Instituto de Investigación en Ingeniería Eléctrica
Alfredo Desages

Project MOMIL 03/2011.
Modem Standard MIL 188-110B para canal Ionosferico

Formas de Onda FSK, QPSK y QAM para velocidades de 75, 150, 300, 600, 1200, 2400, 4800 ,6400, 8000, 9600 y 12800 bps

LAPSyC IIIE

Bahía Blanca Argentina

2014

Prefacio

MODEM MIL 188-110B. Financiado por Ministerio de Defensa (MINDEF).
En el marco de proyectos de Investigación y Desarrollo para la defensa (PIDDEF).

Autor (Director del Proyecto): Dr. Juan E. Cousseau .

Autor (Co-Director del Proyecto): Mg. Marcelo J. Bruno .

Colaborador (Becario): Ing. José Gabbarrini.

Juan E. Cousseau jcousseau@uns.edu,ar
DIEC-UNS, IIIE-CONICET

Marcelo J. Bruno mbruno@criba.edu.ar
DIEC-UNS, IIIE-CONICET

José P. Gabbarrini jose.gabbarrini@gmail.com
DIEC-UNS

Resumen

El MODEM MIL-STD-188-110B para portadora simple en canal HF es usado diariamente por las fuerzas armadas de EE. UU. y otros paises alrededor del mundo. Tipiciamente, las comunicaciones que superan la línea de vista (NLOS) son realizadas por medio de satelites que retransmiten la señal de un punto a otro. Sin embargo, las comunicaciones satelitales son caras, el número de satelites es limitado y desde el punto de vista estrategico son altamente vulnerables a cualquier ataque. Alternativamente, los MODEMs HF de portadora simple ofrecen una opción confiable utilizando la ionosfera, en lugar de un satelite, para reflectar la señal transmitida y asi alcanzar el punto NLOS deseado.

La ionosfera, a su vez, presenta características únicas de canal que obligan a utilizar técnicas de procesamiento de señal mucho mas complejas que aquellas requeridas para las comunicaciones satelitales. Una de las carcateristicas principales de canal ionosferico es que la velocidad máxima que se puede transmitir sin sufrir severas distorsiones es de 500 bps. Otros factores de desmejora importantes a tener en cuenta son el ruido, el efecto Doppler y las interferencias.

En función de lo anterior, para alcanzar tasas de datos relativamente importantes es necesario aplicar el estado de arte en lo que respecta a esquemas de modulación y demodulación multinivel en portadora simple para banda angosta.

El presente estudio y desarrollo del MODEM MIL 188-110B contempla la evaluación e implementación de las técnicas DSP que permitan abordar los aspectos mencionados, detro de los cuales destacamos, ecualización adaptiva de canal, formateo de pulso de transmisión, detección por máxima verosimilitud, filtrado multirat epolifasico, interleaving y codificación FEC.

Contents

List of Figures

List of Tables

Chapter 1

Introducción

1.1 *Desafíos de la comunicación HF*

Los efectos de ruido, Doppler, diferencias de sincronización en el terminal receptor ocurren en todo sistema de comunicación inalámbrica se utilice o no la ionosfera.

El *r*uido ocurre en todo el canal y se puede pensar como fomo una fuente de señal aleatoria aditiva a la señal. En la mayoría de los casos, el ruido se asume aditivo Gaussiano Blanco (AWGN). La diferentes formas de onda requieren un minimo de relación señal a ruido (SNR) especifico para funcionar adecuadamente y recupera la señal corrupta por dicho ruido.

El *d*esplazamiento Doppler es causado por el moviento relativo entre los terminales de recepción y transmisión. Del lado receptor causa qu ela señal detectada sufra una distorsión producto del cambio de frecuencias con los que arriba. Generalmente cada tipo de forma de onda tiene un dezplazamiento Doppler máximo a que puede ser corregido. Si el dezplazamiento Doppler es mayor el receptor corrige por la cantidad máxima y la diferencia de desplazamiento queda sin corregir causando errores y la potencial pérdida de señal.

Las *d*iferencias de sincronismo entre el terminal de transmisión y recepción ocurren debido a la diferencias de los relojes internos de cada terminal. Si el reloj del receptor es mayor o menor que el transmisor la señal detecta presentará un desplazamiento temporal respecto del óptimo que requiere el filtro acoplado y por lo tanto aumenta la probabilidad de que ocurran errores y eventualmente la pérdida de la señal.

Por otro lado los efectos de *d*ispersión en frecuencia y *d*isperión en tiempo también ocurren en las comunicaciones de aire. Para el caso ionosferico el desvanecimiento del canal ocurre cuando la ionosfera

absorbe energía durante la transmisión la diseprsión temporal por los mutipleca caminos que recorre la señal.

La *d*ispersión en frecuencia ocurre porque la velocidad a la que la energía de la señal es absobida no es constante. De acuerdo a [4], debido a que la densidad de la ionosfera no es constante o predecible, es imposible encontrar 'una relación fija entre la distancia y la atenuación de la señal durante la propagación en la ionosfera'.

La *d*ispersión en tiempo ocurre debido a que la señal toma diferentes caminos para llegar al receptor dependiendo de las multiples reflexiones en la ionosfera como lo muestra la figura 1.1. De acuerdo a [4], 'las fuentes del multicamino son las reflexiones de las señales de radio desde las diferentes capas d ela ionosfera. Ademas, multiples reflexiones pueden ocurrir entre la superficie de la tierra y la ionosfera'. Esos multiples caminos causan que la señal recibida contengan 'varios ecos' separados en tiempo en el orden de los mseg [5]. Como resultado de lo anterior existe la posibilidad de que la señal recibida se atenue (o cancele totalmente) debido a una suma destructiva de ecos como lo muestra la figura 1.2.

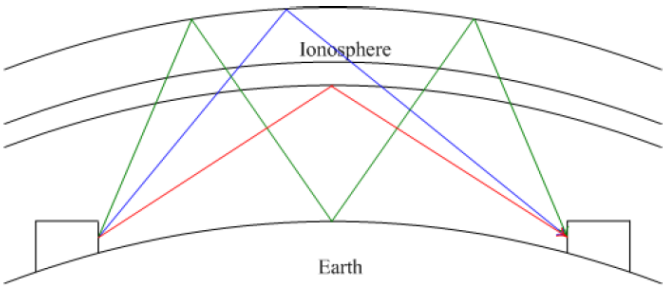

Figure 1.1: Mulficamino Canal Ionosferico

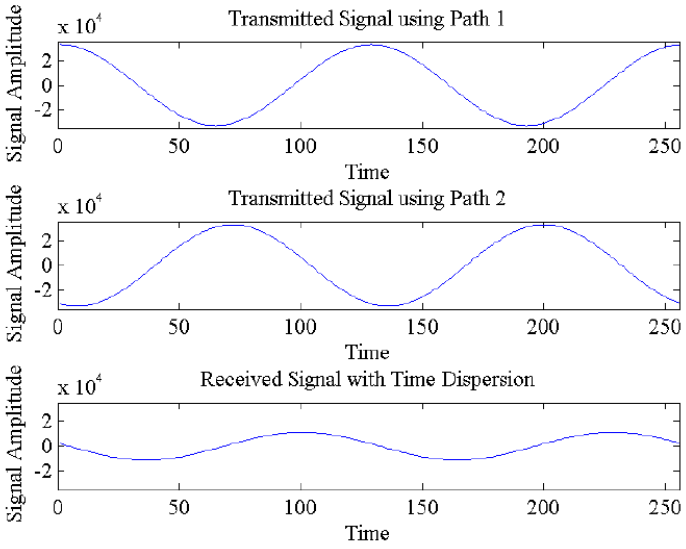

Figure 1.2: Suma de interferencia destructiva

De lo mencionando anteriormente las dispersiones en *t*iempo y en *f*recuencia son los mayores problemas a tratar en el canal ionosferico. Dado que ninguno de los dos efectos pueden ser prevenidos, se deben implementar metodos para minimizar y en el mejor de los casos eliminar los problemas causados por dichos efectos d edispersión.

Los enfoques posibles para resolver la dispersion temporal son [4], [6]:

- Ecualización Adaptiva
- Protección por tiempo de guarda.
- Estimación de Secuencia de Máxima Verosimilitud.

Los enfoques posibles para resolver la dispersion frecuencial son [4], [6]:

- Diversidad en frecuencia.
- Diversidad en tiempo
- Diversida de antenas.
- Expansion de ancho de banda.
- Codificación e intercalado.

Para minimizar los efectos de dispersion en tiempo y en frecuencia los estandares militares evolucionar hacia la inclusión de las tecnicas previamente mencionandas. De las mismas, el STD-MIL-188-110B, incorpora como obligatorio es uso de:

- Ecualización adaptiva.
- Codifcación e intercalado

El uso de *E*stimación de Secuencia de Máxima Verosimilitud es altamente deseable pues aumenta la probabilidad ddetección correcta de los datos o inversamente, permite detectar los datos correctos en peores condiciones de SNR. Productos comerciales como [7] incluyen esta técnica como una opción.

1.2 *Organización del informe*

- Capítulo 1 - Introducción.
- Capítulo 2 - Especificaciones y Diseño MODEM MIL 188-110B.
- Capítulo 3 - Diseño del Transmisor.
- Capítulo 4 - Diseño del Receptor.
- Capítulo 5 - Canal HF. Simulador Canal HF. Analisis canales VHF y UHF.
- Capítulo 6 - Hardware DSP.
- Capítulo 7 - Hardware de Comunicaciones y Enlaces de Prueba.
- Capítulo 8 - Conclusiones y recomendaciones.

Chapter 2

Especificaciones y Diseño MODEM MIL 188-110B

En este Capítulo se describen las especificaciones de transmisión y recepción definidos por el estandard para las dos modalidades previstas en el mismo, baja y alta velocidad. Desde el lado del receptor se presentan los requermientos de desempeño en BER en para ambas velocidades para los diversos canales ionosfericos previstos. Desde el punto de vista del transmisor se detallan las variantes de modulación, codificación e interleaving para cada caso. En virtud de que para baja y alta velocidad las tramas de datos difieren, junto con las especificaciones previas se inluye el detalle de cada una de las tramas respectivas.
A continuación se describe la maquina de estados (y sus diferentes fases) que ejecuta las funciones necesarias para transmitir y recibir los datos. Luego se describe brevemente la filosofía de 'Software Defined Radio' utlizada durante el diseño junto con la descripción en bloque del modulador y demodulador implementados en DSP.
Por último se presenta brevemente las caracteristicas teóricas del canal ionosferico en el contexto de las especificaciones del estandar para contextualizar de manera mas acabada los requemientos de recepción y tansmisión presentados previamente.

2.1 *Desempeño Requerido MIL 188-110B*

La interoperabilidad entre los sistemas es asegurada mediante el uso del estandard en sus diferentes opciones. Como se describió en el Capítulo previo, el ruido AWGN, el desplazamiento Doppler y las diferencias de clock afectan el desempeño de sistema. El MODEM de tono simple requiere que pueda ser capaz de detectar y corregir un máximo desplazamiento Doppler de 75Hz ([3] Apéndice C.6.3 pp 114). No existe una espficación relacionada con la máxima diferencia del tiempo de clock aunque el estandar implementa un mecanismo, mediante secuencias de sincronización, para que el MODEM pueda comunicarse a lo largo de horas. Por último, los valores de SNR requeridos difieren según el esquema de modulación. Los requermientos especificos para cada una de ellas se presentan en la Tabla 2.1 para las modulaciones $R_b < 3200$ bps y la Tabla 2.2 para modulaciones $R_b > 3200$ bps.

En la descripción del canal HF, acorde a [5], 'La cantidad de multicamino puede extenderse hasta 6 mseg y el desvanecimiento puede ser de hasta 5 Hz'. En número de caminos es 2 y la potencia de ambos es igual. La disepresión en frecuencia es representada como la velocidad a la cual la potencia promedio de la señal cambia y es medida en Hz. En relación al simulador de canal HF, las amplitudes máximas y mínimas de la señal deben ser predeterimandas, pero la frecuencia a la que cambia es modificable.

2.1.1 Requerimientos para $R_b < 3200$ bps

Las especificaciones de BER para este rango de velocidades se definen para el mejor caso de intercalado (el mas largo) cuyo retardo oscila entre 2 mseg y 5 mseg según la modulación utlizada (Sección 5.3.2.5 [3]).

Velocidad (bps)	Intercalado	SNR (dB)	SNR (dB)	BER
	Long	AWGN Channel	ITU R Poor Channel	
2400	2ms	10	-	10^{-5}
2400	2ms	-	18	10^{-5}
1200	2ms	-	11	10^{-5}
600	2ms	-	7	10^{-5}
300	5ms	-	7	10^{-5}
150	5ms	-	5	10^{-5}
75	5ms	-	2	10^{-5}

Table 2.1: Requerimientos de SNR para modulaciones de baja velocidad (Tabla XX [3])

Los canales de la Tabla 2.1 son caracterizados de la siguiente manera (Apéndice C [3]) pp 114:

- AWGN: Canal de un solo camino (path) sin desvanecimento.

- ITU Poor: Canal de dos caminos (paths) independientes entre si con estadística Rayleigh con un retardo entre caminos de 2 mseg y el desvanecimento con una varianza (2σ) de 1 Hz.

En ambos canales la SNR se mide sobre un ancho de banda B_T de 3 KHz.

2.1.2 Requerimientos para $R_b > 3200$ bps

Las especificaciones de BER para este rango de velocidades se definen para el mejor caso de intercalado (el mas largo) cuyo retardo es de 8.64 segs (Apéndice C-4 [3]).

Velocidad (bps)	Intercalado	SNR (dB)	SNR (dB)	BER
	Very Long	AWGN Channel	ITU R Poor Channel	
12800	8.64s	27	-	10^{-5}
9600	8.64s	21	33	10^{-5}
8000	8.64s	19	28	10^{-5}
6400	8.64s	16	24	10^{-5}
4800	8.64s	13	20	10^{-5}
3200	8.64s	9	15	10^{-5}

Table 2.2: Requerimientos de SNR para modulaciones de alta velocidad (Tabla C-XV [3])

Los canales de la Tabla 2.2 son definidos de la siguiente manera (Apénic C [3]) pp 114:

- AWGN: Canal de un solo camino (path) sin desvanecimento.
- ITU Poor: Canal de dos caminos (paths) independientes entre si con estadística Rayleigh con un retardo entre caminos de 2 mseg y el desvanecimento con una varianza (2σ) de 1 Hz.

En ambos canales la SNR se mide sobre un ancho de banda B_T de 3 KHz.

2.2 Especificaciones de Transmisión

Como se vió en el capítulo previo , existen dos protocolos (tramas) diferentes de datos según se transmita datos a baja o alta velocidad. A continuación se dan las espeficiaciones del trasnmisor en cada caso:

2.2.1 Especificaciones de Aire Baja Velocidad

Velocidad (bps)	Intercalado	FEC	Modulador
2400	0.6 o 4.8 seg	$\frac{1}{2}$	8PSK
1200	0.6 o 4.8 seg	$\frac{1}{2}$	QPSK
600	0.6 o 4.8 seg	$\frac{1}{2}$	FSK
300	0.6 o 4.8 seg	$\frac{1}{2}$	FSK
150	0.6 o 4.8 seg	$\frac{1}{2}$	FSK
75	0.6 o 4.8 seg	$\frac{1}{2}$	FSK

Table 2.3: Esquemas para baja velocidad

2.2.1.1 Trama de datos para Baja Velocidad

Se transmite la secuencia de preambulo cuyo largo puede ser de 3 o 24 segmentos de 200 mseg. Luego del peambulo los datos son trasmitidos alternados con las sondas según el sistema que se seleccione. La secuencia finaliza con el código d efinal de mensaje.

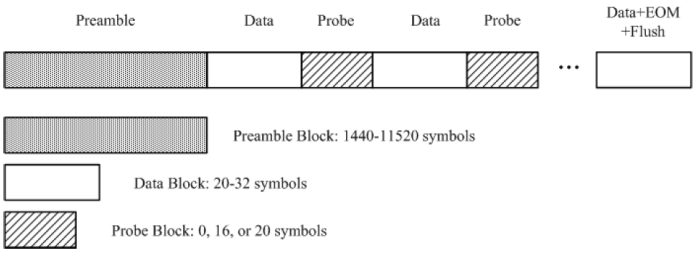

Figure 2.1: Trama de datos para modulaciones baja velocidad

- Preambulo, largo variable (Tabla 2.5).
- Datos, largo variable (Tabla 2.4).
- Sonda, largo variable (Tabla 2.4).

Velocidad (bps)	Datos (símbolos)	Sonda (símbolos)
2400	32	16
1200	32	16
600	20	20
300	20	20
150	20	20
75	32	0

Table 2.4: Largo de datos y sonda para diferentes velocidades

Largo Intercalado	Largo Preambulo
0.6seg	3 segmentos
4.8seg	24 segmentos

Table 2.5: Configuración intercalado y preambulo

El interleaver determina el largo del segmento, cada segmento es de 200 mseg. El detalle de los segmentos se encuentra en 5.3.2.2.1 [3].

2.2.2 Especificaciones de Aire Alta Velocidad

Velocidad (bps)	Intercalado	FEC	Modulador
12800	0.12 - 8.64 seg	$\frac{3}{4}$	64QAM
9600	0.12 - 8.64 seg	$\frac{3}{4}$	64QAM
8000	0.12 - 8.64 seg	$\frac{3}{4}$	32QAM
6400	0.12 - 8.64 seg	$\frac{3}{4}$	16QAM
4800	0.12 - 8.64 seg	$\frac{3}{4}$	8PSK
3200	0.12 - 8.64 seg	$\frac{3}{4}$	QPSK

Table 2.6: Esquemas para alta velocidad

2.2.2.1 Trama de datos para Alta Velocidad

Se transmiten 287 simbolos de preambulo y luego 72 frames (256+31) alternando 256 simbolos de dato con 31 simbolos de sonda. Despues de esos 72 frames se re-inserta un preambulo corto de 72 simbolos (que es un subset del preambulo incial) para sincronizar y corregir Doppler.

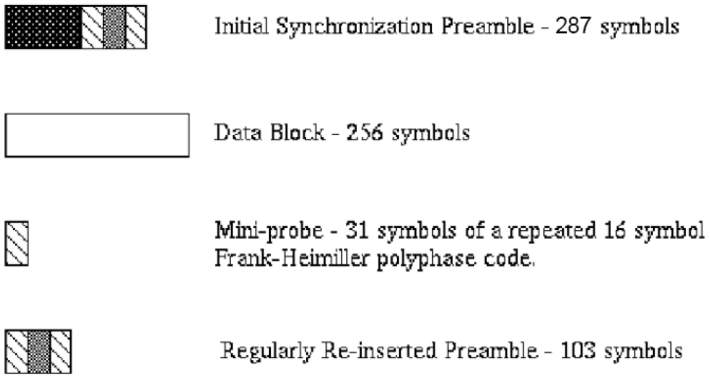

Figure 2.2: Trama de datos para modulaciones alta velocidad

- Preambulo, largo variable (incial y reinserción) (Tabla 2.8).

- Datos, largo fijo (Tabla 2.7).
- Sonda, largo fijo (Tabla 2.7).

Velocidad (bps)	Datos (simbolos)	Sonda (simbolos)
12800	256	31
9600	256	31
8000	256	31
6400	256	31
4800	256	31
3200	256	31

Table 2.7: Largo de datos y sonda para diferentes velocidades

El preambulo es de 287 símbolos. Los primeros 184 son para sincronismo y compensación Doppler mientras que los 103 restantes son para envíar información del interleaver y velocidad.

Largo Intercalado	Largo Preambulo
0.12 seg	287 simbolos
8.64 seg	287 simbolos

Table 2.8: Configuración Intercalado y preambulos

2.3 Procesamiento de los Datos a Alto Nivel (Tramas)

Para transmitir los datos el MODEM pasa por cuatro fases en las cuales, arma la trama de baja o alta velocidad. Las mismas fases que se utilizan para transmisión se utlizan para recepción. Las fases son:

1. Fase preambulo
2. Fase datos y sonda variables.
3. Fase final de mensaje.
4. Fase limpieza de bits.

El MODEM ejectua la transmisión enviando primero el preambulo. Una vez que esta listo, pasa al segundo estado que es envíar los datos y sondas. Una vez que los datos terminan pasa al tercer y cuarto estado que son respectivamente enviar una secuencia de final de mensaje (EOM) y luego vaciar los interleaver y codificadores para que quede todo en cero.
El diagrama en bloques detallado con las fases se puede observar en la figura 2.3

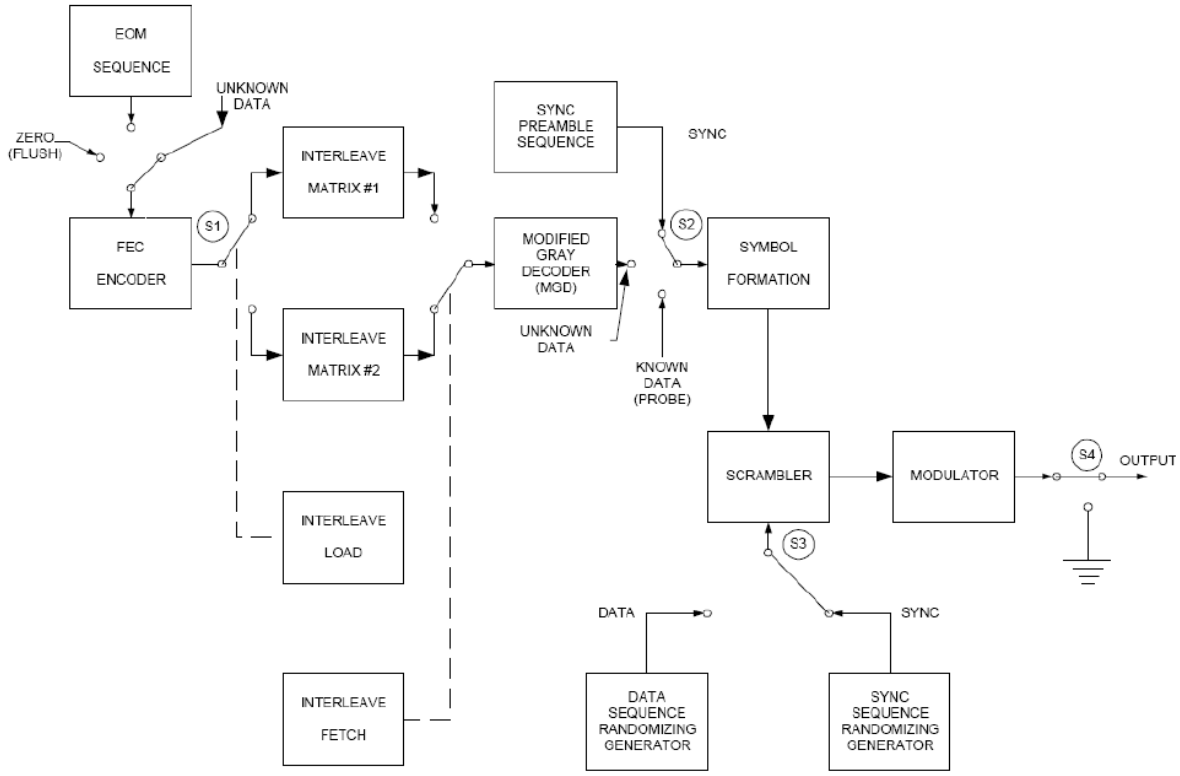

Figure 2.3: Diagrama en bloques funcional del MODEM

En la figura 2.3 se detalla el funcionamiento del MODEM. La maquina de estados funciona como se detalla a continuación:

- *Fase preambulo*: S1 en 'UNKWOWN DATA' Esta fase dura el tiempo de preambulo, que debe ser el mismo en armar el interleaver. Durante ese tiempo los switchs S2 y S3 deben estar en 'SYNC'.
- *Datos*: S1 en 'UNKWOWN DATA', 'S2' conmuta entre 'UNKWOWN DATA' y 'KNOWN DATA (PROBE)'. S3 en 'DATA' indica randomización de 'UNKNOWN DATA' y 'KNOWN DATA'.
- *EOM*: En esta fase, cuando el último 'UNKWNOWN DATA' entra en el codificador, el switch S1 debe pasar a 'EOM'. Esto determina que un patron de 32 bits conocidos se envíen al FEC. Los switchs S2 y S3 siguen funcionando como en la fase Datos.
- *Flush*: Cuando finaliza EOM el switch 'S1' debe pasar a la posición 'FLUSH' y llenar de ceros el FEC.

La siguiente Tabla 2.10 es un resumen de los estados del MODEM:

Fase	S1	S2	S3
Preambulo	UNKWOWN DATA	SYNC	SYNC
Datos	UNKWOWN DATA	UNKWOWN DATA y PROBE	DATA
Final mensaje	EOM	UNKWOWN DATA y PROBE	DATA
Flush	FLUSH	-	-

Table 2.9: Configuración intercalado y preambulos

2.4 *Concepto de Software Defined Radio (SDR)* [1], [2]

El concepto de SDR está relacionado con la implementación de un hardware analógico de comunicaciones utilizando una computadora u otro dispositivo de computación embebida. Dependiendo de la velocidad del hardware de procesamiento se podría reemplazar el hardware analógico (digitalizar) tan cerca de la antena como fuera posible. El estado del arte acutal indica que el margen superior de digitalización ronda los 200MHz [8] y [2] y para estos límites se require de hardware paralelo dedicado implementado con FPGAs. En el caso de presente diseño, debido a la naturaleza de la señal disponible, solamente se puede trbajar con la señal en rango de audio. Por lo tanto para esta implementación lo mas adecuado es implementar el SDR en hardware DSP.

La figura 2.4 muestra donde se aplica el concepto de SDR en el diseño del MODEM MIL-188-110B.

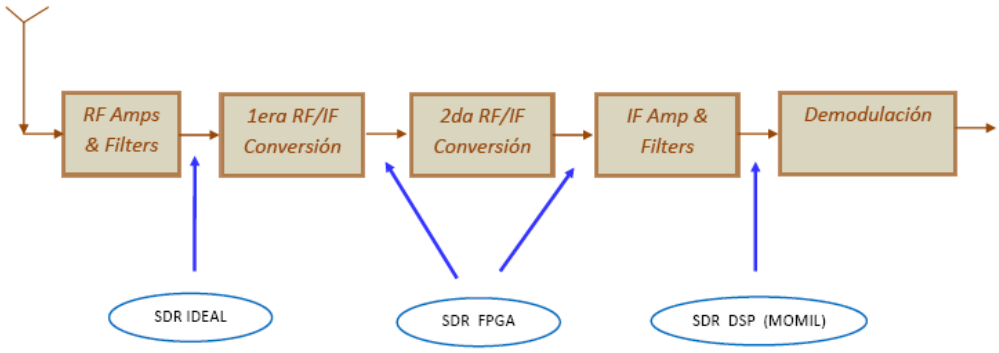

Figure 2.4: SDR en un receptor analógico heterodino

NOTA: Si bien las definiciones SDR y Software Radio (SR) se usan insdistinamente, este último termino se aplica en recepción cuando la digitalización toma lugar despues del LNA. En terminos generales se habla de SDR cuando se tienen un transmisor/receptor totalmente reconfigurables mediante la programación de un hardware.

2.5 *Diseño SDR de MODEM MIL 188-110B para canal HF*

En función del sistema disponible en general la única forma posible de realizar el MODEM es implementando el modulador y el demodulador SDR IQ en el ancho de banda de la señal de Audio operando con esa señal real, es decir, el alcance del SDR es limitado a la banda en cuestión. Por tal motivo la velocidad de datos disponible se reduce la mitad del ancho de banda mencionado tal como lo preveé el estandard [3]. Los diagramas en blouqe spara implementar el modulador y demodulador se presentan a continuación.

2.5.1 Modulador

El modulador se diseña con una arquitectura multifunción (ver figura 2.5). Esta arquitectura realiza dos funciones bien diferenciadas, la primera es modular en CPFSK con modulador IQ utilizando filtro de máscara cuando lo requira y la segunda es un modulador QAM IQ típico con mapeo de bit a símbolo y filtro de formateo de pulso en cada rama.

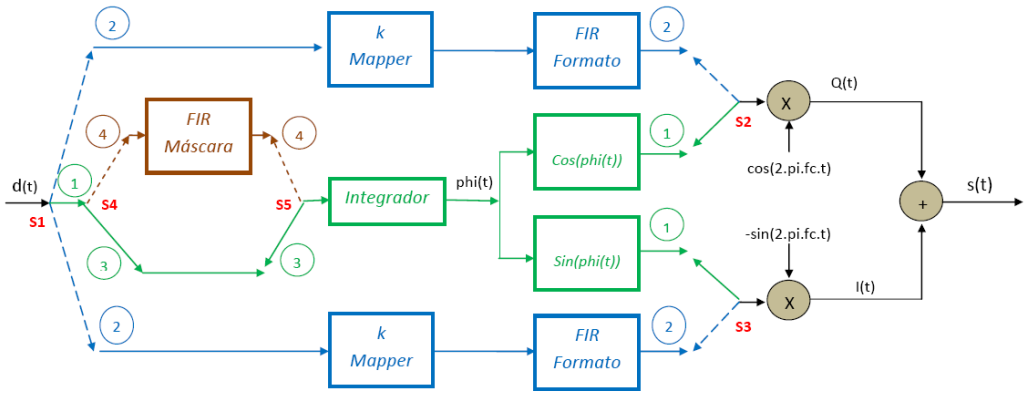

Figure 2.5: Diagrama en bloques del Modulador Multi Modo

En la figura 2.5 se observa que el mismo modulador IQ se utiliza para todas las posibles combinaciones previstas por el estandar. según el tipo de modulación necesaria lo que se debe cambiar es las entradas al modulador IQ.

