10-bit multiplier for signal <acc$mult0005>
       created at line 81.
    Found 20-bit register for signal <acc$mux0000>.
```

```
    Found 70-bit register for signal <reg>.
    Summary:
        inferred  90 D-type flip-flop(s).
        inferred   6 Adder/Subtractor(s).
        inferred   6 Multiplier(s).
Unit <modem_rx_mfilter> synthesized.


Synthesizing Unit <modem_rx_downsampler>.
    Related source file is "C:/PFC/modem_rx_downsampler.vhd
        ".
    Found 10-bit register for signal <downsampler_output>.
    Summary:
        inferred  10 D-type flip-flop(s).
Unit <modem_rx_downsampler> synthesized.


Synthesizing Unit <modem_rx_fine_freq_estimator>.
    Related source file is "C:/PFC/
        modem_rx_fine_freq_estimator.vhd".
    Found 10-bit comparator greatequal for signal <
        acc_phase_rads$cmp_ge0000> created at line 141.
    Found 10-bit comparator greatequal for signal <
        acc_phase_rads$cmp_ge0001> created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0002>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0003>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0004>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0005>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0006>
         created at line 141.
    Found 10-bit addsub for signal <acc_phase_rads$mux0007>
         created at line 141.
    Found 10-bit addsub for signal <angle_temp$mux0000>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0001>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0002>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0003>
        created at line 89.
    Found 10-bit addsub for signal <angle_temp$mux0004>
        created at line 89.
    Found 10-bit adder for signal <
        cordic_temp_rot$addsub0000> created at line 61.
```

```
Found 10-bit adder for signal <
    cordic_temp_rot$addsub0001> created at line 61.
Found 10-bit comparator less for signal <
    cordic_temp_rot$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    cordic_temp_rot$cmp_lt0001> created at line 60.
Found 10-bit adder for signal <
    cordic_temp_vec$addsub0000> created at line 152.
Found 7-bit up counter for signal <count>.
Found 7-bit comparator greatequal for signal <
    count$cmp_ge0000> created at line 88.
Found 10-bit up accumulator for signal <fine_sum>.
Found 7-bit comparator greatequal for signal <
    fine_sum$cmp_ge0000> created at line 109.
Found 10-bit subtractor for signal <fine_sum$sub0000>
    created at line 115.
Found 10-bit adder for signal <i_signed$addsub0000>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0001>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0002>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0003>
    created at line 61.
Found 10-bit adder for signal <i_signed$addsub0004>
    created at line 61.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0000> created at line 141.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0001> created at line 141.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0002> created at line 141.
Found 10-bit comparator greatequal for signal <
    i_signed$cmp_ge0003> created at line 141.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    i_signed$cmp_lt0003> created at line 60.
Found 10-bit addsub for signal <i_signed$mux0004>
    created at line 141.
Found 10-bit addsub for signal <i_signed$mux0006>
    created at line 141.
Found 10-bit addsub for signal <i_signed$mux0008>
    created at line 141.
```

```
Found 10-bit addsub for signal <i_signed$mux0010>
    created at line 141.
Found 10-bit adder for signal <i_signed0$add0000>.
Found 10-bit adder for signal <i_signed0$addsub0000>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0001>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0002>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0003>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0004>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0005>
    created at line 61.
Found 10-bit adder for signal <i_signed0$addsub0006>
    created at line 61.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0003> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0004> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0005> created at line 60.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0006> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0007> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0008> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0009> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0010> created at line 89.
Found 10-bit comparator less for signal <
    i_signed0$cmp_lt0011> created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0002>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0004>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0006>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0008>
    created at line 89.
```

```
Found 10-bit addsub for signal <i_signed0$mux0010>
    created at line 89.
Found 10-bit addsub for signal <i_signed0$mux0012>
    created at line 89.
Found 10-bit register for signal <output_i>.
Found 7-bit comparator less for signal <
    output_i$cmp_lt0000> created at line 109.
Found 10-bit register for signal <output_q>.
Found 10-bit adder for signal <phase_acc$addsub0000>
    created at line 61.
Found 10-bit comparator less for signal <
    phase_acc$cmp_lt0000> created at line 60.
Found 160-bit register for signal <phase_table_0>.
Found 10-bit adder for signal <q_signed$addsub0000>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0001>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0002>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0003>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0004>
    created at line 61.
Found 10-bit adder for signal <q_signed$addsub0005>
    created at line 61.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0003> created at line 60.
Found 10-bit comparator less for signal <
    q_signed$cmp_lt0004> created at line 60.
Found 10-bit addsub for signal <q_signed$mux0004>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0006>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0008>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0010>
    created at line 141.
Found 10-bit addsub for signal <q_signed$mux0012>
    created at line 141.
Found 10-bit adder for signal <q_signed0$add0000>.
Found 10-bit adder for signal <q_signed0$addsub0000>
    created at line 61.
```

