# Tesis de Grado

Módulos de codificación y modulación OFDM de señales digitales implementados en FPGA

Laboratorio de Comunicaciones

Facultad de Ingeniería

Universidad Nacional de Mar del Plata

Alumno: Alejandro Martín Wechsler

Director: Dr. Ing. Jorge Castiñeira Moreira

Codirectora: Mg. Ing. Mónica Cristina Liberatori

# 1. Tabla de contenidos

## Contenido Página

# 2. Introducción

Las comunicaciones digitales son el principal medio de comunicación en la industria, en la investigación, y hasta en la vida cotidiana. Año tras año vemos como lo digital desplaza a lo analógico. Esto se debe, en parte, debido a la simpleza (y a la vez robustez) de diseñar sistemas digitales, independizándose de ciertas problemáticas existentes en el mundo analógico. Ejemplo de esto es la comunicación con teléfonos celulares. Hace ya aproximadamente 20 años, los celulares abandonaron las viejas comunicaciones analógicas, en favor de nuevas tecnologías, como CDMA, y actualmente, GSM. Ambas CDMA y GSM son comunicaciones de índole digital.

Dos procesos indispensables para la transmisión digital de señales son la codificación y la modulación de las mismas. Actualmente, estos procesos se realizan con placas especializadas, poco versátiles, de costo creciente con la complejidad de las mismas, y que muchas veces no pueden conseguirse en el mercado de componentes electrónicos.

En cuanto a la modulación OFDM en particular, podemos decir que se trata de una técnica de modulación en la cual una señal digital de entrada se multiplexa en distintas líneas paralelas, donde la velocidad de cada una de ellas es menor a la de la señal original (N veces menor, siendo N la cantidad de líneas en las que se separa). Luego estas “subseñales” son moduladas en amplitud y/o fase (por ejemplo, con BPSK, QAM, etc.), para luego ser montadas en distintas subportadoras de frecuencias muy cercanas unas a las otras (en una banda de frecuencias elevada). Si se logra que las frecuencias de las subportadoras sean ortogonales entre sí, estas no se interferirán unas a las otras, resultando en una velocidad de envío de información elevada, en un ancho de banda significativamente menor al que se necesitaría si dichas subseñales fueran separadas por su ancho de banda completo. Este proceso puede verse en un diagrama en bloques como el que se presenta en la Figura 2-1.

Como la multiplicación de las señales por osciladores (mezclado) es una operación que tecnológicamente presenta diversos contratiempos (necesidad de mucha circuitería externa al FPGA, alinealidad de los componentes usados, aparición de armónicas indeseadas, necesidad de implementar filtros, entre otros), se reemplaza esta parte del diagrama por la operación de IFFT, que resulta en una señal análoga a usar los mezcladores.

A lo anterior se le agrega el concepto de redundancia (codificación), donde se agregan bits sin información adicional, pero que mejoran cierto aspecto de la comunicación, como puede ser el agregado de la capacidad de corregir (o solo detectar) errores, o mejorar la probabilidad binaria de error en un medio en especial. En nuestro

caso, el medio será el aire, por lo se analizará qué técnica de codificación será la que resulte óptima.

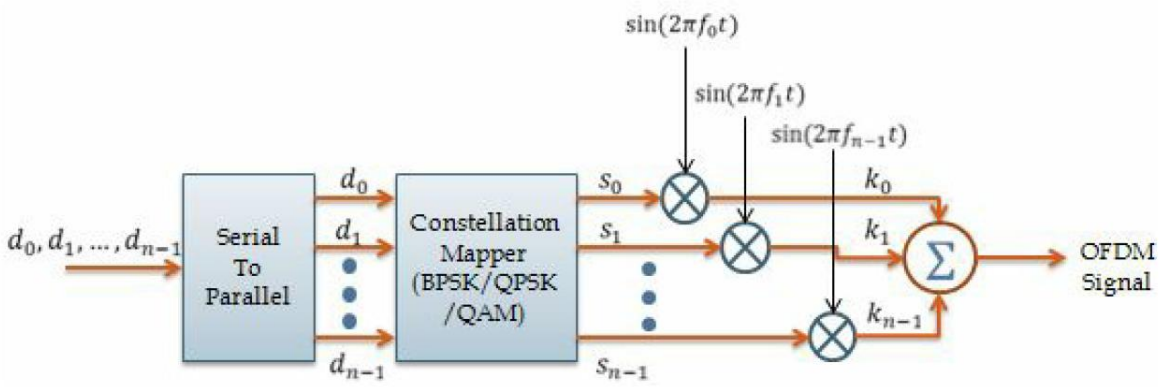



*Figura 2-1:* Diagrama en bloques simplificado de modulación OFDM

En el receptor, deberán hacerse las operaciones inversas a las realizadas en el transmisor. Un diagrama en bloques genérico del sistema completo puede verse en la Figura 2-2.

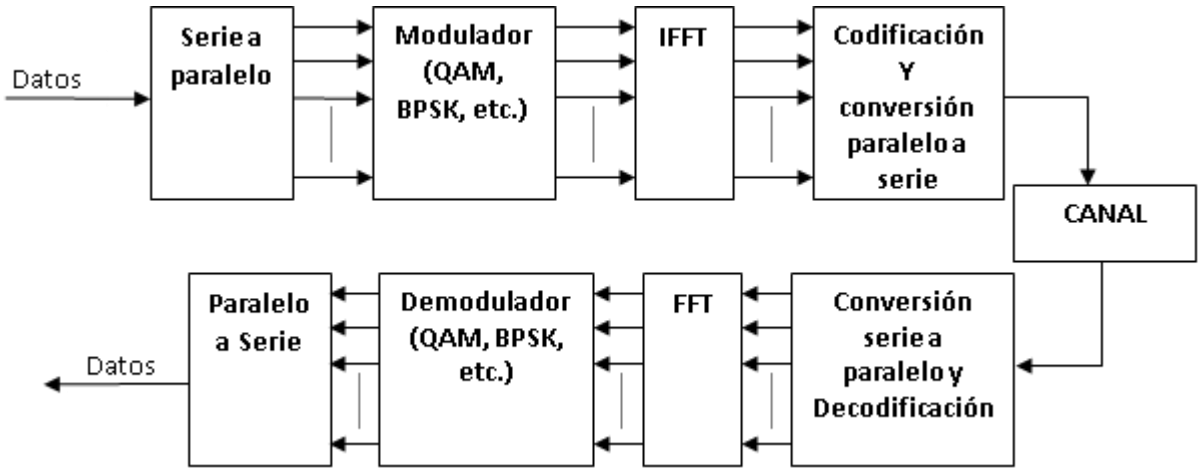



*Figura 2-2:* Diagrama en bloques de sistema de comunicación completo OFDM

La implementación física de este sistema en placas con microcontroladores puede significar un tiempo de procesamiento largo, lo que generaría una latencia incompatible para aplicaciones de tiempo real. Este problema se soluciona usando la tecnología FPGA, donde las operaciones se traducen en circuitos electrónicos de alta velocidad, donde el limitante son las mismas compuertas. Además, al tratarse de un circuito electrónico, es capaz de realizar operaciones en paralelo, siempre y cuando el algoritmo lo permita.

Las placas FPGA son programas con un lenguaje llamado VHDL, del tipo de lenguajes de descripción de hardware. Luego, un software propietario, propio del

fabricante de la placa FPGA, compila el programa, y genera un archivo de programación que, al ser trasladado a la placa de desarrollo, sintetiza un circuito digital que realiza las operaciones necesarias.

# 3. Objetivos

## Objetivos generales

- Reducción de costos en el laboratorio.
- Sumar conocimientos y experiencia en lenguaje VHDL.
- Sumar conocimientos y experiencia en uso de placas FPGA.
- Inserción en el ambiente laboral académico, en un laboratorio.

## Objetivos particulares

- Introducción y estudio de la temática de codificación y modulación.
- Introducción y desarrollo en la tecnología FPGA.
- Desarrollo de módulos de codificación y modulación de señales digitales, usando técnicas de modulación multiportadora.
- Divulgación y publicación de resultados.

# 4. Marco Teórico

Es este apartado, se realizará una introducción a la teoría y conceptos necesarios para poder comprender las ventajas del sistema de comunicaciones OFDM, así como los componentes que lo conforman. Es necesario entonces tener una idea sobre los distintos esquemas de modulación y codificación, y las características de cada uno. A partir de esto, se podrá tener una base para comparar los distintos modelos.

## Elementos de las comunicaciones

La idea detrás de todo esquema de comunicaciones es el de poder transmitir una señal a través de un medio (el aire, por ejemplo), y recibirla satisfactoriamente (tener una representación fiel de la señal transmitida) en otro extremo, usando la mínima cantidad de recursos del medio como sea posible, y teniendo las mínimas perdidas como sea posible. La elección del esquema a utilizar depende de varios factores, como: la facilidad y/o costo de implementación, la inmunidad al ruido que ofrece, la velocidad de transmisión que logra realizar, entre otros. A continuación, listaremos una serie de definiciones para poder comprender a qué nos referimos cada vez que nombramos un concepto.

Mensaje: Es la representación física de la información, producida en la fuente. El objetivo es reproducir en el destino una réplica aceptable del mensaje en la fuente, sin importar su formato.

El mensaje puede ser de naturaleza analógica, si es una cantidad física que varía con el tiempo, usualmente de forma continua; o digital, si es una secuencia ordenada de símbolos, seleccionados de un campo con una cantidad finita de elementos.

En este trabajo, nos concentraremos en mensajes digitales. Nuestro mensaje será una secuencia de unos y ceros, que se agruparán y serán interpretados de manera de poder trabajar sobre ellos.

Señal modulante: Es la representación eléctrica del mensaje. Puede ser generada a partir del mismo, usando un transductor. De igual manera, otro transductor es capaz de generar el mensaje, a partir de la señal modulante. Este proceso puede verse en la Figura 4-1. En este proyecto no haremos distinción entre mensaje y señal modulante, ya que adoptaremos que nuestra fuente genera secuencias de unos y ceros eléctricos.

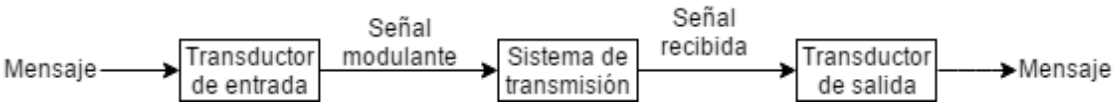



*Figura 4-1:* Proceso de uso de transductores en un sistema de transmisión

Fuente: Es el dispositivo que genera un mensaje (en forma de señal), el cual deberá ser transmitido, a través de un medio, hacia un destino.

Transmisor: Procesa la señal modulante para producir otra señal apta para ser enviada en un canal de transmisión. El procesamiento de la señal para transmisión incluye la modulación y, ocasionalmente, codificación.

Destino: Es el dispositivo que recibe el mensaje transmitido por la fuente.

Receptor: Opera sobre la señal recibida del canal, y la prepara para ser entregada al destino. Las operaciones realizadas incluyen prioritariamente: amplificación, para compensar por pérdidas en la transmisión; y demodulación y decodificación, para obtener el mensaje transmitido. En este trabajo nos enfocaremos en la demodulación de señales.

Señal portadora: Esta señal depende del sistema de comunicaciones que se esté usando, y cambiará alguna de sus características (amplitud, frecuencia o fase, en la mayoría de los casos), dependiendo de la señal modulante. Esta señal es apta para ser transmitida por un medio específico, y los cambios en la misma son luego interpretados en la recepción, para poder recuperar la señal modulante. A este proceso se lo llama modulación y demodulación, respectivamente.

Medio de transmisión: También conocido como “canal”, es el medio eléctrico que une a la fuente, del destino. Ejemplos de esto son: un par de cables, un cable coaxial, un láser, o una onda de radio. Todo canal introduce alguna cantidad de pérdida, o atenuación, provocando que la potencia de la señal decrezca con la distancia.

En la Figura 4-2, se presenta un diagrama en bloques donde puede verse claramente como los elementos típicos de un sistema de transmisión interactúan entre sí. En este trabajo, nos concentramos en una parte específica del transmisor y receptor, la modulación y demodulación, respectivamente.

## Distorsión, ruido e interferencia

Varios efectos no deseados se encuentran a la hora de transmitir una señal. La atenuación es indeseable ya que reduce la potencia de la señal en el receptor. Sin

embargo, también deberemos preocuparnos por la distorsión, la interferencia, y el ruido, que alteran la forma de la señal. Aunque estas contaminaciones pueden ocurrir en cualquier momento, la convención es de culpar enteramente al canal, tratando a los transmisores y receptores como ideales.

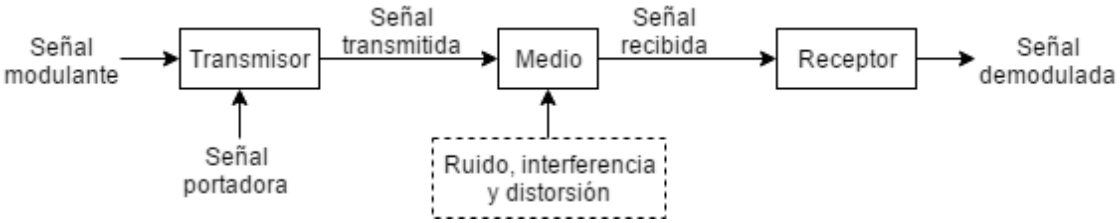
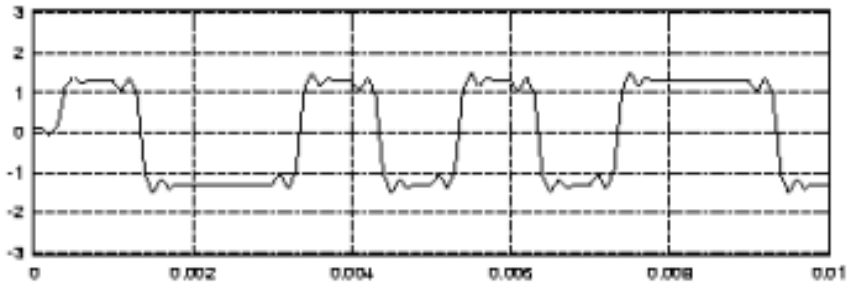
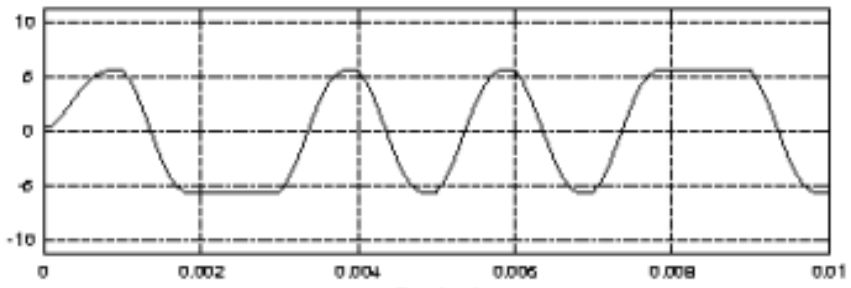



*Figura 4-2:* Esquema que muestra los elementos típicos de un sistema de transmisión.

La distorsión es una perturbación en la señal transmitida, causada por imperfecciones en el medio. A diferencia de la interferencia y el ruido, la distorsión desaparece cuando no hay señal. Si el canal tiene una respuesta lineal en frecuencia, pero distorsiva, entonces la distorsión puede ser corregida, o al menos reducida, con la ayuda de filtros especiales, llamados ecualizadores. Un ejemplo de distorsión puede verse en la Figura 4-3. En 4-3a puede verse una señal transmitida de forma cuadrada, a un medio cuya respuesta al impulso no es constante. En 4-3b podemos ver que la señal recibida mantiene de cierta manera la forma de la señal transmitida, pero con sus bordes recortados, y saturada a valores máximo y mínimo. Esto se debe a que la señal fue distorsionada por el medio.




a_ Señal transmitida a medio distorsivo




b_ Señal recibida, distorsionada por el medio

*Figura 4-3:* Ejemplo de señal enviada por un medio, cuya respuesta al impulso se asemeja a un filtro pasa-bajos.

La interferencia es la contaminación de la señal transmitida por señales externas,

producidas por fuentes humanas, tales como otros transmisores, la línea eléctrica, entre otros. Ocurre a menudo en sistemas de radio cuyas antenas receptoras captan varias señales al mismo tiempo. Un ejemplo de interferencia puede verse en la Figura 4-4. En 4-4a puede verse la señal transmitida al medio, de forma senoidal. En 4-4b se ve una señal interferente, inmersa en el medio. Esta señal se suma a la señal transmitida, por lo que se recibe una señal como la vista en 4-4c.

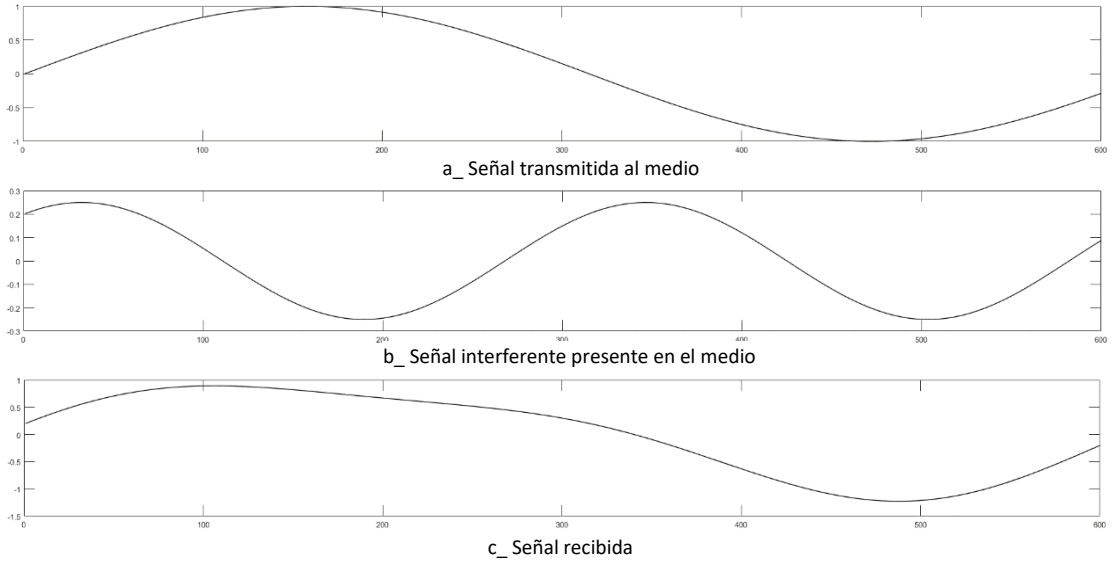



*Figura 4-4:* Ejemplo de recepción de señal transmitida, cuando se encuentra una señal interferente en el medio

El ruido se refiere a señales eléctricas aleatorias e impredecibles, generadas por procesos naturales, tanto internamente como externamente al sistema. Cuando estas señales aleatorias se superponen con la señal que contiene información, el mensaje puede ser parcialmente corrompido, o hasta totalmente destruido. El uso de filtros puede reducir el ruido, pero es imposible eliminarlo completamente. En la Figura 4-5 puede verse como el ruido afecta a la señal transmitida. En 4-5a puede verse una señal senoidal siendo transmitida a un medio que tiene ruido presente, como puede verse en 4-5b. Esto ocasiona que la señal recibida se vea afectada por el ruido, siendo la señal recibida como en 4-5c.

## Fading

El fading es un problema que surge cuando el medio es no guiado (el aire, por ejemplo). En este tipo de medios, la presencia de objetos reflectores en el entorno donde se encuentran el transmisor y el receptor implica la existencia de múltiples caminos para la propagación de la señal. En el receptor, este efecto se traduce en una superposición de copias de la señal transmitida, cada una arribando a través de un camino diferente. Esta superposición de la señal puede ocasionar nulos de señal o efectos de

amplificación. Los nulos se conocen como efecto de desvanecimiento o fading.

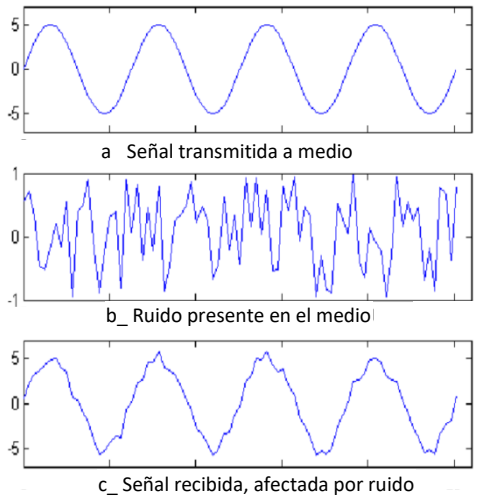



*Figura 4-5:* Ejemplo de cómo el ruido puede afectar a una señal transmitida a un medio

## Fading en frecuencia

El perfil de intensidad multipath $S(\tau)$ se refiere a la variación de la potencia promedio recibida a partir de un impulso transmitido, en función del retardo de propagación que excede al retardo del arribo de la primera señal en el receptor. A partir de este perfil es posible definir el retardo en exceso $T_m$, como el intervalo de tiempo total durante el cual llegan reflexiones con contenido de energía significativo. En un sistema ideal, $S(\tau)$ sería un impulso, ya que la señal llegaría en un solo instante, sin producirse reflexiones ni distintos retrasos.