Los bloques en color verde se utilizan para modular CPFSK, para ello las llaves S1, S2 y S3 deben estar en la posicón 1. Dado que en algunas de las modulaciones mas veloces de CPFSK se requiere de controlar el crecimento espectral es necesario agregar un filtro de máscara. En ese caso las llaves S4 y S5 se colocan en la posición 4. Para el caso las las modulaciones multinivel QPSK o QAM se deben utilizar los bloques en azul colocando para ello las llaves S1 , S2 y S3 en la posición 2.

Se debe destacar que cada uno de los bloques se realiza con un codigo *.C modular que se ejectua en el DSP. Esto permite facilitar la implementación de las diversas variantes requeridas por el estandar en un

único hardware programable.

En el Capítulo 3 se presenta el detalle de los diseños para cada uno de los bloques mencionadaos en esta sección.

2.5.2 De Modulador

El demodulador diseñado para todas las modulaciones es el esquema de la figura 2.6. En ella se puede observar que para convertir la señal de audio real a compleja (IQ) se necesita primero utilizar un filtro de Hilbert. Luego se demodula con las componentes en fase y cuadratura del NCO (Numerically controled oscillator), como la multiplicación no genera nuevos armonicos como en el caso analógico solo es necesario agregar un filtro decimador para obtener el dato banda base. El bloque pos procesamiento es diferente según el tipo de modulación que se reciba. Este bloque genera el control de sincronismo (Sync), el control de amplitud de entrada (AGC) y se encarga de la detección de los datos.

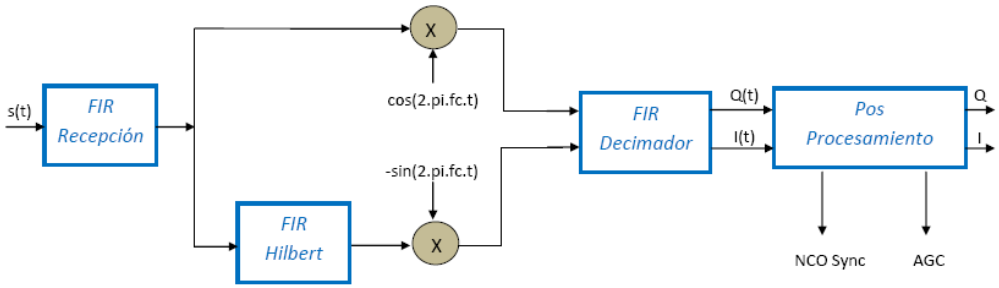

Figure 2.6: Diagrama en bloques Demodulador

El detalle específico de la detección de datos se presentará en el Capítulo 4. La arquitectura incluye el ecualizdor con realimentación por decisión, el esquema de correlación para generar el sincronismo y el generador de correción de fase de portadora.

2.6 *Canal Ionosferico HF*

El canal Ionosferico en su versión completa fue presentado en [9] y es descripto en detalle en [10] (fig. 2.7). Se caracteriza, a grandes rasgos, por ser un canal que genera multicamino de propagación y desvanecimiento. La señal transmitida usualmente toma varios caminos debido a los rebotes en la ionosfer. La altura promedio de las capas E (90km a 120km) y F (200km to 500km) de la ionosfoera cambian con el tiempo introducinedo cambios en frecuencia y en los componentes multicamino.
Si se transmite un onda continua (CW) el espectro que se recibirá será similar al descripto en 2.7. En la misma se observan cuatro caminos que se corresponden con diferentes modos de propagación definidos a continuación:

- A: Camino 1 (Modo 1E, a y b representa la división magnetoionica)
- B: Camino 2 (Modo 1F).
- C: Camino 3 (Modo Mixto).
- D: Camino 4 (Modo 2F).

Los compoentes a y b del modo 1E tienen dispersión similar pero pueden ser resueltos (separados) en frecuencia. En cualquiera de los otros tres modos (1F, Mixto y 2F) las compoentes magnetoionicas aparecen como una sola, a su vez la ganancia es mucho menor que el modo 1E y por ello no se toman en cuenta para le modelado del canal HF [9], [11].

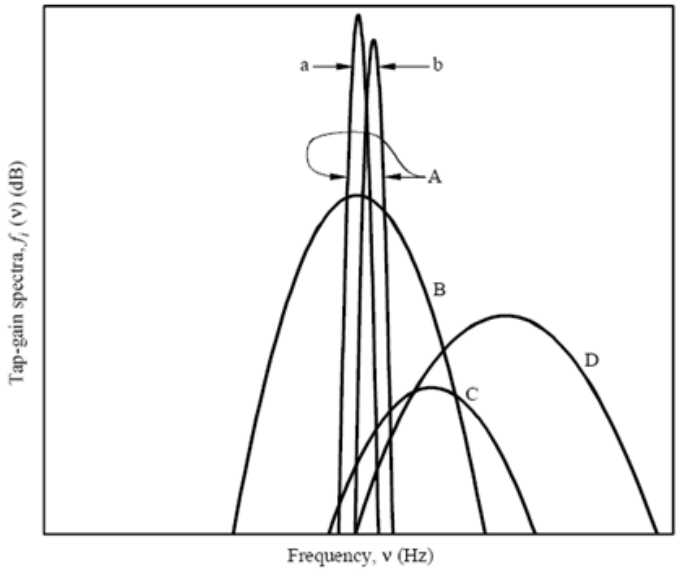

Figure 2.7: Dendisad Espectral para componentes multicamino en respuesta a una onda CW

El modelo general del canal tiene en cuenta entonces ambas compoentes magnetoionicas del modo 1E acorde a la siguiente expresión:

$$G_i(t) = \tilde{G}_{ia}(t)e^{(j.2\pi v_{ia}(t))} + \tilde{G}_{ib}(t)e^{(j.2\pi v_{ib}(t))} \tag{2.1}$$

Don a y b representan cada compoente magentonionica. $\tilde{G}_{ia}$ y $\tilde{G}_{ib}$ son la funciones funciones gaussianas complejas que producen un desvanecimiento Rayleigh. Las exponenciales $e^{(j2\pi v_{ia}(t))}$ y $e^{(j2\pi v_{ib}(t))}$ representan el cambio Doppler.
Cada multicamino tiene su propia función de potencia espectral compuesta por la suma de ambas componentes magnetoionicas expresadas a continuación:

$$f_i = \frac{1}{\tilde{A}_{ia}\sigma_{ia}\sqrt{2\pi}}e^{-\left[\frac{(v-v_{ia})^2}{2\sigma_{ia}^2}\right]} + \frac{1}{\tilde{A}_{ib}\sigma_{ib}\sqrt{2\pi}}e^{-\left[\frac{(v-v_{ib})^2}{2\sigma_{ib}^2}\right]} \tag{2.2}$$

Donde $\tilde{A}_{ia}$ y $\tilde{A}_{ib}$ son las atenuaciones (complejas)y $2\sigma_{ia}^2$ y $2\sigma_{ib}^2$ representan la dispersión Doppler. El modelo general entonces es representado por la figura 2.8:

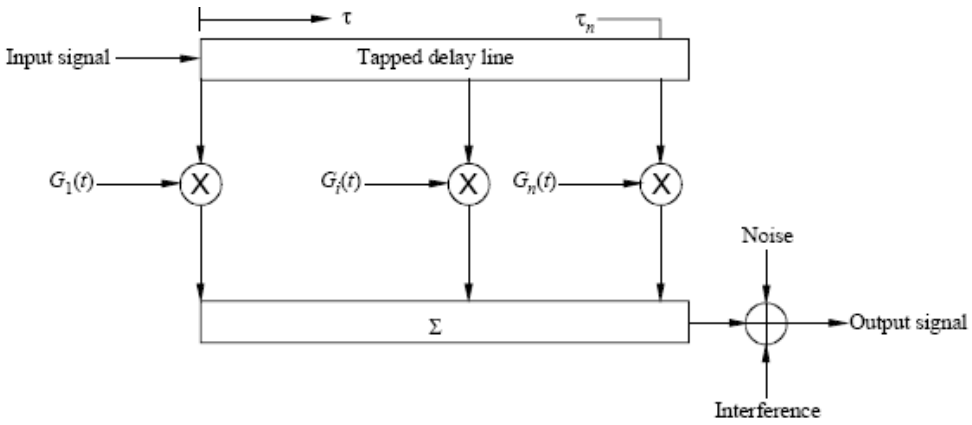

Figure 2.8: Esquema del canal HF ionosferico

La figura 2.9 describe el espectro del canal HF cuando ambas componentes magnetoionicas son resolubles.

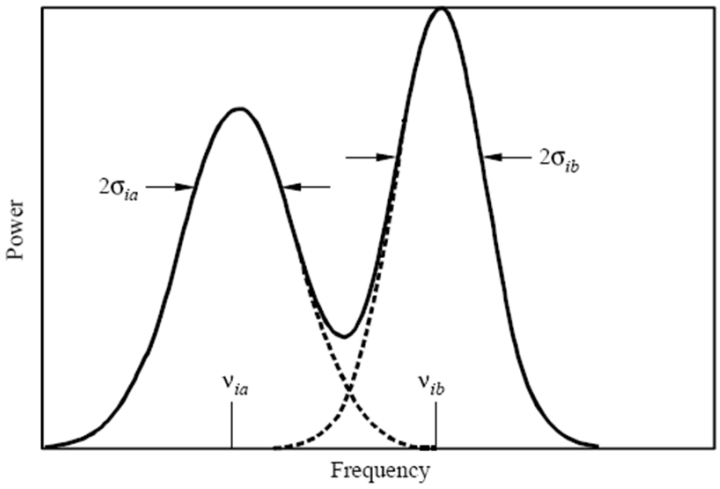

Figure 2.9: Dispersion Gasussiana con dos Compoentes Magnetoionicas

Ahora bien, existen dos casos donde el canal presentado en la ecuación 2.3 (Figura 2.7) se simplifica considerablemente debido a condiciones de propagación y anchos de banda especificos y solo se require uno de los componentes de 2.3. Estas condiciones se decriben a continuación:

- *C*uando la dispersión y cambio Doppler son similares para cada componente magnetoionica y los retardos entre ambas son muy pequeños (del orden de la dispersión Doppler). En ese caso se utiliza un solo espectro como el de la figura 2.10.

- *C*uando las componentes magnetoionicas tienen una diferencia significativa en retardo. En ese caso se utiliza una linea de retardos con el espaciado apropiado. Cada uno de los coeficientes d ela linea de retardos es un espectro como el de la figura 2.10.

Entonces el coeficiente del camino en cuestión queda:

$$G_i(t) = \tilde{G}_{ib}(t)e^{(j2\pi v_{ib}(t))} \tag{2.3}$$

con

$$f_i = \frac{1}{\tilde{A}_{ib}\sigma_{ib}\sqrt{2\pi}} e^{-\left[\frac{(v-v_{ib})^2}{2\sigma_{ib}^2}\right]} \tag{2.4}$$

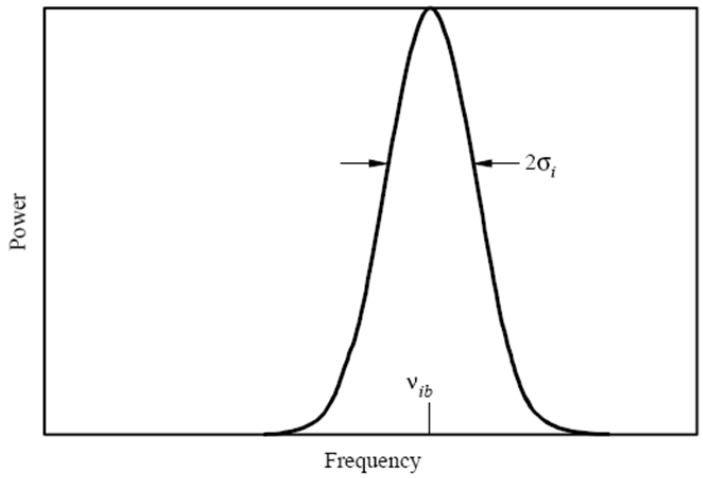

Figure 2.10: Dispersion Gasussiana con una Compoente Magnetoionicas (o dos superpuestas)

La configuración de canal Watterson adoptado por ITU 520-2 [12] tiene en consideración los dos casos previos. A continuación se describirá matematicamente el modelo en cuestión ,en el que se basa el MODEM MIL, y posteriormente se detallará los valores especificos propuestos por la ITU para evaluación y pruebas de conformidad.

2.6.1 Modelo de Canal Ionosferico HF Watterson

Una de las contribuciones claves para describir el comportamiento del canal HF fué el trabajo presentado por Watterson en 1970 [9]. En ese trabajo se presentó un modelo estacionario que fué validado con mediciones. Aunque los canales HF en general no son estaiconarios, este modelo muestra ser valido para tiempos sufientemente cortos (< 10 minutos) y para ancho de banda limitado a 10KHz. De este modo tanto el tiempo de 'estacionaridad' como el ancho de banda entran dentro de las especificaciones del MODEM MIL (en caso de requerir mayor ancho de banda se debe recurrir a otros modelos, por ej [13]).

El modelo Watterson describe al canal HF como un filtro transversal (FIR) donde los coeficientes son complejos y varían con el tiempo. El modelo es expresado por la siguiente ecuación:

$$y_k = \sum_{i=0}^{L-1} h_i x_{k-i} + n_k \tag{2.5}$$

- k es el índice de tiempo.
- y_k es la salida (compleja) del canal en el tiempo k.
- x_k es la entrada (compleja) al canal en el tiempo k.
- h_i es i-esimo (complejo) coeficiente del FIR que representa el canal.
- L es el número de coeficientes.
- n_k es el ruido AWGN en el tiempo k.

Los coeficientes h_i representan los diferentes multicaminos que se reciben en el receptor. Los coeficientes son modelados filtrando AWGN con filtros cuya respuesta en frecuencia es Gaussiana (si bien esta forma se definió en un prmicipio arbitrariamente, sucesivas mediciones mostraron ser la mejor elección [14]). La dispersión Doppler d_k es incluida en el filtro configurando la varianza dle FIR gaussiano $\sigma_k = d_k/2$. La función transferencia de cada coeficiente del canal HF tiene la siguiente expresión:

$$|H_k(f)|^2 = \frac{e^{-\left(\frac{2f^2}{d_k^2}\right)}}{\sqrt{\frac{\pi d_k^2}{2}}} \tag{2.6}$$

la Transformada de Fourier de 2.6 para obtener la expresión temporal de los coeficientes del filtro FIR que representa el canal HF es:

$$h_k(t) = \sqrt{2} e^{-\left(\pi^2 t^2 d_k^2\right)} \tag{2.7}$$

Se debe destacar que en 2.7 los coeficientes tienen también una forma gaussiana. Los coeficientes del FIR HF son entonces descriptos por la ecuación 2.7. La figura 2.11 describe la generación del efecto de un canal ionosferico con la descripción del modelo Watterson:

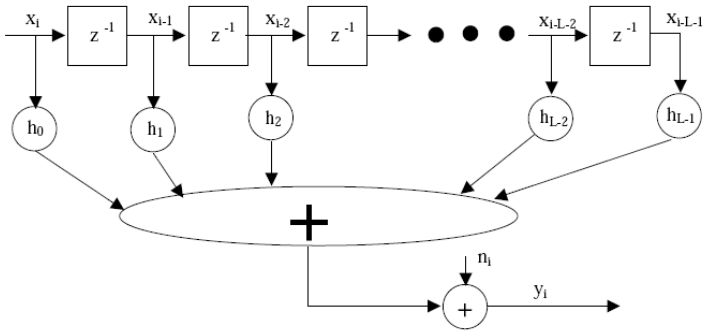

Figure 2.11: Modelo de Canal HF

Se debe recordar que cada coeficiente h_k de la figura 2.11 es un valor complejo generado como lo muestra la figura:

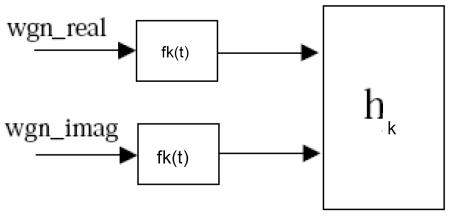

Figure 2.12: Modelo de Canal HF

2.6.2 Clasificación ITU 520.2 del Canal Ionosferico Watterson

El modelo de canal que se utilizó para la definición del estandard y se utiliza para realizar las pruebas de validacion es el canal Watterson [9] presentado previamente. La recomendación ITU 520.2. [12] lo parametriza como se detalla a continuacióin en la Tabla [12].

1. Tipo I (AWGN-FF): Canal Ruido Gaussiano con desvanecimiento plano. Sin dispersión en frecuencia. Sin desplazamiento de frecuencia.
2. Tipo II (AWGN-MP-FF): Canal Ruido Gaussiano con multicamino y desvaneciemiento. Los multicaminos atenuan del igual manera y existe dispersión en frecuencia. Sin dezplazamiento de frecuencia.

Comentarios:

- El canal Tipo I es de desvanecimiento lento ya que la variación canal es mas lenta que el tiempo de símoblo.
- El canal Tipo II es de desvanecimiento rápido ya que la variación del canal es mas rápida que el tiempo de símoblo y por eso se modela con multicamino.

Channel	Condition	Paths	Delay (ms)	Gain (rms)	Doppler Shift	Doppler Spread
I	Good	1	0	1	0	0.2Hz
I	Extreme	1	0	1	0	1Hz
I	CCIR Good	1	0	1	0	0.1Hz
II	CCIR Good	2	0.5	1	0	0.1Hz
I	CCIR Moderate	1	0	1	0	0.5Hz
II	CCIR Moderate	2	1	1	0	0.5Hz
I	CCIR Poor	1	0	1	0	1.0Hz
II	CCIR Poor	2	2	1	0	1.0Hz
I	CCIR High Latitude	1	0	1	0	10Hz
II	CCIR High Latitude	2	10	1	0	50Hz

Table 2.10: Canales HF ITU 520.2

2.7 *Extensión de las especificaciones del estandard MIL 188-110B a canales VHF y UHF*

Si bien el estandar está definido enteramente para los canales HF descriptos en la sección previa, existe la necesidad de utlizar equipos similares en otras bandas de frecuencia. A manera introductoria y, adelantando las conclusiones obtenidas en el Capítulo 'Simulador Canal HF y Análisis de canales VHF y UHF', podemos adelantar que el estandar es viable en estas otras frecuencias. A grandes rasgos la implementación sería similar con la diferencia substancial de que estos canales se comportan con desvanecimiento plano lo cual relajaría las necesidades de ecualización en el receptor. Estos aspectos se discutiran en profundidad en los capitulos subsiguientes.

2.8 Conclusiones

En este Capítulo se presentaron los requerimientos de recepción y las especificaciones de transmisión para cumplir con en el estandar MIL-188-110B. Todo el hardware necesario para realizar la tareas de modulación y demodulación se implementa en soporte DSP aplicando la filosofía de SDR. El hardware se divide en dos grandes grupos, el requerido para implementar tramas de baja velocidad y aquel que se usa para las tramas de balta velocidad. En ambos casos la maquina de estados utlizada es la misma. En algunos casos pueden compartir el mismo modulador y demodulador. La gran diferencia entre la trama de baja y la de alta es que la primera tiene un 'payload' de datos menor puesto que transmite mas seguido las sondas para para actualizar el ecualizador. Tanto el modulador como el demodulador se implementan con un concepto modular que permite adaptar el hardware DSP al tipo de trama y modualción en cada caso.

Por último se realizó una breve descripción teórica del canal HF en el contexto de la clasificación que

describe la recomendación IUT 520-2 para facilitar la interpretación de los requermientos previos. En relación a lo anterior se describe también la posiblidad de utilizar el estandar en otros canales de aire como el VHF y UHF.

Chapter 3

Diseño del Transmisor

En este Capítulo se describen los diseños de los moduladores requeridos para transmitir todas las formas de onda definidas en el estandar MILL 188-110B. Los diseños se dividen en dos grupos bien diferenciados. El primer grupo de moduladores, son los que implementran las formas de onda de baja velocidad de datos que se definen en el cuerpo principal del estandard y son del tipo FSK y QPSK. El segundo grupo de moduladores, los que implementran las formas de onda de alta velocidad de datos descriptas en el Apéndice C del estandard, son del tipo QPSK y QAM.
De acuerdo a las definiciones de [3], el modulador, además dal mapeo de los símbolos a formas de onda, también debe implementar el tramado de datos qu eincluye codificacíón interleaving y las secuencias de sincronimso y de entrenamiento para le ecualizador. Estas tramas de datos son diferentes para baja y alta velocidad y se describirán en detalle en las siguientes secciones.

El presente Capítulo de desarolla en dos grandes bloques, en uno se detallan los diseños de los moduladores propiamiente dichos y en el otro se presenta el diseño de los entramados de datos.
En primer lugar se presentan los moduladores en el siguiente orden, para baja velocidad compatibles con sistemas anteriores ($R_b < 1200$ bps), baja velocidad ($1200 < R_b < 3200$ bps) que corresponen al cuerpo principal del estandard y, alta velocidad ($R_b > 3200$) detalladas en Apéndice C del estandard.
En segundo lugar, y de forma separada, se de describen los diseños de las tramas para baja y alta velocidad.
Las modulaciones $R_b < 1200$ se dividen en dos tipos FSK y QPSK, las primeras son para compatibilidad y las segundas son definidas en el estandar. El segundo grupo de moduladores, compuesto por las modulaciones de velociad $1200 < R_b < 4800$, solo contempla moduladores QPSK. El tercer y último grupo, moduladores de alta velcoidad para lograr $R_b > 4800$ abarca formas de onda QPSK y QAM.
Para las modulaciones de alta velocidad, debido a la relación entre velocidad de datos y ancho de banda de canal, requieren de filtros de formateo de pulso tipo Raised Cosine o Root Raised Cosine. Los diesños estos filtros se presentan a continuación de los moduladores de alta velocidad.

En la segunda parte, se describen los bloques que implementan el armado de las tramas para ambos casos, baja y alta velocidad. En cada caso de describen los detalles y resultados de implementación en algoritmo Matlab con las referencias respectivas al codigo fuente o archivo de datos respectivo.

Por último, previo a las conclusiones, se realiza un analisis de compatibilidad con las formas del presente estandard en relación a las especificaciones de los equipos disponibles actualmente en el SIAG ya sea en HF o en otras frecuencias como ser VHF y UHF.
Finalizando , se dan las conclusiones de los apspectos salientes de los diseños involucrados.

3.1 *Introducción*

El Estandard MIL 188-110B [3] define una serie de formas de ondas de transmisión para poder lograr tasas de datos entre 75 bps y 12800 bps.
Estas formas de ondas pueden ser aplicadas a modem que utilicen portadora simple (Sección 5 y Apéndice C [3]) o modems de multiportadoras (Apéndices A y B [3]). En el diseño presente solo se diseña para portadora simple dado que la arquitectura de comunicación está compuesta por equipos transmisor y receptor de un solo canal de audio (3.4KHz) en simultáneo.
El mínimo de inter operatividad (cuerpo principal del estandar) para sistemas basados en canales HF es definido en los valores de 75, 150 bps, 300 bps, 600 bps, 1200 bps y 2400 bps utilizando formas de onda PSK definidas en el Capítulo 5 de [3].

El rango extendido de inter operatividad se define para las tasas de 3200 bps a 12800 bps utilizando las formas de onda detalladas en el Apéndice C de [3].

Las formas de onda que se tienen en cuenta en este proyecto son las siguientes:

1. Requerimientos de mínima (baja velocidad): las formas de onda (modulaciones) propuestas son las siguientes: FSK band angosta y FSK banda ancha (150 bps, 300 bps , 600 bps y 1200 bps, Sección 5.2 [3]) y BPSK o QPSK.
2. Requermientos de máxima (alta velocidad): las formas de onda (modulaciones) propuestas son las siguientes: QPSK (3200bps), 8QPSK (4800bps), 16QAM (6400 bps) , 32QAM (8000 bps), 64QAM (9600 bps-12800 bps).

Respecto a lo anterior, el estandar define los terminos de inter operatividad tal que debe cumplirse al menos la menor tasa de bps para el caso mínimo cuando el canal HF está establecido y que debe cumplirse la máxima velocidad de bps cuando el canal HF tenga excelentes condiciones para el caso de máxima.

El diseño del MODEM HF implica la implementación de las formas de onda de la Sección 5 (baja velocidad) y del Apéndice C (alta velocidad) en una arquitectura DSP siguiendo el concepto de 'Software Defined Radio' [8].

3.2 *Modulaciones de baja velocidad $R_b < 1200$ bps (pp 23, Secciones 5.1 y 5.2 [3])*

Para velocidades bajas en el estandard [3] se definen modulaciones FSK y QPSK. Las primeras pueden ser de fase discontinua (FSK) o continua (CPFSK). Para el caso QPSK se define el uso de dos mapeos, QPSK

y 8QPSK. A continuación se describirán los diseeños requeridos para las modulaciones FSK y CPFSK. QPSK y 8QPSK se detallaran mas adelante en la sección de modulaciones de alta velocidad dado que el diseño es similar siendo las de baja velocidad un subconjunto de las de alta. Por otro lado dado que con CPSK se puede implementar FSK indistintamente se utilizará una sola arquitectura (CPFSK) que contenga todas las combinaciones.

3.2.1 Fundamentos Teóricos FSK de fase discontinua

En FSK disocntinua simplemente se conmuta entre un par de osciladores de frecuencias f_1 y f_2, sin control del valor de la fase durante dicha conmutación, por lo se producen discontinuidades que generan un crecimiento espectral importante. La expresión de la señal modulada FSK $s(t)$ de fase discontinua es:

$$s_1(t) = A.cos(2\pi f_1 t + \theta_1), -- > bit, 1 \tag{3.1}$$

$$s_2(t) = A.cos(2\pi f_2 t + \theta_2), -- > bit, 0 \tag{3.2}$$

$$\tag{3.3}$$

donde

$$f_1 = f_c - \frac{1}{2T_s} \tag{3.4}$$

$$f_2 = f_c + \frac{1}{2T_s} \tag{3.5}$$

EL espaciamiento descripto por 3.4 y 3.5 implica que ambas portadoras son ortogonales entre si.

El diagrama en bolques para implementar la expresión 3.3 se puede observar en la figura 3.1. Dicha arquitectura es utilizada en las modulaciones de 75 bps, 150 bps banda angosta y 150 bps banda ancha.

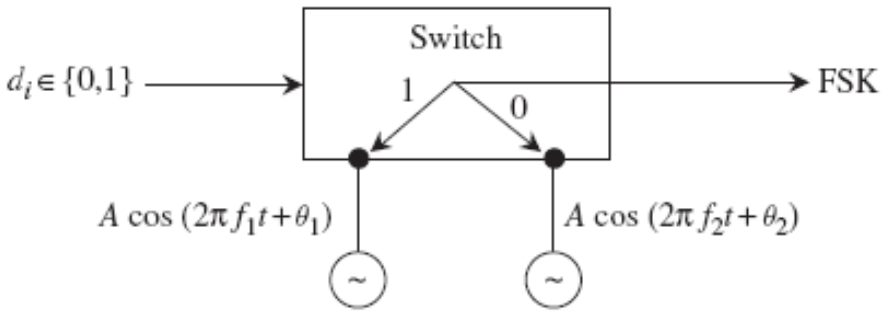

Figure 3.1: Diagrama en bloque para la implementacióin de FSK binaria con fase discontinua

3.2.2 Fundamentos Teóricos de FSK de fase continua

En este método, el cambio entre f_1 y f_2 se realiza coherentemente, es decir , se tiene control de la fase θ y por lo tanto se disiminuye el crecimiento espectral correspndiente. La figura 3.2 describe el esquema

general de la modulación. La expresión matematica de la señal modulada FSK $s(t)$ para fase continua es la siguiente:

$$s(t) = cos(p(t)\frac{\pi t}{T_s})A.cos(2\pi f_c(t)) - sin(p(t)\frac{\pi t}{T_s})A.sin(2\pi f_c(t)) \tag{3.6}$$

Donde:

- $p(t)$, es la forma del pulso (simbolo).
- $cos(p(t)\frac{\pi t}{T_b})$ mapea el símbolo al eje en fase.
- $sin(p(t)\frac{\pi t}{T_b})$ mapea el símbolo al eje en cadratura.

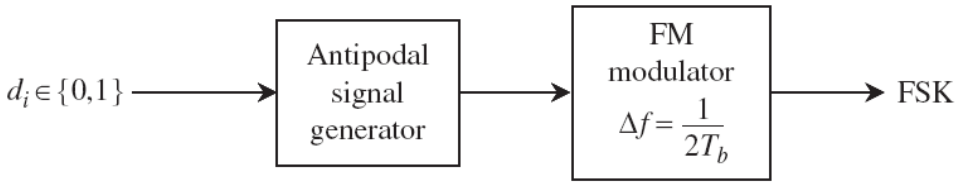

Figure 3.2: Diagrama en bloque para la implementacióin de FSK binaria con fase continua

Para el caso de simbolo NRZ sin filtrado $p(t)$ es ± 1 y entonces la ecuación 3.6 se resume a [15]:

$$s(t) = (\pm 1)cos(\frac{\pi t}{T_b})A.cos(2\pi f_c(t)) - (\pm 1)sin(\frac{\pi t}{T_b})A.sin(2\pi f_c(t)) \tag{3.7}$$

La expresión 3.6 es la implementación con modulador IQ de una modulación CPFSK. El diagrama en bloques para implementar dicha expresión se puede ver en la figura 3.3. Esta modulación es requerida por el estandar para 300 bps , 600 bps y 1200 bps. En el presente diseño la arquitectura de 3.3 también se utiliza para 75 bps y 150 bps banda ancha y 150 bps angosta para simplicidad de diseño.