```
Found 10-bit adder for signal <q_signed0$addsub0001>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0002>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0003>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0004>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0005>
    created at line 61.
Found 10-bit adder for signal <q_signed0$addsub0006>
    created at line 61.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0000> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0001> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0002> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0003> created at line 60.
Found 10-bit comparator less for signal <
    q_signed0$cmp_lt0004> created at line 60.
Found 10-bit addsub for signal <q_signed0$mux0002>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0004>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0006>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0008>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0010>
    created at line 89.
Found 10-bit addsub for signal <q_signed0$mux0012>
    created at line 89.
Found 10-bit adder for signal <temp0$add0000>.
Found 10-bit adder for signal <temp0$sub0000> created
    at line 131.
Found 10-bit adder for signal <temp1$add0000>.
Found 10-bit comparator greater for signal <
    temp1$cmp_gt0000> created at line 126.
Found 10-bit comparator less for signal <
    temp1$cmp_lt0000> created at line 124.
Found 10-bit adder for signal <temp1$sub0000> created
    at line 128.
Found 10-bit adder for signal <temp11$addsub0000>.
Found 10-bit adder for signal <temp12$addsub0000>.
Found 10-bit adder for signal <temp16$addsub0000>.
Found 10-bit adder for signal <temp24$addsub0000>.
Found 10-bit adder for signal <temp29$addsub0000>.
```

```
    Found 10-bit adder for signal <temp3$addsub0000>.
    Found 10-bit adder for signal <temp30$addsub0000>.
    Found 10-bit adder for signal <temp33$addsub0000>.
    Found 10-bit adder for signal <temp34$addsub0000>.
    Found 10-bit adder for signal <temp37$addsub0000>.
    Found 10-bit adder for signal <temp38$addsub0000>.
    Found 10-bit adder for signal <temp4$addsub0000>.
    Found 10-bit adder for signal <temp41$addsub0000>.
    Found 10-bit adder for signal <temp42$addsub0000>.
    Found 10-bit adder for signal <temp45$addsub0000>.
    Found 10-bit adder for signal <temp46$addsub0000>.
    Found 10-bit adder for signal <temp49$addsub0000>.
    Found 10-bit adder for signal <temp50$addsub0000>.
    Found 10-bit adder for signal <temp7$addsub0000>.
    Found 10-bit adder for signal <temp8$addsub0000>.
    Summary:
        inferred   1 Counter(s).
        inferred   1 Accumulator(s).
        inferred  20 D-type flip-flop(s).
        inferred  88 Adder/Subtractor(s).
        inferred  40 Comparator(s).
Unit <modem_rx_fine_freq_estimator> synthesized.


Synthesizing Unit <modem_rx_rz_encoder>.
    Related source file is "C:/PFC/modem_rx_rz_encoder.vhd"
        .
    Found 10-bit comparator greater for signal <
       rz_output_i$cmp_gt0000> created at line 35.
    Found 10-bit comparator greater for signal <
       rz_output_j$cmp_gt0000> created at line 36.
    Summary:
        inferred   2 Comparator(s).
Unit <modem_rx_rz_encoder> synthesized.


Synthesizing Unit <modem_rx_correlator>.
    Related source file is "C:/PFC/modem_rx_correlator.vhd"
        .
WARNING:Xst:646 - Signal <sym<15><15>> is assigned but
    never used. This unconnected signal will be trimmed
    during the optimization process.
WARNING:Xst:646 - Signal <sym<14><15>> is assigned but
    never used. This unconnected signal will be trimmed
    during the optimization process.
WARNING:Xst:646 - Signal <sym<13><15>> is assigned but
    never used. This unconnected signal will be trimmed
    during the optimization process.
```

```
WARNING:Xst:646 - Signal <sym<12><15>> is assigned but
    never used. This unconnected signal will be trimmed
    during the optimization process.
WARNING:Xst:646 - Signal <sym<11><15>> is assigned but
    never used. This unconnected signal will be trimmed
    during the optimization process.
WARNING:Xst:646 - Signal <sym<10><15>> is assigned but
    never used. This unconnected signal will be trimmed
    during the optimization process.
WARNING:Xst:646 - Signal <sym<9><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<8><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<7><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<6><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<5><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<4><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<3><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<2><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<1><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <sym<0><15>> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
WARNING:Xst:646 - Signal <reg<15>> is assigned but never
    used. This unconnected signal will be trimmed during the
     optimization process.
WARNING:Xst:646 - Signal <corr_value> is assigned but never
     used. This unconnected signal will be trimmed during
    the optimization process.
    Found 16x4-bit ROM for signal <
        correlator_symbol$mux0000> created at line 178.
    Found 1-bit register for signal <
        correlator_symbol_valid>.
```

```
Found 4-bit register for signal <correlator_symbol>.
Found 16-bit register for signal <chip_reg>.
Found 32-bit adder for signal <corr_value_0$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_0$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_0$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_0$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_0$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_0$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_0$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_0$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_0$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_1$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_1$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_1$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_1$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_1$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_1$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_1$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_1$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_1$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_10$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_10$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_10$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_10$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_10$add0005>
    created at line 166.
```