Para evitar problemas relacionados con este suceso, es deseable que la duración de un símbolo $T_s$ en el medio sea lo suficientemente mayor al retardo en exceso. En este caso, las componentes multipath arribarían al receptor dentro de la duración del símbolo, sin provocar interferencias con símbolos adyacentes. Caso contrario, se dice que el canal exhibe fading selectivo en frecuencia. Es posible aplicar diferentes técnicas para mitigar este efecto, ya que es posible discriminar las componentes multipath del lado receptor.

En la Figura 4-6 puede verse un ejemplo de perfil multipath, así como su contraparte en el dominio frecuencial, conocida como función correlación de frecuencia espaciada $|R(\Delta f)|$.

A partir de la función correlación de frecuencia espaciada puede definirse el ancho de banda de coherencia $f_0$, que es una medida estadística del rango de frecuencias sobre el cual el canal permite el paso de componentes aproximadamente

con la misma ganancia y fase lineal. Así, todas las componentes de una señal dentro de esta banda sufrirán desvanecimiento en simultáneo o no lo sufrirán. Se verifica la relación $f_0 \cong \frac{1}{T_m}$. Análogamente al análisis temporal, se dice que un canal es selectivo en frecuencia si $f_0 < \frac{1}{T_s} \sim W$, siendo $W$ el ancho de banda de la señal.

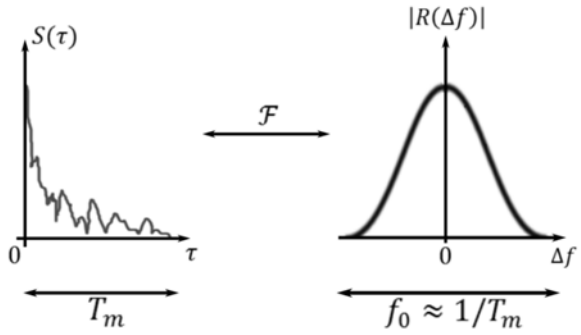



*Figura 4-6:* Ejemplo de perfil multipath y función correlación de frecuencia espaciada

En el caso que $f_0 > W$, el fading es no selectivo en frecuencia, y se le llama flat fading. No se introduce interferencia intersimbólica ISI (en inglés, Inter Symbol Interference), pero genera una degradación de la relación señal a ruido SNR. De esta manera, $f_o$ fija un límite superior en la velocidad de transmisión a la que se podría llegar sin necesidad de ecualizadores.

## Fading por variación en el tiempo

El movimiento del transmisor y receptor, así como el de los objetos en el entorno de los mismos, generan que el medio en el que se envía la señal sufra cambios (será variante en el tiempo). El hecho de que sea variante en el tiempo es lo mismo que decir que es variante en el espacio. Se define la función correlación en el tiempo espaciado $R(\Delta t)$, que especifica la correlación entre la respuesta del canal a una sinusoide transmitida en un instante y la respuesta a la misma sinusoide transmitida un tiempo $\Delta t$ posterior. Un ejemplo de la misma puede verse en la Figura 4-7. Puede verse además su contraparte frecuencial, la densidad espectral de potencial Doppler $S(v)$. El tiempo de coherencia $T_0$ es una medida del tiempo en el que el canal permanece invariable. Este parámetro da una idea de la rapidez del fading.

Cuando $T_0 < T_s$, estamos en presencia de fast fading. Es decir, el canal cambia en el tiempo que un símbolo es enviado. Esta situación debe ser evitada, ya que genera distorsión que a su vez provoca una degradación de la SNR, afectando a la probabilidad binaria de error. A su vez, cuando $T_s < T_0$, se está en presencia de slow fading, y la distorsión característica del fast fading no se hace presente. De esta manera, vemos

que el ancho de banda de fading $f_d \cong \frac{1}{T_0}$ impone un límite inferior a la velocidad de transmisión a la que nuestro sistema debe trabajar, para que el medio no presente fast fading.

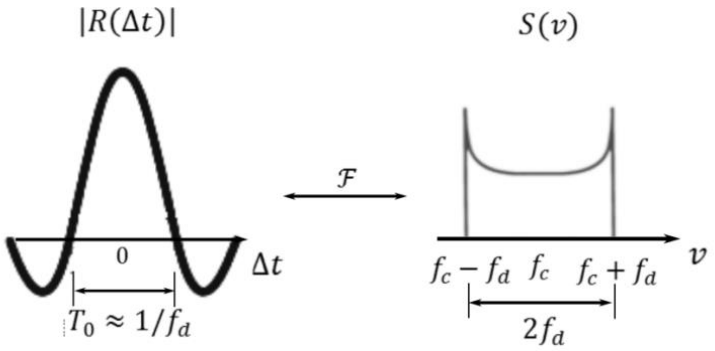



*Figura 4-7:* Función correlación en el tiempo espaciado y densidad espectral de potencia Doppler

## Señales pasabanda

Consideramos que una señal es pasabanda si se cumple la siguiente propiedad:

$$V_{bp}(f) = 0 \; si \; |f| < f_c - W \; y \; |f| > f_c + W$$

Es decir, la señal no tiene contenido espectral fuera de la banda de ancho $2W$, centrada en $f_c$. En la Figura 4-8 puede verse una señal pasabanda. En 4-8a puede verse un espectro pasabanda $V_{bp}(f)$. Su dual en el tiempo puede verse en 4-8b.

Podemos expresar una señal pasabanda como:

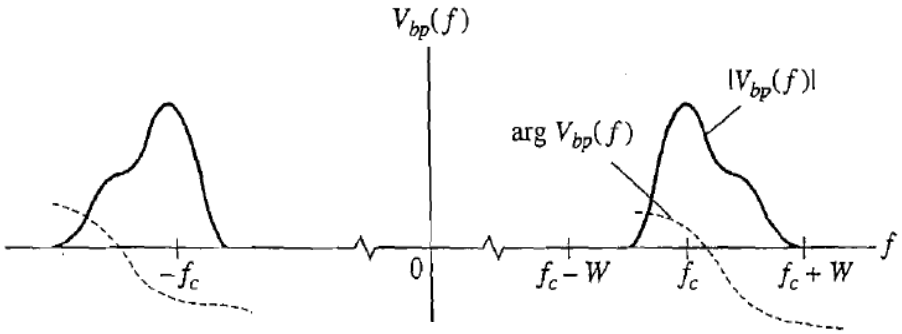
$$v_{bp}(t) = A(t).\cos[w_c.t + \emptyset(t)]$$

Esta es la descripción de envolvente y fase, donde $A(t)$ es la envolvente, y $\emptyset(t)$ es la fase, ambas función del tiempo. En AM, por ejemplo, se nota que $A(t) > 0$. De suceder amplitudes negativas, estas se verán reflejadas como un cambio de fase de $\pm 180°$.

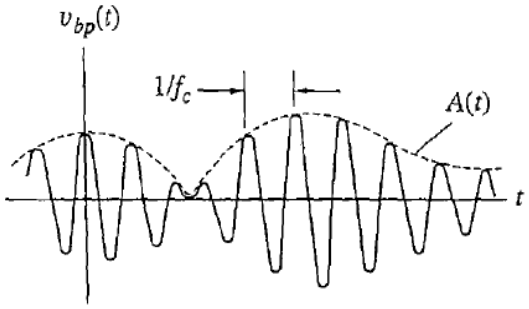
Podemos interpretar también a esta señal como un vector de amplitud $A(t)$ y fase $w_c.t + \emptyset(t)$. Esto puede verse en la *Figura 4-9*. Omitimos la velocidad angular $w_c$, ya que sabemos que la señal real será el producto de rotar este fasor con una velocidad angular $w_c$, en sentido anti horario.

Así, podemos describir la señal como:

$$v_{bp}(t) = v_i(t).\cos(w_c.t) - v_q(t).\text{sen}(w_c.t) =$$
$$= v_i(t).\cos(w_c.t) + v_q(t).\cos(w_c.t + 90°)$$




a_ Espectro de señal pasabanda




b_ Señal en el dominio del tiempo

*Figura 4-8:* Ejemplo de señal pasabanda. Espectro y señal en el tiempo

Donde

$$v_i(t) = A(t).\cos\big(\emptyset(t)\big)$$

$$v_q(t) = A(t).sen\big(\emptyset(t)\big)$$

Esta es la descripción en fase y cuadratura de la señal. $v_i(t)$ y $v_q(t)$ son las componentes en fase y cuadratura, respectivamente. Resulta de interés notar que, en señales pasabanda, ambas componentes tienen espectros pasabajos.

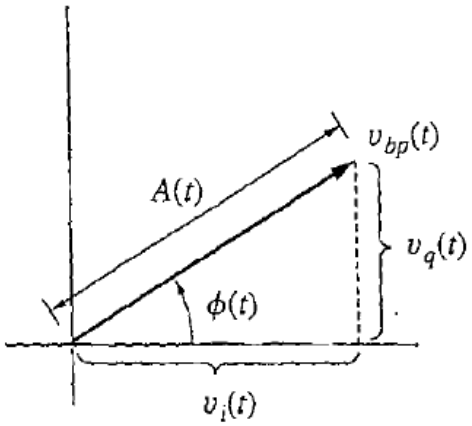



Figura 4-9: Representación de la señal como un fasor.

## Señales pasabanda digitales

Como vimos anteriormente, cualquier señal pasabanda puede ser expresada con la expresión:

$$x_c(t) = A_c.[x_i(t).\cos(w_c.t + \theta) - x_q(t).sen(w_c.t + \theta)]$$

Donde la frecuencia de portadora $f_c$, amplitud $A_c$, y fase $\theta$ son constantes, mientras que las componentes $x_i(t)$ (en fase) y $x_q(t)$ (en cuadratura) dependen del tiempo, y contienen el mensaje. Si estas componentes son estadísticamente independientes, y tienen media cero (ambas condiciones se cumplen en este proyecto), el espectro de potencias de $x_c(t)$ resulta:

$$G_c(f) = \frac{A_c^2}{4}.[G_i(f - f_c) + G_i(f + f_c) + G_q(f - f_c) + G_q(f + f_c)]$$

Donde $G_i(f)$ y $G_q(f)$ son los espectros de potencia de las componentes $i$ y $q$. Por simplicidad, usaremos el equivalente pasabajos:

$$G_{lp}(f) = G_i(f) + G_q(f)$$

Por lo que

$$G_c(f) = \frac{A_c^2}{4}.[G_{lp}(f - f_c) + G_{lp}(f + f_c)]$$

Si suponemos que la componente en fase es una señal digital M-aria (puede tomar M distintos valores), esta tendrá la forma:

$$x_i(t) = \sum_k a_k.p(t - k.D)$$

Donde $a_k$ representa una secuencia de dígitos estadísticamente equiprobables, estadísticamente independientes, no correlacionados, de velocidad $r = \frac{1}{D}$.

El espectro de potencia de esta componente será:

$$G_i(f) = \sigma_a^2.r.|P(f)|^2 + (m_a.r)^2.\sum_{-\infty}^{\infty}|P(n.r)|^2.\delta(f - n.r)$$

El espectro de la componente en cuadratura será el análogo.

Si usamos un pulso $p_D(t)$ rectangular, de duración D, entonces:

$$|P_D(f)^2| = D^2.sinc^2(f.D)$$

Como este espectro no es acotado en frecuencia, podemos concluir que necesitaremos una frecuencia de portadora $f_c \gg r$ para poder producir una señal pasabanda.

## Modulación y sus ventajas

En la modulación se incluyen dos formas de onda: una señal modulante, que representa al mensaje, y una señal portadora, elegida según la aplicación. Un modulador sistemáticamente altera la señal portadora, en función a las variaciones de la señal modulante. Se dice que la señal resultante, modulada, "porta" la información del mensaje. Es necesario, entonces, que la modulación sea una operación reversible, para poder recuperar el mensaje, por el proceso complementario de demodulación.

Este proceso puede verse en la Figura 4-10. Allí, puede verse un ejemplo de modulación de amplitud AM (en inglés, Amplitude Modulation). En 4-10a se observa a la señal portadora, de forma senoidal, de frecuencia alta. En 4-10b, la señal modulante, de frecuencia mucho menor a la señal portadora. Por último, en 4-10c se aprecia la señal resultante, de frecuencia semejante a la portadora, pero cuya envolvente se asemeja a la modulante. Este no es el único método de modulación, pero es el más simple, apto para una primera introducción.

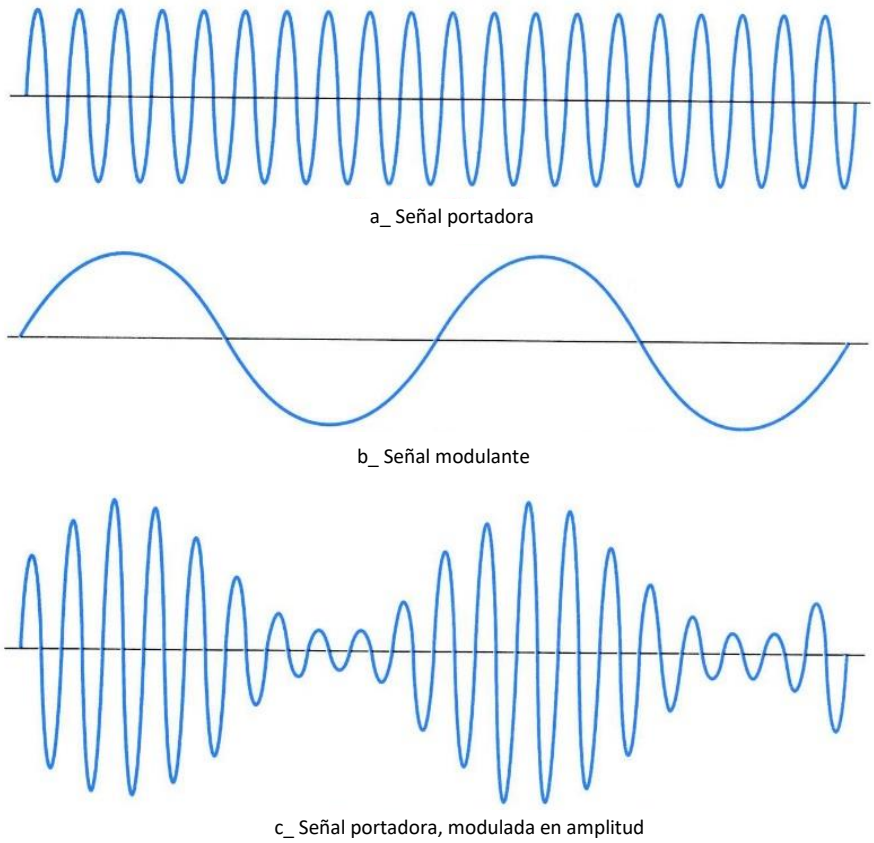



*Figura 4-10:* Ejemplo de modulación en amplitud

La función primaria de la modulación en un sistema de comunicaciones es la de

generar una señal apta para las características del canal de transmisión. Sin embargo, hay muchos beneficios adicionales, y aplicaciones prácticas, como se listará a continuación.

Modulación para transmisión eficiente: La transmisión de señales sobre distancias apreciables incluye una onda electromagnética viajando por el medio, pudiendo ser el mismo guiado (un cable), o no guiado (el aire). La eficiencia de la transmisión depende de la frecuencia a la que se transmita. Como vimos anteriormente, la modulación permite transmitir a una frecuencia deseada, relacionada con la portadora, por lo que cada sistema de transmisión usa el rango de frecuencia que le es más conveniente. En la Figura 4-11 se puede apreciar algunos de los distintos sistemas de comunicación que aprovechan cada rango de frecuencias del espectro radioeléctrico.

Modulación para superar limitaciones de hardware: El diseño de sistemas de comunicaciones puede estar limitado por el costo y disponibilidad de hardware, cuya

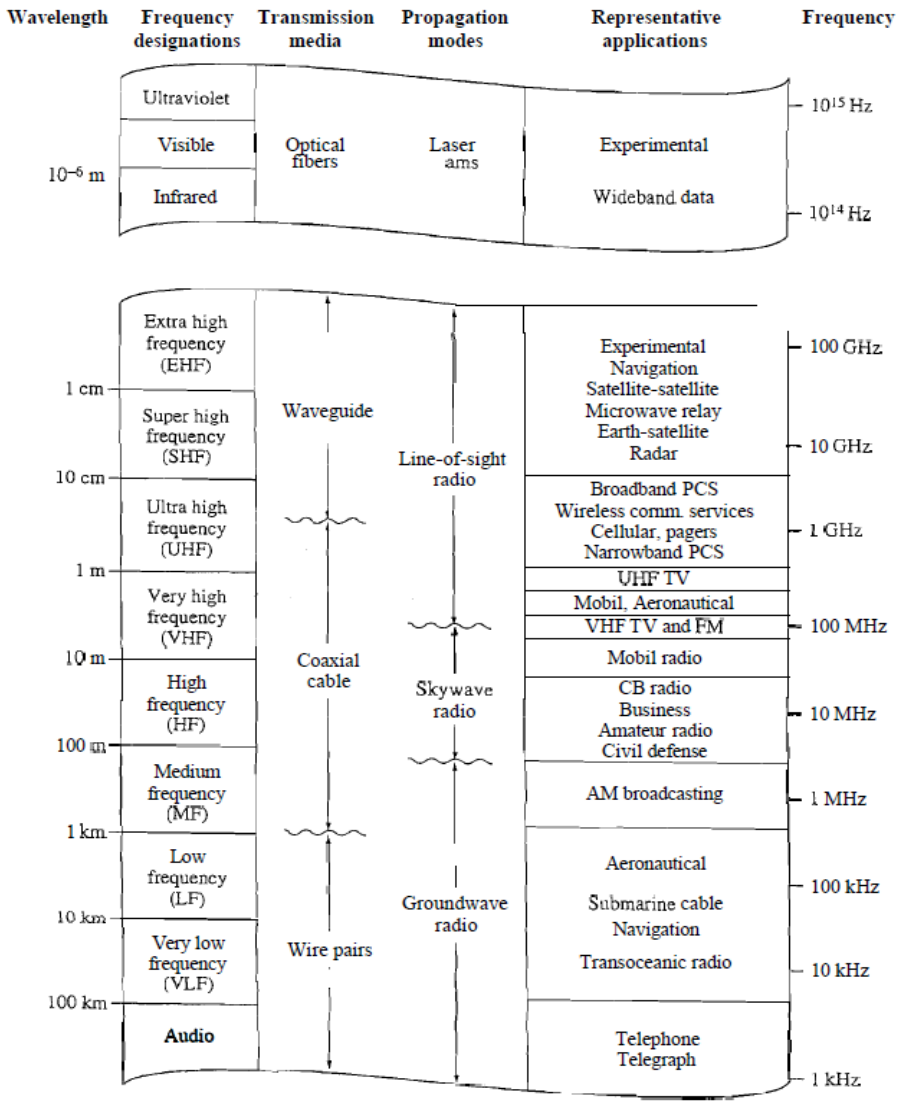



*Figura 4-11:* Uso del espectro radioeléctrico

performance puede depender de las frecuencias que estén involucradas. La modulación permite al diseñador colocar la señal en un rango de frecuencias apropiado para el uso del hardware en cuestión.

Modulación para reducir ruido e interferencia: Un método para combatir el ruido y la interferencia, es el de aumentar la potencia de la señal. Sin embargo, esto puede ser costoso, y puede dañar el hardware utilizado. Afortunadamente, algunos sistemas de modulación tienen propiedades que los hacen relativamente inmunes a estos inconvenientes.

Modulación para asignación de frecuencia: Es posible, a partir de la modulación, tener un control sobre en qué parte del espectro radioeléctrico la potencia de nuestra señal se encontrará. De esta forma, podemos acomodar diversas señales en distintas franjas del espectro, enviando muchas señales al mismo tiempo. Es necesario, sin embargo, el uso de filtros antes de la demodulación, para solo usar la parte del espectro que sea necesaria.

## Modulaciones analógicas

En las modulaciones analógicas, la portadora es una onda sinusoidal, modulada por una señal analógica. Los ejemplos más conocidos son AM y FM, siendo la primera una modulación lineal (hay una translación directa del espectro del mensaje), y la segunda, una modulación exponencial (el espectro resultante no tiene relación directa aparente con el mensaje). Esta clasificación puede verse en la Figura 4-12. No entraremos en mayor detalle, ya que en este trabajo nos dedicaremos a experimentar con señales y modulaciones de naturaleza digital.