Por definición ambos sistemas (continuo y discontinuo) requiren el mismo ancho de banda de canal B_T pero, en caso de FSK de fase continua se genera menos interferencia en el canal adyancente.

El indice de modulación de FSK 'h' se define como $h = 2\Delta f.T_s$, donde $\Delta f = 1/2T_s$. Existen a grandes rasgos tres casos de B_T a tener en cuenta. Cuando h=1, el ancho de banda se aproxima $B_T = 3R_b$, cuando $h > 1$ se utiliza Carson $B_T = 2(2\Delta f + 1/2T_b)$ y cuando $h < 1$ $B_T < R_b$.

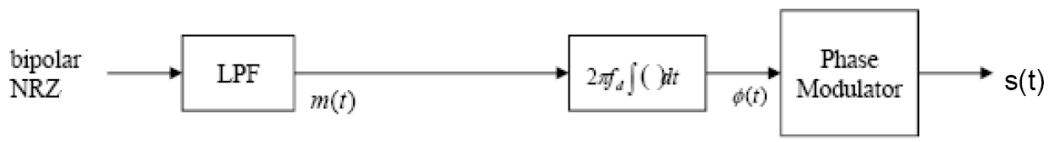

Figure 3.3: Diagrama en bloque para la implementación IQ de FSK binaria con fase continua

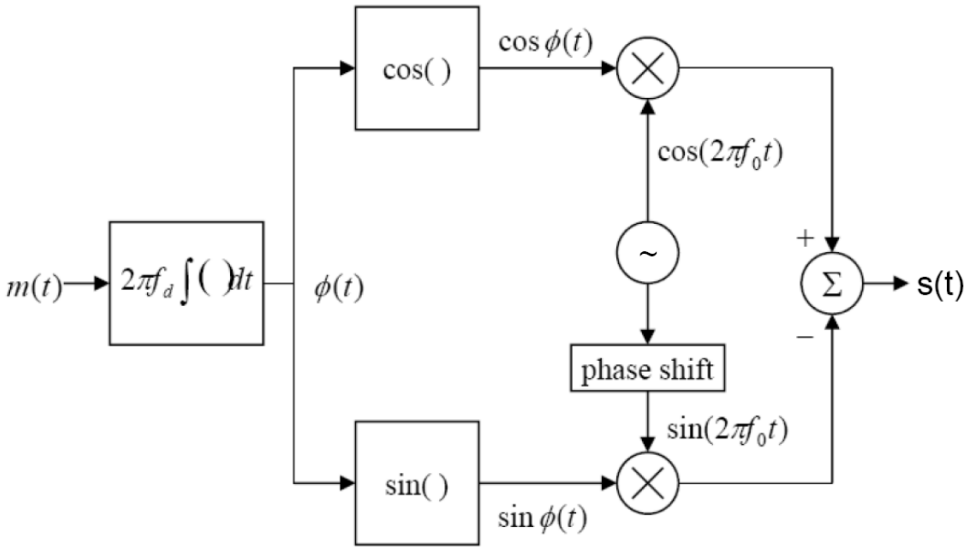

Figure 3.4: Diagrama detallado para la implementación IQ de FSK binaria con fase continua

En la figura 3.4 se puede ver el detalle de implementación de la figura 3.3.

3.2.3 Alfabeto de Pulsos Banda Base

- $d_i \in 0, 1$ para fase discontinua.
- Los d_i se convierten a NRZ $d_k = 2d_i - 1 \in -1, 1$ para fase continua.

3.2.4 Modulación FSK Banda Angosta <= 75 bps (pp 23, Tabla IV [3])

La expresión de la señal modulada $s(t)$ es definida por 3.3. Es una modulación FSK binaria ortogonal cuyo máximo desvío total $2\Delta f = 170Hz$, fase discontinua y frecuencia portadora f_c se elige acorde a la Tabla de la Fig. 3.1. Para el caso HF $f_c = 2000Hz$. Esta moudulación se implementa como la última opción cuando el estado del canal es muy malo.

Canal	MARK (Hz)	CENTRO (Hz)	SPACE (Hz)
LF Radio	915	1000	1085
MM Radio	1615	1700	1785
HF Radio	1575	2000	2425
UHF Radio	500	600	700

Table 3.1: Tabla de frecuencias para FSK banda angosta

El índice de modulación h es:

$$h = 2.\Delta f.T_b = 2.26 \tag{3.8}$$

De lo anterior resulta ser banda ancha y el B_T necesario se calcula por Carson:

$$B_T = 2(\Delta f + 1/T_b) = 2 * (85Hz + 75Hz) = 320Hz \tag{3.9}$$

3.2.5 Señales Temporales FSK banda angosta 75 bps

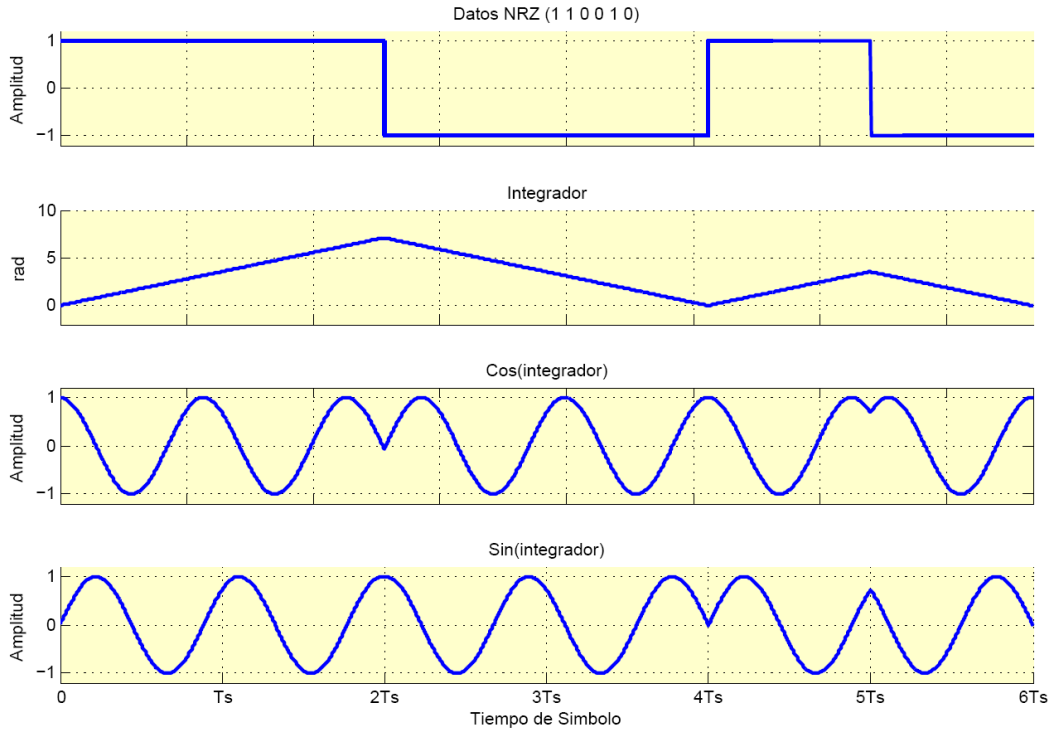

Figure 3.5: $1/T_s = 75Hz$

3.2.6 Espectros FSK banda angosta 75 bps

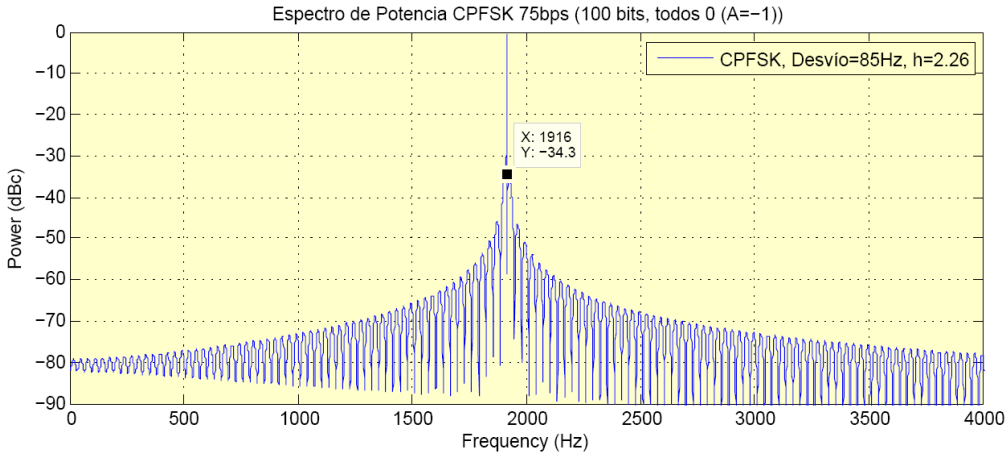

Figure 3.6: Secuencia de datos 'todos 0 $(-\Delta f)$'. $f_c = 2000Hz$

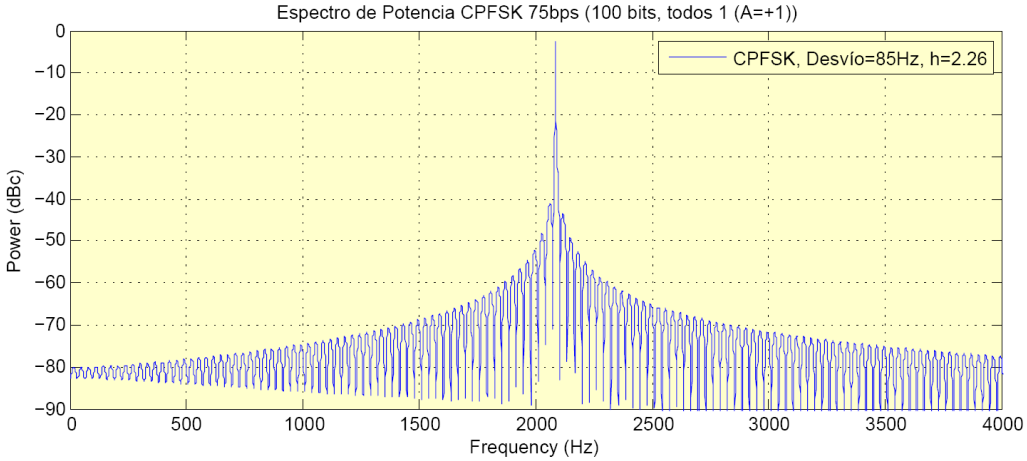

Figure 3.7: Secuencia de datos 'todos 1 $(+\Delta f)$'. $f_c = 2000Hz$

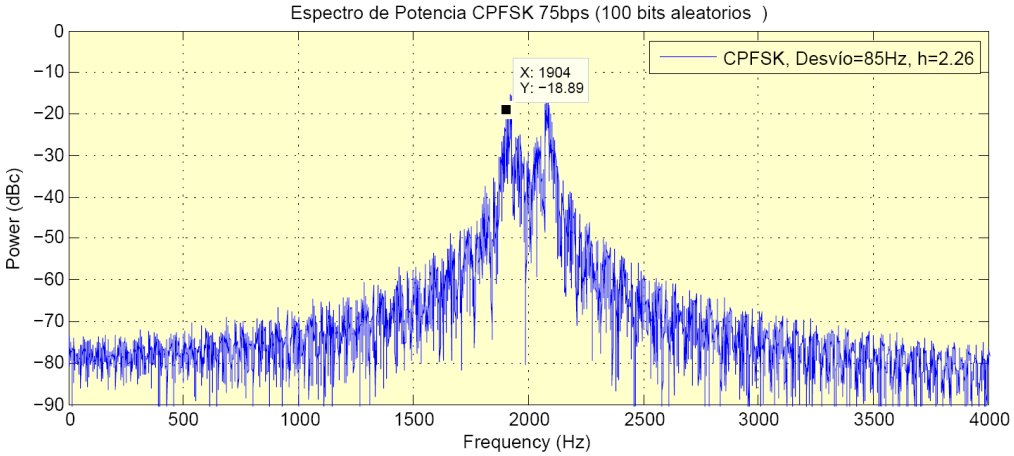

Figure 3.8: Secuencia de datos aleatorios. $f_c = 2000Hz$

Generado con: cpfsk75.m.

3.2.6.1 Resumen FSK Banda Angosta 75 bps

Parametros del Sistema	Valor	Unidades
f_c	2000	Hz
f_{low}	1575	Hz
f_{high}	2425	Hz
B_T	320	Hz
Δf	85	Hz
R_b	75	bps

Table 3.2: Resumen Parametros FSK banda angosta

3.2.6.2 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModData.c, ModIQ.c, Integrador.c, CODEC.c, Interfaz.c, TramaLow.c.
- Tablas Seno / Conseno para f_{low}, f_{up}, f_c.
- Tabla Fases Integrador.
- Tablas Sin(DatoQ), Cos(DatoI).

3.2.7 Modulación FSK 150 banda angosta bps (pp 24, Tabla VI [3])

La expresión de la señal modulada $s(t)$ es 3.3. Esta modulación presenta un desvío total de frecuencia máximo de $2.\Delta f = 85Hz$ con frecuencia portadora f_c de 1275Hz. Se trata de una modulación de fase no continua. Se puede utilizar para tasas de bps menores a la de la especificación.

El índice de modulación h es:

$$h = 2.\Delta f.T_b = 0.56 \tag{3.10}$$

El resultado verifica la condición de banda angosta por lo tanto el el B_T necesario se aproxima como:

$$B_T <= 2 * h * R_b <= 2 * 0.566 * 150 \approx 170Hz \tag{3.11}$$

3.2.7.1 Tabla de valores de Frecuencias (Tabla VI [3])

Parametros	Frecuencia (Hz)
MARK Frec.	1232.5
CENTER Frec.	1275.0
SPACE Frec.	1317.5

Table 3.3: Frecuencias para FSK 150 bps

3.2.8 Señales Temporales FSK banda angosta 150 bps

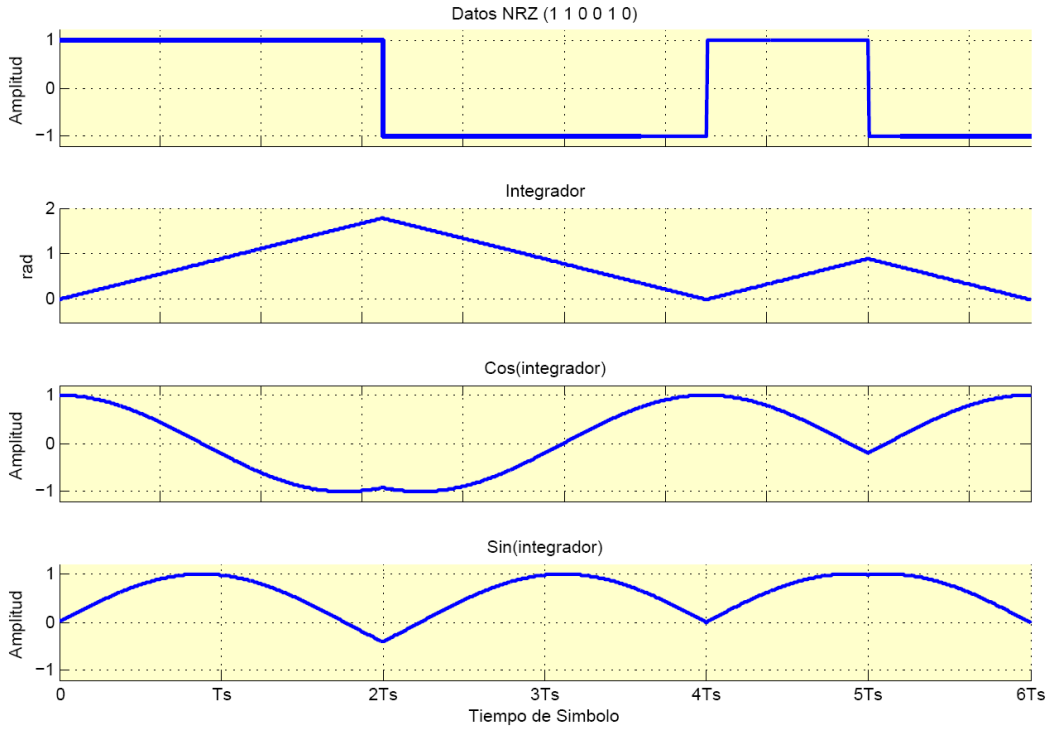

Figure 3.9: $1/T_s = 150Hz$

3.2.9 Espectros FSK banda ancha 150 bps

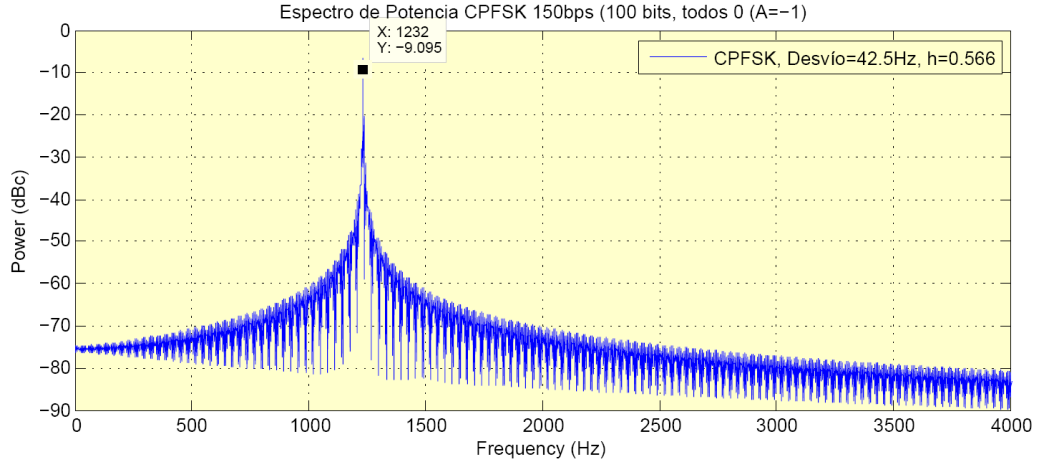

Figure 3.10: Secuencia de datos 'todos 0 $(-\Delta f)$'. $f_c = 1275Hz$

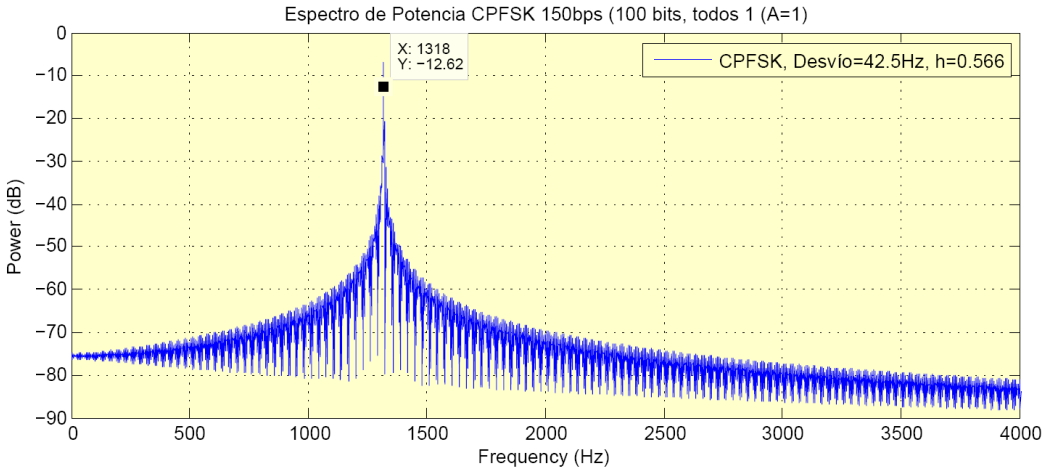

Figure 3.11: Secuencia de datos 'todos 1 $(+\Delta f)$'. $f_c = 1275Hz$

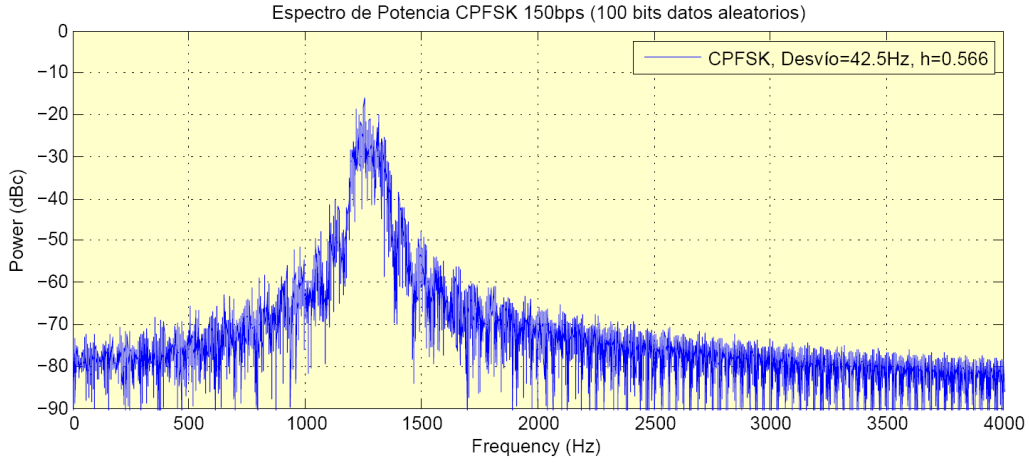

Figure 3.12: Secuencia de datos aleatorios. $f_c = 1275Hz$

Generado con: cpfsk150angosta.m.

3.2.9.1 Resumen FSK 150 banda angosta bps

Parametros del Sistema	Valor	Unidades
f_c	1275	Hz
f_{low}	1232.5	Hz
f_{high}	1317.5	Hz
B_T	170	Hz
Δf	42.5	Hz
R_b	150	bps

Table 3.4: Resumen Parametros FSK banda angosta 150 bps

3.2.9.2 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModData.c, ModIQ.c, Integrador.c, CODEC.c, Interfaz.c, TramaLow.c.
- Tablas Seno / Conseno para f_{low}, f_{up}, f_c.
- Tabla Fases Integrador.
- Tablas Sin(DatoQ), Cos(DatoI).

3.2.10 Modulación FSK Banda Ancha <= 150 bps (pp. 23, Tabla IV [3])

La expresión de la señal modulada $s(t)$ es 3.3. Está modulación esta definida específicamente para el canal HF, el máximo desvío total es $2\Delta f = 850$Hz, no es de fase continua, la frecuencia portadora f_c, acorde a la Tabla de la Fig 3.1 es de 2.000Hz.

El índice de modulación h es:

$$h = 2.\Delta f.T_b = 5.66 \tag{3.12}$$

Como resulta banda ancha el B_T necesario se calcula por Carson:

$$B_T = 2(\Delta f + 1/T_b) = 2 * (425Hz + 150Hz) = 1150Hz \tag{3.13}$$

3.2.11 Señales Temporales FSK banda ancha 150 bps

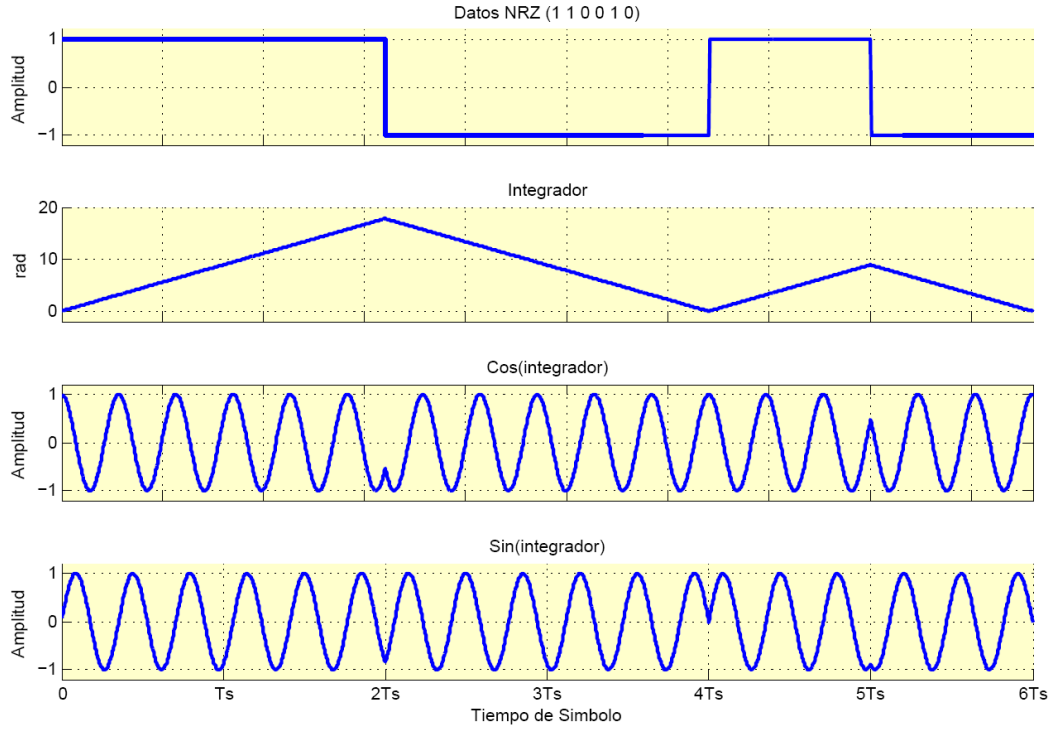

Figure 3.13: $1/T_s = 150Hz$

3.2.12 Espectros FSK banda ancha 150 bps

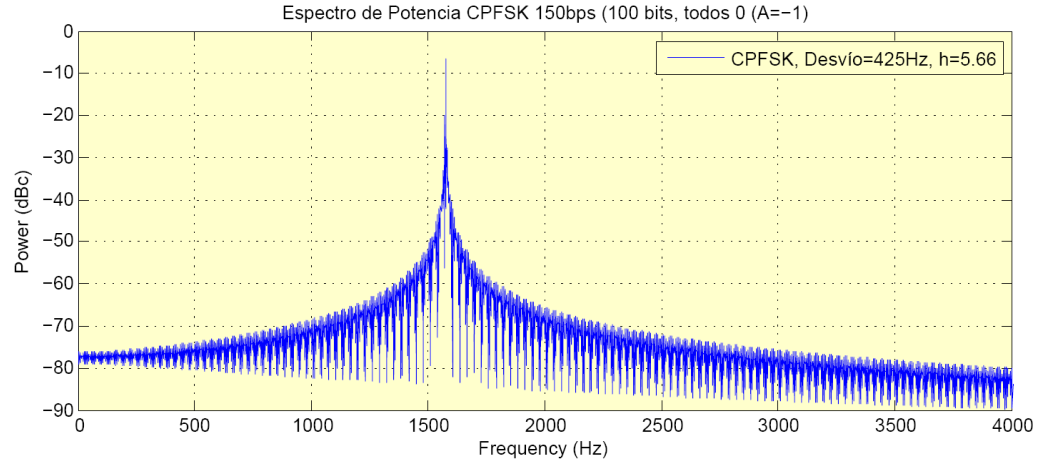

Figure 3.14: Secuencia de datos 'todos 0 $(-\Delta f)$'. $f_c = 2000Hz$

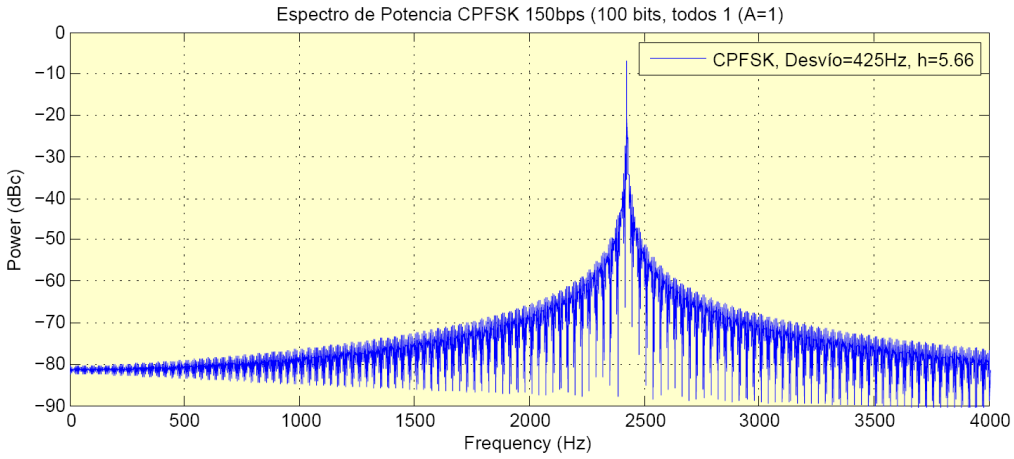

Figure 3.15: Secuencia de datos 'todos 1 $(+\Delta f)$'. $f_c = 2000Hz$

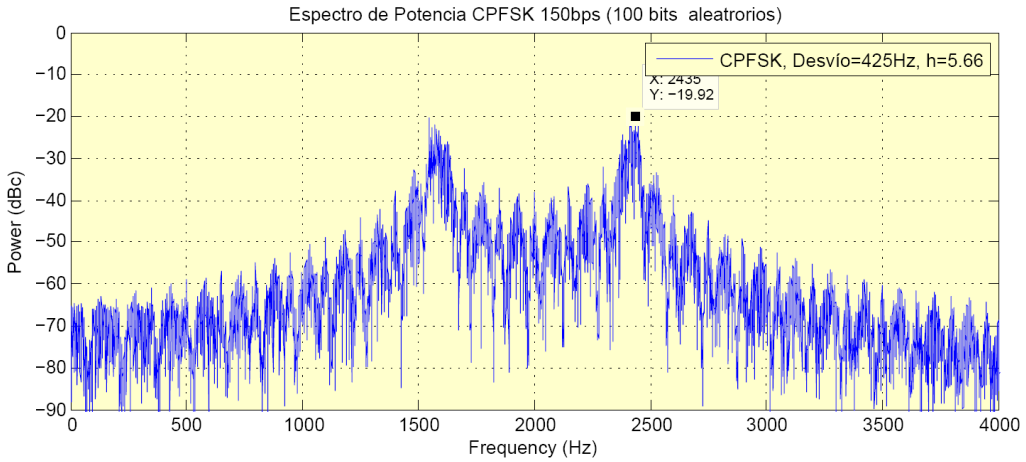

Figure 3.16: Secuencia de datos aleatorios. $f_c = 2000Hz$

Generado con: cpfsk150ancha.m.