```
Found 38-bit adder for signal <corr_value_10$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_10$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_10$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_10$addsub0000
    > created at line 166.
Found 32-bit adder for signal <corr_value_11$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_11$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_11$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_11$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_11$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_11$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_11$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_11$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_11$addsub0000
    > created at line 166.
Found 32-bit adder for signal <corr_value_12$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_12$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_12$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_12$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_12$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_12$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_12$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_12$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_12$addsub0000
    > created at line 166.
Found 32-bit adder for signal <corr_value_13$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_13$add0002>
    created at line 166.
```

```
Found 35-bit adder for signal <corr_value_13$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_13$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_13$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_13$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_13$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_13$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_13$addsub0000
    > created at line 166.
Found 32-bit adder for signal <corr_value_14$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_14$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_14$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_14$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_14$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_14$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_14$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_14$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_14$addsub0000
    > created at line 166.
Found 32-bit adder for signal <corr_value_15$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_15$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_15$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_15$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_15$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_15$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_15$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_15$add0008>
    created at line 166.
```

```
Found 32-bit adder for signal <corr_value_15$addsub0000
    > created at line 166.
Found 32-bit adder for signal <corr_value_2$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_2$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_2$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_2$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_2$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_2$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_2$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_2$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_2$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_3$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_3$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_3$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_3$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_3$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_3$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_3$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_3$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_3$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_4$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_4$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_4$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_4$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_4$add0005>
    created at line 166.
```

```
Found 38-bit adder for signal <corr_value_4$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_4$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_4$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_4$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_5$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_5$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_5$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_5$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_5$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_5$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_5$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_5$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_5$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_6$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_6$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_6$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_6$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_6$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_6$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_6$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_6$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_6$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_7$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_7$add0002>
    created at line 166.
```

```
Found 35-bit adder for signal <corr_value_7$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_7$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_7$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_7$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_7$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_7$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_7$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_8$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_8$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_8$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_8$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_8$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_8$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_8$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_8$add0008>
    created at line 166.
Found 32-bit adder for signal <corr_value_8$addsub0000>
     created at line 166.
Found 32-bit adder for signal <corr_value_9$add0000>
    created at line 166.
Found 34-bit adder for signal <corr_value_9$add0002>
    created at line 166.
Found 35-bit adder for signal <corr_value_9$add0003>
    created at line 166.
Found 36-bit adder for signal <corr_value_9$add0004>
    created at line 166.
Found 37-bit adder for signal <corr_value_9$add0005>
    created at line 166.
Found 38-bit adder for signal <corr_value_9$add0006>
    created at line 166.
Found 39-bit adder for signal <corr_value_9$add0007>
    created at line 166.
Found 40-bit adder for signal <corr_value_9$add0008>
    created at line 166.
```

```
Found 32-bit adder for signal <corr_value_9$addsub0000>
    created at line 166.
Found 5-bit up counter for signal <count>.
Found 5-bit adder for signal <count$addsub0000> created
    at line 114.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0000> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0001> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0002> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0003> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0004> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0005> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0006> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0007> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0008> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0009> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0010> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0011> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0012> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0013> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0014> created at line 85.
Found 32-bit comparator greater for signal <
   n0000_0$cmp_gt0015> created at line 85.
Found 16-bit register for signal <reg>.
Found 256-bit register for signal <sym>.
Found 38-bit adder for signal <sym_pos$add0000> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0001> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0002> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0003> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0004> created
    at line 166.
```

```
Found 38-bit adder for signal <sym_pos$add0005> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0006> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0007> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0008> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0009> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0010> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0011> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0012> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0013> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0014> created
    at line 166.
Found 38-bit adder for signal <sym_pos$add0015> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0016> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0017> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0018> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0019> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0020> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0021> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0022> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0023> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0024> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0025> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0026> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0027> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0028> created
    at line 166.
```

```
Found 36-bit adder for signal <sym_pos$add0029> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0030> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0031> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0032> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0033> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0034> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0035> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0036> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0037> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0038> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0039> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0040> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0041> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0042> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0043> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0044> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0045> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0046> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0047> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0048> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0049> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0050> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0051> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0052> created
    at line 166.
```

```
Found 35-bit adder for signal <sym_pos$add0053> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0054> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0055> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0056> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0057> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0058> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0059> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0060> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0061> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0062> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0063> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0064> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0065> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0066> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0067> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0068> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0069> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0070> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0071> created
    at line 166.
Found 34-bit adder for signal <sym_pos$add0072> created
    at line 166.
Found 35-bit adder for signal <sym_pos$add0073> created
    at line 166.
Found 36-bit adder for signal <sym_pos$add0074> created
    at line 166.
Found 37-bit adder for signal <sym_pos$add0075> created
    at line 166.
Found 33-bit adder for signal <sym_pos$add0076> created
    at line 166.
```