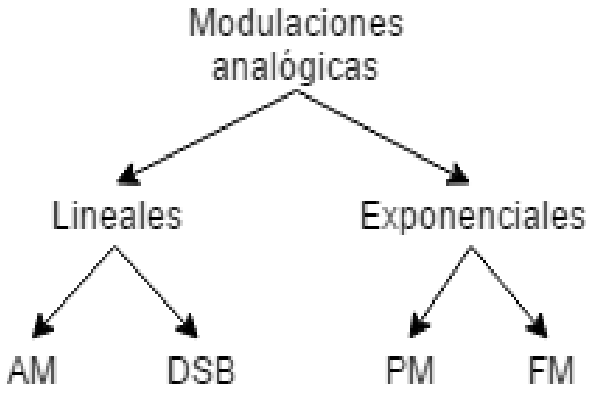



*Figura 4-12:* Clasificación y ejemplos de modulaciones analógicas.

## Modulaciones digitales

Una señal digital puede modular la amplitud, frecuencia, o fase de una portadora sinusoidal. Esta portadora cambiará entre diversos valores discretos, según la modulante. En la Figura 4-13 pueden verse tres ejemplos de modulaciones digitales. En 4-13a vemos una modulación por cambio de amplitud ASK (en inglés, Amplitude Shift Keying), en 4-13b vemos una modulación por cambio de frecuencia FSK (en inglés, Frequency Shift Keying), y en 4-13c vemos una modulación por cambio de fase PSK (en inglés, Phase Shift Keying). Llamaremos eficencia espectral a la relación $\frac{r_b}{B_T}$, donde $r_b$ es la velocidad del flujo de bits a la entrada del transmisor; y $B_T$ es el ancho de banda de la señal modulada.

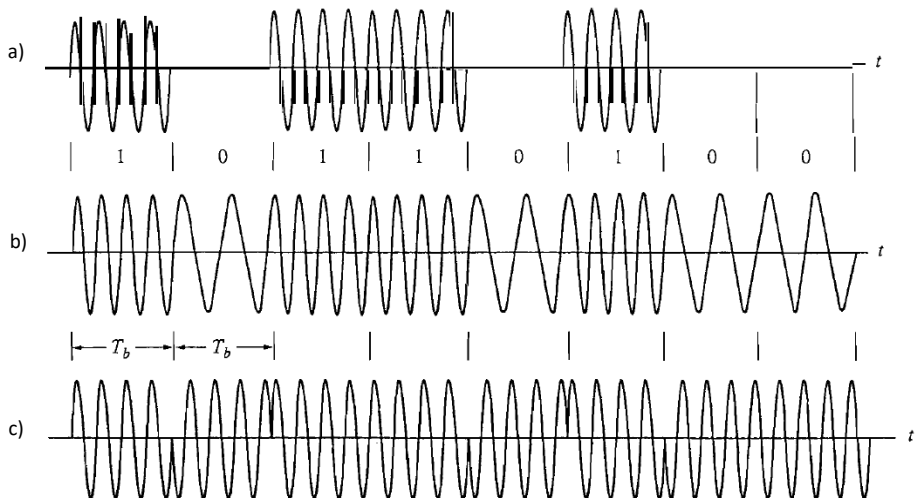



*Figura 4-13:* Ejemplos de modulaciones digitales

## Modulación por cambio de amplitudes - ASK

La forma de onda ASK ilustrada en la figura 4-13a puede ser generada simplemente apagando y prendiendo la señal portadora, proceso llamado OOK (on-off keying). En general, formas de onda ASK M-arias tienen M-1 amplitudes discretas, y el estado de apagado. Como no hay variaciones en la fase de la señal modulada, la componente $q$ es nula, mientras que la componente $i$ es la señal modulante en banda base. La Figura 4-14 muestra el espectro de potencias de una señal ASK M-aria. La velocidad binaria será $r_b = r.log_2(M)$, y el ancho de banda de la señal $B_T = r = \frac{r_b}{log_2(M)}$. Así se logra una eficiencia espectral $\frac{r_b}{B_T} = log_2(M)$.

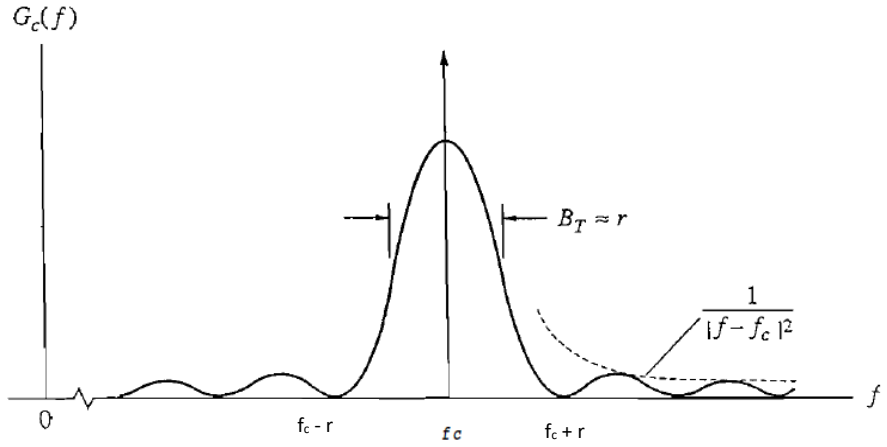



*Figura 4-14:* Espectro de potencia de una señal ASK M-aria

## Modulación por cambio de amplitudes ortogonales - QAM

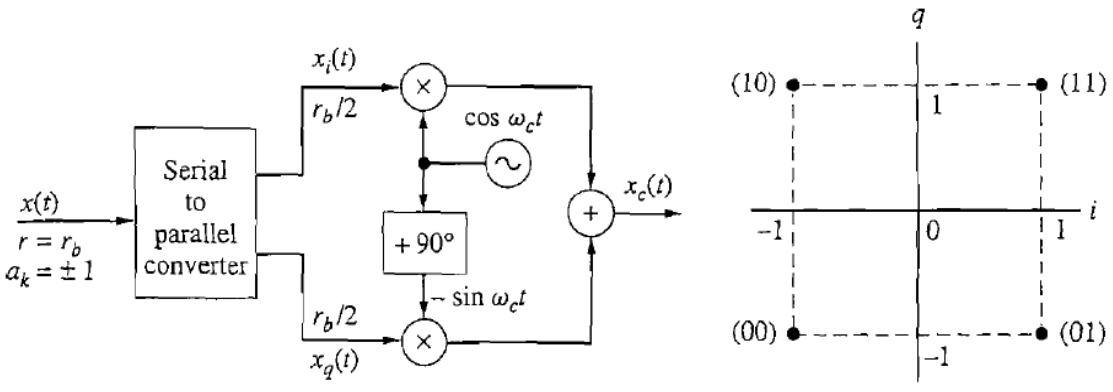
Usando los conceptos de fase y cuadratura, la modulación por cambio de amplitudes ortogonales QAM (en inglés, Quadrature Amplitude Modulation) logra el doble de velocidad de modulación que ASK. La Figura 4-15 muestra los bloques funcionales de un modulador QAM, con una entrada binaria, de velocidad $r_b$. El conversor serie-paralelo divide la entrada en dos flujos de bits, cada uno de velocidad $\frac{r_b}{2}$. Así, las componentes de la señal resultan:

$$x_i(t) = \sum_k a_{2.k}.p_D(t - k.D)$$

$$x_q(t) = \sum_k a_{2.k+1}.p_D(t - k.D)$$

Donde $D = \frac{1}{r} = 2.T_b$ y $a_k = \pm 1$.

La figura 4-15b muestra las combinaciones de valores que pueden tomar $i$ y $q$. A este diagrama se le llama “constelación”.




a_ Diagrama en bloques de modulador QAM

b_ Constelación QAM de 4 puntos

*Figura 4-15:* Modulador QAM

Las componentes son estadísticamente independientes, y tienen los mismos valores estadísticos, $m_a = 0$ y $\sigma^2 = 1$. De esta manera, el espectro de potencia equivalente pasa bajos resulta:

$$G_{lp}(f) = 2.r.|P_D(f)|^2 = \frac{4}{r_b}.sinc^2(\frac{2.f}{r_b})$$

De aquí podemos concluir que este sistema QAM tiene una eficiencia espectral $\frac{r_b}{B_T} = 2\frac{bps}{Hz}$. En este caso, el conversor serie-paralelo "acumula" de a dos bits, para luego enviarlos cada uno por una rama.

También es posible acumular $k = log_2(M)$ bits de entrada al transmisor, y modular las componentes $i$ y $q$ en función de la secuencia de bits a la entrada, y una constelación de M puntos específica. Esto es una modulación QAM M-aria o MQAM. De esta manera, la eficiencia del sistema resulta $\frac{r_b}{B_T} = log_2(M)\frac{bps}{Hz}$. En la Figura 4-16 puede verse una constelación cuadrada de 16 puntos ($M = 16$).

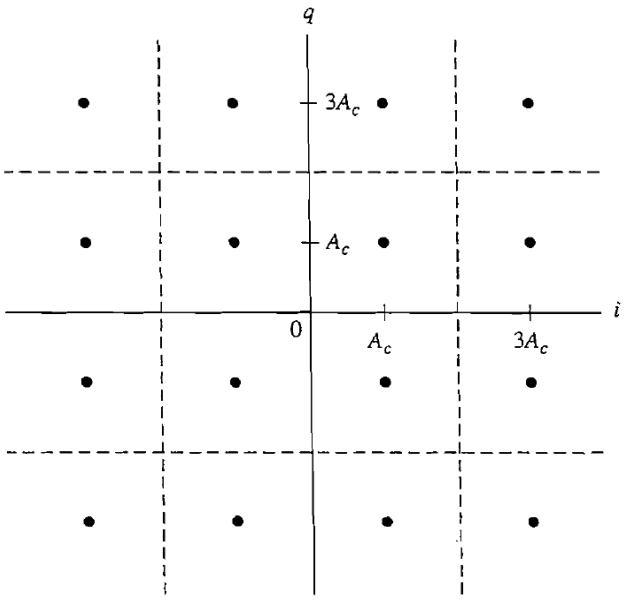



*Figura 4-16:* Constelación cuadrada MQAM con M = 16.

La modulación MQAM es útil para canales con ancho de banda limitado, y provee una menor tasa de errores que otros sistemas M-arios que operan a la misma velocidad. Esta es la modulación elegida para este trabajo.

## Otros tipos de modulaciones digitales

Las modulaciones que hemos repasado hasta ahora pueden ser clasificadas como modulaciones de amplitud. Al igual que en las modulaciones analógicas, es posible modular otras características de la portadora, de forma digital. Por ejemplo, PSK es una modulación de fase, y FSK es una modulación de frecuencia.

Cuando la cantidad de valores de fase posibles en PSK es mayor a 2, la llamamos MPSK (en inglés, Multiple Phase Shift Keying). Podríamos caracterizar a MPSK como un caso especial de MQAM, donde la amplitud del módulo de la señal de salida, es siempre constante. De esta manera, la constelación es de forma circular. Esto puede verse en la Figura 4-17. No nos adentraremos en mayor detalle sobre estos tipos de modulación digital, ya que en este proyecto trabajaremos exclusivamente con modulación MQAM.

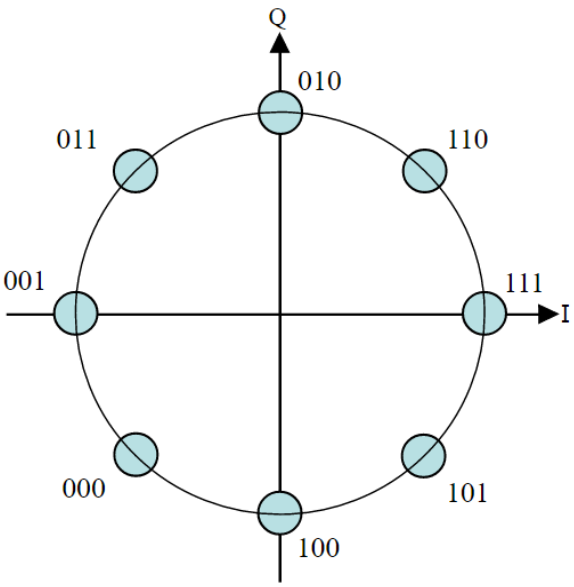



*Figura 4-17:* Constelación MPSK con M = 16

# Multiplexado por división en frecuencias ortogonales – OFDM

## Introducción a OFDM

Al modular, se traslada la información a cambios de frecuencia, fase, o amplitud (o una combinación de ellas) de una portadora. Al multiplexar, acomodamos distintos usuarios en distintas bandas de frecuencia. El multiplexado por división de frecuencias ortogonales OFDM (en inglés, Orthogonal Frequency Division Multiplexing) es una combinación de modulación y multiplexado. En esta técnica, el ancho de banda es compartido por distintas fuentes individuales moduladas (aunque esas fuentes pueden haber modulado distintas partes de un mismo mensaje).

Las técnicas de modulación tradicionales (como AM, FM, ASK, QAM) son modulaciones del tipo de portadora única, donde la información se transfiere en una única portadora. OFDM es una técnica de modulación multiportadora, que emplea más

de una portadora, dentro de un ancho de banda específico, para transferir la información de la fuente hacia el destino. Cada portadora puede usar alguna de las modulaciones digitales previamente mencionadas.

Este método es muy efectivo para la comunicación sobre canales con fading selectivo en frecuencia (cuando diferentes componentes frecuenciales de la señal sufren pérdidas distintas). En vez de combatir al fading selectivo, OFDM mitiga el problema convirtiendo al canal selectivo, en pequeños subcanales con flat fading (cuando las componentes de una señal sufren pérdidas similares). El flat fading es más sencillo de combatir que el selectivo, simplemente usando esquemas de corrección y ecualización.

OFDM es un caso especial de multiplexado por división de frecuencias FDM (en inglés, Frequency Division Multiplexing). En FDM, el ancho de banda es dividido en un grupo de portadoras, en principio sin ninguna relación entre sus respectivas frecuencias. Por ejemplo, si tuviéramos 6 portadoras (digamos a, b, c, d, e, f), cada una podría comportarse de cualquier manera, en sus respectivos subcanales. Si las portadoras fueran armónicos de una misma fundamental, serían “ortogonales” entre sí. Este caso es el llamado OFDM. En la Figura 4-18 podemos ver una comparación entre un espectro FDM, que separa las portadoras para que no se interfieran, y un espectro OFDM, que separa las portadoras para que los ceros de una, correspondan con máximos de otras (es decir, son ortogonales). Puede observarse que, en el caso de OFDM, se envía la misma información, en un ancho de banda reducido, respecto a FDM.

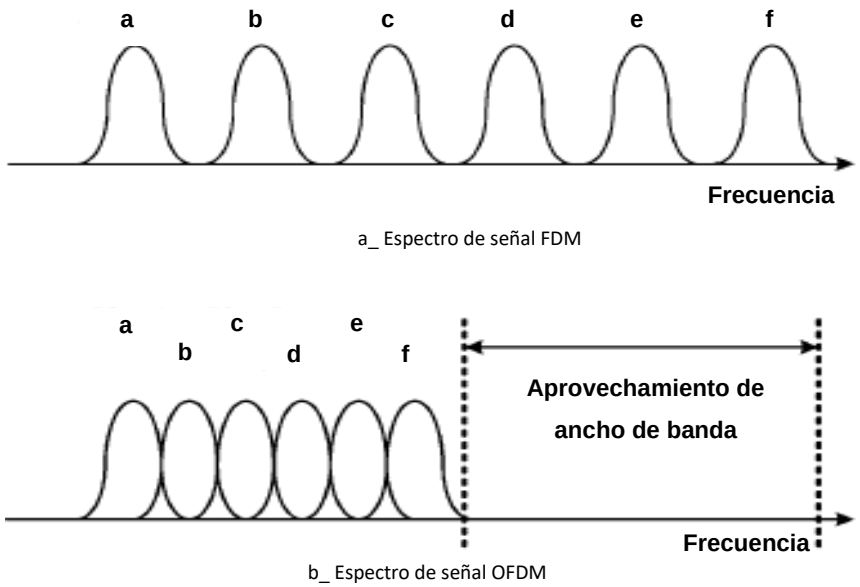



*Figura 4-18:* Diagrama que muestra el mejor aprovechamiento del espectro de OFDM, respecto de FDM.

## Transmisor OFDM

Consideremos que se necesita enviar un conjunto de bits $D = \{d_0, d_1, d_2, \dots\}$ usando OFDM. Lo primero que debemos considerar en el diseño de un transmisor OFDM, es el número de subportadoras requeridas para enviar los datos en cuestión. Como un caso genérico, asumimos que usaremos N subportadoras. Numeraremos a estas subportadoras desde $0$ hasta $N-1$. Cada subportadora estará centrada en frecuencias que son ortogonales unas a las otras.

El segundo parámetro a considerar podría ser la modulación a implementar. Una señal OFDM puede ser construida usando cualquiera de las modulaciones digitales vistas anteriormente. Entre ellas se encuentran MPSK, MQAM, etc.

Los datos deben ser primero convertidos de serie a paralelo, dependiendo del número de subportadoras $N$. El conversor toma el flujo serie de bits y lo divide en $N$ flujos paralelos, indexados desde $0$ hasta $N-1$. Estos flujos paralelos son luego modulados según el esquema requerido (MPSK, MQAM, etc.). Llamaremos a los flujos de salida de los moduladores $S_0, S_1, S_2, \dots, S_{N-1}$. La conversión de datos en paralelo D, a datos digitales modulados S, es usualmente logrado con una tabla de búsqueda LUT (en inglés, Look-up Table). Una vez que estos datos fueron modulados, deben ser montados en las subportadoras de frecuencias ortogonales entre sí, para luego ser transmitidas al medio. Intuitivamente, esto puede realizarse con un banco de $N$ osciladores senoidales, en paralelo, de frecuencias ortogonales. Finalmente, el resultado es sumado para producir la señal OFDM. El proceso descripto puede verse en la Figura 4-19. En este momento, podemos sospechar que el uso de $N$ osciladores puede ser poco práctico, en especial cuando $N$ sea un número grande. La cantidad de circuitería necesaria haría que este sistema sea muy costoso de desarrollar.

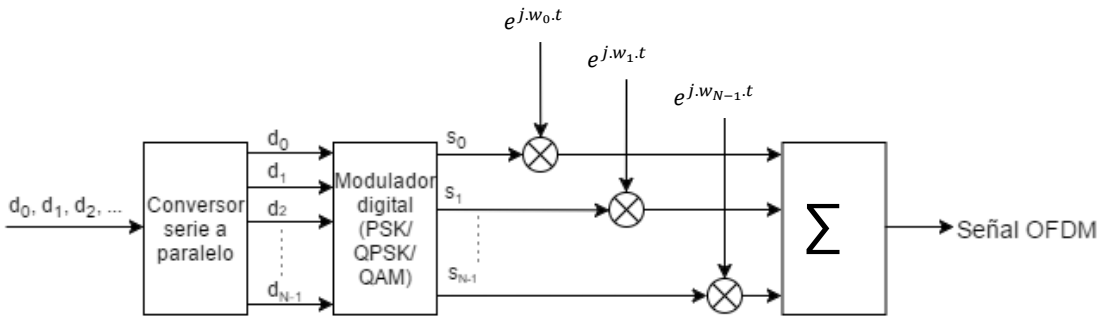



*Figura 4-19:* Diagrama básico de transmisor OFDM, implementando osciladores ortogonales

## La transformada rápida de Fourier FFT en OFDM

En general, podemos expresar una señal OFDM como:

$$c(t) = \sum_{n=0}^{N-1} s_n(t).\, e^{j.2.\pi.f_n.t}$$

Donde $s_n(t)$ representa los símbolos modulados por la constelación utilizada, y $f_n$ son el conjunto de frecuencias ortogonales.

Esta ecuación tiene una íntima relación la transformada inversa de Fourier discreta IDFT. Tanto es así, que el conjunto de osciladores puede ser reemplazado por otro que realice la transformada rápida inversa de Fourier IFFT, algoritmo utilizado para realizar la IDFT. Para más información sobre la DFT/IDFT y su algoritmo optimizado FFT/IFFT, ver el Anexo, sección “La transformada discreta de Fourier - DFT” . Evidentemente, el número de muestras usadas para la operación FFT/IFFT debe ser correspondiente con el número de subportadoras N a implementarse.

Dado que la señal OFDM $c(t)$ está en el dominio del tiempo, tiene sentido utilizar la IFFT en el extremo transmisor, que transformaría las muestras del dominio frecuencial, al dominio temporal. Sin embargo, es posible argumentar que $s_n(t)$ también se encuentra expresada en el dominio temporal. En verdad, se usa la IFFT por su similitud matemática con la operación deseada. En la práctica, es posible utilizar cualquiera de las dos operaciones, siempre y cuando se use su dual en el extremo receptor. En la Figura 4-20 puede verse el sistema transmisor-receptor simplificado, usando OFDM. Notar que en el receptor, los valores de $s_n$ y $d_n$ serán estimados, ya que es posible que se produzcan errores, producto de las imperfecciones en el canal.