3.2.12.1 Resumen FSK Banda Ancha 150 bps

Parametros del Sistema	Valor	Unidades
f_c	2000	Hz
f_{low}	1575	Hz
f_{high}	2425	Hz
B_T	1150	Hz
Δf	425	Hz
R_b	150	bps

Table 3.5: Resumen Parametros FSK banda ancha 150 bps

3.2.12.2 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModData.c, ModIQ.c, Integrador.c, CODEC.c, Interfaz.c, TramaLow.c.
- Tablas Seno / Conseno para f_{low}, f_{up}, f_c.
- Tabla Fases Integrador.
- Tablas Sin(DatoQ), Cos(DatoI).

3.2.13 Modulación FSK 300 bps pp 25, Tabla VII [3])

Esta modulación no está estrictamente definida por el estandar. Se presenta como una opción lenta de la versión FSK 600 bps. Es decir , utiliza la misma portadora y diferente Δf. Se trata de fase continua. El índice de modulación h es:

$$h = 2.\Delta f.T_b = 2 * 200 * (1/300) = 1.33 \tag{3.14}$$

Como resulta banda ancha el B_T necesario se se calcula por Carson:

$$B_T <= 2 * (\Delta f + 1/(2 * T_s)) = 700Hz \tag{3.15}$$

3.2.14 Señales Temporales FSK 300 bps

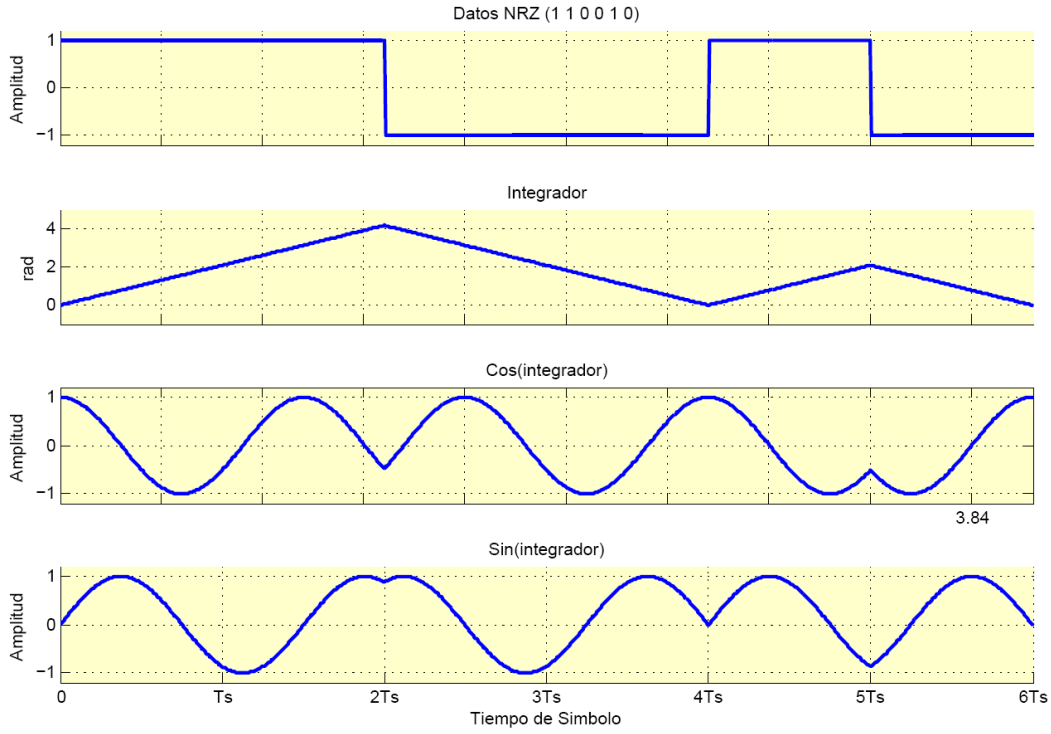

Figure 3.17: $1/T_s = 300Hz$

3.2.15 Espectros FSK banda ancha 300 bps

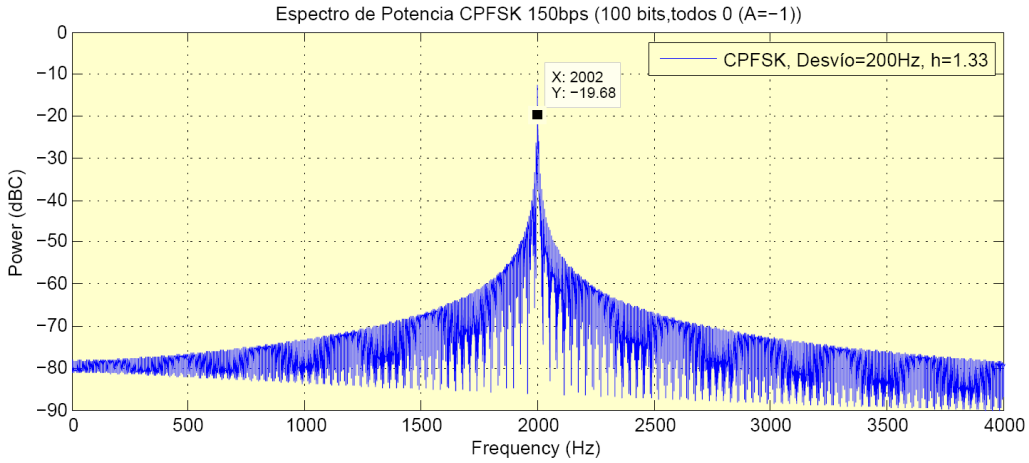

Figure 3.18: Secuencia de datos 'todos 0 $(-\Delta f)$'. $f_c = 2200Hz$

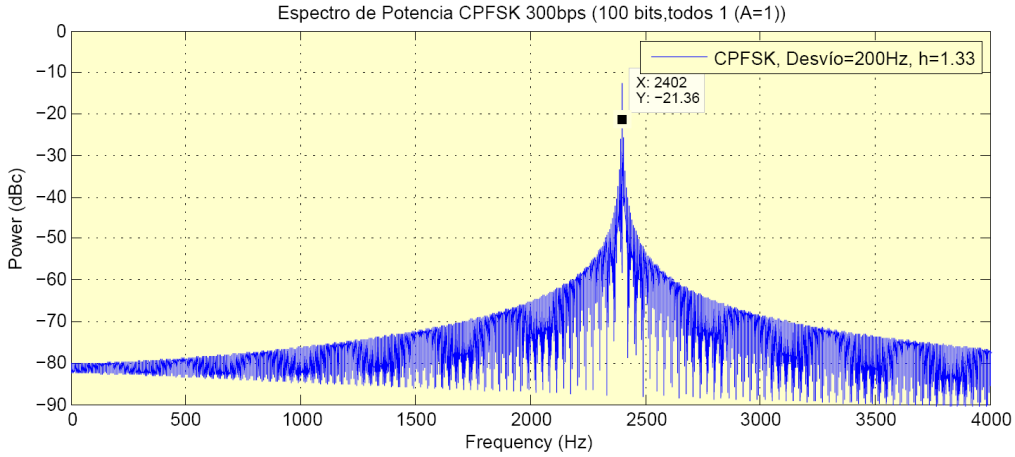

Figure 3.19: Secuencia de datos 'todos 1 $(+\Delta f)$'. $f_c = 2200Hz$

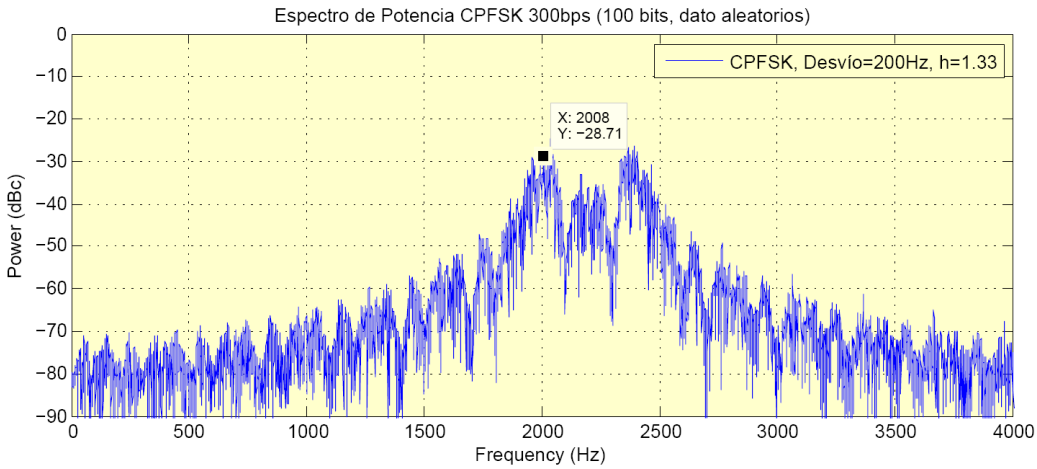

Figure 3.20: Secuencia de datos aleatorios. $f_c = 2200Hz$

Generado con: cpfsk300.m.

3.2.15.1 Resumen FSK banda ancha 300 bps

Parametros del Sistema	Valor	Unidades
f_c	2200	Hz
f_{low}	2000	Hz
f_{high}	2400	Hz
B_T	700	Hz
Δf	200	Hz
R_b	300	bps

Table 3.6: Resumen Parametros FSK banda ancha 300 bps

3.2.15.2 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModData.c, ModIQ.c, Integrador.c, CODEC.c, Interfaz.c, TramaLow.c.
- Tablas Seno / Conseno para f_{low}, f_{up}, f_c.
- Tabla Fases Integrador.
- Tablas Sin(DatoQ), Cos(DatoI).

3.2.16 Modulación FSK 600 bps (pp 25, Tabla VII [3])

La expresión de la señal modulada $s(t)$ es 3.6. Esta modulación presenta un desvío total de frecuencia máximo de $2.\Delta f = 400Hz$. Se trata de una modulación de fase continua. Se puede utilizar para tasas de bps menores a la de la especificación (hasta 150 bps) utilizando la misma portadora de 1500Hz, teniendo en cuenta los cambios correspondientes en los desvíos y anchos de banda.

El índice de modulación h es:

$$h = 2.\Delta f.T_b = 0.66 \tag{3.16}$$

Como resulta banda angonsta el B_T se aproxima de la siguiente manera:

$$B_T =<= 2 * h * R_b <= 2 * 0.66 * 600 \approx 800Hz \tag{3.17}$$

3.2.16.1 Tabla de valores de Frecuencias (Tabla VII [3])

Parametros	Frecuencia (Hz)
	600 bps (o menos)
	$\Delta f = 400Hz$
MARK Frec.	1300
CENTER Frec.	1500
SPACE Frec.	1700

Table 3.7: Frecuencias para FSK 600 bps

3.2.17 Señales Temporales FSK 600 bps

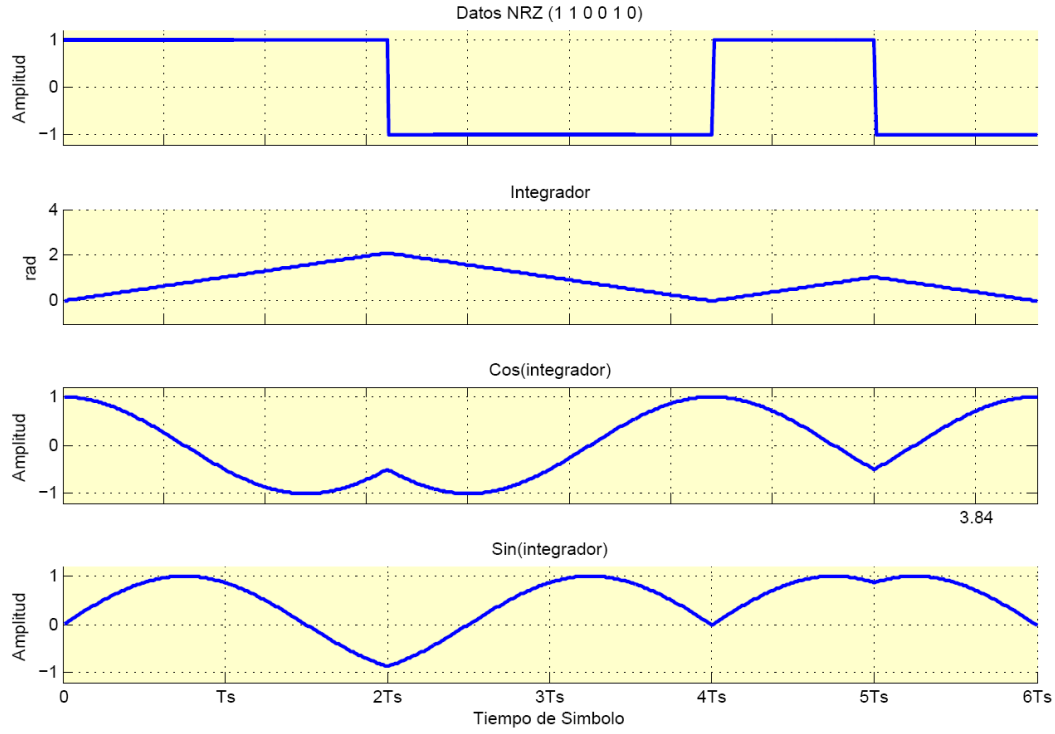

Figure 3.21: $1/T_s = 600Hz$

3.2.18 Espectros FSK banda ancha 600 bps

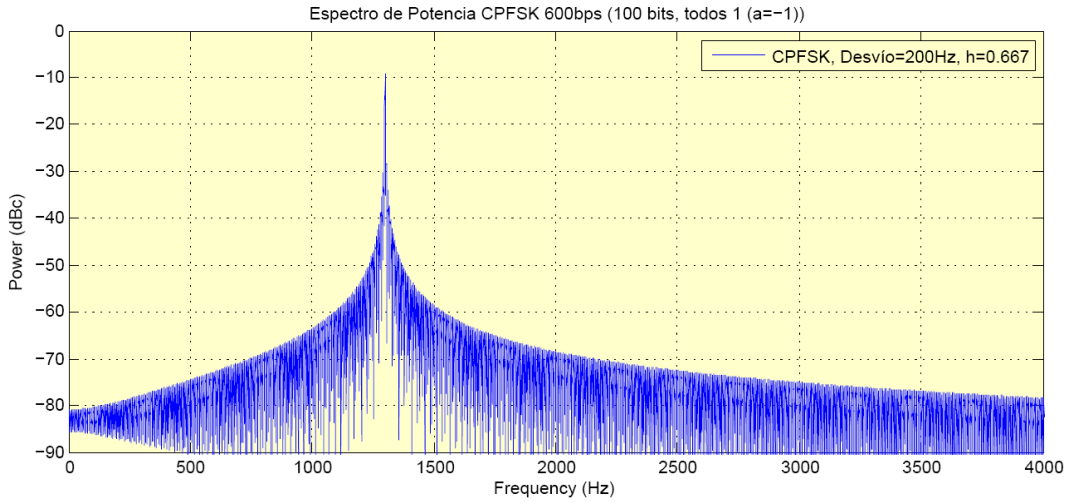

Figure 3.22: Secuencia de datos 'todos 0 $(-\Delta f)$'. $f_c = 1500Hz$

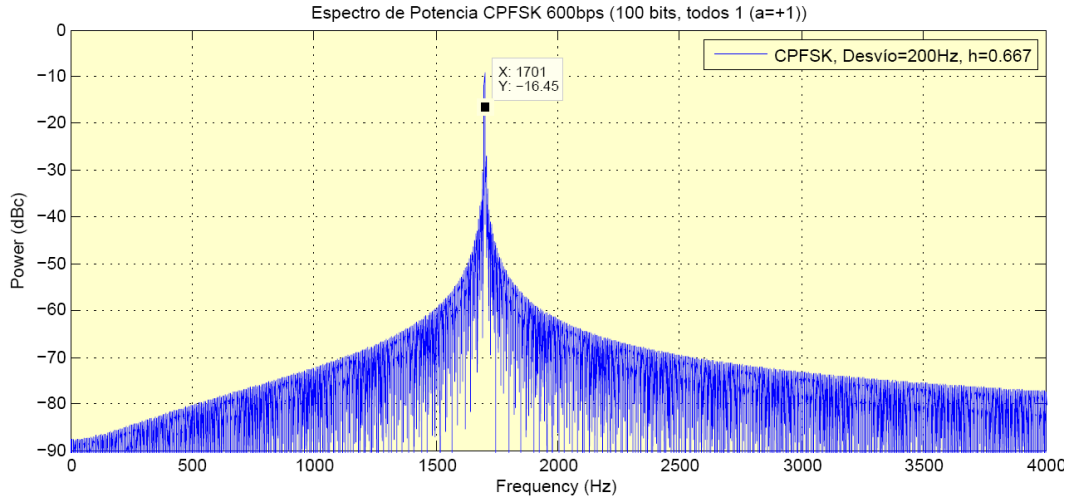

Figure 3.23: Secuencia de datos 'todos 1 $(+\Delta f)$'. $f_c = 1500Hz$

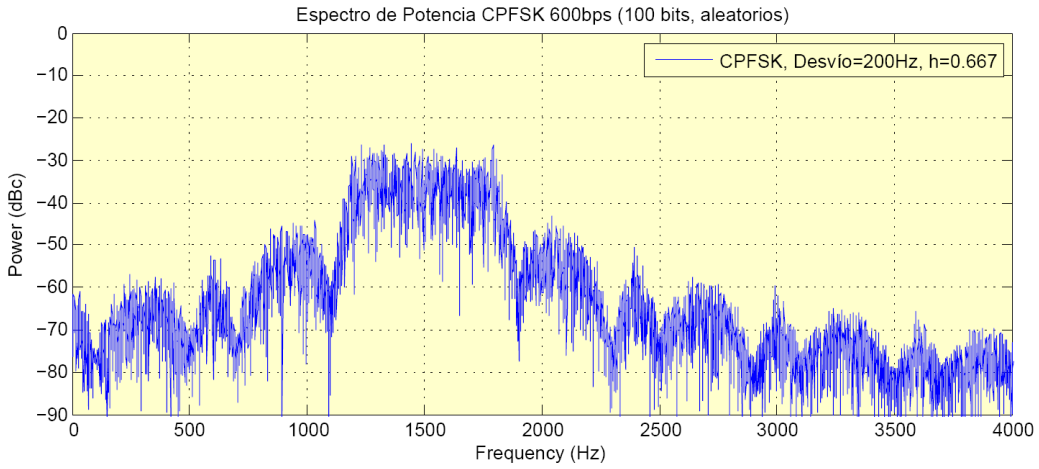

Figure 3.24: Secuencia de datos aleatorios. $f_c = 1500Hz$

Generado con: cpfsk600.m.

3.2.18.1 Resumen FSK 600 bps

Parametros del Sistema	Valor	Unidades
f_c	1500	Hz
f_{low}	1300	Hz
f_{high}	1700	Hz
B_T	1600	Hz
Δf	200	Hz
R_b	600	bps

Table 3.8: Resumen Parametros FSK banda angosta 600 bps

3.2.18.2 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModData.c, ModIQ.c, Integrador.c, CODEC.c, MascFilter.c, Interfaz.c, Trama.c.
- Tablas Seno / Conseno para f_{low}, f_{up}, f_c.
- Tabla Fases Integrador.
- Tablas Sin(DatoQ), Cos(DatoI).
- Tabla Filtro Máscara.

3.2.19 Modulación FSK 1200 bps (pp 25, Tabla VII [3])

La expresión de la señal modulada $s(t)$ es 3.6. Esta modulación presenta un desvío total de frecuencia máximo de $2.\Delta f = 800Hz$. Se trata de una modulación de fase continua. Se puede utilizar para tasas de bps menores a la de la especificación (hasta 600 bps) utilizando la misma portadora de 1700Hz, teniendo en cuenta los cambios correspondientes én los desvíos y anchos de banda.

El índice de modulación h es:

$$h = 2.\Delta f.T_b = 0.66 \tag{3.18}$$

Como resulta banda ancha el B_T necesario se aproxima de la siguiente manera:

$$B_T =<= 2 * h * R_b <= 2 * 0.66 * 1200 \approx 1600Hz \tag{3.19}$$

3.2.19.1 Tabla de valores de Frecuencias (Tabla VII [3])

Parametros	Frecuencia (Hz)
	1200 bsp (o menos)
	$\Delta f 800Hz$
MARK Frec.	1300
CENTER Frec.	1700
SPACE Frec.	2100

Table 3.9: Frecuencias para FSK 1200 bps

3.2.20 Señales Temporales FSK 1200 bps

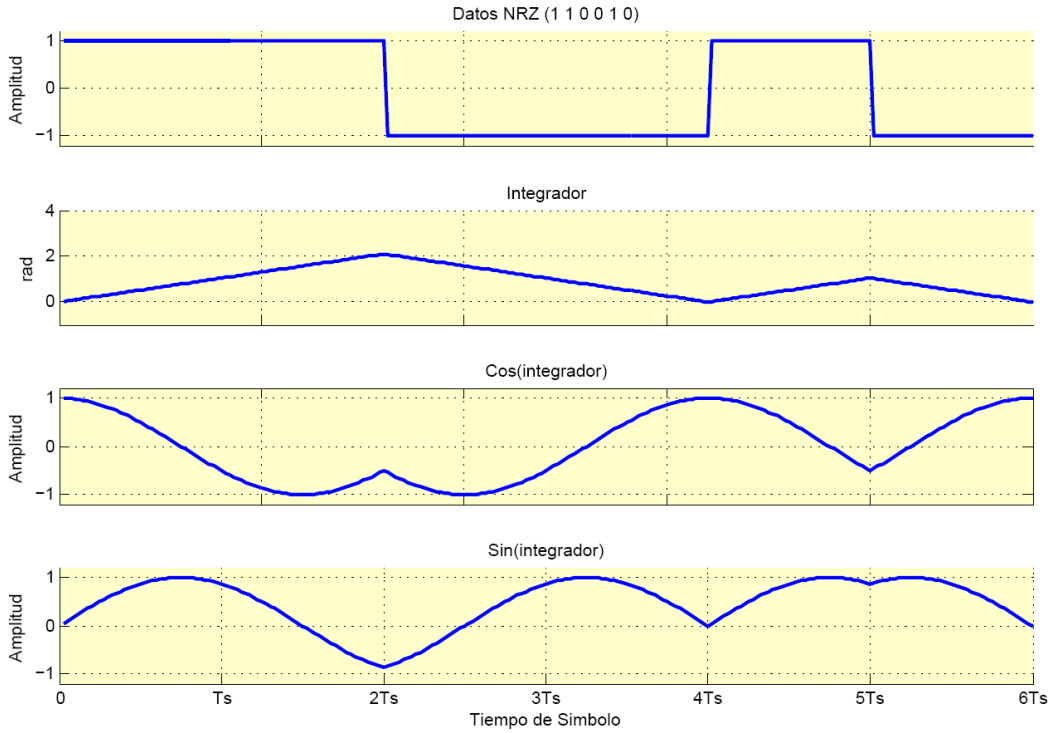

Figure 3.25: $1/T_s = 1200Hz$

3.2.21 Espectros FSK banda ancha 1200 bps

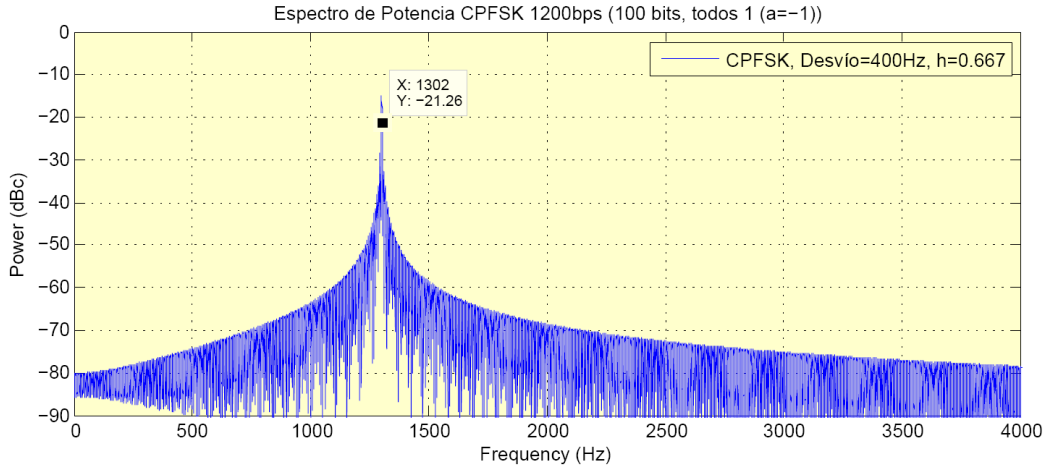

Figure 3.26: Secuencia de datos 'todos 0 $(-\Delta f)$'. $f_c = 1700Hz$

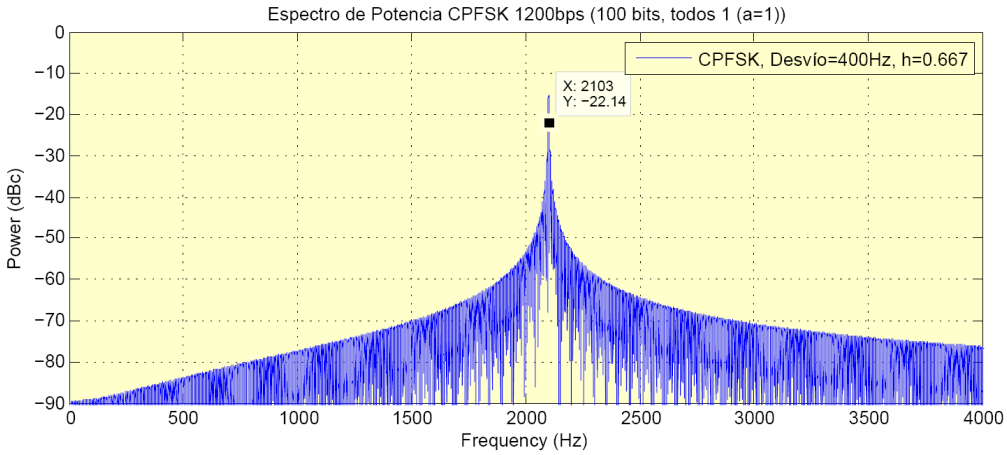

Figure 3.27: Secuencia de datos 'todos 1 $(+\Delta f)$'. $f_c = 1700Hz$

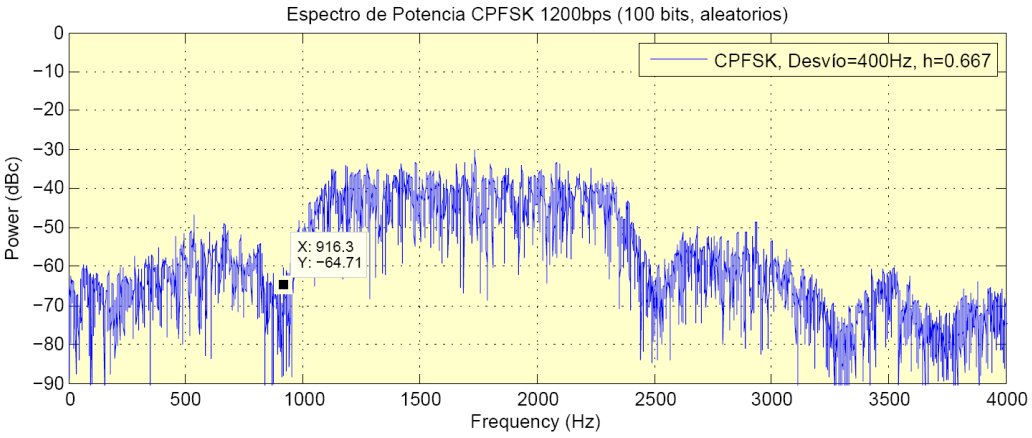

Figure 3.28: Secuencia de datos aleatorios. $f_c = 1700Hz$

Generado con: cpfsk1200.m.

3.2.21.1 Resumen FSK 1200 bps

Parametros del Sistema	Valor	Unidades
f_c	1700	Hz
f_{low}	1300	Hz
f_{high}	2100	Hz
B_T	1600	Hz
Δf	400	Hz
R_b	1200	bps

Table 3.10: Resumen Parametros FSK banda ancha 1200 bps

3.2.21.2 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModData.c, ModIQ.c, Integrador.c, CODEC.c, MascFilter.c, Interfaz.c, Trama.c.
- Tablas Seno / Conseno para f_{low}, f_{up}, f_c.
- Tabla Fases Integrador.
- Tablas Sin(DatoQ), Cos(DatoI).
- Tabla Filtro Máscara.