```
    Found 34-bit adder for signal <sym_pos$add0077> created
         at line 166.
    Found 35-bit adder for signal <sym_pos$add0078> created
         at line 166.
    Found 36-bit adder for signal <sym_pos$add0079> created
         at line 166.
    Found 37-bit adder for signal <sym_pos$add0080> created
         at line 166.
    Found 33-bit adder for signal <sym_pos$add0081> created
         at line 166.
    Found 34-bit adder for signal <sym_pos$add0082> created
         at line 166.
    Found 35-bit adder for signal <sym_pos$add0083> created
         at line 166.
    Found 36-bit adder for signal <sym_pos$add0084> created
         at line 166.
    Found 37-bit adder for signal <sym_pos$add0085> created
         at line 166.
    Found 33-bit adder for signal <sym_pos$add0086> created
         at line 166.
    Found 34-bit adder for signal <sym_pos$add0087> created
         at line 166.
    Found 35-bit adder for signal <sym_pos$add0088> created
         at line 166.
    Found 36-bit adder for signal <sym_pos$add0089> created
         at line 166.
    Found 37-bit adder for signal <sym_pos$add0090> created
         at line 166.
    Found 33-bit adder for signal <sym_pos$add0091> created
         at line 166.
    Found 34-bit adder for signal <sym_pos$add0092> created
         at line 166.
    Found 35-bit adder for signal <sym_pos$add0093> created
         at line 166.
    Found 36-bit adder for signal <sym_pos$add0094> created
         at line 166.
    Found 37-bit adder for signal <sym_pos$add0095> created
         at line 166.
INFO:Xst:738 - HDL ADVISOR - 256 flip-flops were inferred
    for signal <sym>. You may be trying to describe a RAM in
     a way that is incompatible with block and distributed
    RAM resources available on Xilinx devices, or with a
    specific template that is not supported. Please review
    the Xilinx resources documentation and the XST user
    manual for coding guidelines. Taking advantage of RAM
    resources will lead to improved device usage and reduced
     synthesis time.
    Summary:
         inferred    1 ROM(s).
```

```
        inferred    1 Counter(s).
        inferred  277 D-type flip-flop(s).
        inferred  241 Adder/Subtractor(s).
        inferred   16 Comparator(s).
Unit <modem_rx_correlator> synthesized.


Synthesizing Unit <modem_rx_symbol_to_bit>.
    Related source file is "C:/PFC/modem_rx_symbol_to_bit.
       vhd".
WARNING:Xst:737 - Found 1-bit latch for signal <
    symbol_reg_0>. Latches may be generated from incomplete
    case or if statements. We do not recommend the use of
    latches in FPGA/CPLD designs, as they may lead to timing
     problems.
    Found 1-bit register for signal <bit_temp>.
    Found 3-bit register for signal <symbol_reg<3:1>>.
    Summary:
        inferred    4 D-type flip-flop(s).
Unit <modem_rx_symbol_to_bit> synthesized.


Synthesizing Unit <modem_tx>.
    Related source file is "C:/PFC/modem_tx.vhd".
Unit <modem_tx> synthesized.


Synthesizing Unit <modem_rx>.
    Related source file is "C:/PFC/modem_rx.vhd".
Unit <modem_rx> synthesized.


Synthesizing Unit <modem>.
    Related source file is "C:/PFC/modem.vhd".
Unit <modem> synthesized.

INFO:Xst:1767 - HDL ADVISOR - Resource sharing has
    identified that some arithmetic operations in this
    design can share the same physical resources for reduced
     device utilization. For improved clock frequency you
    may try to disable resource sharing.
```

10.3.6. HDL Synthesis Report

```
Macro Statistics
# ROMs                                                 : 1
 16x4-bit ROM                                          : 1
```

```
# Multipliers                                    : 12
 10x10-bit multiplier                           : 12
# Adders/Subtractors                             :
    442
 10-bit adder                                   :
    110
 10-bit addsub                                  : 76
 10-bit subtractor                              : 2
 20-bit adder                                   : 12
 3-bit adder                                    : 1
 32-bit adder                                   : 32
 33-bit adder                                   : 16
 34-bit adder                                   : 32
 35-bit adder                                   : 32
 36-bit adder                                   : 32
 37-bit adder                                   : 32
 38-bit adder                                   : 32
 39-bit adder                                   : 16
 40-bit adder                                   : 16
 5-bit adder                                    : 1
# Counters                                       : 6
 11-bit up counter                              : 1
 4-bit up counter                               : 2
 5-bit up counter                               : 1
 7-bit up counter                               : 1
 9-bit up counter                               : 1
# Accumulators                                   : 2
 10-bit up accumulator                          : 2
# Registers                                      : 83
 1-bit register                                 : 5
 10-bit register                                : 39
 16-bit register                                : 20
 2-bit register                                 : 14
 20-bit register                                : 2
 3-bit register                                 : 1
 4-bit register                                 : 2
# Latches                                        : 3
 1-bit latch                                    : 2
 4-bit latch                                    : 1
# Comparators                                    :
    101
 10-bit comparator greatequal                   : 12
 10-bit comparator greater                      : 4
 10-bit comparator less                         : 60
 11-bit comparator greatequal                   : 2
 11-bit comparator greater                      : 1
 11-bit comparator less                         : 1
 32-bit comparator greater                      : 16
 7-bit comparator greatequal                    : 2
```

```
 7-bit comparator less                                  : 1
 9-bit comparator greatequal                            : 1
 9-bit comparator less                                  : 1
# Multiplexers                                          : 2
 1-bit 8-to-1 multiplexer                               : 2
# Xors                                                  : 8
 1-bit xor2                                             : 8
```

10.3.7. Advanced HDL Synthesis

```
Loading device for application Rf_Device from file '4vlx15.
   nph' in environment C:\Xilinx\10.1\ISE.