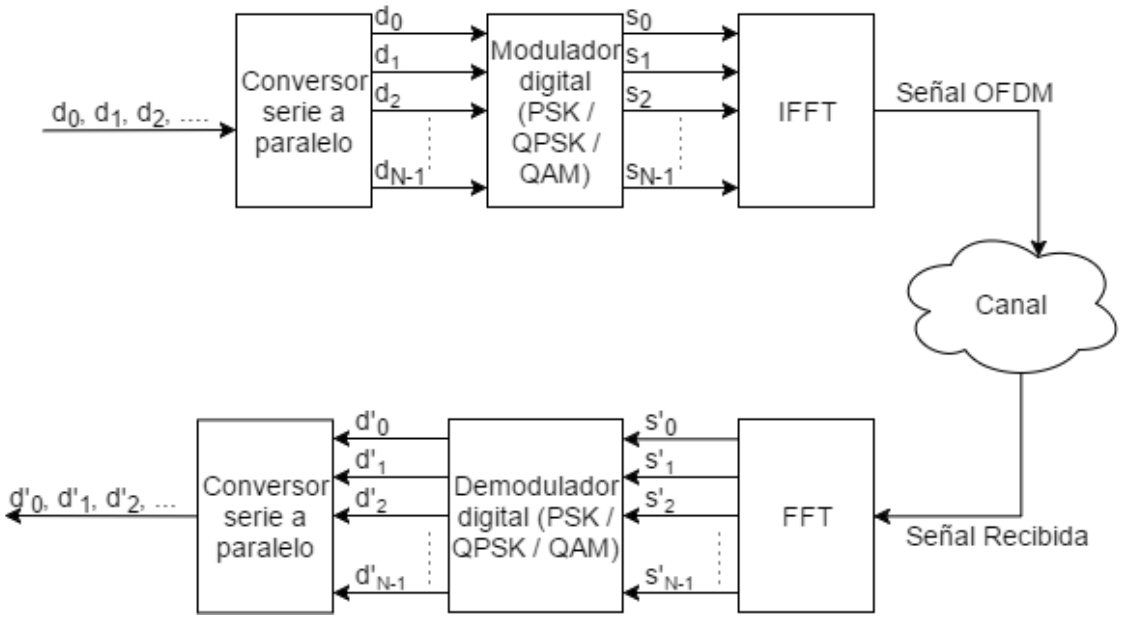



*Figura 4-20:* Diagrama en bloques de sistema transmisor-receptor OFDM

## Tecnología FPGA

Una alternativa más que interesante para la implementación de sistemas de comunicaciones se apoya en una tecnología que hace ya varios años está ganando lugar en el mercado. Se trata de la tecnología de dispositivos de hardware reconfigurables, también conocidos como FPGA (en inglés, “Field Programable Gate Array”).

Los FGPA poseen cada vez más áreas de aplicación (por ejemplo, radioastronomía, emulación de hardware, bioinformática, criptografía, sistemas de comunicación, entre otros). Las universidades están extendiendo su uso como excelente herramienta didáctica y de prototipado.

Un FPGA es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada, utilizando un lenguaje de descripción especializado (hablaremos sobre el lenguaje utilizado más adelante). La lógica programable puede reproducir desde funciones sencillas, como las llevadas a cabo por una compuerta lógica, hasta complejos sistemas.

A diferencia de circuitos microcontroladores, donde solo una instrucción puede ser ejecutada en un instante de tiempo, las FPGA pueden realizar cualquier cantidad de tareas en paralelo, según sus recursos lo permitan.

Las especificaciones de recursos de FPGA a menudo incluyen el número de bloques de lógica configurables, número de bloques de lógica de función fijos, como multiplicadores, y el tamaño de los recursos de memoria, como RAM, en bloques embebidos. De las muchas partes del chip FPGA, estos son generalmente los más importantes cuando se seleccionan y comparan FPGAs para una aplicación en particular.

Las principales empresas (líderes de mercado) productoras de estos dispositivos, son Xilinx y Altera. Entre ambas, satisfacen a gran parte de la demanda de FPGA. Cada una de ellas posee distintas series de productos, desde chips económicos, ideados para pequeños diseños, hasta otros con infinidad de recursos, ideados para diseños de gran tamaño. En este proyecto, se utilizó una FPGA de Altera. Más precisamente, una de la serie Cyclone V.

La serie Cyclone está construida para la realización de diseños de poca potencia, siendo el precio un factor clave, permitiendo a economías de bajos recursos poder adentrarse en el diseño en FPGA. Cada iteración de esta serie fue creciendo en integración y performance, siendo hoy una gran alternativa para proyectos universitarios.

## Lenguaje VHDL

Los lenguajes utilizados para el diseño de sistemas en FPGA son llamados HDLs (en inglés, “Hardware Description Language”). El más utilizado es VHDL, definido por el IEEE. Originalmente, el lenguaje VHDL fue desarrollado por el Departamento de Defensa de los Estados Unidos a inicios de los años 80's, con el fin de realizar simulación de circuitos eléctricos digitales; sin embargo, posteriormente se desarrollaron las herramientas de síntesis e implementación en hardware a partir de los archivos VHDL.

Dentro del VHDL hay varias formas con las que se puede diseñar el mismo circuito y es tarea del diseñador elegir la más apropiada. Entre ellas se encuentra:

1- Funcional o Comportamental: Se describe la forma en que se comporta el circuito digital, se tiene en cuenta solo las características del circuito respecto al comportamiento de las entradas y las salidas. Esta es la forma que más se parece a los lenguajes de software ya que la descripción puede ser secuencial, además de combinar características concurrentes. Estas sentencias secuenciales se encuentran dentro de los llamados procesos en VHDL. Los procesos son ejecutados en paralelo entre sí, y en paralelo con asignaciones concurrentes de señales y con las instancias a otros componentes.

2- Flujo de datos: Se describen asignaciones concurrentes (en paralelo) de señales.

3- Estructural: Se describe el circuito con instancias de componentes. Estas instancias forman un diseño de jerarquía superior, al conectar los puertos de estas instancias con las señales internas del circuito, o con puertos del circuito de jerarquía superior. Es la recomendada cuando el diseño digital se vuelve complejo o está conformado por múltiples bloques de hardware.

4- Mixta: combinación de todas o algunas de las anteriores.

En VHDL también existen formas metódicas para el diseño de máquinas de estados, filtros digitales, bancos de pruebas, etc.

El primer paso del diseño consiste en la construcción del diagrama en bloque del sistema. En diseños complejos como en software los programas son generalmente jerárquicos y VHDL ofrece un buen marco de trabajo para definir los módulos que integran el sistema y sus interfaces, dejando los detalles para pasos posteriores.

El segundo paso es la elaboración del código en VHDL para cada módulo, para sus interfaces y sus detalles internos. Como el VHDL es un lenguaje basado en texto, se puede utilizar cualquier editor para esta tarea, aunque el entorno de los programas de VHDL incluye su propio editor de texto. Después de que se ha escrito algún código

se hace necesario compilarlo. El compilador de VHDL analiza este código y determina los errores de sintaxis y chequea la compatibilidad entre módulos.

# 5. Desarrollo práctico

El desarrollo de este proyecto se realizó sobre una plataforma FPGA. Más específicamente, se utilizó una placa de desarrollo Cyclone V GX Starter Kit, diseñada por el fabricante taiwanés Terasic, cuya imagen puede verse en la Figura 5-1. La misma cuenta con un dispositivo FPGA Cyclone V GX 5CGXFC5C6F27C7N, de Altera (recientemente adquirida por Intel), que contiene 77000 elementos lógicos, además de seis PLL y 150 bloques DSP, entre otras utilidades. Además, la placa cuenta con diversos periféricos, como 18 LEDS, 4 botones, 10 switches, etc. En la Figura 5-2 puede verse un resumen de los periféricos presentes en esta placa.


*Figura 5-1:* Ilustración de la placa de desarrollo utilizada

El software utilizado para programar esta placa fue el Quartus Primer Lite Edition, versión 16.1.1. Esta interface presenta diversas herramientas para el diseño de los bloques funcionales que componen un proyecto. En esta ocasión se utilizaron principalmente dos utilidades: el editor de texto, para escribir código en lenguaje VHDL; y archivos esquemáticos, para interconectar los distintos bloques funcionales que componen el diseño completo. Estos archivos esquemáticos resultan de especial utilidad, ya que permiten una rápida comprensión de la funcionalidad de cada bloque.

Una vez completo el proyecto, la compilación la realiza Quartus. La misma consta fundamentalmente de 4 pasos: análisis y síntesis, que verifica que el código escrito no presente problemas sintácticos; fitter, que utiliza un algoritmo para calcular el circuito que debe ser sintetizado en la placa; assembler, que genera los archivos que son transmitidos a la placa, para la sintetización del circuito; y TimeQuest Timing

Analyzer, que da información sobre tiempos de retardo, entre otras cosas.

Además, se utilizó el software MATLAB desde una PC, para la comprobación

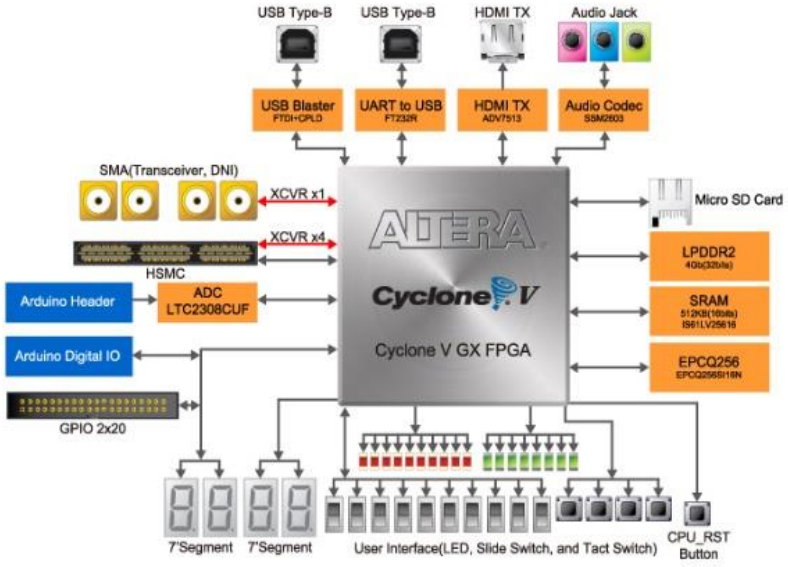



*Figura 5-2:* Información de periféricos presentes en la placa de desarrollo.

funcional y simulaciones del sistema frente a valores teóricos de probabilidad binaria de error en función de la relación señal a ruido, para distintos medios. La comunicación entre la computadora y la placa fue realizada a partir de una comunicación serie UART, utilizando el chip FT232R presente en la placa.

Un diagrama en bloques del sistema completo puede verse en la Figura 5-3. Se trata de un circuito cerrado (aunque fue representado como una línea recta). En principio, la PC genera una secuencia binaria aleatoria, que es transmitida a la FPGA. Esta modula la secuencia recibida, y devuelve a la computadora un conjunto de símbolos, equivalentes a los datos de entrada modulados. Luego, se simula el medio (en la PC) para añadir ruido o interferencia de algún tipo a los símbolos, para después demodular los mismo en la FPGA. Finalmente, se calcula la tasa de error que presenta este sistema de comunicaciones frente al medio simulado. Todas las comunicaciones entre la PC y la FPGA se hacen implementando una UART.


*Figura 5-3:* Diagrama en bloques del sistema completo

Se eligió utilizar un sistema OFDM de 8 subportadoras, ya que este presenta una mejora notoria frente a sistemas uniportadora, si tenemos en cuenta la velocidad de

transmisión que puede alcanzarse en un ancho de banda determinado. También presenta una mejora en función del fading presente por reflexiones en el medio, ya que el ancho de banda total de la señal es subdividido en 8, haciendo que un sistema que presenta fading selectivo en frecuencia, en el ancho de banda de la señal completa, pueda utilizarse, si es que el fading pasa a ser plano al dividir el espectro en 8. Se evitaron sistemas con mayor cantidad de subportadoras ya que su diseño en lenguaje VHDL sería poco práctico, debido al tiempo de desarrollo que requeriría. Para este tipo de diseños, es recomendable el uso de placas SOC (en inglés, “System On Chip”), que poseen tanto dispositivos FPGA como microcontroladores, interconectados. Estas placas pueden ser programadas en lenguaje C, para luego implementar sus distintos bloques en el microcontrolador, o en la FPGA, según sea conveniente. El circuito implementado en la FPGA no es tan eficiente como cuando se desarrolla en VHDL, pero permite diseños que requieran operaciones matemáticas que pueden llegar a ser dificultosas en lenguajes HDL.

En las próximas secciones, analizaremos en detalle los bloques funcionales de este sistema, para así poder entender de qué se tratan los resultados subsiguientes. Los bloques implementados en la computadora se presentarán en los capítulos “Simulaciones” y “Resultados experimentales”.

# Diseños en FPGA

## Modulador OFDM

El diseño del modulador, en FPGA, consiste básicamente de 6 bloques funcionales. En la Figura 5-4 puede verse el mismo, así como las interconexiones entre

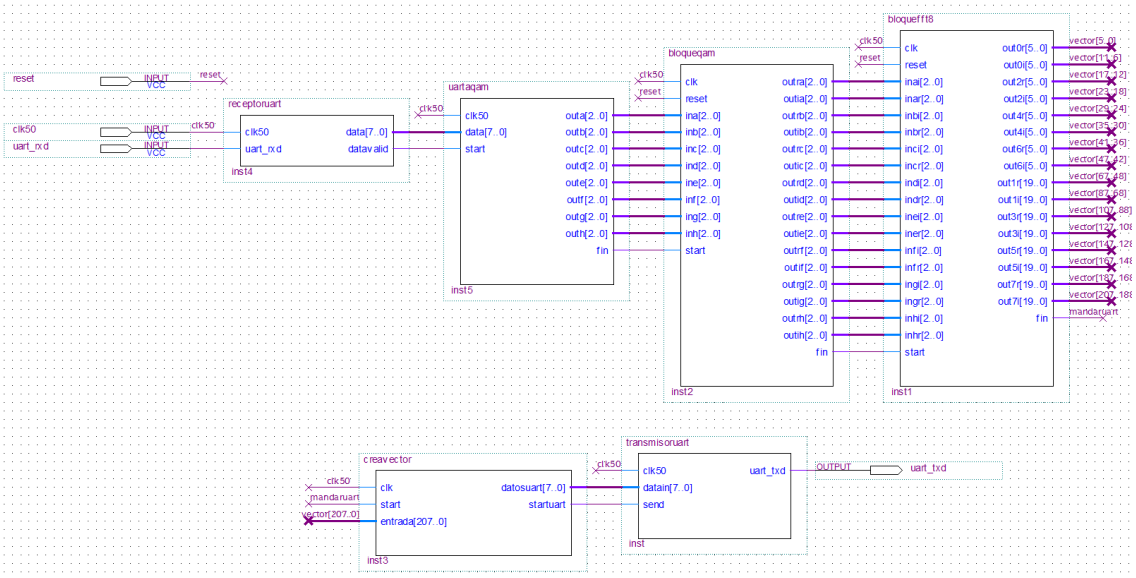



*Figura 5-4:* Diagrama en bloques, visto en Quartus, del modulador OFDM.

los bloques. Los pines “reset”, “clk50” y “uart_rxd” son las entradas al circuito. El pin “uart_rxd” es la salida del circuito. “reset” usa lógica negada (se activa cuando la línea tiene un estado de tensión bajo). “clk50” es un reloj de 50 MHz, incluido en la placa de desarrollo. “uart_rxd” es la línea de recepción UART, por donde se reciben los datos binarios. El sistema espera recibir 3 bytes por comunicación UART, y luego los procesa. Entrega, nuevamente por UART, los datos modulados, siguiendo un orden específico, que veremos más adelante. Todos los bloques están diseñados como máquinas de estado, y utilizan un reloj de 50 MHz.

Los bloques “receptoruart”, “uartaqam”, “creavector” y “transmisoruart” se usan para transmitir, recibir, y poner en orden, los datos que son recibidos y enviados por UART, ya que este tipo de comunicación es serie, pero el resto de los bloques toma las entradas en paralelo, y dispone sus salidas también en paralelo. La descripción de los mismos puede verse en el anexo, en la sección “Bloques Quartus auxiliares”.

En la Figura 5-5, puede verse un resumen que ofrece Quartus, con información sobre la implementación del circuito en la FPGA, como cantidad de elementos lógicos utilizados, bloques DSP utilizados, entre otros. Como podemos ver, la utilización de esta parte del proyecto sobre la FPGA es mínima, en comparación de los recursos disponibles.

| | |
|---|---|
| Flow Status | Successful - Tue Mar 21 15:09:10 2017 |
| Quartus Prime Version | 16.1.1 Build 200 11/30/2016 SJ Lite Edition |
| Revision Name | ofdm8 |
| Top-level Entity Name | ofdm8 |
| Family | Cyclone V |
| Device | 5CGXFC5C6F27C7 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 389 / 29,080 ( 1 % ) |
| Total registers | 423 |
| Total pins | 4 / 364 ( 1 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 4,567,040 ( 0 % ) |
| Total DSP Blocks | 8 / 150 ( 5 % ) |
| Total HSSI RX PCSs | 0 / 6 ( 0 % ) |
| Total HSSI PMA RX Deserializers | 0 / 6 ( 0 % ) |
| Total HSSI TX PCSs | 0 / 6 ( 0 % ) |
| Total HSSI PMA TX Serializers | 0 / 6 ( 0 % ) |
| Total PLLs | 0 / 12 ( 0 % ) |
| Total DLLs | 0 / 4 ( 0 % ) |

*Figura 5-5:* Resumen de los recursos utilizados por este proyecto de la placa de desarrollo.

En esta sección, analizaremos el funcionamiento de los bloques responsables de la modulación de los datos de entrada.

### Bloque “bloqueqam”

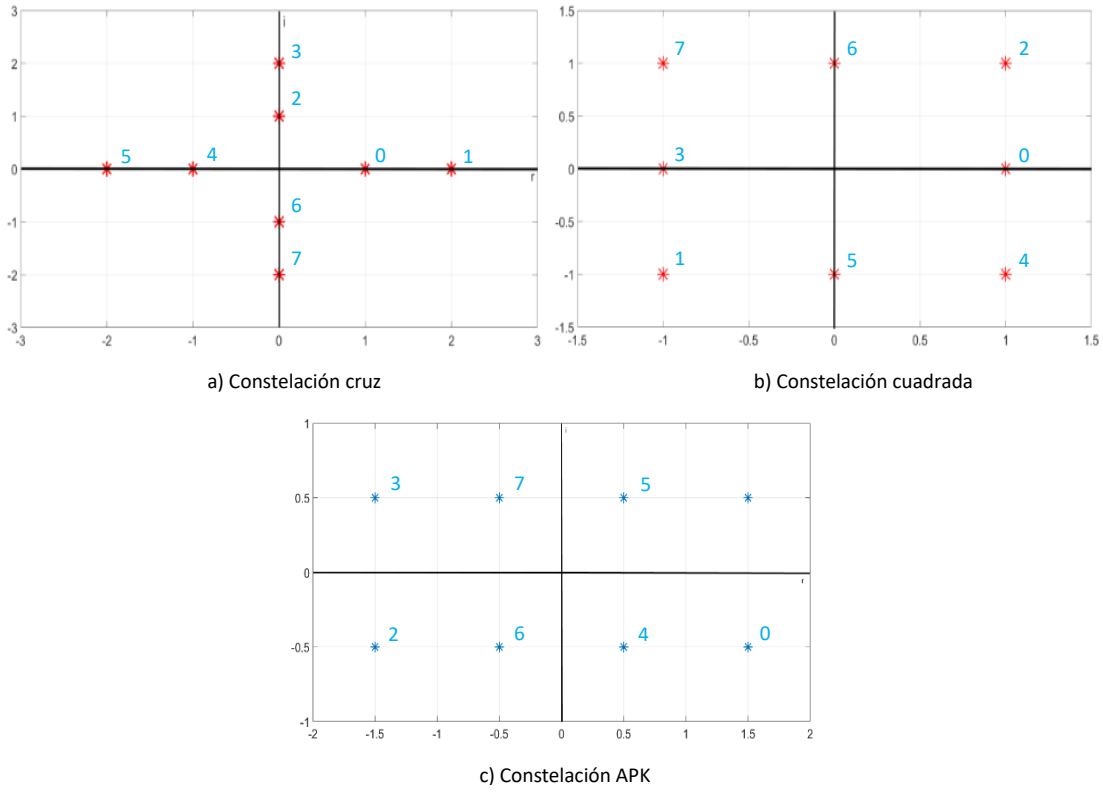
Este bloque espera un pulso negativo en la línea “start”, para luego modular cada uno de los buses de entrada, de forma independiente, usando una modulación 8QAM.