3.2.22 Máscara Espectral para las modulaciones FSK (pp 25, Sección 5.2.2.2 [3])

El valor de la mascara espectral indica que fuera de la banda (¿3400Hz) debe ser al menos de 40dB por debajo del máximo valor espectral dentro de la banda. Este valor se debe cumplir para cualquiera sea la tasa de bps que se utilice.

Como se puede ver, existen dos casos donde el requerimiento de máscara espectral no se cumple, para FSK600 y FSK1200. Por lo tanto se deben aplicar los filtros correspoondientes en cada caso como se describe a continuación.

3.2.22.1 Filtro de mascara para FSK600

El detalle del diseño se decribe en la sesión de FDATOOL $'Filtro600.fda'$. Se trata un FIR de 64 Taps cuya respuesta impulsiva y en frecuencia se muestra a continuación en la figura 3.29:

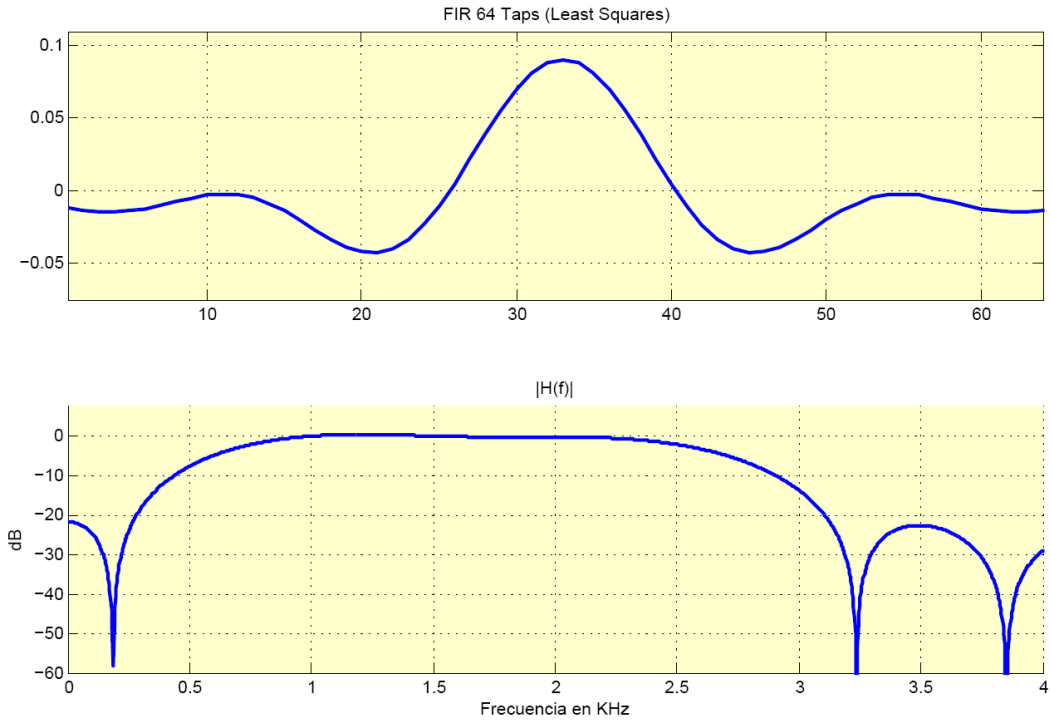

Figure 3.29: FIR least squares pasa banda

Los coeficientes del filtro se encuentran en el archivo 'FIRcpfsk600.mat'.
En la siguiente figura 3.30 se puede verificar que luego de la aplicación del filtro se cumple con la especificación de máscara espectral.

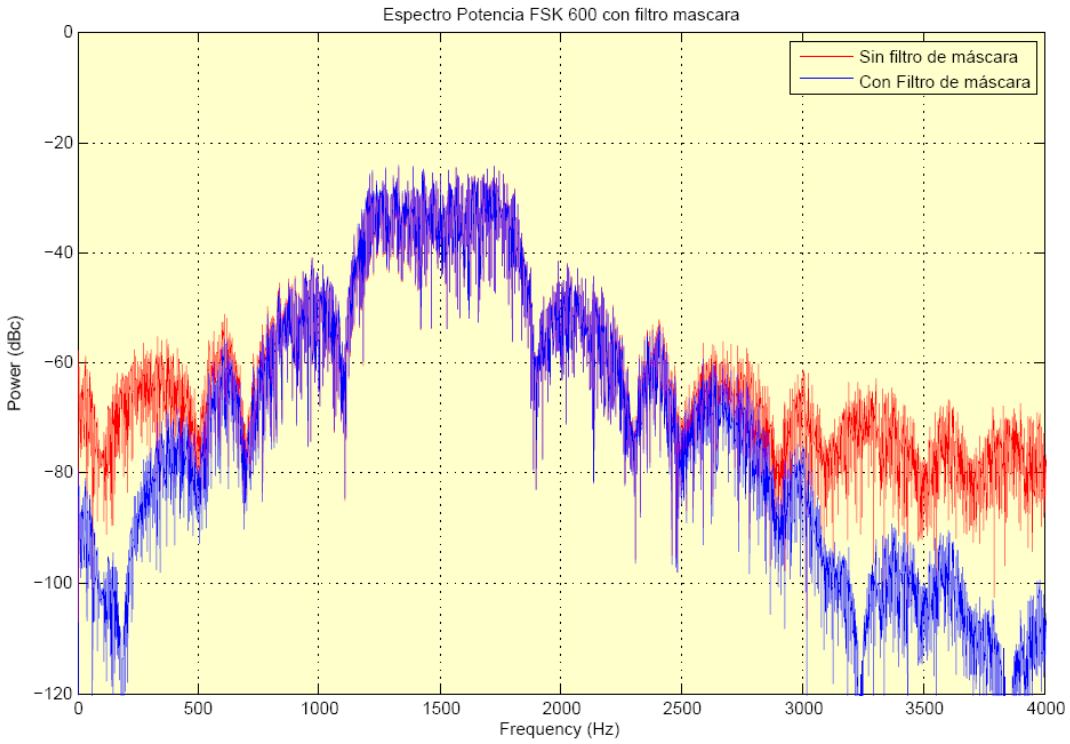

Figure 3.30: Comparación espectros FSK 600 con y sin filtro

3.2.22.2 Filtro de mascara para FSK1200

El detalle del diseño se decribe en la sesion de FDATOOL $'Filtro1200.fda'$. Se trata un FIR de 64 Taps cuya respuesta impulsiva y en frecuencia se muestra a continuacion en la figura 3.31:

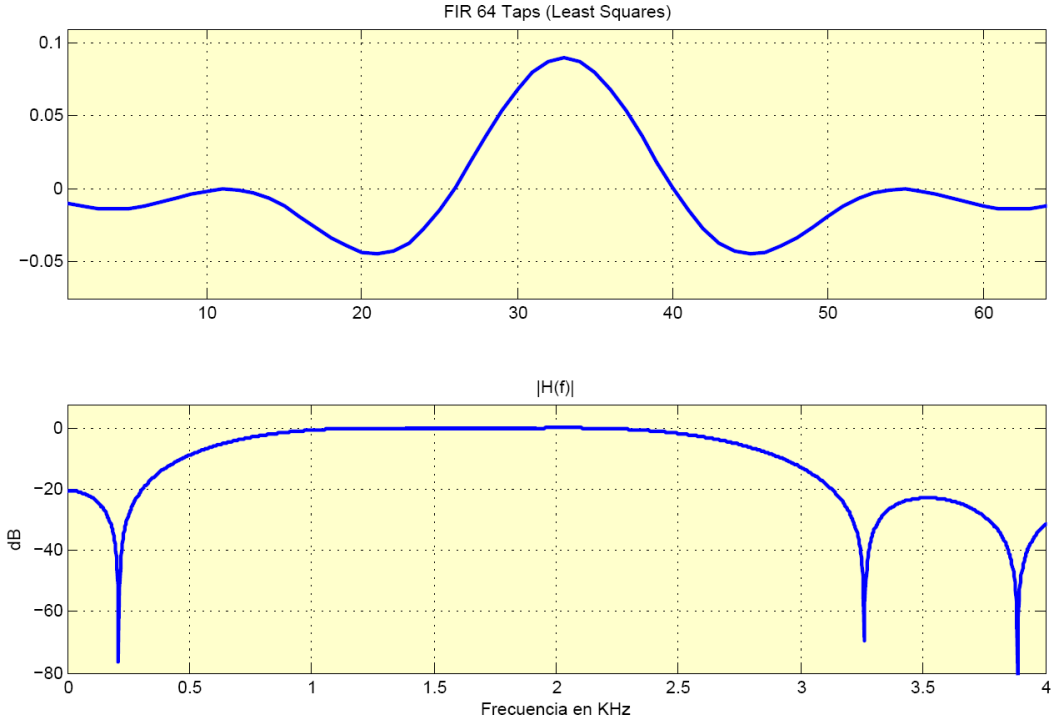

Figure 3.31: FIR least squares pasa banda

Los coeficientes del filtro se encuentran en el archivo 'FIRcpfsk1200.mat'.
En la siguiente figura 3.32 se puede verificar que luego de la aplicación del filtro se cumple con la especificación de máscara espectral.

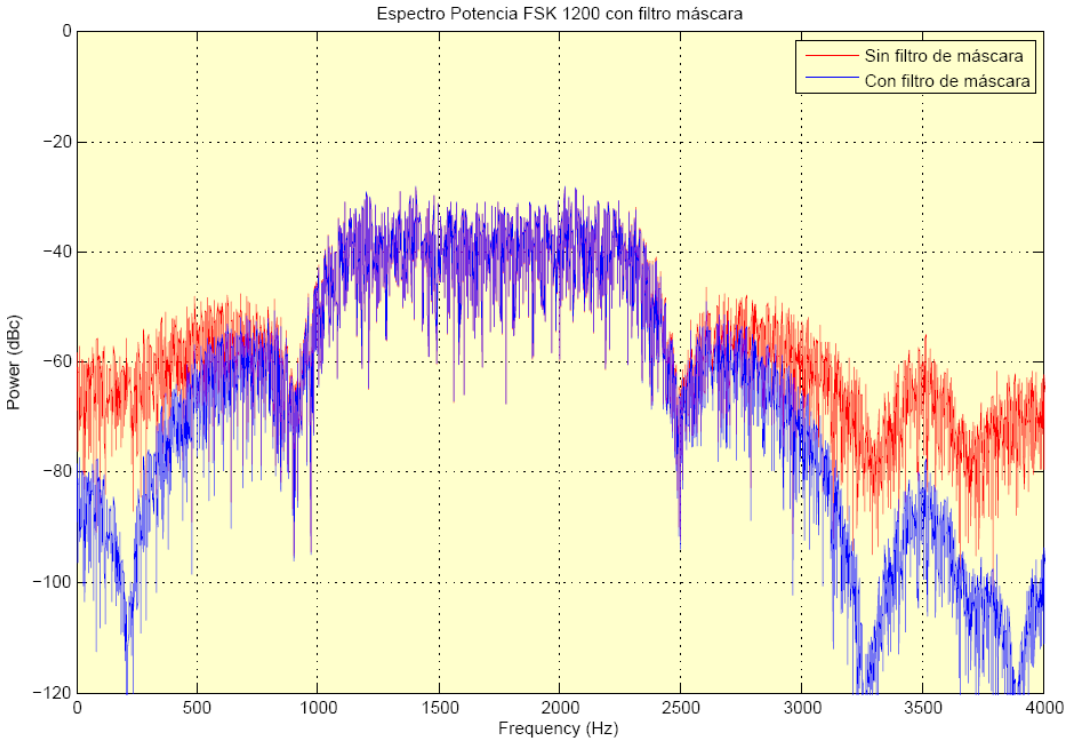

Figure 3.32: Comparación espectros FSK 1200 con y sin filtro

3.3 *Resumen general modulaciones* $R_b < 1200$ *bps*

La Tabla 3.3 muestra las caracteristicas principales de las modulaciones FSK:

MODEM	R_b bit/s	$2\Delta f(Hz)$	$f_c(Hz)$	$f_{low}(Hz)$	$f_{high}(Hz)$	Fase	Modulador
FSK-NB 75	75	170	2000	1575	2425	no CPM	fig 3.1 o fig 3.4
FSK-WB 150	150	850	2000	1575	2425	no CPM	fig 3.1 o fig 3.4
FSK150	150	85	1275	1232.5	1317.5	no CPM	fig 3.1 o fig 3.4
FSK300	<=300	300	2200	2100	2300	CPM	fig 3.4
FSK600	<=600	400	1500	1300	1700	CPM	fig 3.4
FSK1200	<=1200	800	1700	1300	2100	CPM	fig 3.4

Table 3.11: Comparación parametros moduladores FSK

3.3.1 Modulaciones QPSK 75 bps, 150 bps, 300 bps, y 600 bps

Las modulaciones FSK y CPFSK son previstas por el estandard para mantener compatibilididad con diversos sistemas (moduladores y demoduladores) pre-existentes hasta los 1200 bps. A su vez estandar en si mismo define sus propias modulaciones para estas velocidades mas la opción de 2400 bps en su

cuerpo principal. Todas las modulaciones son del tipo PSK y todas comparten la misma portadora de 1800Hz. La velocidad de símbolo será en función de la tasa d edatos. Las modulaciones de 75 bps - 1200 bps PSK son un subconjunto de las de 2400 bps, por lo tanto los detalles de implementación se darán la siguiente Sección dejando la presenta sección para los esquemas de ocmpatibilidad anterior (ver Tabla) 3.5 para los detalles.

3.4 Pulsos de formateo Raised Cosine (RC) y Root Raised Cosine (RRC) $\alpha = 0.33$

Previo a introducir las modulaciones que utlizan velocidad de símbolo $R_s = 2400Hz$ harermos una breve descripción del formateo de pulso requerido para evitar el ailiasing cuando el canal es limitado en banda y, de este modo transmitir una velocidad mayoren dicho canal. Según la teoría de Nyquist [16] la relación entre R_s(velocidad de datos), B_T (ancho de banda de transmisión) y el factor de roll-off (α) del filtro es:

$$B_T = \frac{R_S/2}{1+\alpha} \tag{3.20}$$

$$1600Hz = \frac{2400/2Hz}{1+\alpha} \tag{3.21}$$

Donde 1600 Hz es el ancho de banda disponible y 2400 Hz el ancho de banda requerido para puslos PAM. Entonces, el valor de roll off se calcula como sigue:

$$\alpha = \frac{1600Hz}{1200Hz} - 1 = 0.33 \tag{3.22}$$

Dado el factor de roll of se verifica, la resupuesta impulsiva (memoria) del filtro se extiende hasta $3T_s$. La ecuación que describe la respuesta impulsiva del filtro Raised Cosine (RC) es 3.23 [16]:

$$h(t) = \frac{sin\left(\frac{\pi t}{T_s}\right)}{\frac{\pi t}{T_s}} \frac{cos\left(\frac{\pi \alpha t}{T_s}\right)}{1-\left(\frac{2\alpha t}{T_s}\right)^2} \tag{3.23}$$

La figura 3.33 muestra el pulso coseno elevado para diferentes factores de roll off:

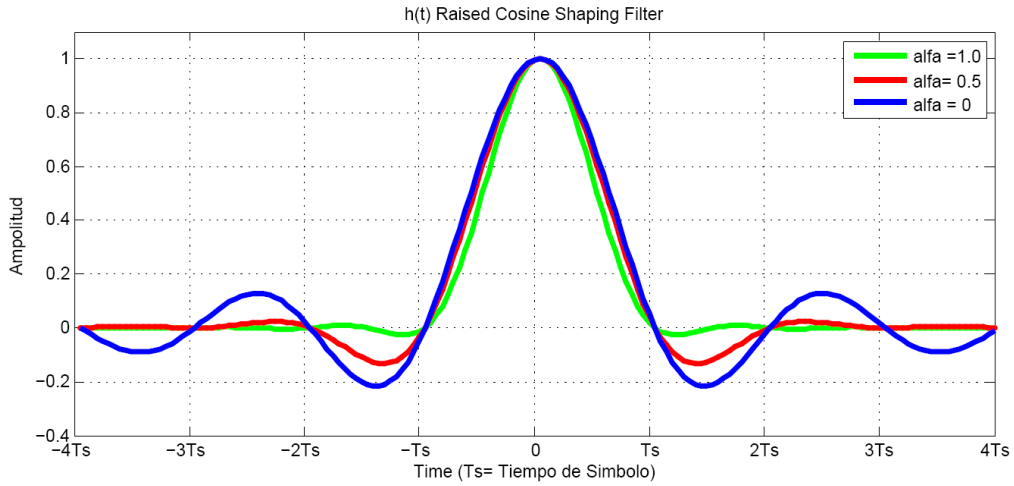

Figure 3.33: Pulso RC para α =0.0, 0.5 y 1.0

Cuya respectiva respuesta en frecuencia es descriptapor la ecuación 3.26 (Fig 3.34):

$$H(f) = T_s \Rightarrow 0 \leq |f| \leq \frac{1-\alpha}{2T_s} \tag{3.24}$$

$$H(f) = \frac{T_s}{2}\left[1 + cos\left[\frac{\pi T_s}{\alpha}\left(|f| + \frac{1-\alpha}{2T_s}\right)\right]\right] \Rightarrow \frac{1-\alpha}{2T_s} \leq |f| \leq \frac{1+\alpha}{2T_s} \tag{3.25}$$

$$H(f) = 0 \Rightarrow |f| > \frac{1+\alpha}{2T_s} \tag{3.26}$$

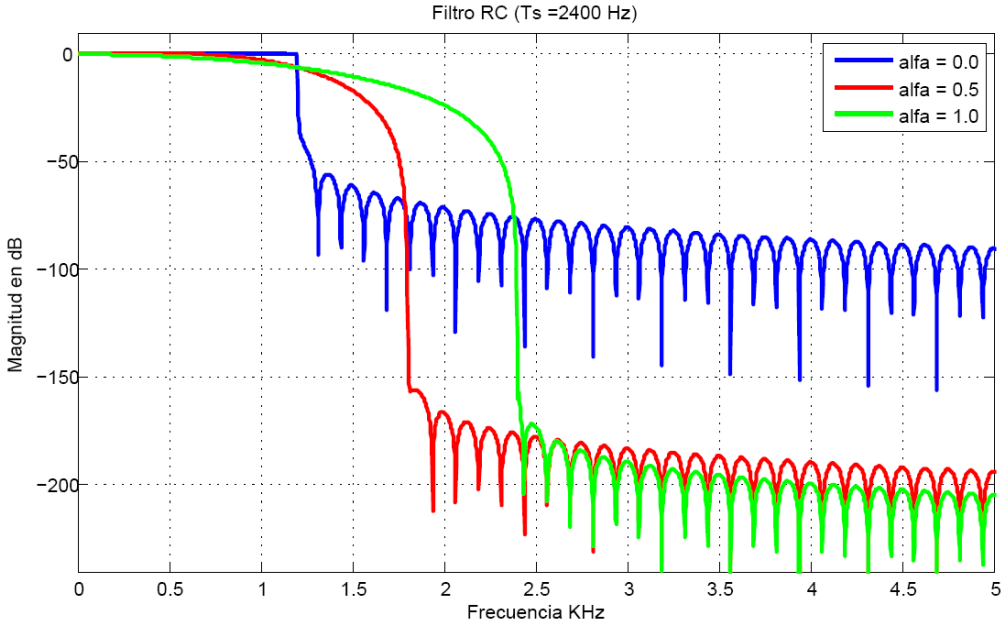

Figure 3.34: Espectro Pulso RC para α =0.0, 0.5 y 1.0

Cuando no se utiliza ecualizador y se usa matched filter es necesario usar Root Raised Cosine (RRC). La ecuación 3.27 describe la respuesta impulsiva del mismo [17]:

$$h(t) = \frac{4\alpha}{\pi\sqrt{T_s}} \left[\frac{cos\frac{\pi(1+\alpha)t}{T_s} + \frac{T_s}{4\alpha t} sin\frac{\pi(1-\alpha)t}{T_s}}{1 - \left(\frac{4\alpha t}{T_s}\right)^2} \right] \tag{3.27}$$

La figura 3.35 muestra el pulso coseno elevado para diferentes factores de roll off:

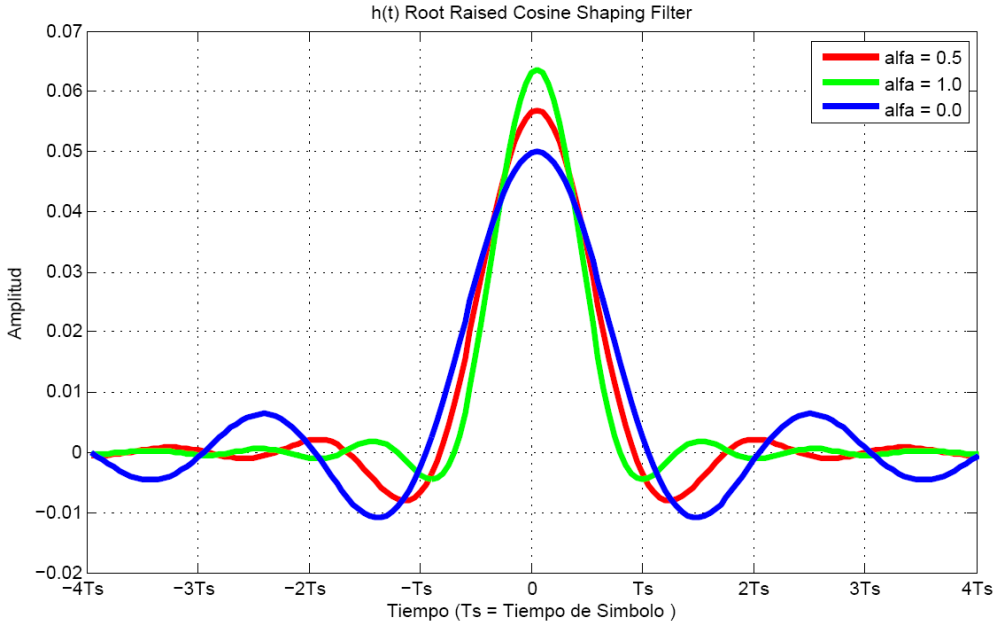

Figure 3.35: Pulso RRC para α =0.0, 0.5 y 1.0

La respuesta en frecuencia tiene la sgiuiente expresión 3.28 (Fig 3.36):

$$H_{RRC}(f) = \sqrt{|H_{RC}(f)|} \tag{3.28}$$

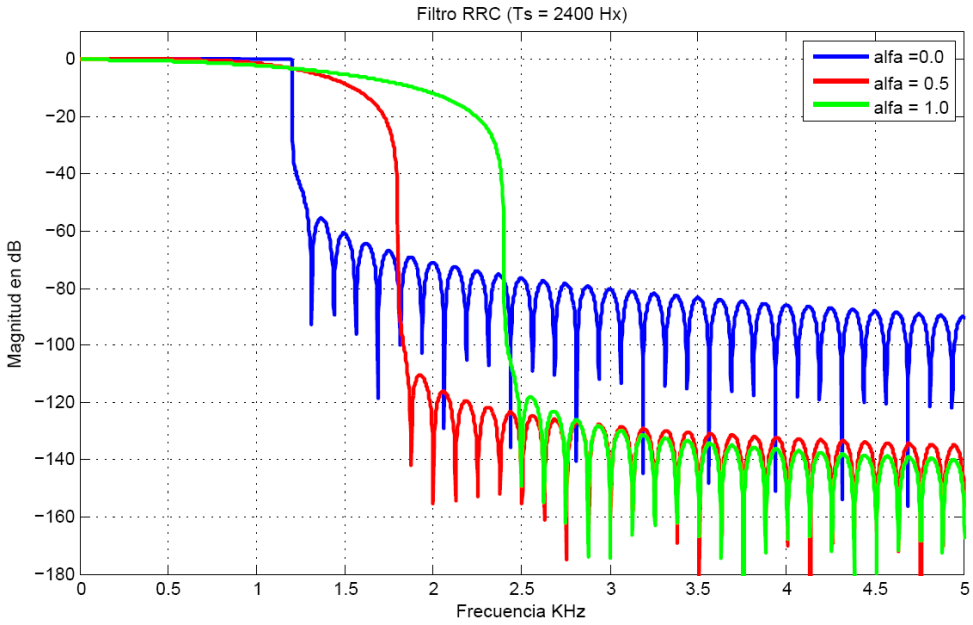

Figure 3.36: Espectro Pulso RRC para α =0.0, 0.5 y 1.0

3.5 *Modulaciones para* $1200 < R_b < 2400$ *bps (pp 94, Tabla C-II Apéndice C [3])*

La modulaciones que se utlizan para esta tasa de velocidad son dos, QPSK y 8QPSK. El mapeo está definido como lo muestra la figura 3.37 (es diferente al definido en el Apendice C). Para ambas modulaciones se utiliza la trama de baja velocidad. La portadora es siempre de 1800Hz y la velocidad de simoblo 2400Hz. Para evitar alisasing se utliza filtro de formtato de pulso coseno elevado $\alpha = 0.33$.

Vel. (bps)	Vel.	Método	Bits/Sím.	Vel.	Símbolos	Símbolos
Información	Código			Canal	Desconocidos	Conocidos
2400	1/2	1/2	3	4800	32	16
1200	1/2	1/2	2	2400	20	20
600	1/2	1/2	1	1200	20	20
300	1/4	1/2 rep. 2	1	1200	20	20
150	1/8	1/2 rep. 4	1	1200	20	20
75	1/2	1/2	2	150	Todos	0

Table 3.12: Parametros para formas de onda QPSK baja velocidad

Esta modulaciones estan definidas en el apartado 5.3.2.4 de [3]. En el mismo tambien se definen modulaciones para 4800 bps, 600 bps, 300 bps, 150 bps y 75 bps. Para el caso de la primera, 480 bps, no

se implementará pues se superpone en especificaciones con la similar presentada en la siguiente seccion (Apendice C).

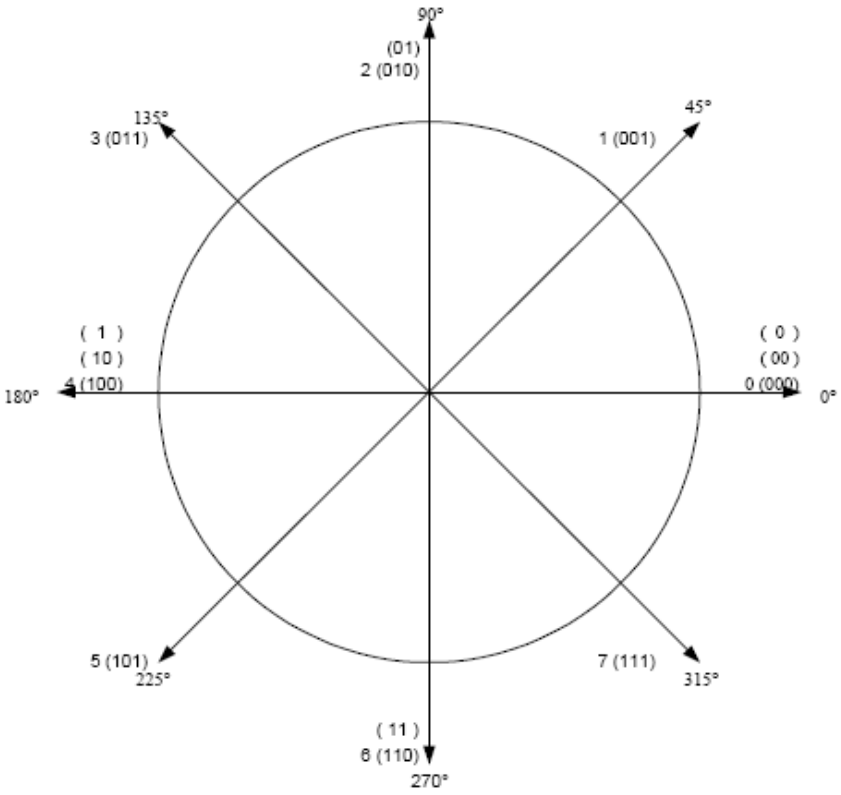

Figure 3.37: Mapeo para modulaciones PSK baja velocidad

La ecuación que describe el funcionamiento del modulador QPSK es la siguiente:

$$s(t) = p((k), t)cos(2\pi f_t t) + p((k-1), t)sin(2\pi f_t t) \tag{3.29}$$

La arquitectura de modulador que implementa la ecuación 3.29 se muestra en la figura 3.38

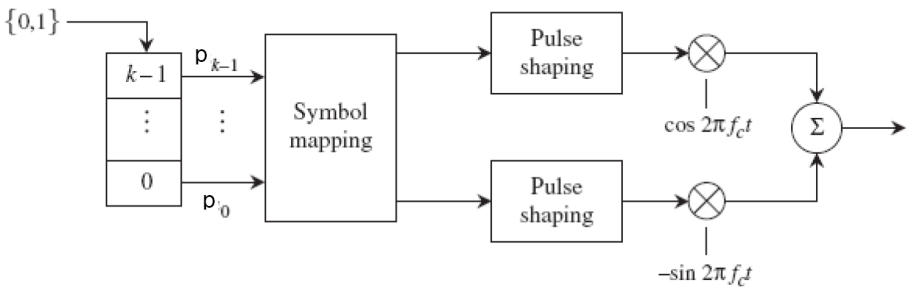

Figure 3.38: Modulador QPSK

NOTA: En la implementación el filtro RRC se realiza primero utilizando un modulador impulso y luego se aplica el filtro RRC.