Synthesizing (advanced) Unit <modem_rx_correlator>.
INFO:Xst - In order to maximize performance and save block
   RAM resources, the small ROM <
   Mrom_correlator_symbol_mux0000> will be implemented on
   LUT. If you want to force its implementation on block,
   use option/constraint rom_style.
Unit <modem_rx_correlator> synthesized (advanced).

Synthesizing (advanced) Unit <modem_rx_mfilter>.
        Multiplier <Mmult_acc_mult0005> in block <
           modem_rx_mfilter> and adder/subtractor <
           Madd_acc_add0001> in block <modem_rx_mfilter>
           are combined into a MAC<Maddsub_acc_mult0005>.
        The following registers are also absorbed by the
           MAC: <acc_mux0000> in block <modem_rx_mfilter>.
        Multiplier <Mmult_acc_mult0004> in block <
           modem_rx_mfilter> and adder/subtractor <
           Madd_acc_add0000> in block <modem_rx_mfilter>
           are combined into a MAC<Maddsub_acc_mult0004>.
        Multiplier <Mmult_acc_mult0003> in block <
           modem_rx_mfilter> and adder/subtractor <
           Madd_acc_addsub0003> in block <modem_rx_mfilter>
            are combined into a MAC<Maddsub_acc_mult0003>.
        Multiplier <Mmult_acc_mult0002> in block <
           modem_rx_mfilter> and adder/subtractor <
           Madd_acc_addsub0001> in block <modem_rx_mfilter>
            are combined into a MAC<Maddsub_acc_mult0002>.
        Multiplier <Mmult_acc_mult0001> in block <
           modem_rx_mfilter> and adder/subtractor <
           Madd_acc_addsub0000> in block <modem_rx_mfilter>
            are combined into a MAC<Maddsub_acc_mult0001>.
INFO:Xst:2385 - HDL ADVISOR - You can improve the
   performance of the multiplier Mmult_acc_mult0000 by
   adding 2 register level(s).
```

```
INFO:Xst:2385 - HDL ADVISOR - You can improve the
    performance of the multiplier Mmult_acc_mult0000 by
    adding 2 register level(s).
Unit <modem_rx_mfilter> synthesized (advanced).
WARNING:Xst:2677 - Node <reg_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_10_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_10_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_10_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_11_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_11_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_11_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_12_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_12_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_12_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_13_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_13_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_13_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_14_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_14_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_14_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_15_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_15_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_15_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_8_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_8_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_8_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
```

```
WARNING:Xst:2677 - Node <sym_9_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_9_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_9_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_0_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_0_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_0_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_1_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_1_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_1_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_2_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_2_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_2_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_3_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_3_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_3_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_4_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_4_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_4_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_5_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_5_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_5_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_6_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_6_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_6_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
```

```
WARNING:Xst:2677 - Node <sym_7_13> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_7_14> of sequential type is
    unconnected in block <modem_rx_correlator>.
WARNING:Xst:2677 - Node <sym_7_15> of sequential type is
    unconnected in block <modem_rx_correlator>.
```

10.3.8. Advanced HDL Synthesis Report

```
Macro Statistics
# ROMs                                          : 1
 16x4-bit ROM                                   : 1
# MACs                                          : 10
 10x10-to-20-bit MAC                            : 10
# Multipliers                                   : 2
 10x10-bit multiplier                           : 2
# Adders/Subtractors                            :
    400
 10-bit adder                                   :
    110
 10-bit addsub                                  : 76
 10-bit subtractor                              : 2
 20-bit adder                                   : 2
 3-bit adder                                    : 1
 32-bit adder                                   : 64
 32-bit adder carry in                          : 32
 33-bit adder                                   : 16
 34-bit adder                                   : 32
 35-bit adder                                   : 32
 36-bit adder                                   : 16
 37-bit adder                                   : 16
 5-bit adder                                    : 1
# Counters                                      : 6
 11-bit up counter                              : 1
 4-bit up counter                               : 2
 5-bit up counter                               : 1
 7-bit up counter                               : 1
 9-bit up counter                               : 1
# Accumulators                                  : 2
 10-bit up accumulator                          : 2
# Registers                                     :
    705
 Flip-Flops                                     :
    705
# Latches                                       : 3
 1-bit latch                                    : 2
 4-bit latch                                    : 1
```