Luego, los datos modulados son enviados a la salida, discriminando las partes real e imaginaria. Los datos de entrada y salida son interpretados usando un formato de punto fijo $Q_{3.0}$ con signo. Para más información sobre este formato, referenciarse al anexo, en la sección “Formato Qm.n con signo”.

Para la modulación 8QAM, se plantearon tres constelaciones, que pueden verse en la Figura 5-6. Se decidió usar la constelación cuadrada, ya que esta ofreció la misma respuesta frente a distintos canales, en las simulaciones, que la constelación APK (las mismas se presentarán en el capítulo próximo), necesitando un bit menos para ser representada en formato $Q_{m.n}$.

La máquina de estados implementada para este bloque puede verse en la Figura 5-7.

Para generar los símbolos 8QAM, el modulador implementa 8 circuitos idénticos, de modo que todos los símbolos se procesen en paralelo. El código de este circuito se muestra a continuación. El mismo le asigna un valor a la salida, dependiendo de la entrada. Las salidas son los valores real e imaginario, en formato $Q_{3.0}$, del punto cartesiano que corresponda con el punto de la constelación numerado igual que la entrada.




a) Constelación cruz

b) Constelación cuadrada

c) Constelación APK

*Figura 5-6*: Constelaciones 8QAM tenidas en cuenta a la hora del diseño

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
```

```
library work;
    use work.fixed_pkg.all;

entity qam8 is
    port(
            inm : in std_logic_vector(2 downto 0);
            re, im: out sfixed(2 downto 0)
            );
end qam8;

architecture cuerpo of qam8 is
begin
    process(inm)
    begin
        if (inm = "000") then
            re <= "001";
            im <= "000";
        elsif (inm = "001") then
            re <= "111";
            im <= "111";
        elsif (inm = "010") then
            re <= "001";
            im <= "001";
        elsif (inm = "011") then
            re <= "111";
            im <= "000";
        elsif (inm = "100") then
            re <= "001";
            im <= "111";
        elsif (inm = "101") then
            re <= "000";
            im <= "111";
        elsif (inm = "110") then
            re <= "000";
            im <= "001";
        elsif (inm = "111") then
            re <= "111";
            im <= "001";
        end if;
    end process;
end cuerpo;
```

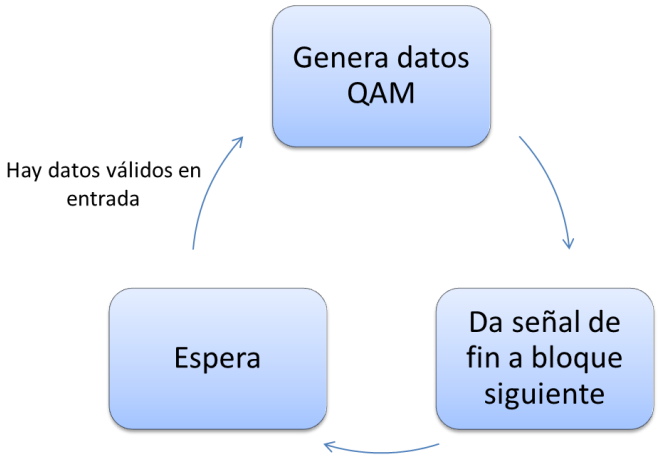



*Figura 5-7:* Máquina de estados del bloque modulador QAM.

### Bloque “bloquefft8”

Este bloque realiza la transformada de Fourier de 8 valores complejos de entrada. Se usa este bloque, y la técnica descripta en el anexo, sección “La transformada discreta de Fourier - DFT”, para obtener la IFFT.

Los datos de entrada se interpretan en formato $Q_{3.0}$, y se ingresan las partes real e imaginaria por separado, al igual que a la salida. Sin embargo, debido a las cuentas que se realizan para obtener la FFT, los coeficientes pares tienen un formato $Q_{3.3}$, y los impares, un formato $Q_{3.17}$. La razón de esto puede entenderse al desarrollar la fórmula de la FFT, para encontrar los valores de las salidas, en función de los valores de entrada. Esto puede verse en el anexo, sección nombrada anteriormente. Allí, vemos que los valores pares de salida solo son sumas de las entradas, mientras que las salidas impares contienen multiplicaciones por un factor $tw$. Para tener un balance entre precisión y velocidad, se decidió discretizar este valor en formato $Q_{2.14}$.

La máquina de estados implementada es la de la Figura 5-8. Puede apreciarse que su estructura es muy similar a la del modulador 8QAM.

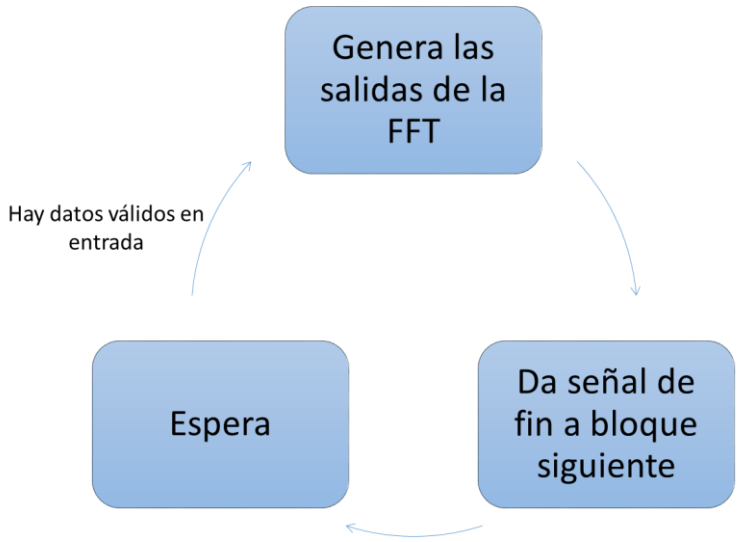



*Figura 5-8:* Máquina de estados del bloque que realiza la operación FFT.

Los códigos de los circuitos que realizan los cálculos de las salidas de la FFT tienen todos el mismo formato. A continuación, vemos a modo de ejemplo, dos de ellos.

Código para cálculo de $X(2)$:

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.fixed_pkg.all;
```

```
entity fft8b is
    generic(
                tw : sfixed(1 downto -14) := "0010110101000001"
        );
    port(
            inar, inai, inbr, inbi, incr, inci, indr, indi: in
sfixed(2 downto 0);
            iner, inei, infr, infi, ingr, ingi, inhr, inhi: in
sfixed(2 downto 0);
            out1r, out1i: out std_logic_vector(26 downto 0)
        );
end fft8b;

architecture cuerpo of fft8b is
signal out1rtmp, out1itmp : sfixed(12 downto -14);
begin
    out1rtmp <= (inar - iner + inci - ingi) + (tw * (inbr + inbi -
infr - infi - indr + indi + inhr - inhi));
    out1itmp <= (inai - inei - incr + ingr) + (tw * (inbi - inbr -
infi + infr - indi - indr + inhi + inhr));
    out1r <= std_logic_vector(out1rtmp);
    out1i <= std_logic_vector(out1itmp);
end cuerpo;
```

Código para cálculo de $X(3)$:

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.fixed_pkg.all;

entity fft8c is
    generic(
                tw : sfixed(1 downto -14) := "0010110101000001"
        );
    port(
            inar, inai, inbr, inbi, incr, inci, indr, indi: in
sfixed(2 downto 0);
            iner, inei, infr, infi, ingr, ingi, inhr, inhi: in
sfixed(2 downto 0);
            out2r, out2i: out std_logic_vector(9 downto 0)
        );
end fft8c;

architecture cuerpo of fft8c is
signal out2rtmp, out2itmp : sfixed(9 downto 0);
begin
    out2rtmp <= inar + iner - incr - ingr + inbi + infi - indi - inhi;
    out2itmp <= inai + inei - inci - ingi - inbr - infr + indr + inhr;
    out2r <= std_logic_vector(out2rtmp);
    out2i <= std_logic_vector(out2itmp);
end cuerpo;
```

## Demodulador OFDM

El diseño del demodulador, en FPGA, consiste de 7 bloques, que representan 7 circuitos distintos. El mismo puede verse en la Figura 5-9. Las entradas al demodulador

son: “reset”, que trabaja con lógica negada (se activa cuando la línea tiene un valor lógico bajo); “clk50”, un reloj de 50 MHz, incluido en la placa de desarrollo; y “uart_rxd”, que se corresponde con la línea de recepción UART. Este diseño cuenta con una sola salida, “uart_txd”, encargada de manejar la línea de transmisión UART. El demodulador espera recibir datos modulados por UART, en un orden y formato específicos. Luego, a esos datos se les aplica una transformada de Fourier, y su salida es ingresada a un demodulador 8QAM. Todos los bloques fueron diseñados como máquinas de estado, usando el reloj de 50 MHz.

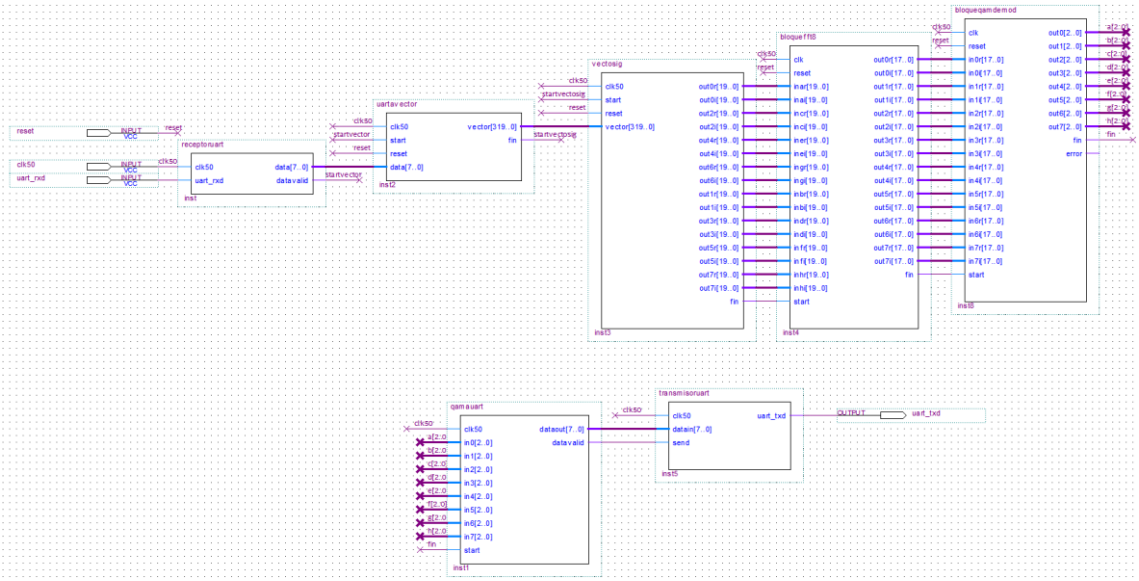



*Figura 5-9:* Diagrama completo, visto en Quartus, del demodulador OFDM

Los bloques “receptoruart”, “uartavector”, “vectosig”, “qamauart” y “transmisoruart” se usan para transmitir, recibir, y poner en orden, los datos que son recibidos y enviados por UART, ya que este tipo de comunicación es serie, pero el resto de los bloques toma las entradas en paralelo, y dispone sus salidas también en paralelo. La descripción de los mismos puede verse en el anexo, en la sección “Bloques Quartus auxiliares”.

Al igual que en el modulador, Quartus nos ofrece un resumen de los recursos de la placa utilizados por este diseño. Este puede verse en la Figura 5-10. Observamos que la utilización de elementos lógicos supera ampliamente a la del modulador, aunque no llegando al 10% de los recursos totales de la placa. Esta diferencia en la cantidad de elementos lógicos utilizados puede explicarse por la mayor cantidad de bits que debe manejar este diseño. Además, al tener que hacer comparaciones de números en formato punto fijo, con muchos bits (en el bloque responsable de la demodulación

8QAM), hace que el circuito sea considerablemente más amplio que en el modulador.

| | |
|---|---|
| Flow Status | Successful - Thu Mar 23 11:50:42 2017 |
| Quartus Prime Version | 16.1.1 Build 200 11/30/2016 SJ Lite Edition |
| Revision Name | ofdm8receptor |
| Top-level Entity Name | ofdm8receptor |
| Family | Cyclone V |
| Device | 5CGXFC5C6F27C7 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 2,155 / 29,080 ( 7 % ) |
| Total registers | 1329 |
| Total pins | 4 / 364 ( 1 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 4,567,040 ( 0 % ) |
| Total DSP Blocks | 8 / 150 ( 5 % ) |
| Total HSSI RX PCSs | 0 / 6 ( 0 % ) |
| Total HSSI PMA RX Deserializers | 0 / 6 ( 0 % ) |
| Total HSSI TX PCSs | 0 / 6 ( 0 % ) |
| Total HSSI PMA TX Serializers | 0 / 6 ( 0 % ) |
| Total PLLs | 0 / 12 ( 0 % ) |
| Total DLLs | 0 / 4 ( 0 % ) |

*Figura 5-10:* Resumen de los recursos utilizados de la placa, por el diseño del demodulador.

En esta sección, analizaremos el funcionamiento de los bloques responsables de la demodulación de los datos de entrada.

### Bloque “fft8”

Este bloque realiza las mismas acciones que el bloque homónimo, presente en el modulador. La única diferencia es el formato con el que son interpretadas las entradas y salidas. Las entradas se interpretan con formato $Q_{3.17}$. Esto se debe a que, en esta instancia, los símbolos pueden haber sufrido ligeras modificaciones, debido al ruido presente en el medio. De haberse utilizado una cantidad pequeña de bits para representar la parte fraccionaria, se hubiera redondeado la señal con ruido, provocando que el sistema falle en la demodulación.

Experimentalmente, se llegó a la conclusión de que 17 bits de parte fraccionaria son suficientes para representar nuestra señal, con ruido agregado. Se utilizaron 3 bits de parte entera, por más que nuestros símbolos tengan un valor máximo de “1”, para evitar que el ruido sature la capacidad del formato punto fijo utilizado. De esta manera, ruidos mayores a los $4{,}77\ dB$ (respecto a la señal), no provocarán que esto suceda. Las salidas son interpretadas con formato $Q_{5.13}$.

### Bloque “qamdemod”

Este bloque toma los símbolos recibidos, luego de ser transformados, y los interpreta en formato $Q_{5.13}$. Al suceder un pulso negativo en “start”, estos símbolos son demodulados en paralelo, por 8 circuitos idénticos. Esta demodulación se hace

calculando, a partir de las partes real e imaginaria de cada símbolo, qué punto de la constelación se encuentra más cercana. Finalmente, las partes real e imaginaria correspondientes a ese punto cartesiano, son enviadas a la salida. La constelación usada fue la cuadrada, que puede verse en la Figura 5-6.

La máquina de estados implementada por este bloque puede verse en la Figura 5-11.

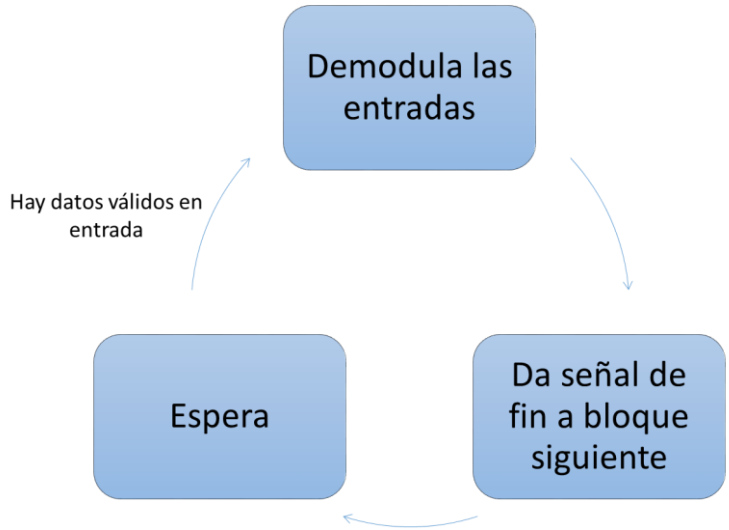



*Figura 5-11:* Máquina de estados correspondiente al demodulador QAM..

A continuación, se encuentra el código del circuito encargado de reconocer qué punto de la constelación, en el plano cartesiano, se encuentra más cercano al símbolo recibido.

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

library work;
    use work.fixed_pkg.all;

entity qam8demod is
    generic(
        cerocinco : sfixed(0 downto -1) := "01";
        menoscerocinco : sfixed(0 downto -1) := "11";
        unocinco : sfixed(1 downto -1) :=  "011";
        menosunocinco : sfixed(1 downto -1) := "101"
    );
    port(
        inr, ini : in std_logic_vector(17 downto 0);
        salida : out std_logic_vector(2 downto 0);
        error : out std_logic := '1'
    );
end qam8demod;

architecture cuerpo of qam8demod is
```

```
signal inrpf, inipf : sfixed(4 downto -13);
begin

    inrpf <= to_sfixed(inr,4,-13);
    inipf <= to_sfixed(ini,4,-13);

    process(inrpf,inipf)
    begin

        if ((abs(inrpf)) >= (abs(inipf))) then
            if (inrpf >= 0) then
                if (inipf >= cerocinco) then
                    salida <= "010";
                elsif (inipf <= menoscerocinco) then
                    salida <= "100";
                else
                    salida <= "000";
                end if;
            else
                if (inipf >= cerocinco) then
                    salida <= "111";
                elsif (inipf <= menoscerocinco) then
                    salida <= "001";
                else
                    salida <= "011";
                end if;
            end if;
        else
            if (inipf >= 0) then
                if (inrpf >= cerocinco) then
                    salida <= "010";
                elsif (inrpf <= menoscerocinco) then
                    salida <= "111";
                else
                    salida <= "110";
                end if;
            else
                if (inrpf >= cerocinco) then
                    salida <= "100";
                elsif (inrpf <= menoscerocinco) then
                    salida <= "001";
                else
                    salida <= "101";
                end if;
            end if;
        end if;
    end process;

end cuerpo;
```

# 6. Simulaciones

Previo a la implementación de los sistemas en la FPGA, se simularon las tres constelaciones presentadas en la Figura 5-6*: Constelaciones 8QAM tenidas en cuenta a la hora del diseño*, para ver cuál es la que mejor responde a las necesidades de este proyecto. Se simuló la curva que relaciona la probabilidad binaria de error $BER$, en función de la relación señal a ruido de los bit $(E_b/N_0)_{dB}$, para dos medios. Uno de los medios fue un canal Gaussiano. En el mismo, los símbolos sufren de interferencia por ruido Gaussiano. El otro medio, más acorde a la utilización real de este sistema, es el canal de Rayleigh, que, además de agregar ruido Gaussiano, tiene en cuenta posibles reflexiones en el medio. Estas reflexiones son simuladas multiplicando a los símbolos por una variable aleatoria con densidad de probabilidad exponencial.

Para realizar las simulaciones, se cumplió el siguiente procedimiento:

1- Fijar la cantidad "$nSym$" de símbolos a simular, por cada $(E_b/N_0)_{dB}$.
2- Fijar una $(E_b/N_0)_{dB}$.
3- Generación de 24 bits aleatorios.
4- Modulación 8QAM de los 24 bits aleatorios, con la constelación a simular.
5- Transformación de los símbolos 8QAM a símbolos OFDM, utilizando la IFFT.
6- Intervención del canal simulado, degenerando los símbolos OFDM, de manera normalizada, para poder comparar los resultados de las distintas constelaciones, utilizando una relación $(E_b/N_0)_{dB}$ prefijada.
7- Transformación de los símbolos recibidos, utilizando la FFT.
8- Demodulación 8QAM de los símbolos transformados, para recuperar los datos generados en el paso 3.
9- Cálculo de cantidad de errores cometidos, comparando los datos recibidos con los enviados.
10- Aumenta el contador de errores totales para la $(E_b/N_0)_{dB}$ prefijada.
11- Si no se simularon "$nSym$" símbolos para la $(E_b/N_0)_{dB}$ dada, se vuelve al paso 3. Si no, sigue al paso siguiente.
12- Cálculo de BER para la $(E_b/N_0)_{dB}$ utilizada. Esta se calcula como la cantidad de errores cometidos, dividido por la cantidad de bits transmitidos.
13- Cambio de la $(E_b/N_0)_{dB}$ y vuelta al paso 3.

En todas las simulaciones, se tomó como válida la $BER$ calculada, siempre y cuando la cantidad de errores cometidos en la demodulación fuera igual o mayor que 200.

A continuación, veremos los códigos y resultados de las simulaciones de nuestro sistema, en los medios mencionados anteriormente, para las distintas constelaciones planteadas. Las funciones “bintodec”, “binuntodec” y “dectobin” fueron diseñadas para conversión de formatos, según fueron necesarios, y se funcionamiento puede verse en el anexo, sección “Códigos auxiliares Matlab”.

## Canal Gaussiano

La primera simulación hecha con el canal Gaussiano, fue la de la constelación APK, obteniendo una curva como la Figura 6-1.