3.5.1 QPSK $R_b = 1200$

3.5.1.1 Señales Temporales QPSK 1200 bps

Las siguientes figuras muestran una secuencia de datos determinada y su resepctiva salida cuando se utliza el filtro de formateo. Dado que en la implementación final es con filtro (3.43) la figura 3.42 se igual se incluye como referencia de testeo durante la fase de diseño.

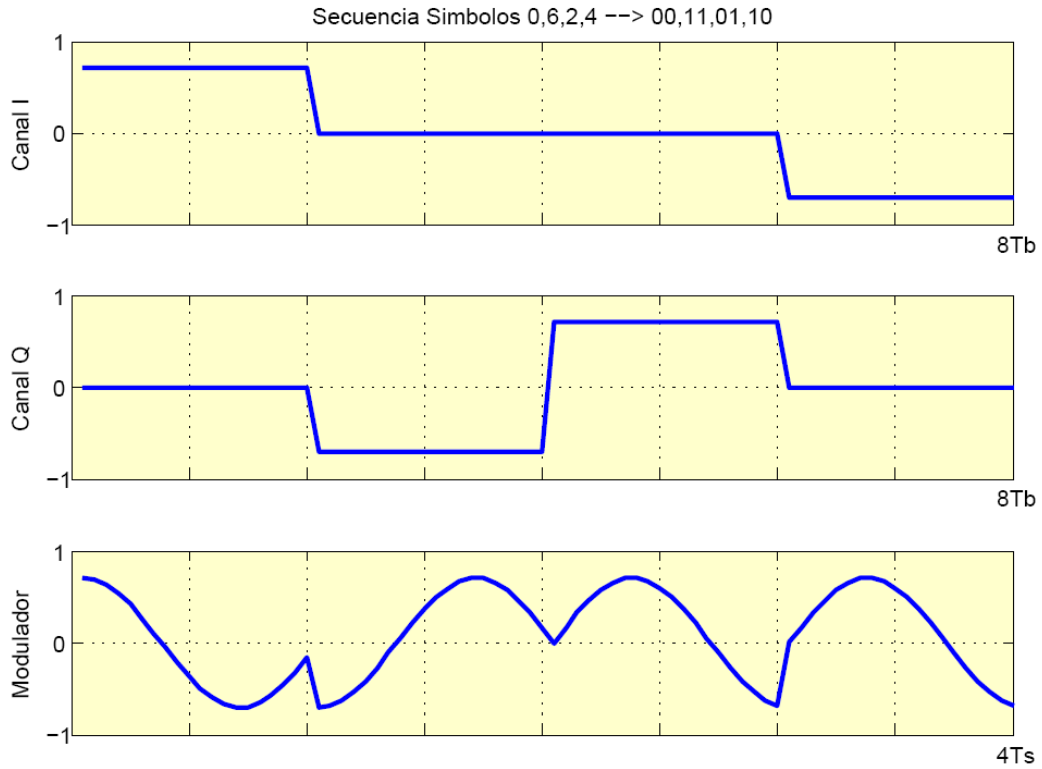

Figure 3.39: QPSK en tiempo para secuencia binaria de testeo fija

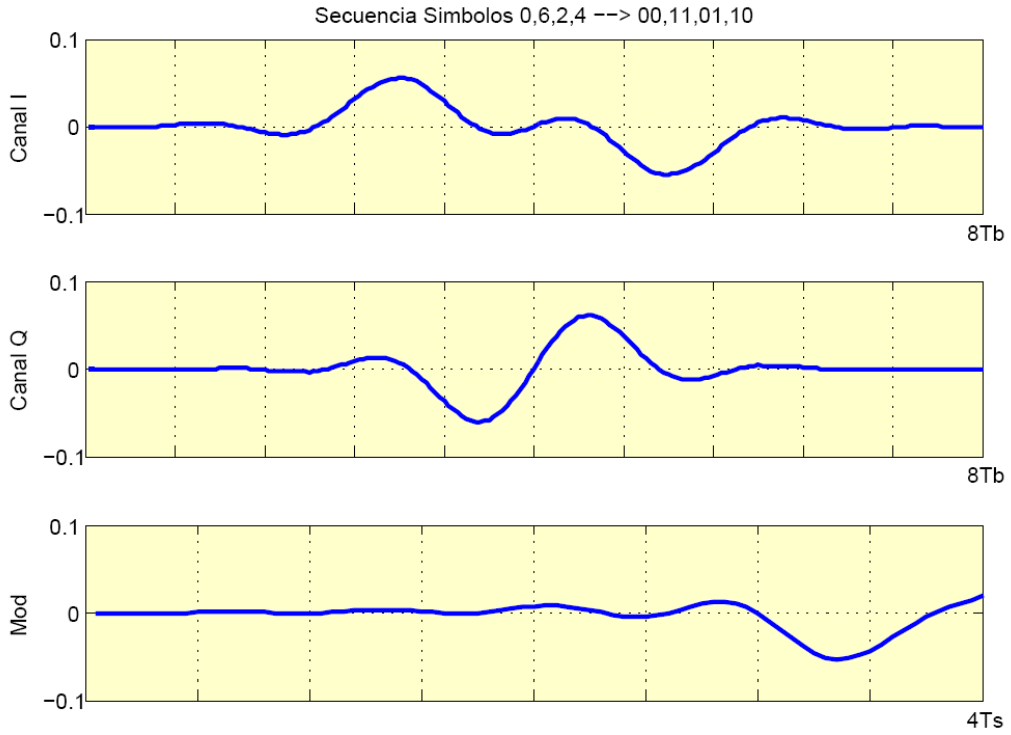

Figure 3.40: QPSK en tiempo con filtro de formato para secuencia binaria de testeo fija

3.5.1.2 Espectro QPSK 1200 bps

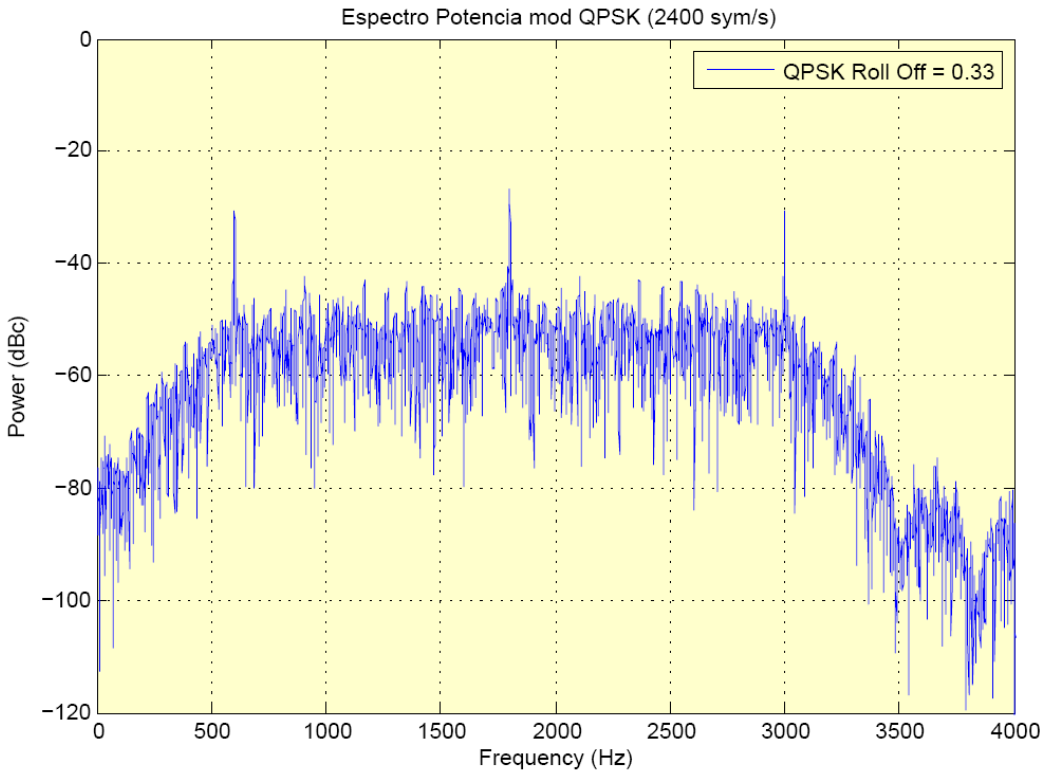

Figure 3.41: Espectro QPSK para secuencia aleatoria 600 símbolos

Las figuras fueron generadas con QPSKlow.m, los coeficientes del filtro RRC archivados en 'FIRQPSK48000.mat'.

3.5.2 Resumen QPSK $R_b = 1200$ bps

Parametros del Sistema	Valor	Unidades
f_c	1800	Hz
B_T	3200	Hz
R_s	2400	sps
R_b	1200	bps

Table 3.13: Resumen Parametros QPSK 1200 bps

3.5.2.1 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModIQ.c, CODEC.c, FormatFilter.c Interfaz.c, TramaLow.c.

- Tablas Seno / Conseno para f_c.
- Tabla Filtro Formato.

3.5.3 8QPSK $R_b = 2400$

Las siguientes figuras muestran una secuencia de datos determinada y su resepctiva salida cuando se utliza el filtro de formateo. Dado que en la implementación final es con filtro (**??**) la figura **??** se igual se incluye como referencia de testeo durante la fase de diseño.

3.5.3.1 Señales Temporales 8QPSK 2400 bps

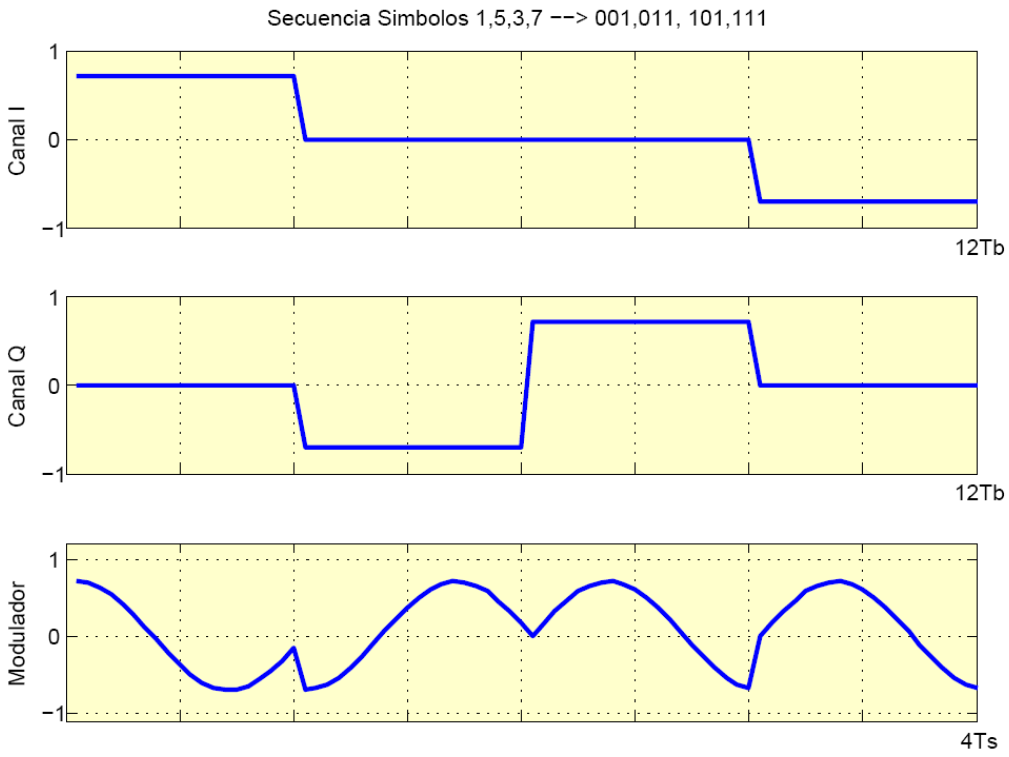

Figure 3.42: QPSK en tiempo para secuencia binaria de testeo fija

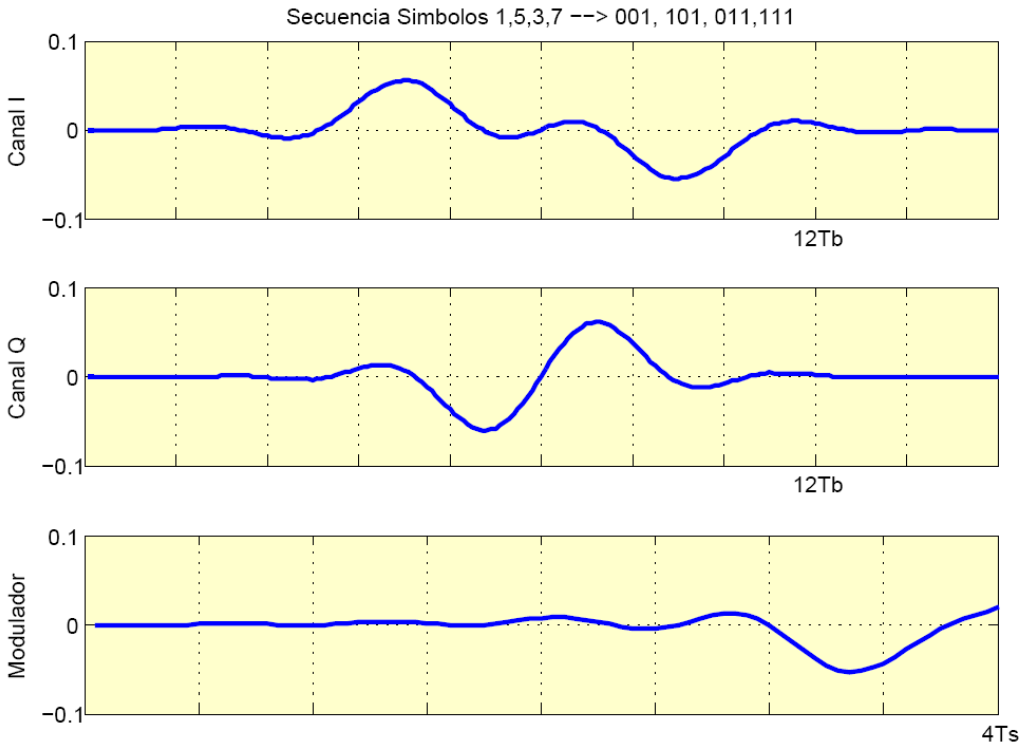

Figure 3.43: 8QPSK en tiempo con filtro de formato para secuencia binaria de testeo fija

3.5.3.2 Espectro 8QPSK 2400 bps

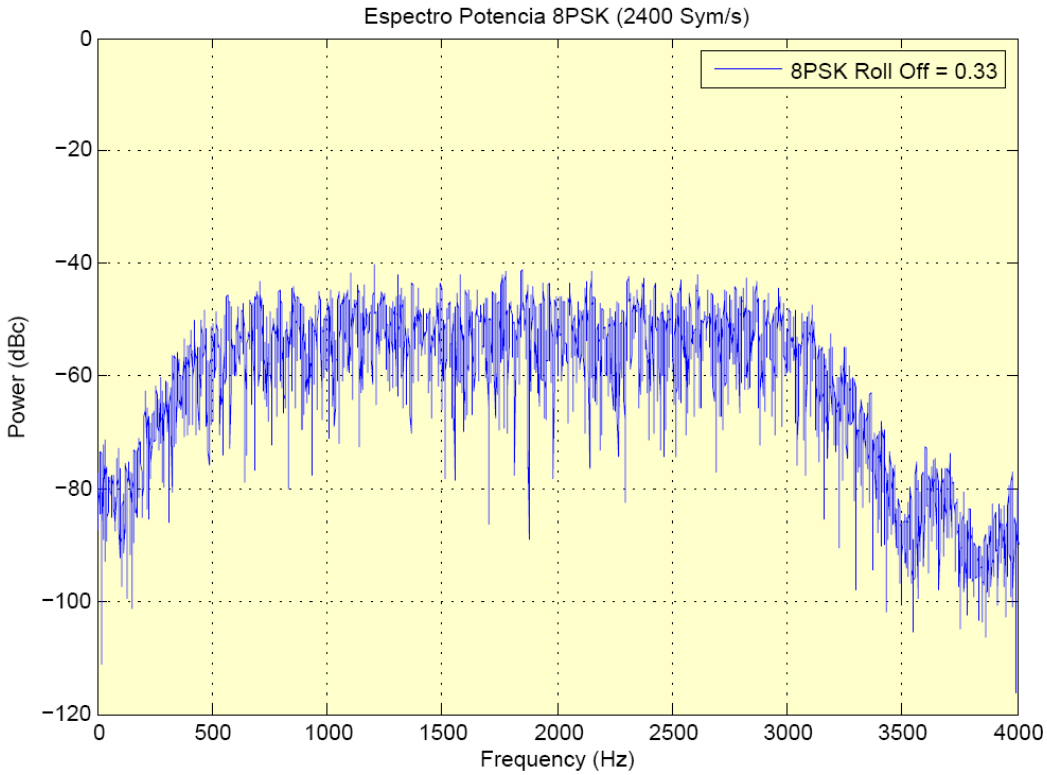

Figure 3.44: Espectro 8QPSK para secuencia aleatoria 600 símbolos

Las figuras fueron generadas con eigthQPSKlow.m, los coeficientes del filtro RRC archivados en 'FIRQPSK48000.mat'.

3.5.4 Máscara Espectral

En estas modulación la mascara espectral esta resuelta al utlilizar filtro de formato de pulsos Raised Cosine. Por lo tanto no require filtro de máscara espectral.

3.5.5 Resumen 8QPSK $R_b = 2400$ bps

Parametros del Sistema	Valor	Unidades
f_c	1800	Hz
B_T	3200	Hz
R_s	2400	sps
R_b	2400	bps

Table 3.14: Resumen Parametros 8QPSK 2400 bps

3.5.5.1 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModIQ.c, CODEC.c, FormatFilter.c Interfaz.c, TramaLow.c.
- Tablas Seno / Conseno para f_c.
- Tabla Filtro Formato.

3.5.6 Resumen General modulaciones $1200 < R_b < 3200$

MODEM	bps	$R_s(sym/s)$	$f_c(Hz)$	$f_{Null,low}(Hz)$	$f_{Null,high}(Hz)$	Filtro	Roll-off	Modulador
QPSK	1200	2400	1800	200	3400	RRC	0.33	fig 7.1
8QPSK	2400	2400	1800	200	3400	RRC	0.33	fig 7.1

Table 3.15: Comparación parametros moduladores QPSK y 8QPSK para canal [3](200Hz-3400Hz). Con filtro de formateo de pulso $\alpha = 0.33$

3.5.7 Acerca de las Velociades bps de las formas de onda $1200 < R_b < 3200$ [3]

Las velocidades bps que se muestran en la Tabla 3.15 para cada modulación son menores a las velocidades toericas respectivas. Esto de debe a dos motivos: primero, se agregan secuencias 'probe' para el entrenamiento del ecualizador y segundo los bits estan codificados por FEC. La máxima velocidad que el alcanza el estanard es aproximadamente 66% de la velocidad teórica un modular y se calcula de la siguiente manera:

$$Bits_{Tx,Total} = Bits_{Dato} + Bits_{Probe} = 32 + 16 = 48 \tag{3.30}$$

Por lo tanto el porcentaje de datos es:

$$\% = (Bits_{Dato}/Bits_{TxTotal}) * 100 = 0.66 \tag{3.31}$$

Pero de los $Bits_{Dato}$ solo el 50% son datos pues están codificado con un código 1/2 [3]. Entonces los datos reales transmitidos son:

$$Bits_{Tx,Dato} = 0.66 * .5 = 0.33 \tag{3.32}$$

3.6 *Modulaciones para* $3200 < R_b < 4800$ *bps (pp 94 Tabla C-II Apéndice C [3])*

Las modulaciones que permiten una tasa superior a los 3200 bps se presentan en la siguiente Tabla 3.16. Basicamente, se trata de QPSK y 8QPSK con los mapeos de bits descriptos en la Tablas C-I y C-III con velocidad de símbolo de $R_s = 2400sps$ y frecuencia de portadora $1800Hz$. La diferencia con las respectivas modulaciones QPSK y 8QPSK (presentadas en la sección anterior) es que estas modulaciones mapean los datos organizados con la trama de alta velocidad.

La ecuación que describe el funcionamiento del modulador QPSK es la siguiente (idem 3.29 y **??**):

$$s(t) = p((k), t)cos(2\pi f_t t) + p((k-1), t)sin(2\pi f_t t) \tag{3.33}$$

Las tasas de bits alcanzadas se muestran en la tabla 3.16

$R_b bps$	Modulación
3200	QPSK
4800	8QPSK

Table 3.16: Modulaciones propuestas por el estandard para lograr $R_b > 3200 < 4800$ bps

En función de los parametros disponibles del sistema se verifica [16] que el ancho de banda necesario para la tranamisión de las tasa de datos especificada no es suficiente por lo que es indispensable agregar filtro de formateo que permita aumentar la tasa de bits en el canal disponible. EL valor de roll off del filtro requerido es 0.33 (ver ec 3.22)
El canal banda base disponible es el el canal de voz telefónica que tiene las siguientes caracterisiticas:

- $B_T = 3200Hz$
- $f_{Null,min} = 200Hz$
- $f_{Null,max} = 3400Hz$

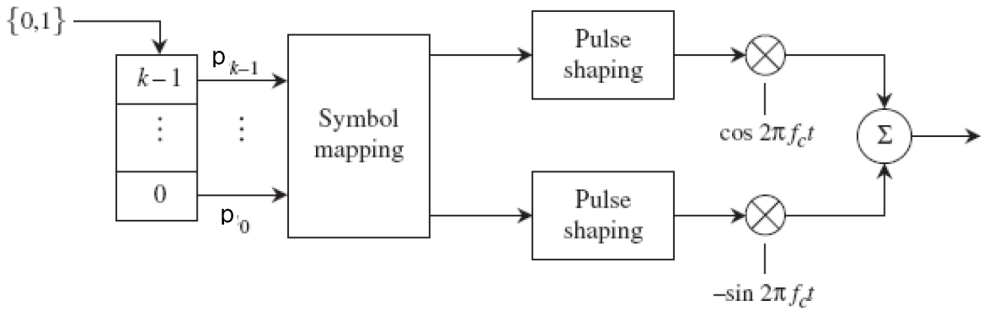

Figure 3.45: Diagrama en bloques para modulaciones QPSK (idem 3.37)

3.6.1 Alfabeto de Simbolos Binarios

$$d_k \in 0, 1 \tag{3.34}$$

3.6.2 Modulación QPSK 3200 bps (pp 94 Tabla C-III Apéndice C [3])

Esta primera modulación permite un máximo de 3200 bps. Para la misma el mapeo de 2 bits a símbolo se realiza con la siguiente Tabla 3.17, los simbolos mapean a la constelacion que muestra la figura 3.46 cuyos voltages I y Q son descriptos por la Tabla 3.21.

Dibit	Symbol
00	0
01	2
11	4
10	6

Table 3.17: Agrupación de 2 bits/símbolo para QPSK

3.6.2.1 Constelación QPSK Alta Velocidad

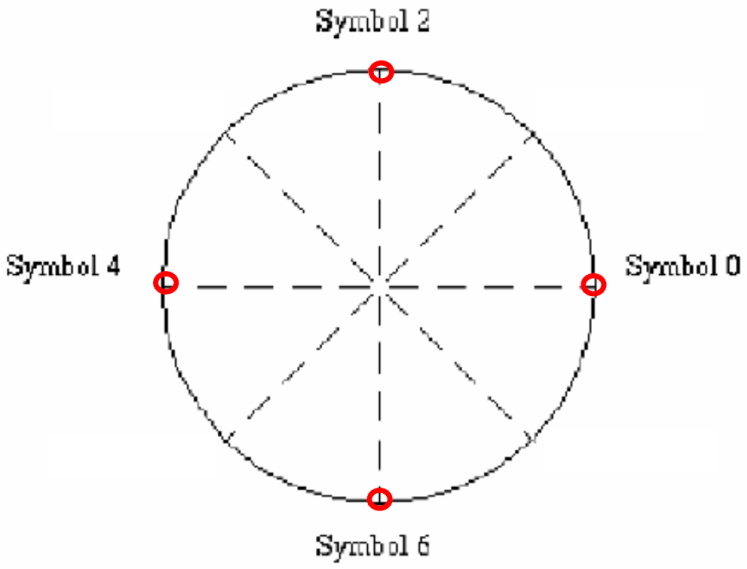

Figure 3.46: Constelación QPSK

3.6.2.2 Tabla de valores IQ (pp 94 Tabla C III [3])

El mapeo IQ para QPSK es un subconjunto de entre todos los valores posibles de la Tabla 3.21, la Tabla 3.21 muestra las fases con sus valores IQ:

Symbol	Phase	In-Phase	Quadrature
0	0	1.000000	0.000000
2	$\pi/2$	0.000000	1.000000
4	π	-1.000000	0.000000
6	$3\pi/4$	0.000000	-1.000000

Table 3.18: Mapeo de símbolos y fases de referencia para QPSK

3.6.2.3 Señales Temporales QPSK 3200 bps

Las siguientes figuras muestran una secuencia de datos determinada y su resepctiva salida con y sin filtro de formateo. A pesar que la implementación final es con filtro (3.48) la figura 3.47 igual se incluye como referencia de testeo durante la fase de diseño.

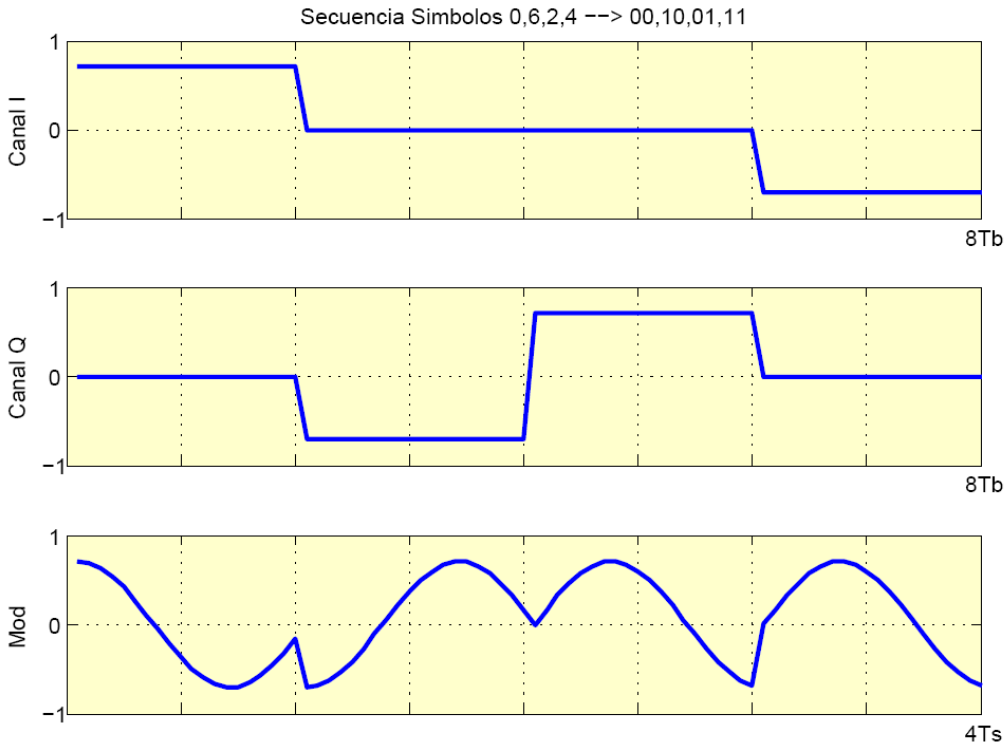

Figure 3.47: QPSK en tiempo para secuencia binaria de testeo fija

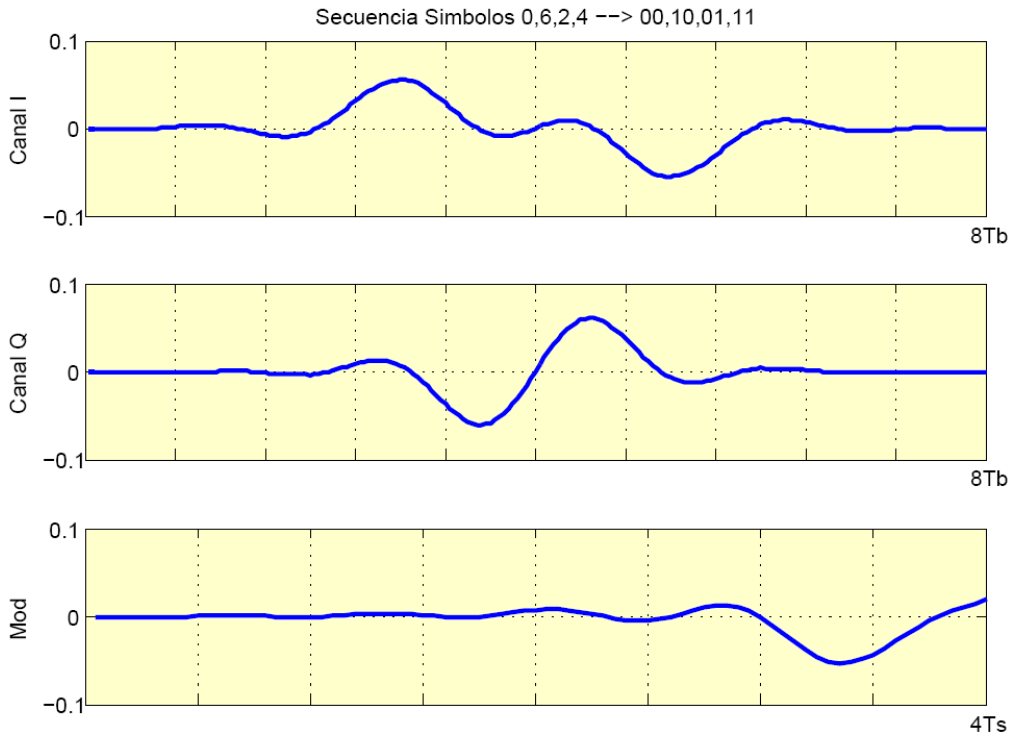

Figure 3.48: QPSK en tiempo con filtro de formato para secuencia binaria de testeo fija

3.6.2.4 Espectro QPSK 3200 bps

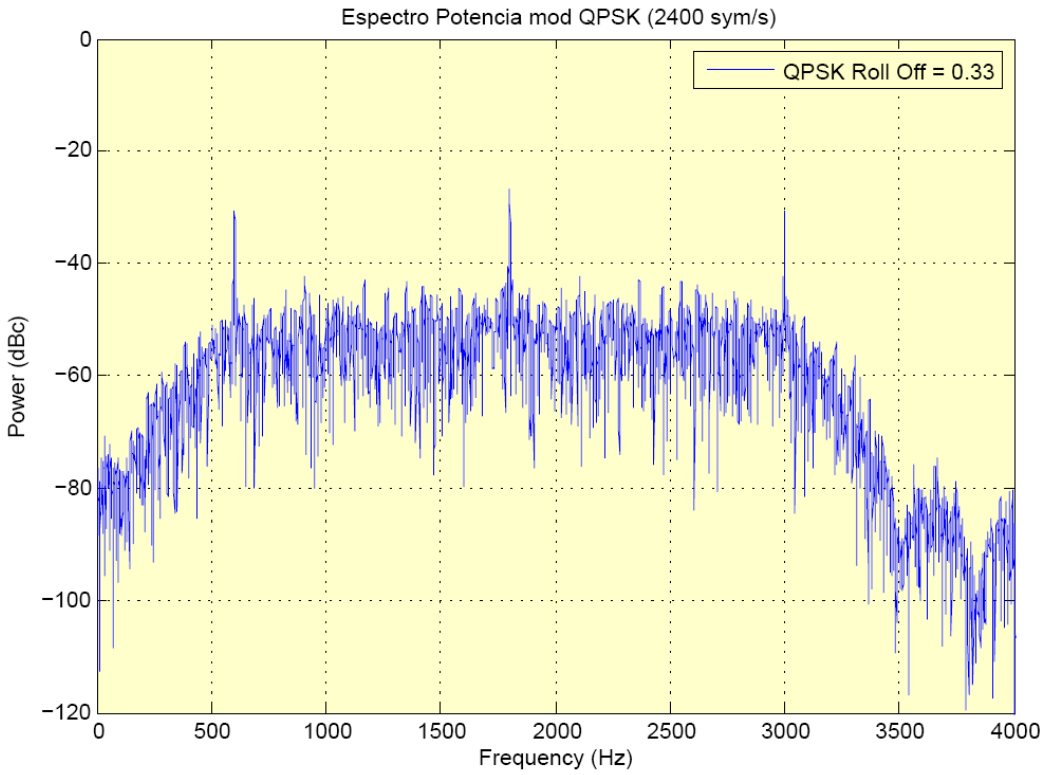

Figure 3.49: Espectro QPSK para secuencia aleatoria 600 símbolos

Las figuras fueron generadas con QPSKhigh.m, los coeficientes del filtro RRC archivados en 'FIRQPSK48000.mat'.