```
# Comparators                                    :
    101
 10-bit comparator greatequal                    : 12
 10-bit comparator greater                       : 4
 10-bit comparator less                          : 60
 11-bit comparator greatequal                    : 2
 11-bit comparator greater                       : 1
 11-bit comparator less                          : 1
 32-bit comparator greater                       : 16
 7-bit comparator greatequal                     : 2
 7-bit comparator less                           : 1
 9-bit comparator greatequal                     : 1
 9-bit comparator less                           : 1
# Multiplexers                                   : 2
 1-bit 8-to-1 multiplexer                        : 2
# Xors                                           : 8
 1-bit xor2                                      : 8
```

10.3.9. Low Level Synthesis

```
INFO:Xst:2261 - The FF/Latch <sym_6_0> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <chip_reg_13>
INFO:Xst:2261 - The FF/Latch <sym_1_0> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <chip_reg_3>
INFO:Xst:2261 - The FF/Latch <sym_2_0> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <chip_reg_5>
INFO:Xst:2261 - The FF/Latch <sym_4_0> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <chip_reg_9>
INFO:Xst:2261 - The FF/Latch <sym_1_1> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <chip_reg_4>
INFO:Xst:2261 - The FF/Latch <sym_4_1> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <chip_reg_10>
INFO:Xst:2261 - The FF/Latch <sym_1_2> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_0>
INFO:Xst:2261 - The FF/Latch <sym_4_2> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_0>
INFO:Xst:2261 - The FF/Latch <sym_1_3> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_1>
```

```
INFO:Xst:2261 - The FF/Latch <sym_4_3> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_1>
INFO:Xst:2261 - The FF/Latch <sym_1_4> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_2>
INFO:Xst:2261 - The FF/Latch <sym_4_4> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_2>
INFO:Xst:2261 - The FF/Latch <sym_1_5> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_3>
INFO:Xst:2261 - The FF/Latch <sym_4_5> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_3>
INFO:Xst:2261 - The FF/Latch <sym_1_6> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_4>
INFO:Xst:2261 - The FF/Latch <sym_4_6> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_4>
INFO:Xst:2261 - The FF/Latch <sym_1_7> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_5>
INFO:Xst:2261 - The FF/Latch <sym_4_7> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_5>
INFO:Xst:2261 - The FF/Latch <sym_1_8> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_6>
INFO:Xst:2261 - The FF/Latch <sym_4_8> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_6>
INFO:Xst:2261 - The FF/Latch <sym_1_9> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_7>
INFO:Xst:2261 - The FF/Latch <sym_4_9> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_7>
INFO:Xst:2261 - The FF/Latch <sym_1_10> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_8>
INFO:Xst:2261 - The FF/Latch <sym_4_10> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_8>
INFO:Xst:2261 - The FF/Latch <sym_4_11> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_9>
```

```
INFO:Xst:2261 - The FF/Latch <sym_1_11> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_9>
INFO:Xst:2261 - The FF/Latch <sym_4_12> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_5_10>
INFO:Xst:2261 - The FF/Latch <sym_1_12> in Unit <
    modem_rx_correlator> is equivalent to the following FF/
    Latch, which will be removed : <sym_2_10>

Optimizing unit <modem> ...

Optimizing unit <modem_tx_controller> ...

Optimizing unit <modem_tx_mfilter> ...

Optimizing unit <modem_rx_coarse_freq_estimator> ...

Optimizing unit <modem_rx_mfilter> ...

Optimizing unit <modem_rx_fine_freq_estimator> ...

Optimizing unit <modem_rx_correlator> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block modem,
    actual ratio is 30.
FlipFlop M_RX/DOW_i/downsampler_output_4 has been
    replicated 1 time(s)
FlipFlop M_RX/DOW_i/downsampler_output_5 has been
    replicated 1 time(s)
FlipFlop M_RX/DOW_i/downsampler_output_9 has been
    replicated 3 time(s)
FlipFlop M_RX/DOW_q/downsampler_output_4 has been
    replicated 1 time(s)
FlipFlop M_RX/DOW_q/downsampler_output_9 has been
    replicated 2 time(s)
FlipFlop M_RX/DOW_i/downsampler_output_9 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_8 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_7 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_6 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_5 has been
    replicated 1 time(s) to handle iob=true attribute.
```

```
FlipFlop M_RX/DOW_i/downsampler_output_4 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_3 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_2 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_1 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_i/downsampler_output_0 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_9 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_8 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_7 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_6 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_5 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_4 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_3 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_2 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_1 has been
    replicated 1 time(s) to handle iob=true attribute.
FlipFlop M_RX/DOW_q/downsampler_output_0 has been
    replicated 1 time(s) to handle iob=true attribute.

Final Macro Processing ...