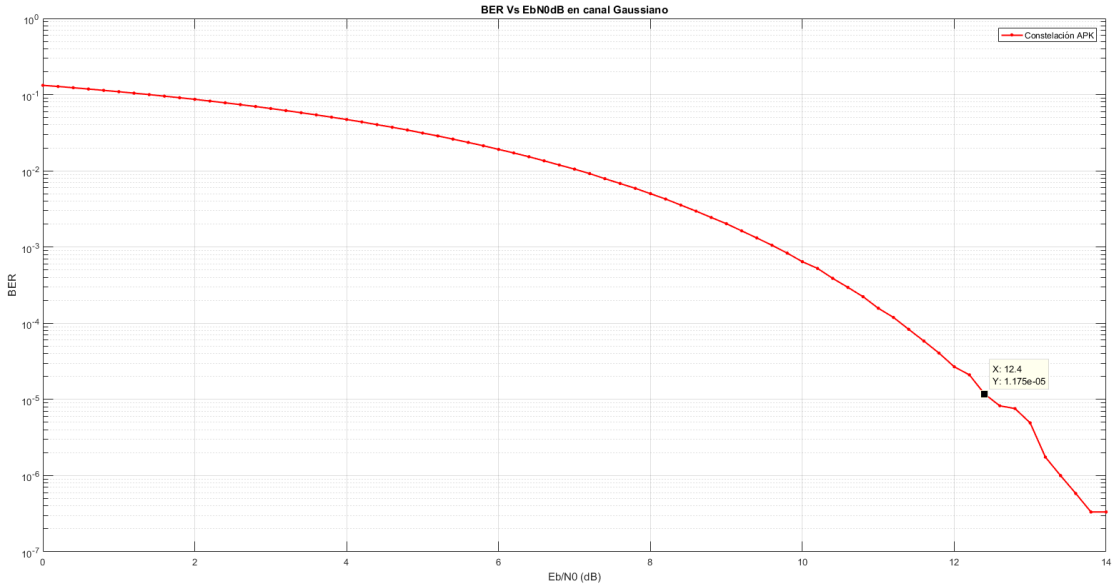



*Figura 6-1:* Simulación del sistema OFDM, en canal Gaussiano, utilizando la constelación APK.

Luego, se simularon las constelaciones cuadrada y cruz. Las curvas resultantes

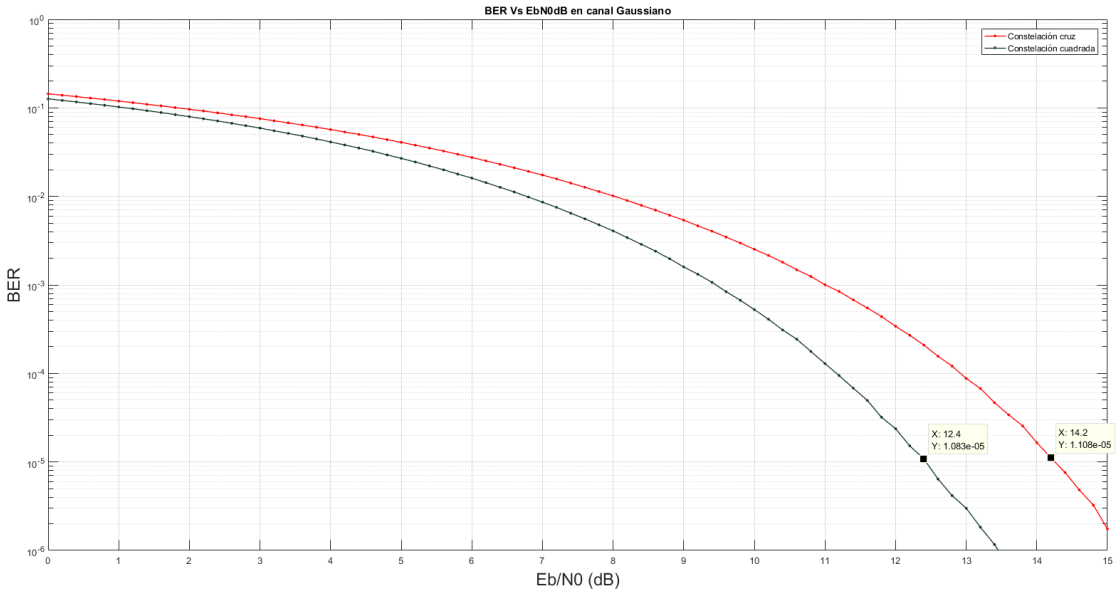



*Figura 6-2:* Simulaciones del sistema OFDM, en canal Gaussiano, para las constelaciones cruz y cuadrada.

se encuentran en la Figura 6-2.

En este momento, podemos comparar las curvas de las tres constelaciones. Mientras que la constelación cruz demostró la peor resistencia frente al ruido Gaussiano, las constelaciones cuadrada y APK tuvieron un rendimiento muy similar entre sí.

Los códigos para el cálculo de estas curvas son similares en los tres casos, cambiando los valores para modular, y las comparaciones sobre los símbolos recibidos, para demodular. A continuación, veremos el código para la simulación de la constelación APK:

```
nSym = 500000;      % Número de símbolos por SNR a simular
EbN0dB = 0:0.2:16;  % SNRs a simular
N = 8;

EsN0dB = EbN0dB + 10*log10(3);  % Traduce SNR del bit a SNR del símbolo
errores = zeros(1,length(EsN0dB));

for j=1:length(EbN0dB)  % Recorre el vector con las SNR a simular.
    for k=1:nSym

        %Transmisor
        datosbin = round(rand(1,3*N));
        s = zeros(1,N);
        for l=1:N
            switch binuntodec(datosbin(1,3*l-2:3*l))
                case 0
                    s(1,l) = 1.5-0.5i;
                case 1
                    s(1,l) = 1.5+0.5i;
                case 2
                    s(1,l) = -1.5-0.5i;
                case 3
                    s(1,l) = -1.5+0.5i;
                case 4
                    s(1,l) = 0.5-0.5i;
                case 5
                    s(1,l) = 0.5+0.5i;
                case 6
                    s(1,l) = -0.5-0.5i;
                case 7
                    s(1,l) = -0.5+0.5i;
            end
        end

        X_Frec = s;
        ofdm_senal = ifft(X_Frec);
```

```
        %Canal
        varruido = 1/(10.^(EsN0dB(j)/10));  % Calcula la varianza de ruido
        sigmaruido = sqrt(varruido/2);
        ruido = sigmaruido * (randn(1,length(ofdm_senal)) +
1i*randn(1,length(ofdm_senal)));    % Genera ruido Gaussiano
        r = ofdm_senal + ruido * sqrt(0.1875) ; % Suma el ruido, normalizado por la
varianza de la señal

        %Receptor
        R_Frec = fft(r);
        s_cap = zeros(1,N);
        for m=1:N
            if (imag(R_Frec(1,m)) > 0)
                if (real(R_Frec(1,m)) > 1)
                    s_cap(m) = 1;
                    s_capbin(1,3*m-2:3*m) = [0 0 1];
                else if (real(R_Frec(1,m)) > 0)
                        s_cap(m) = 5;
                        s_capbin(1,3*m-2:3*m) = [1 0 1];
                    else if (real(R_Frec(1,m)) > -1)
                        s_cap(m) = 7;
                        s_capbin(1,3*m-2:3*m) = [1 1 1];
                        else
                            s_cap(m) = 3;
                            s_capbin(1,3*m-2:3*m) = [0 1 1];
                        end
                    end
                end
            else
                if (real(R_Frec(1,m)) > 1)
                    s_cap(m) = 0;
                    s_capbin(1,3*m-2:3*m) = [0 0 0];
                else if (real(R_Frec(1,m)) > 0)
                        s_cap(m) = 4;
                        s_capbin(1,3*m-2:3*m) = [1 0 0];
                    else if (real(R_Frec(1,m)) > -1)
                        s_cap(m) = 6;
                        s_capbin(1,3*m-2:3*m) = [1 1 0];
                        else
                            s_cap(m) = 2;
                            s_capbin(1,3*m-2:3*m) = [0 1 0];
                        end
                    end
                end
            end
        end

        numErrors = sum(sum(xor(s_capbin,datosbin)));   % Calcula cantidad de errores de
la corrida actual
        errores(j) = errores(j) + numErrors;        % Aumenta la cuenta de errores
totales, para la SNR actual
```

```
    end
end
simBer = errores/(nSym*24); % Calcula la BER
figure(1)
semilogy(EbN0dB,simBer,'r-o');
grid on
title('BER Vs EbN0dB');
xlabel('Eb/N0 (dB)'); ylabel('BER');
```

Para las otras constelaciones, solo presentaremos los códigos para modular y demodular los símbolos.

Códigos para modulación y demodulación de símbolos, para constelación cuadrada:

```
%Transmisor
 datosbin = round(rand(1,3*N));
 s = zeros(1,N);
 for l=1:N
     switch binuntodec(datosbin(1,3*l-2:3*l))
         case 0
             s(1,l) = 1;
         case 1
             s(1,l) = -1-1i;
         case 2
             s(1,l) = 1+1i;
         case 3
             s(1,l) = -1;
         case 4
             s(1,l) = 1-1i;
         case 5
             s(1,l) = -1i;
         case 6
             s(1,l) = 1i;
         case 7
             s(1,l) = -1+1i;
     end
 end

%Receptor
 R_Frec = fft(r);
 s_cap = zeros(1,N);
 for m=1:N
     if (real(R_Frec(1,m)) > 0.5) && (imag(R_Frec(1,m)) > 0.5)
         s_cap(m) = 2;
         s_capbin(1,3*m-2:3*m) =[0 1 0];
     else if (real(R_Frec(1,m)) > 0.5) && (imag(R_Frec(1,m)) < -0.5)
             s_cap(m) = 4;
             s_capbin(1,3*m-2:3*m) = [1 0 0];
         else if (real(R_Frec(1,m)) < -0.5) && (imag(R_Frec(1,m)) > 0.5)
```

```
                s_cap(m) = 7;
                s_capbin(1,3*m-2:3*m) = [1 1 1];
            else if (real(R_Frec(1,m)) < -0.5) && (imag(R_Frec(1,m)) < -0.5)
                    s_cap(m) = 1;
                    s_capbin(1,3*m-2:3*m) = [0 0 1];
                else if abs(real(R_Frec(1,m))) > abs(imag(R_Frec(1,m)))
                        if real(R_Frec(1,m)) > 0
                            s_cap(m) = 0;
                            s_capbin(1,3*m-2:3*m) = [0 0 0];
                        else
                            s_cap(m) = 3;
                            s_capbin(1,3*m-2:3*m) = [0 1 1];
                        end
                    else
                        if imag(R_Frec(1,m)) > 0
                            s_cap(m) = 6;
                            s_capbin(1,3*m-2:3*m) = [1 1 0];
                        else
                            s_cap(m) = 5;
                            s_capbin(1,3*m-2:3*m) = [1 0 1];
                        end
                    end
                end
            end
        end
    end
```

Códigos para modulación y demodulación de símbolos, para constelación cruz:

```
%Transmisor
datosbin = round(rand(1,3*N));
s = zeros(1,N);
for l=1:N
    switch binuntodec(datosbin(1,3*l-2:3*l))
        case 0
            s(1,l) = 1;
        case 1
            s(1,l) = 2;
        case 2
            s(1,l) = 1i;
        case 3
            s(1,l) = 2i;
        case 4
            s(1,l) = -1;
        case 5
            s(1,l) = -2;
        case 6
            s(1,l) = -1i;
        case 7
            s(1,l) = -2i;
    end
end
```

```
%Receptor
R_Frec = fft(r);
for m=1:N
    s_cap = zeros(1,N);
    if ((abs(real(R_Frec(1,m)))) > (abs(imag(R_Frec(1,m)))))
        if (real(R_Frec(1,m)) < -1.5)
            s_cap(m) = -2;
            s_capbin(1,3*m-2:3*m) = [1 0 1];
        else
            if (real(R_Frec(1,m)) < 0)
                s_cap(m) = -1;
                s_capbin(1,3*m-2:3*m) = [1 0 0];
            else
                if (real(R_Frec(1,m)) > 1.5)
                    s_cap(m) = 2;
                    s_capbin(1,3*m-2:3*m) = [0 0 1];
                else
                    s_cap(m) = 1;
                    s_capbin(1,3*m-2:3*m) = [0 0 0];
                end
            end
        end
    else
        if (imag(R_Frec(1,m)) < -1.5)
            s_cap(m) = -2i;
            s_capbin(1,3*m-2:3*m) = [1 1 1];
        else
            if (imag(R_Frec(1,m)) < 0)
                s_cap(m) = -1i;
                s_capbin(1,3*m-2:3*m) = [1 1 0];
            else
                if (imag(R_Frec(1,m)) > 1.5)
                    s_cap(m) = 2i;
                    s_capbin(1,3*m-2:3*m) = [0 1 1];
                else
                    s_cap(m) = 1i;
                    s_capbin(1,3*m-2:3*m) = [0 1 0];
                end
            end
        end
    end
end
```

## Canal de Rayleigh

El segundo medio simulado fue el canal de Rayleigh. El mismo, además de agregar ruido Gaussiano, tiene cuenta posibles reflexiones que pudieran suceder en el medio, a partir de la multiplicación de cada símbolo por una variable aleatoria con distribución exponencial.

En la Figura 6-3, podemos ver la simulación de la curva de rendimiento de la

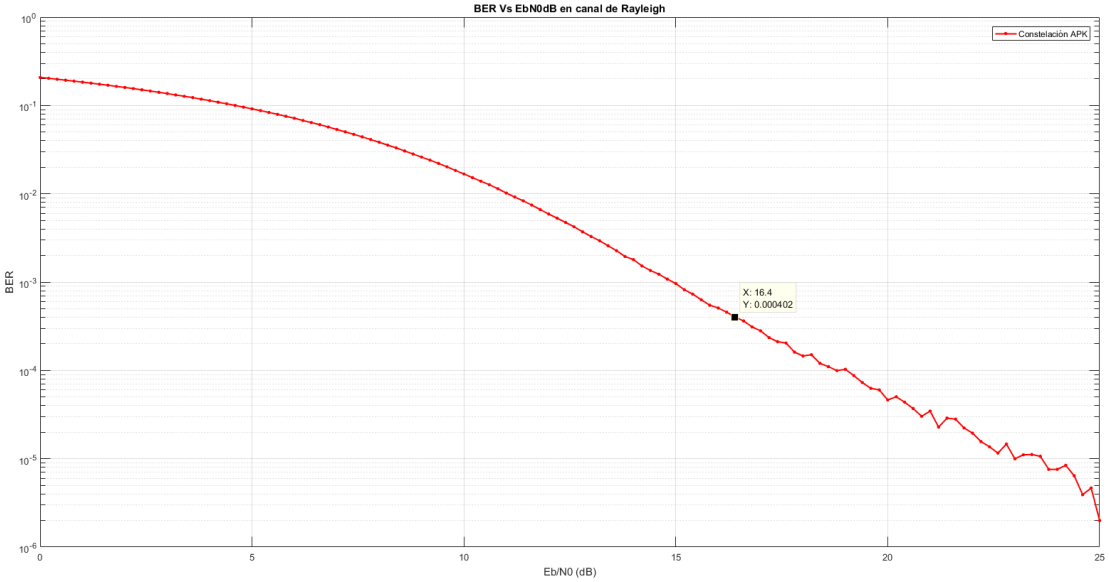



*Figura 6-3:* Simulación del sistema OFDM, en canal de Rayleigh, utilizando la constelación APK.

constelación APK, en un canal de Rayleigh. Como era de esperarse, se observa que la misma sufre de una peor degradación que en el canal Gaussiano.

En la Figura 6-4, podemos ver la simulación de la curva de rendimiento de las constelaciones cuadrada y cruz, en un canal de Rayleigh.

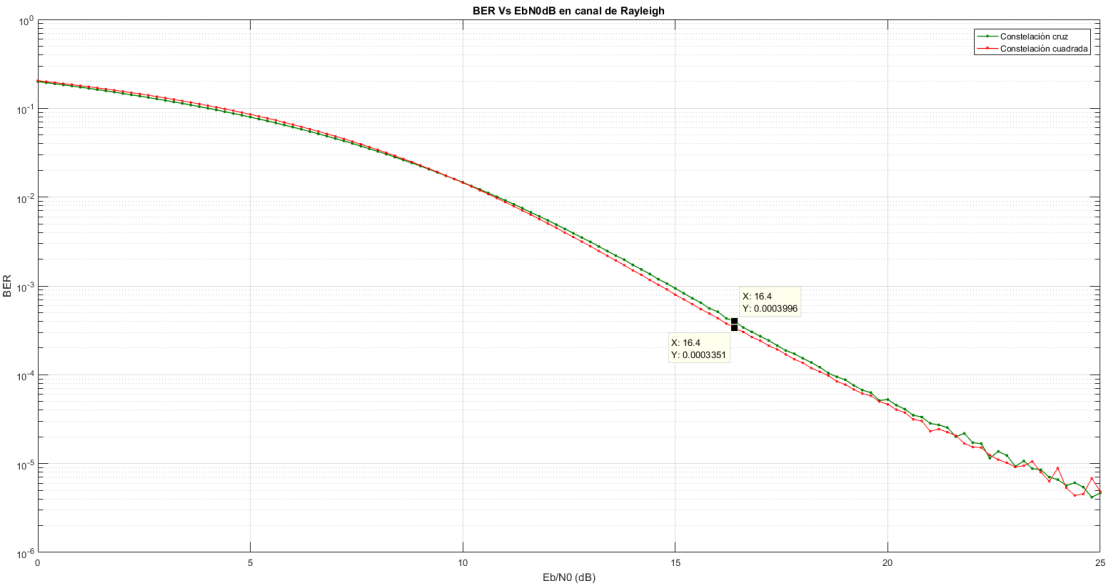



*Figura 6-4:* Simulaciones del sistema OFDM, en canal de Rayleigh, para las constelaciones cruz y cuadrada.

Sorpresivamente, las tres constelaciones mostraron resultados prácticamente idénticos.

A partir de estos resultados, se decidió utilizar la constelación cuadrada, ya que los puntos de la constelación de la misma requieren menor cantidad de bits para ser

representado en formato $Q_{m.n}$. Esto se traduce en un diseño más simple, y en un circuito más eficiente, ya que disminuye la circuitería necesaria.

Los códigos para las simulaciones son análogos a los del canal Gaussiano, con la diferencia de la parte del mismo que se encarga de simular el medio. En el canal de Rayleigh, los símbolos son multiplicados por variables aleatorias con distribución exponencial, normalizados por la media cuadrática de las señales.

A continuación, vemos el código para la simulación del medio, para la constelación APK:

```
        %Canal
         varruido = 1/(10.^(EsN0dB(j)/10));  % Calcula la varianza del ruido
         sigmaruido = sqrt(varruido/2);
         rayleigh = sqrt(exprnd(0.433,[1,8])) + 1i*sqrt(exprnd(0.433,[1,8])); % Genera
variables exponenciales
         ruido = sigmaruido * (randn(1,length(ofdm_senal)) +
1i*randn(1,length(ofdm_senal))); % Genera ruido Gaussiano
         r = ofdm_senal .* rayleigh + ruido * sqrt(0.1875) ; % Afecta a la señal por
canal de Raylegih
         r = r ./ rayleigh;  % Divide el resultado por las variables exponenciles
```

Los códigos para las constelaciones cuadrada y cruz son similares al mostrado, cambiando la normalización del ruido y de las variables exponenciales.

# 7. Resultados experimentales

Para la comprobación funcional de los diseños en FPGA se realizaron pruebas, con ayuda de una PC. Se crearon dos funciones en MATLAB, llamadas “parteuno” y “partedos”. La primera, recibe $nSym$ símbolos modulados por la FPGA, luego de enviar $24 . nSym$ bits aleatorios a la misma. También entrega los datos binarios generados. La segunda función se encarga de enviar $nSym$ símbolos modulados a la FPGA, y recibir los datos demodulados de la misma. Estos datos demodulados deberían ser idénticos a los datos generados por la función anterior, si se le ingresan los símbolos generados por “parteuno”. Los códigos de estas funciones pueden verse en el anexo, sección “Códigos auxiliares Matlab”.

La primera prueba que se realizó fue la de modular una gran cantidad de datos en la FPGA, generados por la PC, y luego demodularlos nuevamente con la FPGA. Como era esperable, todos los datos generados fueron demodulados sin errores.

Luego, se prosiguió a degenerar los símbolos modulados por la FPGA, según alguno de los dos medios tenidos en cuenta. Estos símbolos degenerados fueron luego demodulados por la FPGA, y recibidos por la computadora, para poder trazar las curvas previamente simuladas. Se implementó la constelación cuadrada ya que, al simular, tuvo una performance idéntica a la constelación APK, aun cuando su diseño en FPGA es más simple.