3.6.3 Resumen QPSK 3200 bps

Parametros del Sistema	Valor	Unidades
f_c	1800	Hz
B_T	3200	Hz
R_s	2400	sps
R_b	3200	bps

Table 3.19: Resumen Parametros 8QPSK 2400 bps

3.6.3.1 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModIQ.c, CODEC.c, FormatFilter.c Interfaz.c, TramaLow.c.

- Tablas Seno / Conseno para f_c.
- Tabla Filtro Formato.

3.6.4 Modulación 8QPSK 4800 bps (pp 94, Tabla C-I Apéndice C [3])

Expresón de la señal modulada $s(t)$:

(3.35)

Esta modulación permite un máximo de 4800 bps. Para esta modulación el mapeo de 3 bits a símbolo se realiza con la Tabla 3.20, los simbolos mapean a la constelacion que muestra la figura 3.50 cuyos voltages I y Q son descriptos por la Tabla 3.21.

Tribit	Symbol
000	1
001	0
010	2
011	3
100	6
101	7
110	5
111	4

Table 3.20: Agrupación de 3 bits/símbolo para 8QPSK

3.6.4.1 Constelación

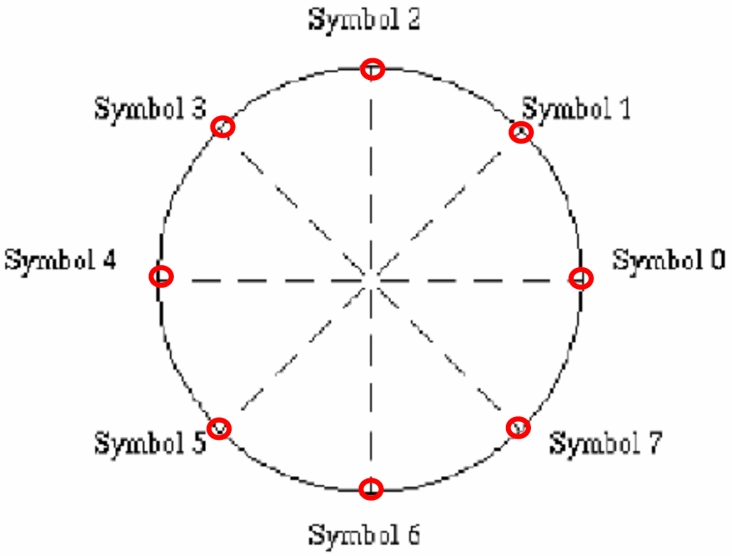

Figure 3.50: Constelación 8PSK

3.6.4.2 Tabla de valores IQ (Tabla C I [3])

El mapeo IQ para QPSK es un subconjunto de entre todos los valores posibles de la Tabla 3.21, que reproducimos a contniuación por conveniencia.

Symbol	Phase	In-Phase	Quadrature
0	0	1.000000	0.000000
1	$\pi/4$	0.707107	0.707107
2	$\pi/2$	0.000000	1.000000
3	$3\pi/4$	-0.707107	0.707107
4	π	1.000000	0.000000
5	$5\pi/4$	-0.707107	-0.707107
6	$3\pi/4$	0.000000	-1.000000
7	$7\pi/4$	0.707107	-0.707107

Table 3.21: Mapeo de símbolos y fases de referencia para 8QPSK

3.6.4.3 Señales Temporales 8QPSK 4800 bps

Las siguientes figuras muestran una secuencia de datos determinada y su resepctiva salida con y sin filtro de formateo. A pesar que la implementación final es con filtro (**??**) la figura 3.51 igual se incluye como referencia de testeo durante la fase de diseño.

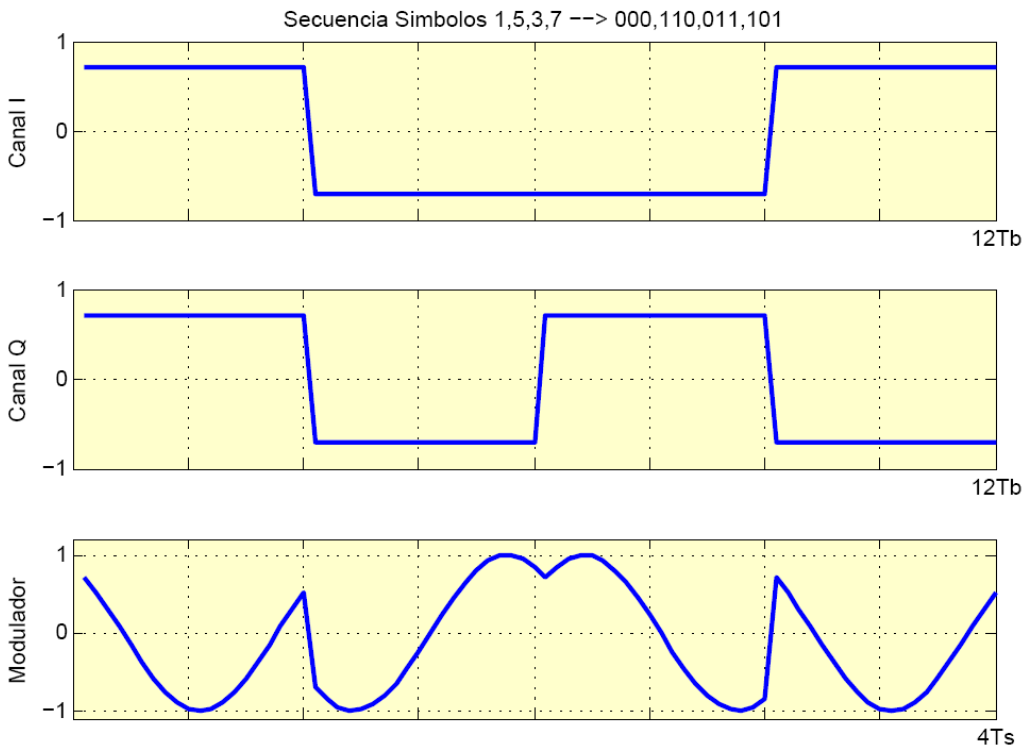

Figure 3.51: 8QPSK en tiempo para secuencia binaria de testeo fija

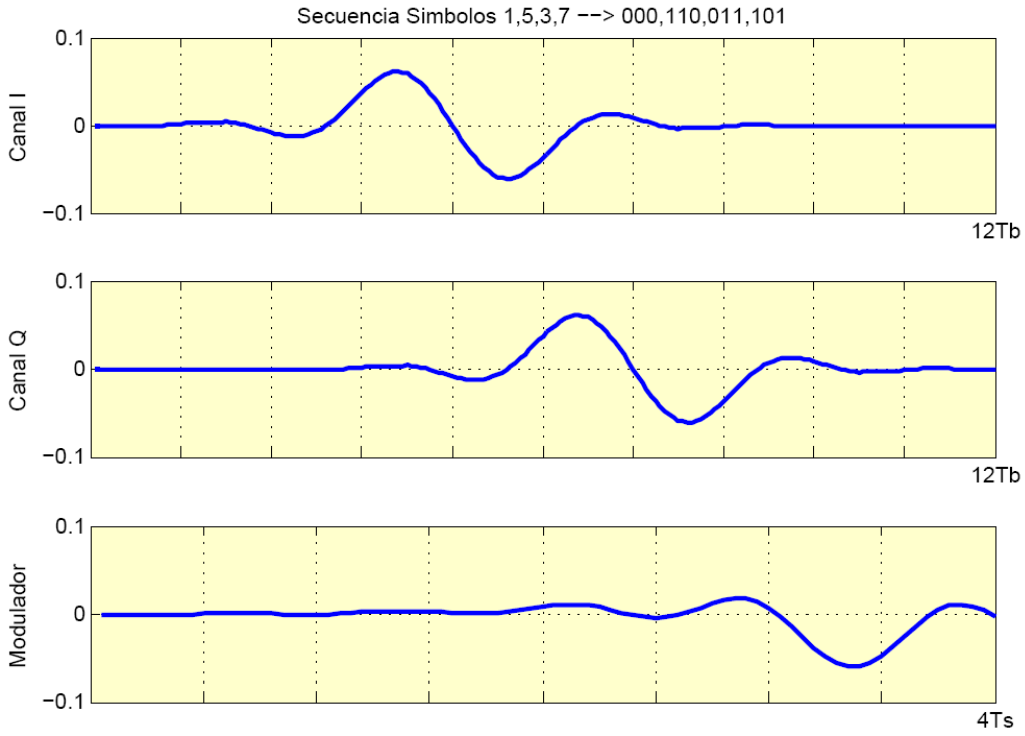

Figure 3.52: 8QPSK en tiempo con filtro de formato para secuencia binaria de testeo fija

3.6.4.4 Espectro 8QPSK 4800 bps

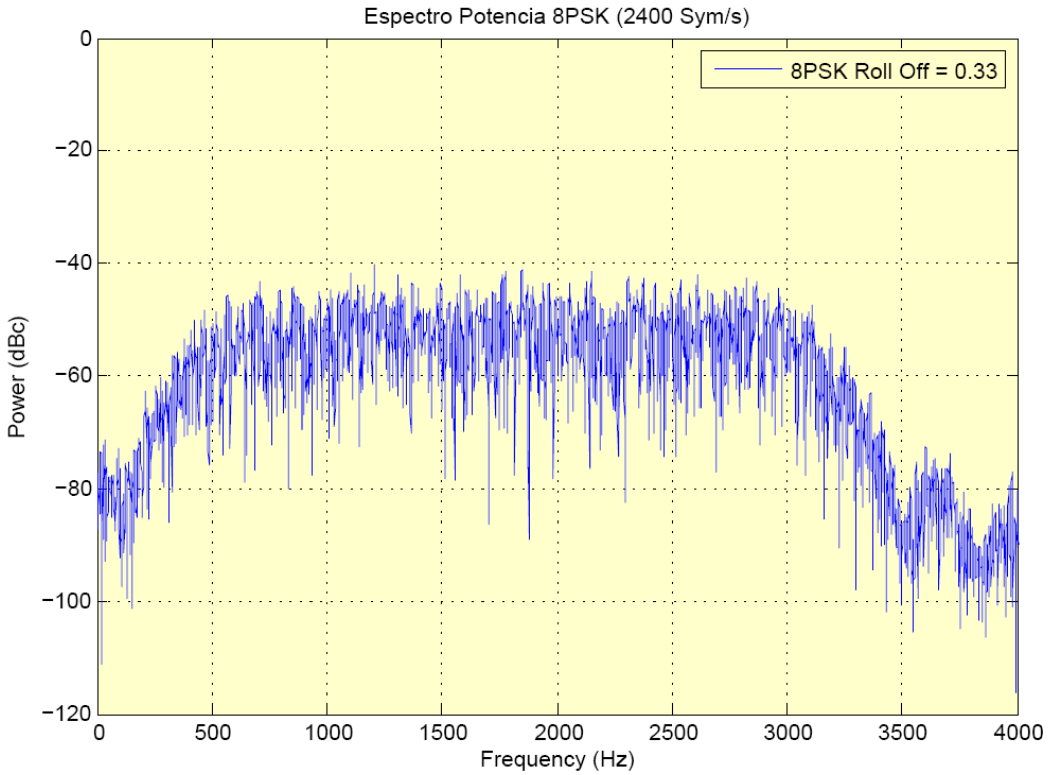

Figure 3.53: Espectro QPSK para secuencia aleatoria 600 símbolos

Las figuras fueron generadas con eigthQPSKhigh.m, los coeficientes del filtro RRC archivados en 'FIRQPSK48000.mat'.

3.6.5 Resumen 8PSK 4800 bps

Parametros del Sistema	Valor	Unidades
f_c	1800	Hz
B_T	3200	Hz
R_s	2400	sps
R_b	4800	bps

Table 3.22: Resumen Parametros 8QPSK 2400 bps

3.6.5.1 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModIQ.c, CODEC.c, FormatFilter.c Interfaz.c, TramaLow.c.

- Tablas Seno / Conseno para f_c.
- Tabla Filtro Formato.

3.6.6 Máscara Espectral para modulaciones QPSK (pp 92, Appendix C.5.1 [3])

La densidad de pontencia espectral debe estar poder debajo en 20dB (resepcto del espectro dentro de la banda) cuando se mide 200Hz por encima o por debajo de la banda de interes.

3.7 *Modulaciones para $R_b > 4800$ bps (pp 94 Tabla C-II Apéndice C [3])*

Las formas de onda para esta velocidades de bits están descriptas en el Apéndice C de [3]. La tabla 3.23 muestra las modulaciones utlizadas para para lograr tasas de bits/s mayoeres a 4800.

$R_b bps$	Modulación
6400	16QAM
8000	32QAM
9600	64QAM
12800	64QAM

Table 3.23: Modulaciones propuestas por el estandard para lograr $R_b > 4800$ bps

La ecuación que describe el funcionamiento del modulador QAM es la siguiente:

$$s(t) = p((k), t)cos(2\pi f_t t) + p((k-1), t)sin(2\pi f_t t) \tag{3.36}$$

La arquitectura de modulador que implementa la ecuación 3.36 se muestra en la figura 3.54

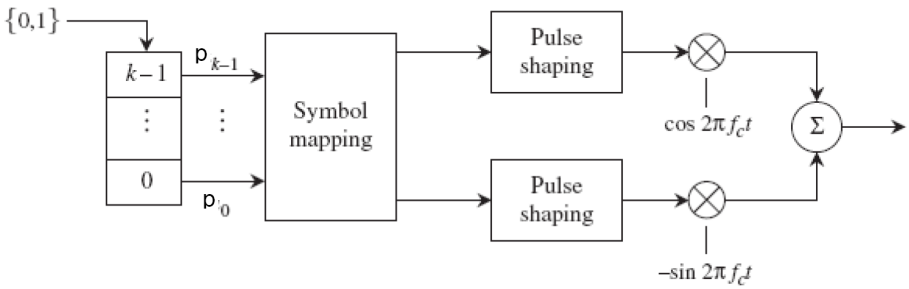

Figure 3.54: Modulador QAM

El esquema de modulación previa se utiliza para transmitir 16QAM - 32QAM y 64QAM.

NOTA: En la implementación el filtro RRC se realiza primero utilizando un modulador impulso y luego se aplica el filtro RRC.

3.7.1 Alfabeto de Simbolos Binarios

$$d_k \in 0, 1 \tag{3.37}$$

3.7.2 Modulación 16QAM 6400 bps (pp 96 Tabla C-V Apéndice C [3])

Esta modulación permite un máximo de 6400 bps. Se realiza un mapeo de 4 bits a símbolo, los simbolos mapean a la constelacion que muestra la figura 3.55 cuyos voltages I y Q son descriptos por la Tabla 3.24.

(3.38)

3.7.2.1 Constelación 16QAM

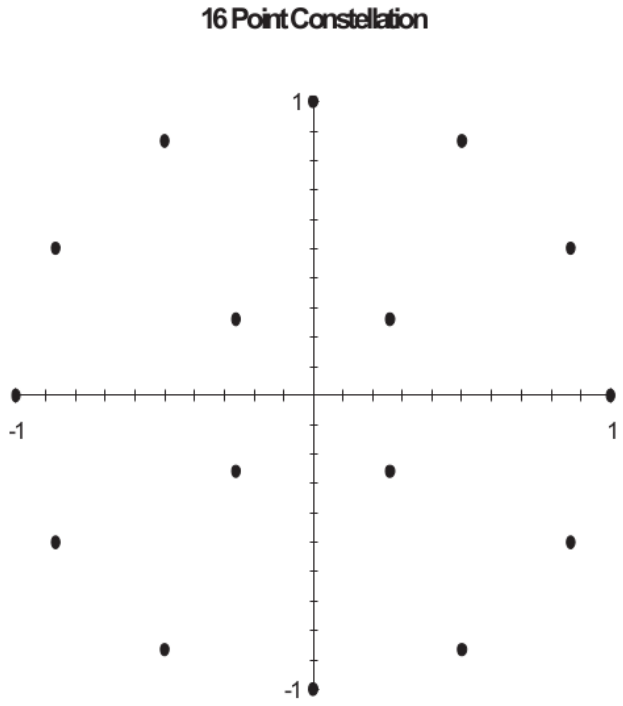

Figure 3.55: Constelación 16QAM

3.7.2.2 Tabla de valores IQ (Tabla C V [3])

Símbolo	In Phase	Quadrature
0	0.866025	0.500000
1	0.500000	0.866025
2	1.000000	0.000000
3	0.258819	0.258819
4	-0.500000	0.866025
5	0.000000	1.000000
6	-0.866025	0.500000
7	-0.258819	0.258819
8	0.500000	-0.866025
9	0.000000	-1.000000
10	0.866025	-0.500000
11	0.258819	-0.258819
12	-0.866025	-0.500000
13	-0.500000	-0.866025
14	-1.000000	0.000000
15	-0.258819	-0.258819

Table 3.24: Componentes en Fase y Cuadratura para 16QAM

3.7.2.3 Señales Temporales 16QAM 6400 bps

Las siguientes figuras muestran una secuencia de datos determinada y su resepctiva salida con y sin filtro de formateo. A pesar que la implementación final es con filtro (3.57) la figura **??** igual se incluye como referencia de testeo durante la fase de diseño.

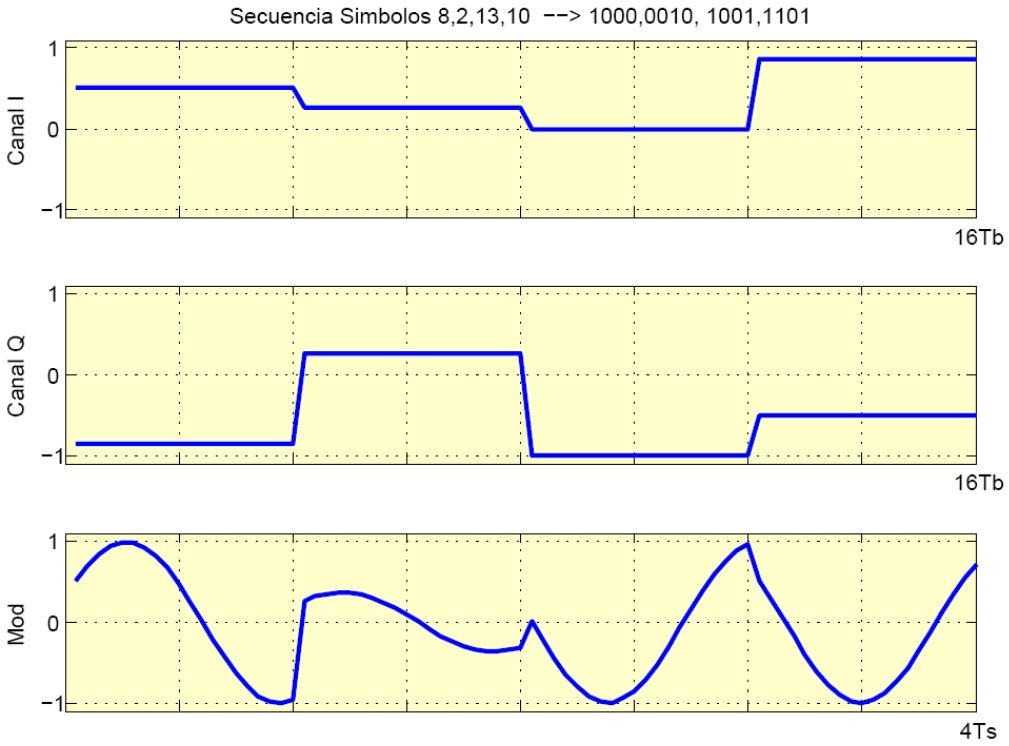

Figure 3.56: 16QAM en tiempo para secuencia binaria de testeo fija

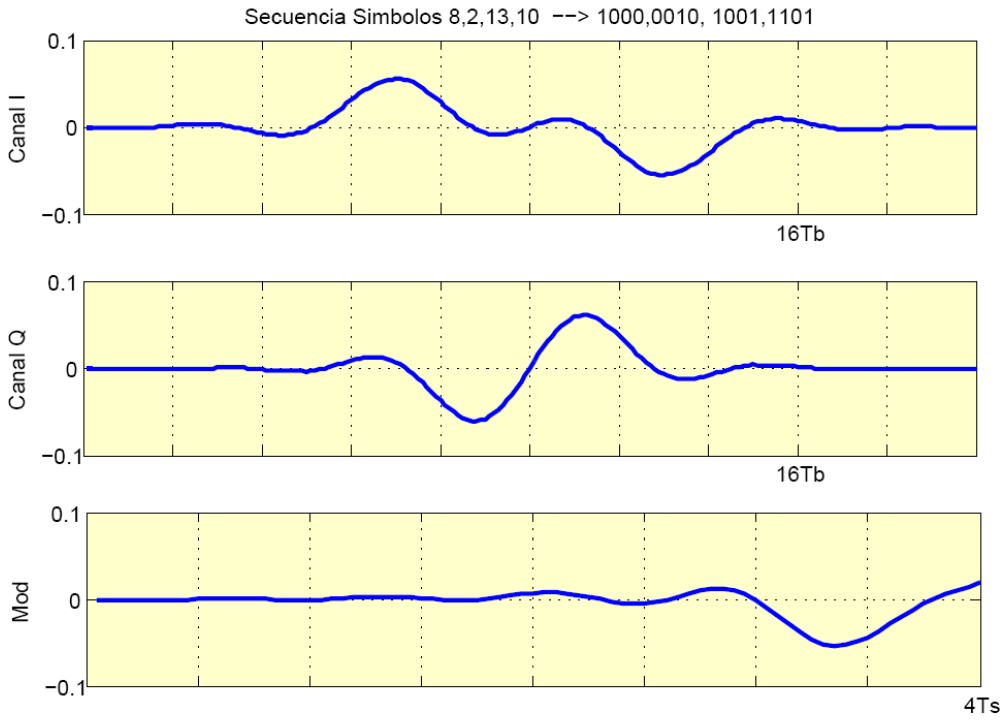

Figure 3.57: 16QAM en tiempo con filtro de formato para secuencia binaria de testeo fija

3.7.2.4 Espectro 16QAM 6400 bps

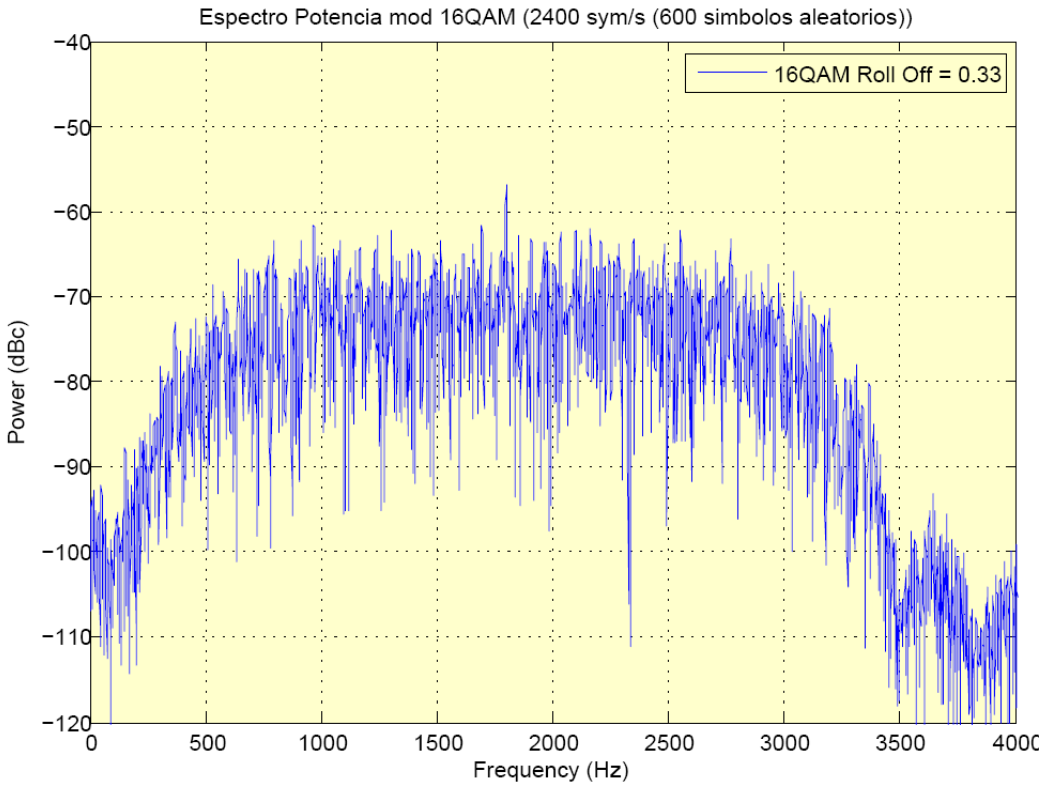

Figure 3.58: Espectro 16QAM para secuencia aleatoria 600 símbolos

Las figuras fueron generadas con sixteenQAM.m, los coeficientes del filtro RRC archivados en 'FIRQPSK48000.mat'.

3.7.3 Resumen 16QAM 6400 bps

Parametros del Sistema	Valor	Unidades
f_c	1800	Hz
B_T	3200	Hz
R_s	2400	sps
R_b	6400	bps

Table 3.25: Resumen Parametros 32QAM 6400 bps

3.7.3.1 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModIQ.c, CODEC.c, FormatFilter.c Interfaz.c, TramaHigh.c.

- Tablas Seno / Conseno para f_c.
- Tabla Filtro Formato.

3.7.4 Modulación 32QAM 8000 bps (pp 97 Table C-VI Apéndice C [3])

Esta modulación permite un máximo de 8000 bps. Se realiza un mapeo de 5 bits a símbolo, los simbolos mapean a la constelacion que muestra la figura 3.59 cuyos voltages I y Q son descriptos por la Tabla 3.7.4.2.