Processing Unit <modem> :
        Found 4-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_10_2>.
        Found 4-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_10_0>.
        Found 4-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_10_3>.
        Found 4-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_10_5>.
        Found 9-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_1_0>.
        Found 4-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_4_1>.
        Found 4-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_4_4>.
```

```
        Found 4-bit shift register for signal <M_RX/FR_FINE
            /phase_table_0_4_6>.
Unit <modem> processed.
```

10.3.10. Final Register Report

```
Macro Statistics
# Registers                                          :
    728
 Flip-Flops                                         :
     728
# Shift Registers                                   : 8
 4-bit shift register                               : 7
 9-bit shift register                               : 1
```

10.3.11. Final Report

```
Final Results
RTL Top Level Output File Name     : modem.ngr
Top Level Output File Name         : modem
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 162

Cell Usage :
# BELS                             : 4995
#      BUF                         : 1
#      GND                         : 1
#      INV                         : 34
#      LUT1                        : 121
#      LUT2                        : 373
#      LUT2_D                      : 22
#      LUT2_L                      : 42
#      LUT3                        : 774
#      LUT3_D                      : 51
#      LUT3_L                      : 41
#      LUT4                        : 1571
#      LUT4_D                      : 159
#      LUT4_L                      : 90
#      MULT_AND                    : 12
#      MUXCY                       : 770
#      MUXF5                       : 189
```

```
#       MUXF6                             : 3
#       VCC                               : 1
#       XORCY                             : 740
# FlipFlops/Latches                       : 767
#       FDC                               : 168
#       FDCE                              : 365
#       FDCPE                             : 3
#       FDE                               : 66
#       FDE_1                             : 48
#       FDPE                              : 103
#       FDRE                              : 8
#       LD                                : 4
#       LDC                               : 1
#       LDE                               : 1
# Shift Registers                         : 8
#       SRL16E                            : 8
# Clock Buffers                           : 3
#       BUFGP                             : 3
# IO Buffers                              : 159
#       IBUF                              : 34
#       OBUF                              : 125
# DSPs                                    : 12
#       DSP48                             : 12
=========================================================================
```

10.3.12. Device utilization summary

```
Selected Device : 4vlx15sf363-12

 Number of Slices:                     1772  out of   6144
       28%
 Number of Slice Flip Flops:            747  out of  12288
        6%
 Number of 4 input LUTs:               3286  out of  12288
       26%
    Number used as logic:              3278
    Number used as Shift registers:       8
 Number of IOs:                         162
 Number of bonded IOBs:                 162  out of    240
       67%
    IOB Flip Flops:                     20
 Number of GCLKs:                        3  out of     32
        9%
 Number of DSP48s:                      12  out of     32
       37%
```

10.3.13. Timing Summary

```
Speed Grade: -12

   Minimum period: 36.776ns (Maximum Frequency: 27.192MHz)
   Minimum input arrival time before clock: 26.266ns
   Maximum output required time after clock: 6.670ns
   Maximum combinational path delay: 8.680ns