A continuación, vemos los códigos y resultados de estas pruebas.

## Prueba sobre canal Gaussiano

Para realizar esta prueba, se degeneraron los símbolos, agregándoles ruido Gaussiano, de manera similar a la hecha en las simulaciones. Los símbolos fueron previamente generados por la FPGA.

El código usado en MATLAB es el siguiente:

```
nSym = 50000;  % Cantidad de símbolos a simular, por cada SNR
EbN0dB =0:10;   % SNRs a simular.
N = 8;          % Bits por Símbolo

EsN0dB = EbN0dB + 10*log10(3);  % Transforma SNR de bit a SNR de símbolo
errores = zeros(1,length(EsN0dB));

for k=1:length(EbN0dB)  % Recorre el vector de SNRs a simular
    varruido = 1/(10.^(EsN0dB(k)/10));  % Calcula varianza de ruido
```

```
    sigmaruido = sqrt(varruido/2);
    ruido = sigmaruido * (randn(nSym,8) + 1i*randn(nSym,8));
    r = medio50000 + ruido * sqrt(0.1875) ; % Agrega ruido a los símbolos calculados por
el modulador
    s_capbin = partedos(r,nSym); % Envía símbolos degenerados, y recibe datos
demodulados
    numErrors = sumaerrores(s_capbin,datos50000);   % Calcula errores para la SNR actual

    errores(k) = errores(k) + numErrors;
end

simBer = errores/(nSym*24); % Calcula la BER
figure(1)
semilogy(EbN0dB,simBer,'r-o');  % Grafica la BER, en función de la SNR
grid on
title('BER Vs EbN0dB');
xlabel('Eb/N0 (dB)'); ylabel('BER');
```

La curva resultante puede verse en la Figura 7-1.

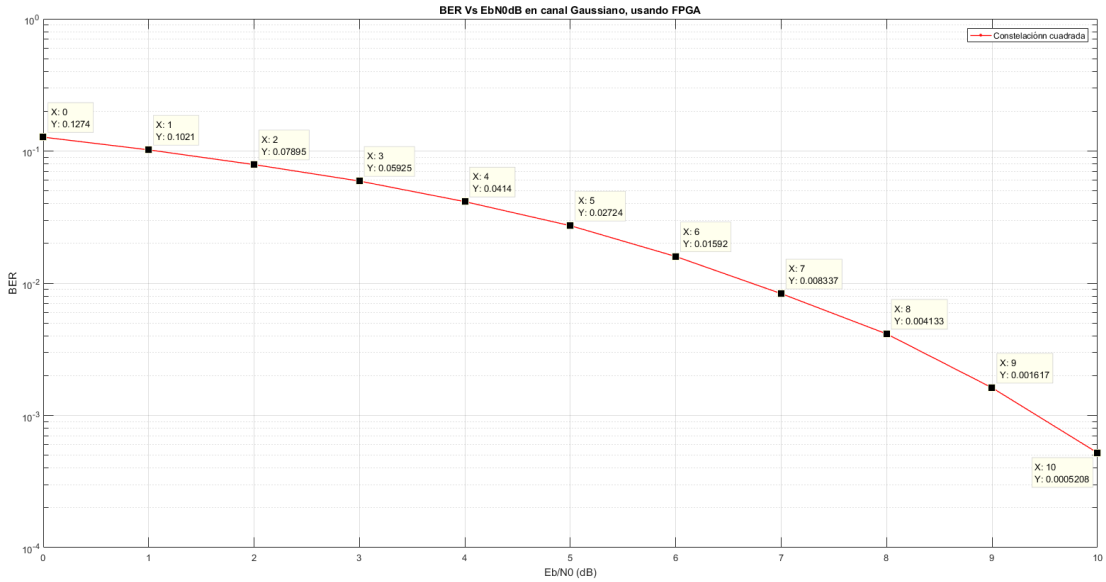



*Figura 7-1:* Curva de rendimiento para el sistema OFDM, modulando y demodulando con FPGA, para canal Gaussiano.

Esta curva es muy similar a la simulada, por lo que podemos inferir que el sistema funciona correctamente para un medio Gaussiano.

## Prueba sobre canal de Rayleigh

Esta prueba es similar a la anterior, con la diferencia de que los símbolos fueron degenerados en consistencia con un canal de Rayleigh, de igual manera que en la simulación de este sistema.

La Figura 7-2 muestra el resultado de esta prueba.

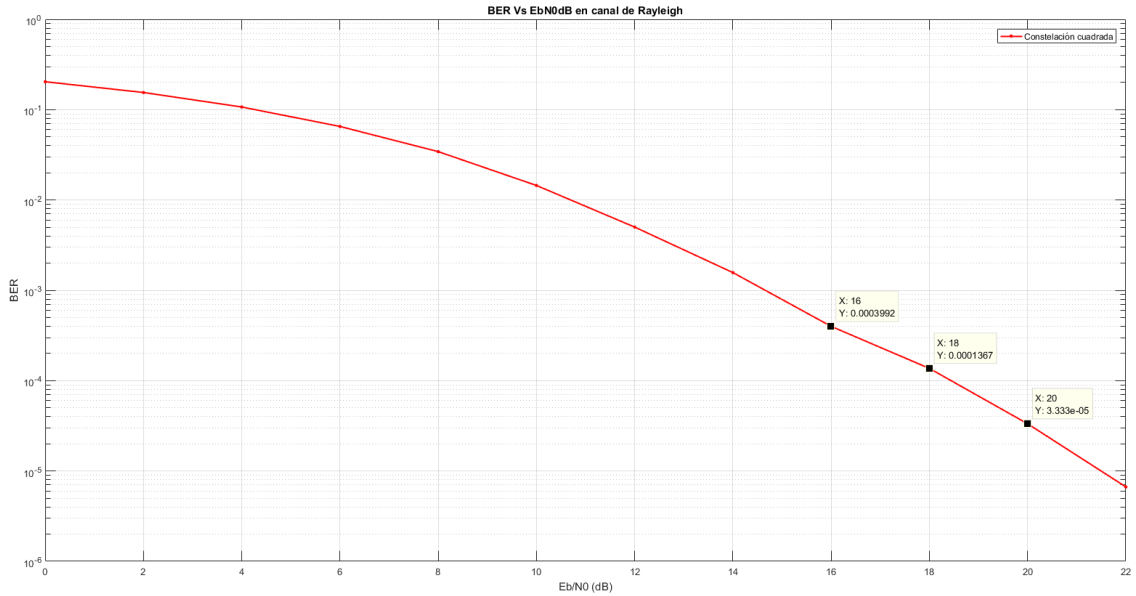



*Figura 7-2:* Curva de rendimiento para el sistema OFDM, modulando y demodulando con FPGA, para canal de Rayleigh.

Al igual que en caso de canal Gaussiano, esta curva se adapta perfectamente a la simulada.

# 8. Conclusiones

En este proyecto, se desarrolló exitosamente un conjunto modulador/demodulador OFDM de 8 subportadoras, en una plataforma FPGA, en lenguaje VHDL.

Además del desafío intelectual de comprender los conceptos que requiere este sistema de comunicaciones, se sumaron otros contratiempos, propios de desarrollos tecnológicos. Uno de estos contratiempos fue la necesidad de trabajar a nivel bit, como requiere el lenguaje VHDL. De esta manera, se debió razonar la mejor manera de realizar cada una de las operaciones, teniendo en cuenta que el código VHDL sería luego traducido en un circuito. Además, se debió realizar código que desarrolle operaciones complejas, como la DFT. Esto le dio una dificultad adicional al desarrollo del proyecto. Algunas de estas operaciones podrían haberse resuelto usando IP Cores, bloques pre programados por Altera. Sin embargo, se optó por el desarrollo manual de los mismo, ya que los IP Cores son licenciados y, si se desearan usar para un desarrollo comercial, se debería pagar una licencia.

Teniendo en cuenta la magnitud de los diseños implementados en FPGA, si se deseara diseñar una placa que sirva para el propósito de este proyecto, se podrían utilizar dispositivos FPGA más económicos que el presente en nuestra placa de desarrollo. Por ejemplo, se podría usar el Cyclone IV EP4CE6E22C8N, que contiene la cantidad de necesaria de elementos lógicos y multiplicadores, a un precio de 11 dólares.

Una vez finalizado el proyecto, surgieron algunas mejores que podrían implementarse en el futuro. Entre ellas se encuentran:

1- Uso de herramientas para desarrollo en FPGA, en lenguaje C: Estas herramientas permiten un desarrollo mucho más grande que el lenguaje VHDL, a un costo de mayor uso de recursos del dispositivo FPGA. Para esto, Altera ofrece un SDK (en inglés, “Software Development Kit”), que se integra a Quartus, para la utilización de librerías OpenCL (librerías basadas en C, utilizadas para la realización de algoritmos que permitan la ejecución de diversas tareas en paralelo).

2- Uso de placas de desarrollo SOC: estas placas contienen, además de un dispositivo FPGA, uno o más microcontroladores. De esta manera, se tiene una plataforma más versátil, donde puede elegirse qué dispositivo se encargará de cuales tareas.

3- Desarrollo de sistemas OFDM con más subportadoras. Esto permitiría un sistema aún más rápido, y más inmune al fading, que el desarrollado.

Por ejemplo, los sistemas actuales de internet móvil LTE, usan hasta 2048 subportadoras.

4- Agregado de codificación adicional: podrían agregarse métodos de codificación adicionales (como prefijo y sufijo cíclicos). Estos le darían al sistema una inmunidad adicional frente a las reflexiones del medio.

Todas estas mejoras pueden desarrollarse en un futuro, ya que la totalidad de este proyecto quedará a disposición del Laboratorio de Comunicaciones, del Departamento de Electrónica, en la Universidad Nacional de Mar del Plata. Esto abre una posibilidad para futuros alumnos que deseen adentrarse en el estudio de sistemas multiportadora, o en el desarrollo de diseños en FPGA.

# 9. Bibliografía

[1] **A. Bruce Carlson, Paul B. Crilly, Janet C. Rutledge**, *Communications Systems: An Introduction to Signals and Noise in Electrical Communication*, Estados Unidos de América, 2002, 4ta. Edición.

[2] **Jorge Castiñeira Moreira**, *Comunicaciones Digitales: Apunte completo de Cátedra*, apunte de la materia "Comunicaciones Digitales", Universidad Nacional de Mar del Plata.

[3] **Mónica Cristina Liberatori**, *Redes de datos y sus Protocolos*, Mar del Plata, Editorial de la Universidad Nacional de Mar del Plata, 2016.

[4] **Mathuranathan Viswanathan**, *Simulation of Digital Communication Systems Using Matlab*, Amazon, 2013, 2da. Edición.

[5] **E. Oran Brigham**, *The Fast Fourier Transform*, Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1974.

[6] **Ye (Geoffrey) Li, Gordon L. Stuber**, *Orthogonal Frequency Divison Multiplexing for Wireless Communications,* Georgia Institute of Technology, Springer, 2006.

[7] **Tzi-Dar Chiueh, Pey-Yun Tsai**, *OFDM Baseband Receiver Design for Wireless Communications*, John Wiley & Sons Pte. Ltd, 2007

[8] **Ahmad R. S. Bahain, Burton R. Saltzberg, Mustafa Ergen,** *Multi-Carrier Digital Communications Theory and Applications of OFDM,* Springer, 2004

[9] **Ramjee Prasad**, *OFDM for Wireless Communications Systems,* Artech House, Inc, 2004

[10] **Yong Soo Cho, Jaekwon Kim, Won Young Yang, Chung-Gu Kang,** MIMO-OFDM Wireless Communications with MATLAB

# 10. Anexo

## La transformada discreta de Fourier - DFT

La transformada discreta de Fourier DFT es un caso especial de la transformada de Fourier, de forma que la misma pueda ser computada por una máquina.

Para comenzar, consideremos una señal continua $h(t)$. El primer paso es discretizar esta señal, multiplicándola por un tren de pulsos de período T, que llamaremos $\Delta_0(t)$. Así,

$$h(t).\Delta_0(t) = h(t).\sum_{k=-\infty}^{\infty}\delta(t-k.T)$$

Seguido, la función muestreada es truncada, multiplicándola por una función rectangular $x(t)$, definida como:

$$x(t) = 1\ cuando\ -\frac{T}{2} < t < T_0 - \frac{T}{2}$$

$$x(t) = 0, en\ otro\ caso$$

Donde $T_0$ es igual a la duración de la función truncada. Esta función comienza antes de $t = 0$ y finaliza antes de $t = T_0$ para evitar confusión sobre qué muestras son usadas luego de truncar.

De esta manera,

$$h(t).\Delta_0(t).x(t) = \left[h(t).\sum_{k=-\infty}^{\infty}\delta(t-k.T)\right].x(t) = \sum_{k=0}^{N-1}h(k.T).\delta(t-k.T)$$

Donde asumimos que entran $N$ muestras dentro del intervalo que la función $x(t)$ es igual a 1.

El último paso es el de muestrear la transformada de Fourier de la ecuación anterior. Esto es equivalente a convolucionar en el dominio del tiempo con una función $\Delta_1(t) = T_0.\sum_{r=-\infty}^{\infty}\delta(t-r.T_0)$, resultando en la función:

$$\tilde{h}(t) = T_0.\sum_{r=-\infty}^{\infty}\left[\sum_{k=-\infty}^{\infty}h(k.T).\delta(t-k.T-r.T_0)\right]$$

La transformada de esta señal puede ser expresada como:

$$G\left(\frac{n}{N.T}\right) = \sum_{k=0}^{N-1}h(k.T).e^{-j.2.\pi.n.\frac{k}{N}}\ con\ n = 0,1,\dots,N-1$$

Esta es la transformada de Fourier discreta, cuya expresión relaciona N muestras en el tiempo con N muestras en frecuencia, a partir de la transformada de Fourier continua, asumiendo que las N muestras temporales son un período de la señal continua. Este procedimiento puede verse en la Figura 10-1.

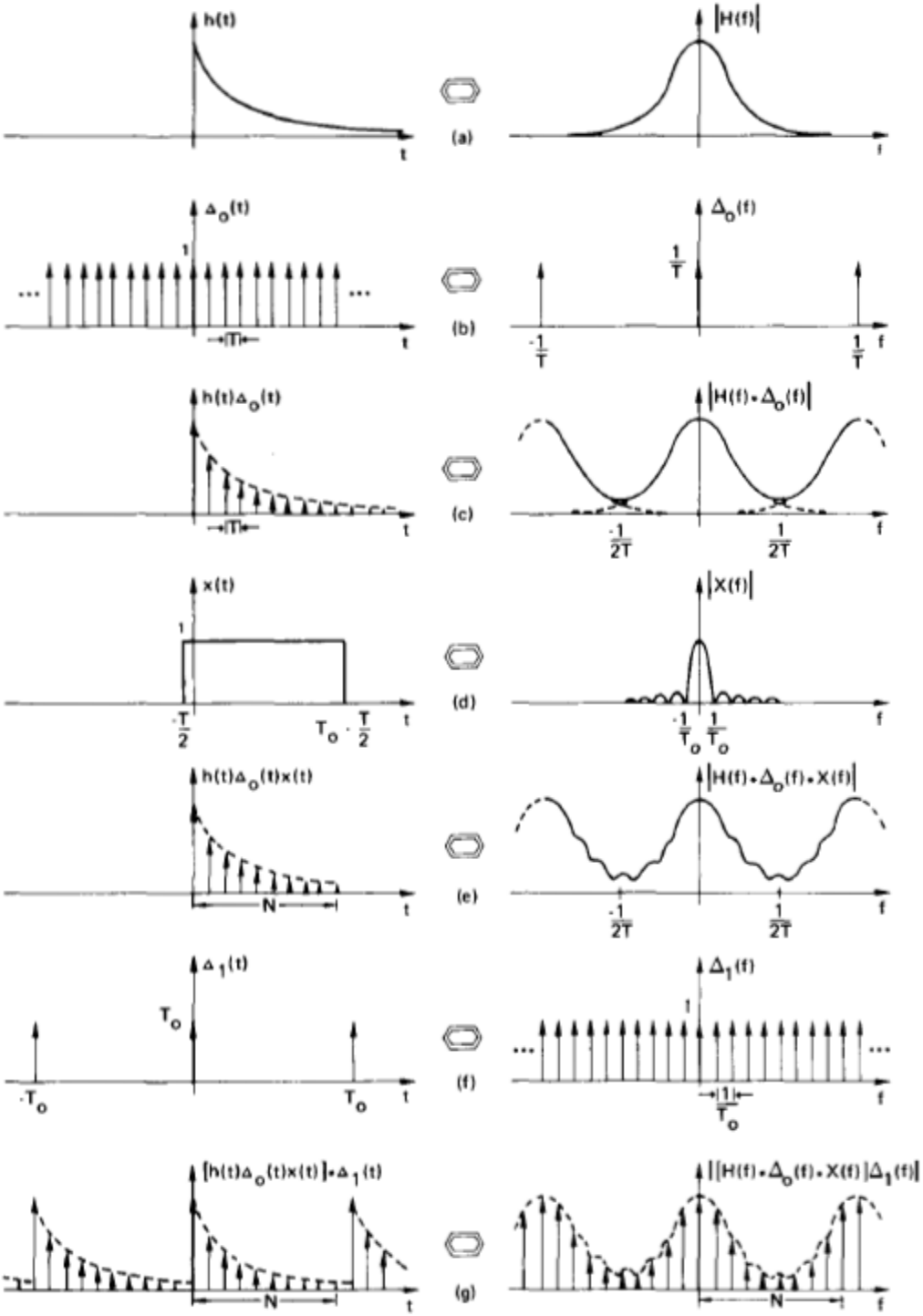



*Figura 10-1:* Procedimiento gráfico de obtención de la transformada discreta de Fourier, a partir de una señal continua, con espectro continuo.

La transformada discreta inversa de Fourier puede expresarse como:

$$g(k.T) = \frac{1}{N}.\sum_{n=0}^{N-1} G\left(\frac{n}{N.T}\right).e^{j.2.\pi.n.\frac{k}{N}}$$

Es importante recordar que el par transformado requiere que ambas funciones

$g(k.t)$ y $G\left(\frac{n}{N.T}\right)$ sean periódicas.

La implementación de esta operación en una computadora (especialmente cuando las computadoras no poseían la capacidad de realizar operaciones de manera rápida como ahora) requiere de un tiempo muy grande, el cual puede no ser práctico para aplicaciones de tiempo real. Es por esto que se desarrollaron diversos algoritmos que disminuyen el tiempo de procesamiento, a partir del tratamiento matemático de la DFT. A estos algoritmos se los llama transformada rápida de Fourier (FFT).

Puede resultar de utilidad el desarrollo de las partes real e imaginaria de las salidas de la FFT de 8 puntos (la que implementaremos en este trabajo), en función de las entradas. Al hacerlo, resulta:

$$X(0)^r = x(0)^r + x(1)^r + x(2)^r + x(3)^r + x(4)^r + x(5)^r + x(6)^r + x(7)^r$$

$$X(0)^i = x(0)^i + x(1)^i + x(2)^i + x(3)^i + x(4)^i + x(5)^i + x(6)^i + x(7)^i$$

$$X(1)^r = x(0)^r - x(4)^r + x(2)^i - x(6)^i + tw.(x(1)^r + x(1)^i - x(5)^r - x(5)^i - x(3)^r + x(3)^i + x(7)^r - x(7)^i)$$

$$X(1)^i = x(0)^i - x(4)^i - x(2)^r + x(6)^r + tw.(x(1)^i - x(1)^r - x(5)^i + x(5)^r - x(3)^i - x(3)^r + x(7)^i + x(7)^r)$$

$$X(2)^r = x(0)^r + x(4)^r - x(2)^r - x(6)^r + x(1)^i + x(5)^i - x(3)^i - x(7)^i$$

$$X(2)^i = x(0)^i + x(4)^i - x(2)^i - x(6)^i - x(1)^r - x(5)^r + x(3)^r + x(7)^r$$

$$X(3)^r = x(0)^r - x(4)^r - x(2)^i + x(6)^i + tw.(x(3)^r - x(1)^r + x(1)^i + x(5)^r - x(5)^i + x(3)^i - x(7)^r - x(7)^i)$$

$$X(3)^i = x(0)^i - x(4)^i + x(2)^r - x(6)^r + tw.(x(3)^i - x(1)^i - x(1)^r + x(5)^i + x(5)^r - x(3)^r - x(7)^i + x(7)^r)$$

$$X(4)^r = x(0)^r - x(1)^r + x(2)^r - x(3)^r + x(4)^r - x(5)^r + x(6)^r - x(7)^r$$

$$X(4)^i = x(0)^i - x(1)^i + x(2)^i - x(3)^i + x(4)^i - x(5)^i + x(6)^i - x(7)^i$$

$$X(5)^r = x(0)^r - x(4)^r + x(2)^i - x(6)^i + tw.(x(5)^r - x(1)^r - x(1)^i + x(5)^i + x(3)^r - x(3)^i - x(7)^r + x(7)^i)$$

$$X(5)^i = x(0)^i - x(4)^i - x(2)^r + x(6)^r + tw.(x(5)^i - x(1)^i + x(1)^r - x(5)^r + x(3)^i + x(3)^r - x(7)^i - x(7)^r)$$

$$X(6)^r = x(0)^r + x(5)^r - x(3)^r - x(6)^r - x(1)^i - x(5)^i + x(3)^i + x(7)^i$$

$$X(6)^i = x(0)^i + x(5)^i - x(3)^i - x(6)^i + x(1)^r + x(5)^r - x(3)^r - x(7)^r$$

$$X(7)^r = x(0)^r - x(4)^r - x(2)^i + x(6)^i + tw.(x(1)^r - x(1)^i - x(5)^r + x(5)^i - x(3)^r - x(3)^i + x(7)^r + x(7)^i)$$

$$X(7)^i = x(0)^i - x(4)^i + x(2)^r - x(6)^r + tw.(x(1)^i + x(1)^r - x(5)^i - x(5)^r - x(3)^i + x(3)^r + x(7)^i - x(7)^r)$$

Donde $tw = real(e^{j.\frac{\pi}{4}}) \cong 0{,}7071$.