Number of errors   :    0 (   0 filtered)
Number of warnings :   71 (   0 filtered)
Number of infos    :   51 (   0 filtered)
```

Capítulo 11

Conclusiones y trabajo futuro

En este proyecto se ha realizado un modelo VHDL sintetizable de un módem digital siguiendo las indicaciones del estándar IEEE 802.15.4 (ZigBee). ZigBee es en la actualidad el estándar dominante para redes inalámbricas personales de corto alcance y baja velocidad (LR-WPAN) y cuenta con numerosas aplicaciones dentro del campo de la automatización industrial y la domótica, en este último caso, habiendo desbancado a estándares anteriores como HOME-RF (Puede encontrarse documentación técnica del grupo, aunque disuelto, en [15]).

Lo que hace a ZigBee tan atractivo son sus costes y consumos extremadamente bajos, debido en parte a la posibilidad de ralizar los procesos de modulación y demodulación completamente en digital, destinando a la parte analógica sólo las tareas de recepción, transmisión y conversión en banda.

ZigBee ha despertado el interés de muchos productores de hardware y software como Motorola, Texas Instruments, Atmel, Microchip o Ember, que ofrecen muchísimas soluciones. Sin embargo, a pesar de ser ZigBee un estándar abierto, estas soluciones se basan en hardware propietario y por tanto podrían no ser idóneas en aplicaciones donde son necesarias soluciones a medida para cumplir con algunas restricciones de diseño. La solución a este problema pasa por pagar los derechos de explotación de propiedad intelectual y utilizar un softcore o hardcore ZigBee proporcionado por terceros para desarrollar una nueva solución hardware. Este método es sin duda el más rápido, pero no el más económico sobre todo si se piensa que ZigBee es un estándar abierto, por lo que sus especificaciones son sabidas, y que existen en el mercado herramientas de desarrollo (basadas en lenguajes de descripción de hardware,y que usan FPGAS como tecnología objetivo) de bajo coste que permitirían un desarrollo de un prototipo en un periodo de

tiempo relativamente corto.

De la misma forma, utilizar una descripción RTL o de comportamiento escrita en un lenguaje de descripción de hardware como VHDL para generar un softcore del módem, tiene el valor añadido de poder escoger la tecnología objetivo, ya que permite decidir entre matrices programables como FPGAs o soluciones parcialmente a medida basadas en celdas estándar de una librería.

El trabajo desarrollado en este proyecto representa una contribución muy valiosa ya que, debidamente mejorada, puede ser el núcleo de una serie de productos basados en uno de los estándares dominantes y de más amplia aceptación en el mercado. Para que esto sea posible, de cara al futuro sería necesario aportar las siguientes mejoras al modelo desarrollado:

Implementación del sistema de RSSI

Como se describía en el Capítulo 2, el sistema indicador de nivel de señal o RSSI (Received Signal Strength Indicator) es una estimación de la potencia de la señal recibida dentro del ancho de banda de un canal dado. Podemos calcular este valor midiendo la energía de las muestras entrantes y acumulándola durante un periodo de tiempo determinado. Calcularemos la potencia dividiendo la energía acumulada por el número (N) de muestras analizadas. Así, la potencia de la señal vendrá dada por:

$$RSSI = \frac{1}{N}\sum_{n=0}^{N-1}|m(n)|^2 = \frac{1}{N}\sum_{n=0}^{N-1}s_I^2(n) + s_Q^2(n)$$

donde el módulo de $m(n)$ es el módulo de las muestras complejas entrantes moduladas en cuadratura de fase y N es el número de muestras medidas. El RSSI puede implementarse mediante el CORDIC en modo vectorial que nos permite convertir un valor complejo de forma cartesiana a forma polar (módulo, fase). Así, solo tenemos que ir acumulando el módulo de las muestras entrantes (calculado con el CORDIC rotacional) durante N muestras y al final, dividir por las N muestras analizadas. Esta división puede implementarse mediante un desplazamiento a la derecha si el número N de muestras es una potencia de 2.

Además, el CCA o *Clear Channel Assessment* y el indicador de la calidad del enlace (LQI) pueden implementarse a partir del RSSI.

Mejoras en los controladores de modulación/demodulación

Actualmente, los controladores de modulación y demodulación dependen de señales externas para iniciar y terminar su funcionamiento:

start_tx y *start_rx*. La necesidad de tener un componente externo que controle ambas señales puede eliminarse utilizando el campo *length* de la PDU de ZigBee. Al iniciar la modulación, se inicializaría un conta-

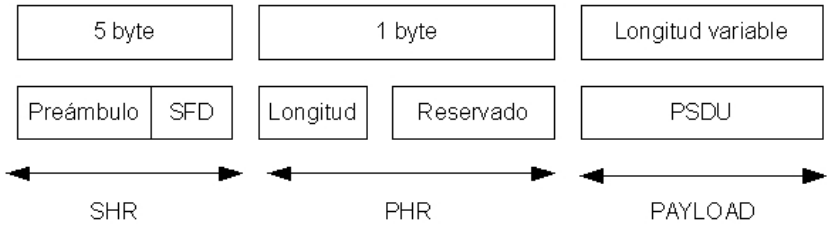

Figura 11.1: Detalle de la PDU del estándar 802.15.4

dor con el valor del campo *length* y el proceso no terminaría hasta que el contador se decrementase hasta 0. De la misma manera, en el proceso de demodulación, una vez detectado el campo *length* se inicializaría otro contador, y así, decrementándolo en cada bit recibido, se podría controlar la recepción total del paquete sin necesidad de contar con una unidad de control externa.

Bibliografía

[1] Roger Martinsen Koteng. *Evaluation of SDR-implementation of IEEE 802.15.4 Physical Layer.* 2006.

[2] ZigBee Alliance. *ZigBee Specification.* 2008.

[3] IEEE Standard for Information Technology. *Part 15.4: Wireless Medium Access Control and Physical Layer Specifications for Low-Rate Wireless Personal Area Networks.* 2006.

[4] Volnei A. Pedroni. *Circuit Design with VHDL.* MIT Press, Cambridge, Massachusetts, London, England, 2004.

[5] Pong P. Chu. *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version.* Wiley-Interscience, United States of America, 2008.

[6] Bernard Sklar. *Digital Communications. Fundamentals and Applications.* Prentice Hall, Upper Saddle River, NJ, 2001.

[7] Steven Kay. *A Fast and Accurate Single Frequency Estimator.* IEEE Trans. Acoust. Speech Signal Process. v37 i12. 1987-1990.

[8] P.J. Kootsookos. *A Review of the Frequency Estimation and Tracking Problems.* Systems Engineering Department, Australian National University, 1993.

[9] Jack E. Volder. *The CORDIC Trigonometric Computing Technique, IRE Transactions on Electronic Computers.* 1959.

[10] J. S. Walther. *The Story of Unified CORDIC, VLSI Signal Processing 25, 107.* 2000.

[11] Andraka, Ray. *A survey of CORDIC algorithms for FPGA based computers.*

[12] Richard Herveille. *CORDIC Core. OpenCores.* www.opencores.org, 2001.

[13] G. Cornetta. *Technical Report: Design and Implementation of a 802.15.4-Compilant Modem.* 2008.

[14] B. Razavi. *RF Microelectronics* Prentice Hall, Upper Saddle River, NJ, 1998.

[15] HomeRF Archive. *http://www.cazitech.com/HomeRF_Archives.htm.*