Una forma de obtener la IFFT, si se tiene un sistema que realiza la FFT, es ingresar al mismo con las partes real e imaginaria de los coeficientes de entrada intercambiados, volver a intercambiar las partes real e imaginaria a la salida, y dividir el resultado por N. Esto puede verse en la Figura 10-2.

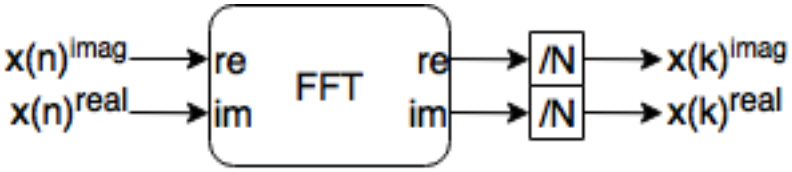



*Figura 10-2:* Diagrama que muestra el procedimiento para el cálculo de la IFFT, a partir de un bloque FFT.

## Formato $Q_{m.n}$ con signo

El formato $Q_{m.n}$ es una forma de interpretar números binarios, usando punto fijo. Este formato consta de $m$ bits de parte entera, y $n$ bits de parte fraccionaria, en complemento a 2, para poder representar números negativos. Este formato permite representar números desde $-2^{m-1}$ hasta $2^{m-1} - 2^{-n}$, en pasos de $2^{-n}$. Se eligió este formato para reducir la cantidad de líneas necesarias, ajustándose a las necesidades, en cada situación. Esta flexibilidad en la cantidad de bits a utilizar nos da la posibilidad de reducir la cantidad de elementos lógicos necesarios para la realización en FPGA de nuestro proyecto.

Resulta de interés analizar qué sucede con las dimensiones necesarias para evitar que el número no pueda ser representado, cuando se realizan operaciones matemáticas entre dos números en formato $Q_{m_1.n_1}$ y $Q_{m_2.n_2}$. En el caso de las sumas y restas, el formato que debe tener el resultado es $m = \max(m_1, m_2) + 1$ y $n = \max(n_1, n_2)$. En el caso de las multiplicaciones, el formato que debe tener el resultado es $m = m_1 + m_2$ y $n = n_1 + n_2$. Solo estas operaciones nos interesan en este trabajo.

## Bloques Quartus auxiliares

### Bloque “receptoruart”

Este bloque atiende a la línea de recepción UART, usando una velocidad de 115200 baudios, con un bit de start, un bit de stop, y sin bit de paridad. Una vez que detecta que hay datos los recibe, y escribe los mismos en el bus de salida data[7..0]. Un ciclo de reloj más tarde, envía un pulso negativo en la salida “datavalid” (la misma se encuentra en un nivel de tensión alto si no hay datos válidos). Este bloque puede verse en la Figura 10-3.

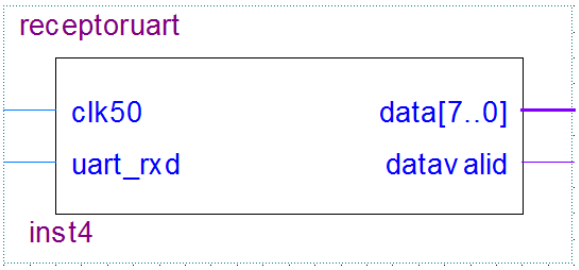



*Figura 10-3:* Bloque en Quartus del receptor UART.

## Bloque “uartaqam”

Este bloque espera recibir 3 bytes de “receptoruart”, luego separa estos datos en 8 buses de 3 bits cada uno. Finalmente, transmite un pulso negativo en la línea “fin” (esta línea está en estado alto, cuando no hay datos válidos a la salida). Este bloque puede verse en la figura Figura 10-4. Se usa como conversor serie/paralelo.

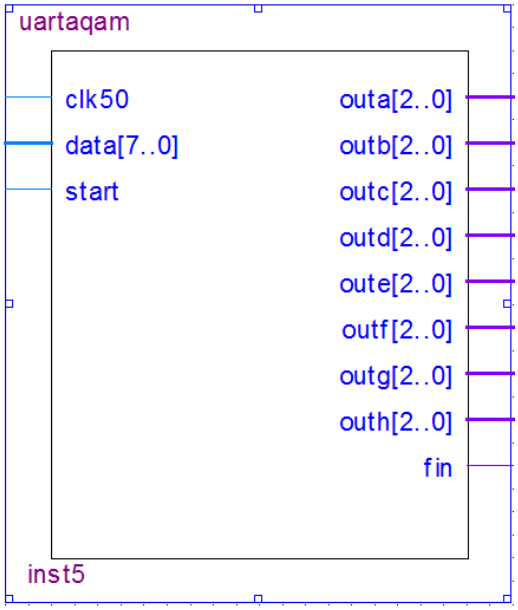



*Figura 10-4:* Bloque en Quartus que hace la conversión serie/paralelo de los bits de entrada.

## Bloque “creavector”

Este bloque toma un vector de 208 bits y, al recibir un pulso negativo en la entrada “mandaruart”, los transfiere, de un byte por vez, hacia su salida “datosuart[7..0]”, enviando un pulso negativo en la salida “startuart”, por cada uno. Entre cada uno de los bytes a transferir espera un tiempo, para que el byte anterior pueda ser transferido por la UART. Esto debe hacerse ya que el reloj es de 50 MHz, pero la UART funciona a una

velocidad de 115200 Baudios. Este bloque puede verse en la Figura 10-5.

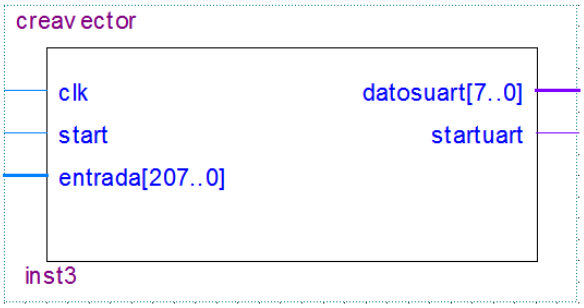



*Figura 10-5:* Bloque en Quartus que prepara los bytes para ser enviados por la UART.

## Bloque "transmisoruart"

Este bloque se encarga de enviar los datos en el bus de entrada "datain[7..0]" por UART, una vez que haya un pulso negativo en la entrada "send". Usa una velocidad de 115200 baudios con un bit de start, un bit de stop, y sin bit de paridad. De no haber datos para enviar, la línea de salida se mantendrá en nivel alto. Este bloque puede verse en la Figura 10-6.

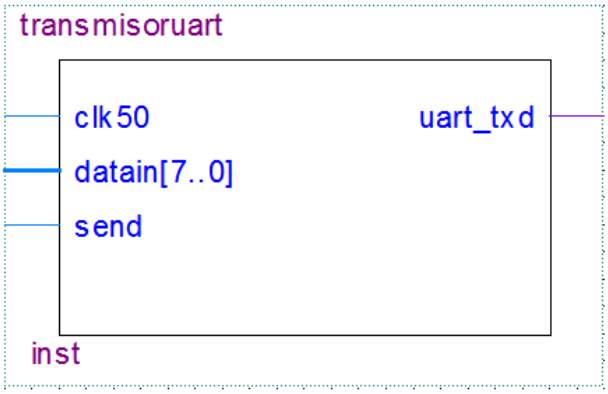



*Figura 10-6:* Bloque en Quartus que se encarga de transmitir datos por UART, de ser necesario.

## Bloque "uartavector"

Este bloque toma los datos que son recibidos por UART, disponibles en la línea data[7..0], y los ordena en un vector de 320 elementos, cuando la línea "start" se encuentra en estado bajo (normalmente se encuentra en estado alto). Cuando termina, envía un pulso negativo en la línea "fin". Este bloque se puede ver en la Figura 10-7.

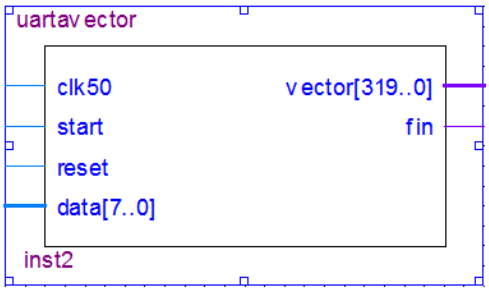



*Figura 10-7:* Bloque en Quartus que arma un vector con los datos recibidos por UART.

## Bloque "vectosig"

Este bloque recibe el vector armado por "uartavector", y hace la conversión serie a paralelo, siguiendo una secuencia conocida, cuando la línea "start" tiene un pulso negativo. Al finalizar, envía un pulso negativo en la línea "fin". A su salida, se encuentran las partes real e imaginarias de los 8 símbolos recibidos, en formato $Q_{3.17}$, listos para ser demodulados. El mismo puede verse en la Figura 10-8.

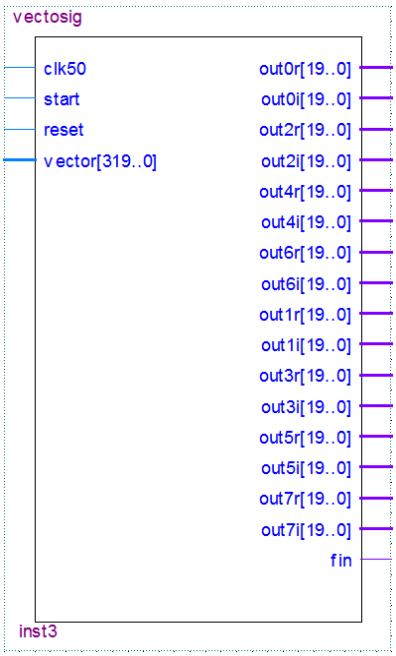



*Figura 10-8:* Bloque en Quartus que se encarga de hacer la conversión serie a paralelo, de los símbolos recibidos.

## Bloque "qamauart"

Este bloque tiene en su entrada, los datos demodulados, en formato $Q_{3.0}$, en 8 buses independientes, correspondientes a los 8 símbolos recibidos. Cuando sucede un pulso negativo en "start", este bloque traslada a la salida data[7..0], las entradas, de un

byte a la vez. Por cada byte, envía un pulso negativo en “datavalid”. Se espera un tiempo entre cada byte, ya que se debe esperar que el bloque “transmisoruart” envíe el byte anterior, antes de poder procesar el siguiente. Este bloque puede verse en la Figura 10-9.

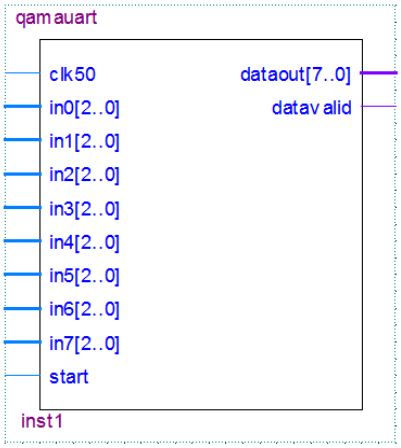



*Figura 10-9:* Bloque en Quartus que dispone los datos demodulados, para ser enviados por el transmisor UART.

# Códigos auxiliares Matlab

## Función “bintodec”

Esta función transforma un número binario en formato $Q_{m.n}$ con signo, a decimal. Toma como entradas: $x$, un vector de unos y ceros, que simboliza un número binario; $m$, la cantidad de bits de parte entera de $x$; y $n$, la cantidad de bits de parte fraccionaria de $x$.

El código es el siguiente:

```
Function y = bintodec(x, m, n)  % Pasa de x en formato Qm.n con signo a decimal
    rango = m+n;                    %x debe ser un string
    xvector = x - '0'; % Pasa de string a vector
    tmp = ones(1,rango);
    if (xvector(1) == 1)    % Reconoce si es negativo
        tmp = xor(tmp,xvector);
        neg = 1;
    else
        neg = 0;
        tmp = xvector;
    end
    expo = zeros(1,rango);
    for i=1:1:rango  % Recorre el vector, sumando las potencias de 2 que correspondan
        expo(i) = tmp(i) .* (2^(m-i));
    end
```

```
    ytmp = sum(expo);
    if (neg == 1)    % Si es negativo, calcula el complemento a 2.
        ytmp = ytmp + 2^(-n);
        ytmp = -ytmp;
    end
    y = ytmp;    % Entrega el número decimal
end
```

## Función “binuntodec”

Esta función transforma un número binario, representado por un vector de unos y ceros, a entero. Interpreta al vector como un binario sin signo.

El código es el siguiente:

```
function y = binuntodec(in) %Convierte un vector binario sin signo a entero.
    len = length(in);
    outtmp = 0;
    for i=(len-1):-1:0  % Recorre el vector, sumando las potencias de 2 que corresponda
        outtmp = outtmp + in(len-i) * 2^(i);
    end
    y = outtmp;
```

## Función “dectobin”

Esta función transforma un número decimal, a formato $Q_{m.n}$. Las entradas son: $in$, el número decimal; $m$, cantidad de bits con los que se desea representar la parte entera de $in$; y $n$, cantidad de bits con los que se desea representar la parte fraccionaria de $in$. La salida es un vector, que representa el número decimal, en formato $Q_{m.n}$.

El código es el siguiente:

```
function [out] = dectobin(in,m,n)    % Transforma un decimal a formato Qm.n
    intmp = in;
    outtmp = zeros(1,(m+n));
    if in < 0    % Reconoce si es negativo. Si lo es, niega el número, y le
        neg = 1;% quita el peso del bit menos significativo en el formato Qm.n
        intmp = - intmp;
        intmp = intmp - 2^(-n);
    else
        neg = 0;
    end;

    for j=(m-1):-1:-n           % Arma el vector de salida, recorriéndolo y
        if intmp >= (2^j)       % reconociéndo si debe ingresar un 1 o 0.
            outtmp(m-j) = 1;
            intmp = intmp - (2^j);
        else
            outtmp(m-j) = 0;
```

```
        end;
    end;

    if neg == 1     % Si es negativo, niega los bits.
        outtmp = xor(ones(1,(m+n)),outtmp);
    end;
    out = outtmp;
```

## Función “parteuno”

Esta función genera $24 . nSym$ datos binarios aleatorios, los envía a la FPGA, y recibe $nSym$ símbolos de la misma, correspondientes a los datos modulados.

La entrada es la cantidad de símbolos a generar y enviar. Las salidas son los datos aleatorios generados, y los símbolos recibidos de la FPGA.

El código es el siguiente:

```
function [salidarandom,salidamedio] = parteuno(nsimbolos) % Genera datos binarios
aleatorios y recibe esos datos modulados

nbytes = 3 * nsimbolos; % Cantidad de bytes a generar
datosbinarios = floor(rand(nbytes,8)*2);    % Genera los bytes aleatorios
datosdecimales = zeros(nbytes,1);
for j=1:nbytes  % Transforma los bytes en números decimales
    datosdecimales(j,1) = binuntodec(datosbinarios(j,1:8));
end

% A continuación se abre y configura el puerto serie
s1 = instrfind;
if ~isempty( s1 )
    fclose( s1 );
    delete( s1 );
    clear s1
end
s1 = serial('COM4', 'BaudRate', 115200);
set(s1,'DataBits',8);
set(s1,'Parity','none');
set(s1,'StopBits',1);
s1.InputBufferSize=1024;
fopen(s1);

premedio = zeros(nsimbolos,8);
xdec = zeros(1,26);
for k=1:nsimbolos  % Recorre los símbolos generados aleatoriamente
    for l=(3*k-2):(3*k); % Envía por UART los datos de un símbolo
        pause(0.00001);
        fwrite(s1,datosdecimales(l,1),'uint8');
    end
```

```
    for m=1:26  % Recibe por UART 26 bytes, donde se encuentra la información del
símbolo.
        xdec(m) = fread(s1,1);
    end

    % A continuación se ordenan los bytes recibidos, para poder interpretar
    % los datos enviados por la FPGA.
    xbin = dec2bin(xdec,8);

    aux = zeros(1,208);
    for n = 1:26
        aux(209-8*n:216-8*n) = xbin(n,1:8);
    end

    out0i = bintodec(aux(203:208),3,3);
    out0r = bintodec(aux(197:202),3,3);
    out2i = bintodec(aux(191:196),3,3);
    out2r = bintodec(aux(185:190),3,3);
    out4i = bintodec(aux(179:184),3,3);
    out4r = bintodec(aux(173:178),3,3);
    out6i = bintodec(aux(167:172),3,3);
    out6r = bintodec(aux(161:166),3,3);
    out1i = bintodec(aux(141:160),3,17);
    out1r = bintodec(aux(121:140),3,17);
    out3i = bintodec(aux(101:120),3,17);
    out3r = bintodec(aux(81:100),3,17);
    out5i = bintodec(aux(61:80),3,17);
    out5r = bintodec(aux(41:60),3,17);
    out7i = bintodec(aux(21:40),3,17);
    out7r = bintodec(aux(1:20),3,17);
    out0 = out0r + 1i * out0i;
    out1 = out1r + 1i * out1i;
    out2 = out2r + 1i * out2i;
    out3 = out3r + 1i * out3i;
    out4 = out4r + 1i * out4i;
    out5 = out5r + 1i * out5i;
    out6 = out6r + 1i * out6i;
    out7 = out7r + 1i * out7i;
    premedio(k,1:8) = [out0 out1 out2 out3 out4 out5 out6 out7];
end
fclose(s1)  % Cierra el puerto serie
salidarandom = datosbinarios;
salidamedio = premedio;
```

## Función “partedos”

Esta función envía $nSym$ símbolos modulados a la FPGA, y recibe $24 . nSym$ bits demodulados de la misma. Las entradas son los datos modulados, y la cantidad de símbolos enviados. La salida es una matriz con los datos demodulados

El código es el siguiente:

```
function [outbinario] = partedos(in,nsimbolos) % Envía símbolos y recibe datos
demodulados

% A continuación se abre y configura el puerto serie
s1 = instrfind;
if ~isempty( s1 )
    fclose( s1 );
    delete( s1 );
    clear s1
end
s1 = serial('COM4', 'BaudRate', 115200);
set(s1,'DataBits',8);
set(s1,'Parity','none');
set(s1,'StopBits',1);
s1.InputBufferSize = 1024;
fopen(s1);

% A continuación se ordenan los datos a enviar por UART, y se transforman
% los símbolos a formato Qm.n
salidadatos = zeros(nsimbolos,3);
vector = zeros(nsimbolos,320);
realin = real(in);
imagin = imag(in);
for n=1:nsimbolos
    vector(n,1:320) = [dectobin(realin(n,1),3,17) dectobin(imagin(n,1),3,17)
dectobin(realin(n,3),3,17) dectobin(imagin(n,3),3,17) dectobin(realin(n,5),3,17)
dectobin(imagin(n,5),3,17) dectobin(realin(n,7),3,17) dectobin(imagin(n,7),3,17)
dectobin(realin(n,2),3,17) dectobin(imagin(n,2),3,17) dectobin(realin(n,4),3,17)
dectobin(imagin(n,4),3,17) dectobin(realin(n,6),3,17) dectobin(imagin(n,6),3,17)
dectobin(realin(n,8),3,17) dectobin(imagin(n,8),3,17)];
end

for j=1:nsimbolos  % Se recorren los símbolos a enviar
    datos = zeros(1,3);
    for k=1:40  % Se envía un símbolo a la FPGA
        pause(0.0001)
        fwrite(s1,binuntodec(vector(j,8*k-7:8*k)),'uint8');
    end
    for l=1:3   % Se recibe 24 bits demodulados
        datos(1,l) = fread(s1,1);
    end
    salidadatos(j,1:3) = datos(1,1:3);

    for m=1:3   % Se pasa los datos recibidos a binario
        outbinario(3*j-3+m,1:8) = dectobin(salidadatos(j,m),8,0);
    end
end
fclose(s1); % Se cierra el puerto serie
```