3.7.4.1 Constelación 32 QAM

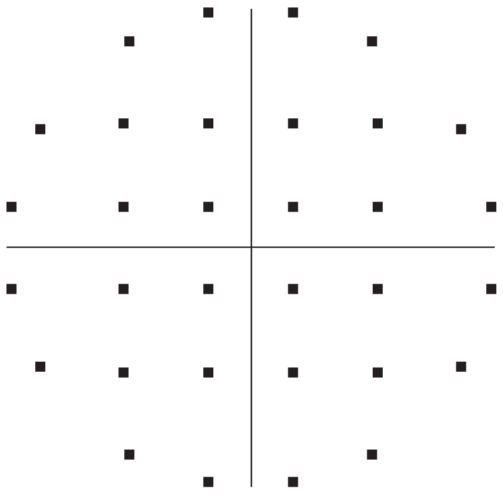

Figure 3.59: Constelación 32QAM

3.7.4.2 Tabla de valores IQ (Tabla C VI [3])

Símbolo	In Phase	Quadrature	Símbolo	In Phase	Quadrature
0	0.866380	0.499386	16	0.866380	-0.499386
1	0.984849	0.173415	17	0.984849	-0.173415
2	0.499386	0.866380	18	0.499386	-0.866380
3	0.173415	0.984849	19	0.173415	-0.984849
4	0.520246	0.520246	20	0.520246	-0.520246
5	0.520246	0.173415	21	0.520246	-0.173415
6	0.173415	0.520246	22	0.173415	-0.520246
7	0.173415	0.173415	23	0.173415	-0.173415
8	-0.866380	0.499386	24	-0.866380	-0.499386
9	-0.984849	0.173415	25	-0.984849	-0.173415
10	-0.499386	0.866380	26	-0.499386	-0.866380
11	-0.173415	0.984849	27	-0.173415	-0.984849
12	-0.520246	0.520246	28	-0.520246	-0.520246
13	-0.520246	0.173415	29	-0.520246	-0.173415
14	-0.173415	0.520246	30	-0.173415	-0.520246
15	-0.173415	0.173415	31	-0.173415	-0.173415

Table 3.26: Componentes en Fase y Cuadratura para 32QAM

3.7.4.3 Señales Temporales 32QAM 8000 bps

Las siguientes figuras muestran una secuencia de datos determinada y su resepctiva salida con y sin filtro de formateo. A pesar que la implementación final es con filtro (3.61) la figura 3.60 igual se incluye como referencia de testeo durante la fase de diseño.

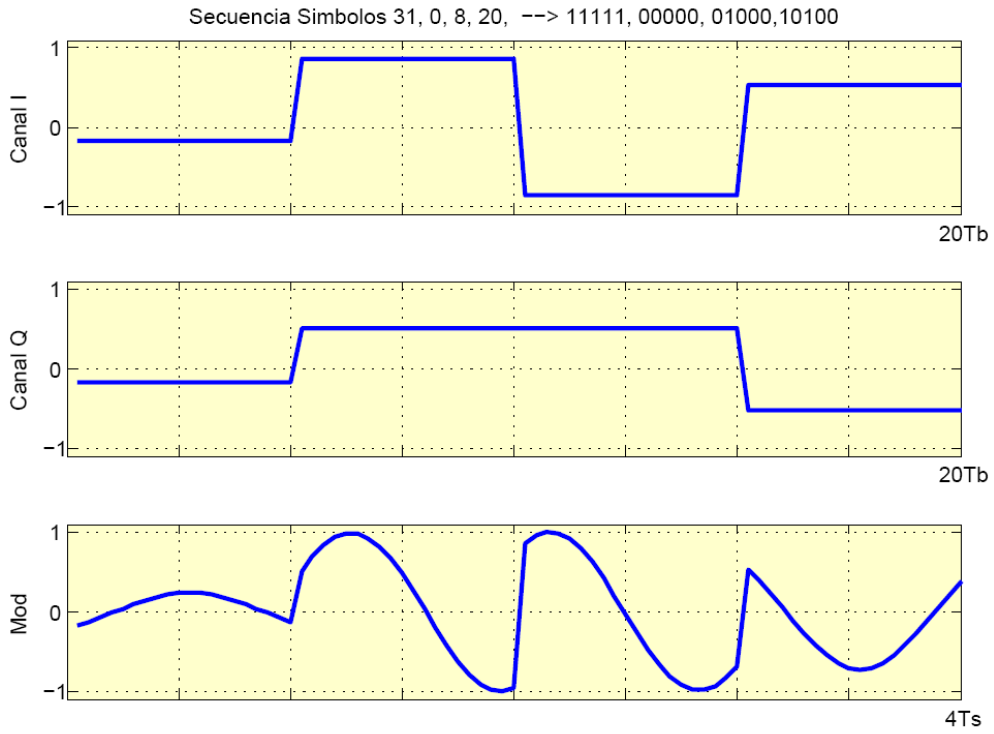

Figure 3.60: 32QAM en tiempo para secuencia binaria de testeo fija

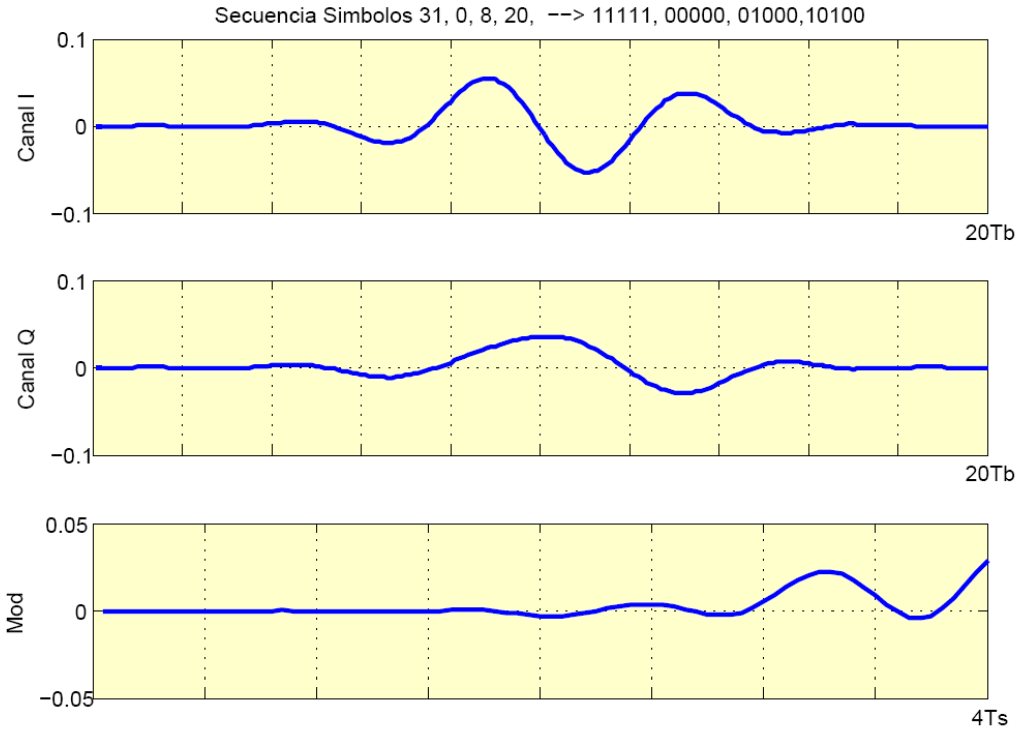

Figure 3.61: 32QAM en tiempo con filtro de formato para secuencia binaria de testeo fija

3.7.4.4 Espectro 32QAM 8000 bps

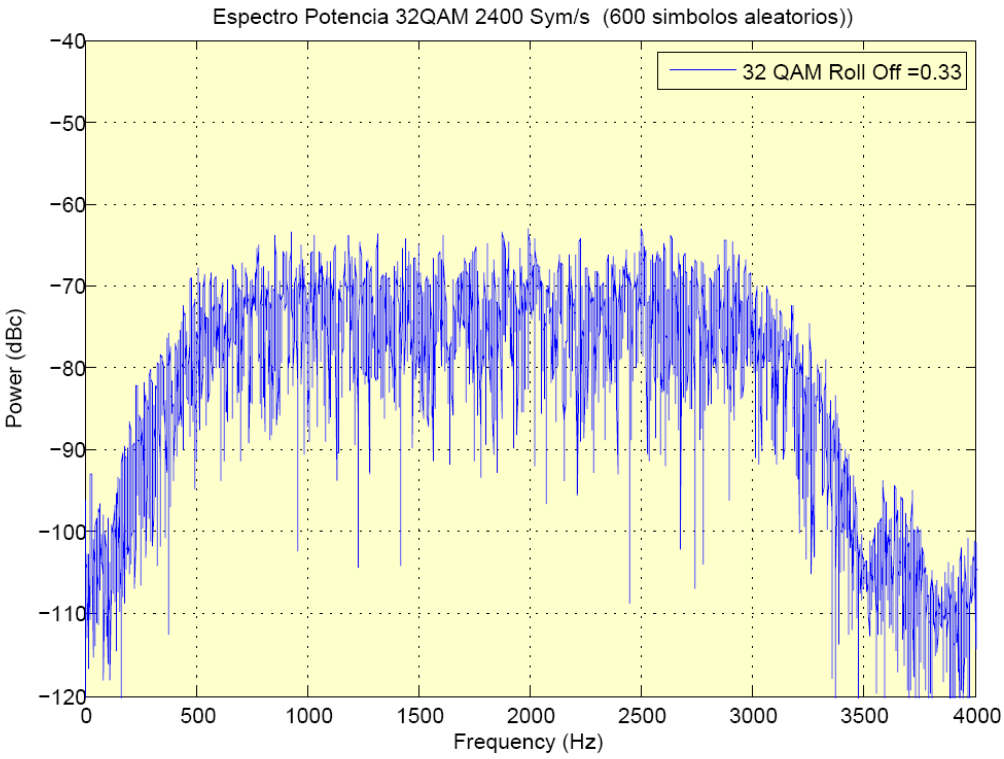

Figure 3.62: Espectro 32QAM para secuencia aleatoria 600 símbolos

Las figuras fueron generadas con thirtytwoQAM.m, los coeficientes del filtro RRC archivados en 'FIRQPSK48000.mat'.

3.7.5 Resumen 32 QAM 8000 bps

Parametros del Sistema	Valor	Unidades
f_c	1800	Hz
B_T	3200	Hz
R_s	2400	sps
R_b	8000	bps

Table 3.27: Resumen Parametros 32QAM 8000 bps

3.7.5.1 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModIQ.c, CODEC.c, FormatFilter.c Interfaz.c, TramaHigh.c.

- Tablas Seno / Conseno para f_c.
- Tabla Filtro Formato.

3.7.6 Modulación 64QAM $9600 - 12800$ bps (pp 99 Tabla C-VII Apéndice C [3])

Esta modulación permite un máximo de 9600 bps. Se realiza un mapeo de 6 bits a símbolo, los simbolos mapean a la constelacion que muestra la figura 3.63 cuyos voltages I y Q son descriptos por la Tabla 3.28.

Expresión de la señal modulada $s(t)$:

(3.39)

3.7.6.1 Constelación 64 QAM

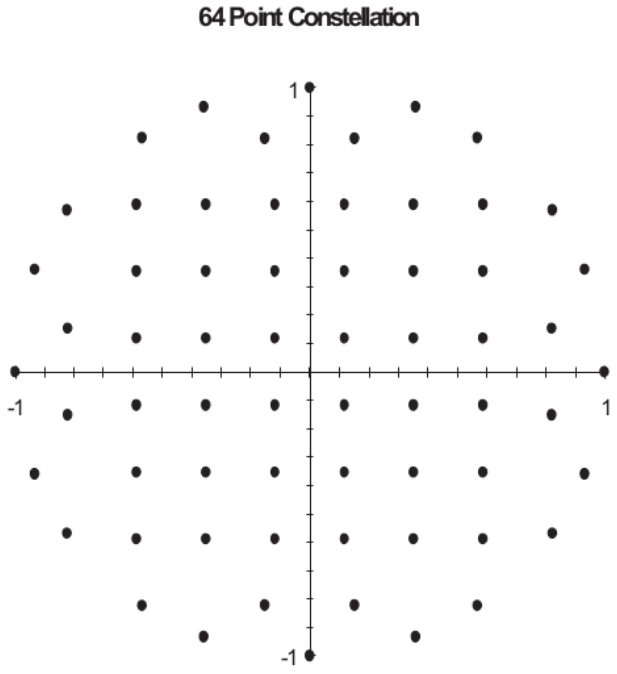

Figure 3.63: 64-QAM Constellation

3.7.6.2 Tabla de valores IQ (Tabla C VII [3])

Símbolo	In Phase	Quadrature	Símbolo	In Phase	Quadrature
0	1.000000	0.000000	32	0.000000	1.000000
1	0.822878	0.568218	33	-0.822878	0.568218
2	0.821137	0.152996	34	-0.821137	0.152996
3	0.932897	0.360142	35	-0.932897	0.360142
4	0.000000	-1.000000	36	-1.000000	0.000000
5	0.822878	-0.568218	37	-0.822878	-0.568218
6	0.821137	-0.152996	38	-0.821137	-0.152996
7	0.932897	-0.360142	39	-0.932897	-0.360142
8	0.568218	0.822878	40	-0.568218	0.822878
9	0.588429	0.588429	41	-0.588429	0.588429
10	0.588429	0.117686	42	-0.588429	0.117686
11	0.588429	0.353057	43	-0.588429	0.353057
12	0.568218	-0.822878	44	-0.568218	-0.822878
13	0.588429	-0.588429	45	-0.588429	-0.588429
14	0.588429	-0.117686	46	-0.588429	-0.117686
15	0.588429	-0.353057	47	-0.588429	-0.353057
16	0.152996	0.821137	48	-0.152996	0.821137
17	0.117686	0.588429	49	-0.117686	0.588429
18	0.117686	0.117686	50	-0.117686	0.117686
19	0.117686	0.353057	51	-0.117686	0.353057
20	0.152996	-0.821137	52	-0.152996	-0.821137
21	0.117686	-0.588429	53	-0.117686	-0.588429
22	0.117686	-0.117686	54	-0.117686	-0.117686
23	0.117686	-0.353057	55	-0.117686	-0.353057
24	0.360142	0.932897	56	-0.360142	0.932897
25	0.353057	0.588429	57	-0.353057	0.588429
26	0.353057	0.117686	58	-0.353057	0.117686
27	0.353057	0.353057	59	-0.360142	-0.932897
29	0.353057	-0.588429	61	-0.353057	-0.588429
30	0.353057	-0.117686	62	-0.353057	-0.117686
31	0.353057	-0.353057	63	-0.353057	-0.353057

Table 3.28: Componentes en Fase y Cuadratura para 64QAM

3.7.6.3 Señales Temporales 64QAM 9600 bps

Las siguientes figuras muestran una secuencia de datos determinada y su resepctiva salida con y sin filtro de formateo. A pesar que la implementación final es con filtro (3.65) la figura 3.64 igual se incluye como

referencia de testeo durante la fase de diseño.

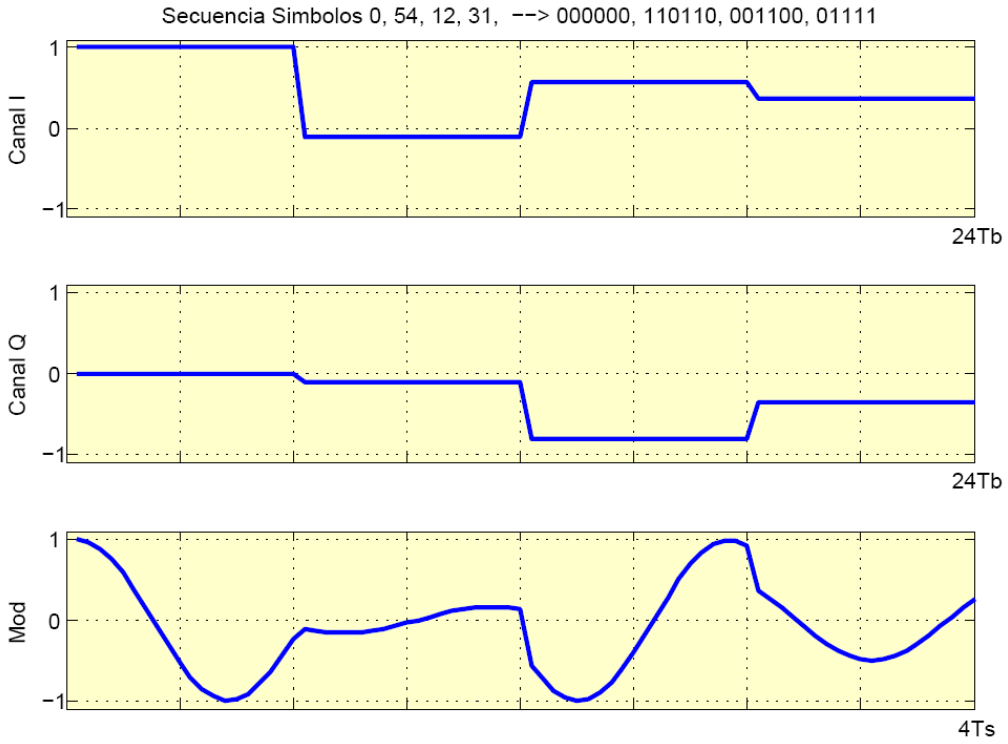

Figure 3.64: 64QAM en tiempo para secuencia binaria de testeo fija

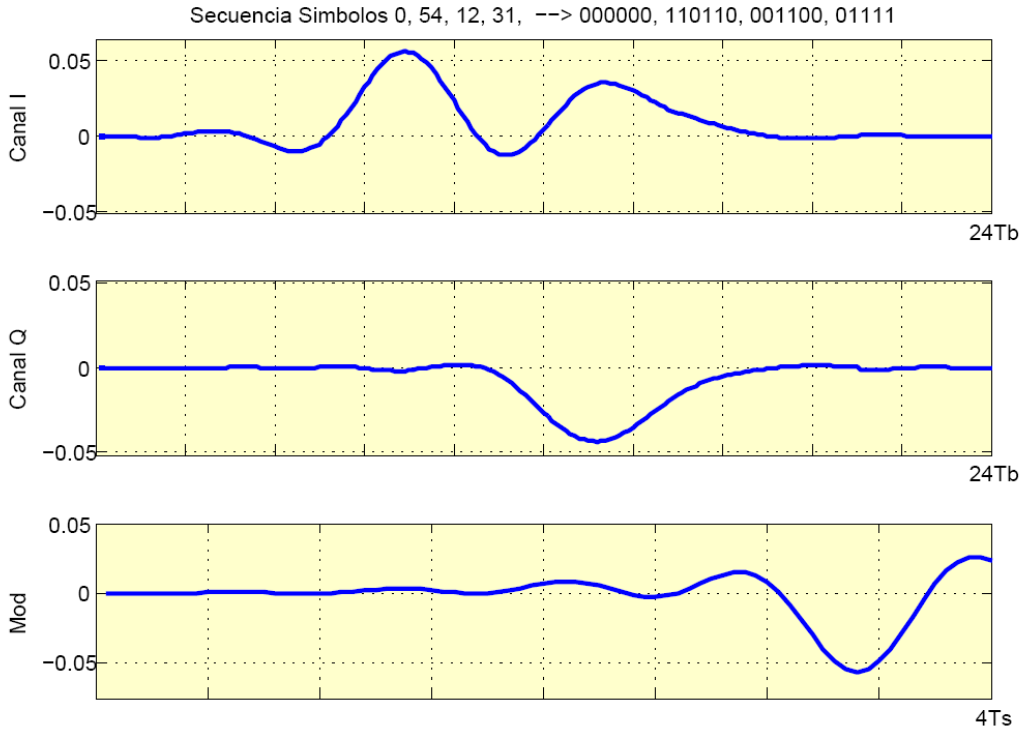

Figure 3.65: 64QAM en tiempo con filtro de formato para secuencia binaria de testeo fija

3.7.6.4 Espectro 64QAM 9600 bps

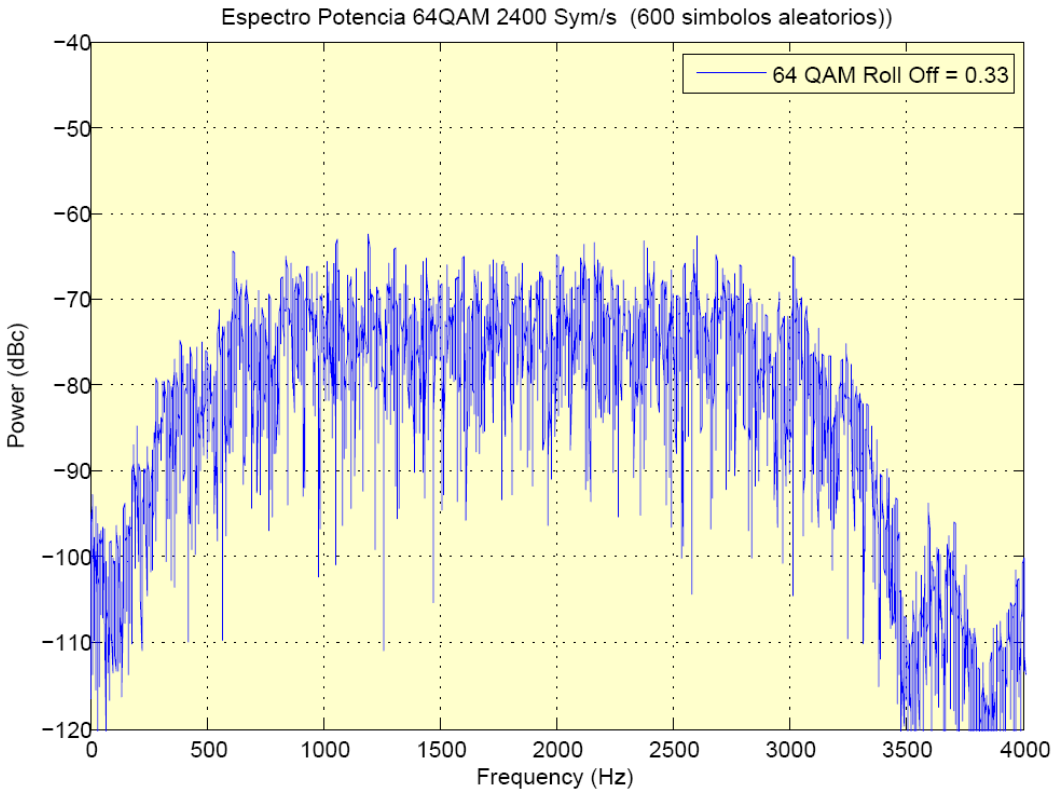

Figure 3.66: Espectro 64QAM para secuencia aleatoria 600 símbolos

Las figuras fueron generadas con sixtyfourQAM.m, los coeficientes del filtro RRC archivados en 'FIRQPSK48000.mat'.

3.7.7 Resumen 64QAM 9600-12800 bps

Parametros del Sistema	Valor	Unidades
f_c	1800	Hz
B_T	3200	Hz
R_s	2400	sps
R_b	9600	bps

Table 3.29: Resumen Parametros 64QAM 9600 bps

3.7.7.1 Algoritmos y Tablas DSP Relacionados (Apéndice DSP)

- ImpulseMod.c, ModIQ.c, CODEC.c, FormatFilter.c Interfaz.c, TramaHigh.c.

- Tablas Seno / Conseno para f_c.
- Tabla Filtro Formato.

3.7.8 Máscara Espectral para modulaciones QAM (pp 92 Appendix C.5.1 [3])

La densidad de pontencia espectral debe estar poder debajo en 20dB (resepcto del espectro dentro de la banda) cuando se mide 200Hz por encima o por debajo de la banda de interes.

3.8 *Resumen general modulaciones* $R_b > 3200$ *bps*

MODEM	bps	$R_s(sym/s)$	$f_c(Hz)$	$f_{Null,low}(Hz)$	$f_{Null,high}(Hz)$	Filtro	Roll-off	Modulador
QPSK	3200	2400	1800	200	3400	RRC	0.33	fig 3.54
8QPSK	4800	2400	1800	200	3400	RRC	0.33	fig 3.54
QAM16	6400	2400	1800	200	3400	RRC	0.33	fig 3.54
QAM32	8000	2400	1800	200	3400	RRC	0.33	fig 3.54
QAM64	9600	2400	1800	200	3400	RRC	0.33	fig 3.54
QAM64	12800	2400	1800	200	3400	RRC	0.3	fig 3.54

Table 3.30: Comparación parametros moduladores QPSK y QAM para canal [3](200Hz-3400Hz). Con filtro de formateo de pulso $\alpha = 0.33$

NOTA: Todas las modulaciones anteriores usan codigo convolucional de 1/2 - perforado (punctured) 3/4.

3.8.1 Acerca de las Velociades bps de las formas de onda del Apéndice C [3]

Las velocidades bps que se muestran en la Tabla 3.30 para cada modulación son menores a las velocidades toericas respectivas. Esto de debe a dos motivos: primero, se agregan secuencias 'probe' para el entrenamiento del ecualizador y segundo los bits estan codificados con FEC. La máxima velocidad que el alcanza el estanard es aproximadamente 66% de la velocidad teórica de un modulador y se calcula de la siguiente manera:

$$Bits_{Tx,Total} = Bits_{Dato} + Bits_{Probe} = 256 + 31 = 287 \tag{3.40}$$

Por lo tanto el porcentaje de datos es:

$$\% = (Bits_{Dato}/Bits_{Tx_Total}) * 100 = 0.89 \tag{3.41}$$

Pero de los $Bits_{Dato}$ solo el 75% son datos pues están codificado con un código 3/4 [3] Appendix C, pp 102. Entonces los datos reales transmitidos son:

$$Bits_{Tx,Dato} = 0.89 * .75 = 0.67 \tag{3.42}$$

El valor anerior se redondea a 0.66 para incorporar la secuencia de preambulo.

3.9 *Tramas de Datos*

3.9.1 Trama de baja velocidad

3.9.1.1 Datos Desconocidos

El mapeo de los datos debe ser realizado acorde a la figura 3.37 cuando se utiliza modulación QPSK de M=2, M=4 y M=8 simobolos (Tabla 3.31). Para M = 2 los simbolos son 0 y 4 (600-150 bps), M=4 simoblos 0,2,4,6 (1200 bps) y para M=8 simbolos 0,1,2,3,4,5,6 y 7 (2400 bps). En el caso de modulaciones FSK M=2 entonces mapea a las dos frecuencias posibles f_{low} y f_{up}.

Bit Rate (bps)	Bits/Simbolo
2400	3
1200	2
600	1
300	1
150	1
75	2

Table 3.31: Bits por símbolo para cada velocidad

3.9.1.2 Codificador FEC para los Datos Desconocidos

El codificador FEC debe ser usado para velocidades hasta 2400 bps. El diagrama en bloque del codificador se muestra a ocntinuación 3.67 :

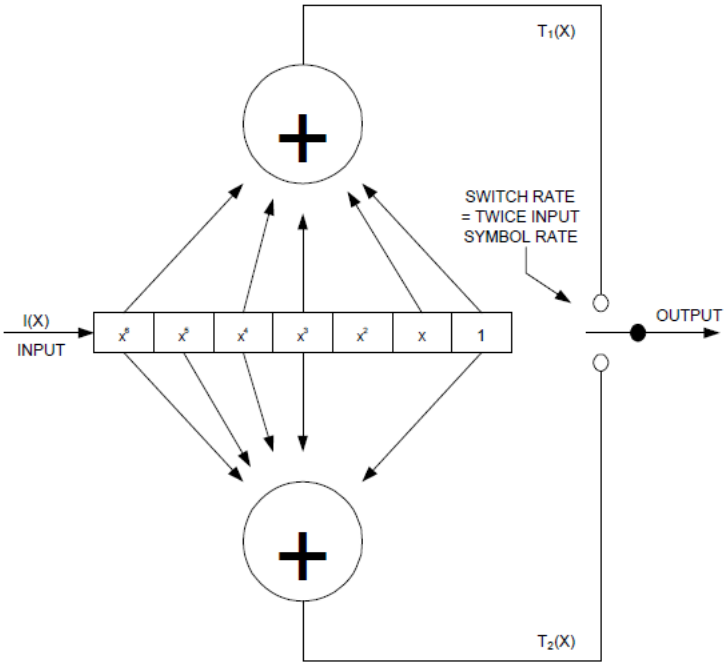

Figure 3.67: Codificador FEC

La función del FEC es llevada a cabo por un codificador convolucional de largo 7 y velocidad $\frac{1}{2}$. Los dos nodos de suma de la figura 3.67 representan sumas de módulo 2. Por cada bit de entrada al codificador ($I(X)$), salen 2 bits de salida por el codificador $T_1(x)$ y $T_2(x)$, siendo el bit de salida $T_1(x)$ el primero en salir.

Las diferentes codificaciones según la velocidad de bits se pueden encontrar en la Tabla (3.5). NOTA: Cuando existe repetición de código primero se repite $T_1(x)$, seguido por las repeticiones de $T_2(x)$.

Los polinomios generadores de largo restringido 7 son:

$$T_1 = X^6 + X^4 + X^3 + X + 1; \tag{3.43}$$

y

$$T_2 = X^6 + X^5 + X^4 + X^3 + 1 \tag{3.44}$$

3.9.1.3 Interleaver para los Datos Desconocidos

El interleaver se hace organizando los datos desconocidos en formato matriz que pemrite acomodar bolques que contengan 0.0 segs, 0.6 segs or 4.8 segs de datos para cada uno de los sistemas de modulación disponible. EL proceso de interleaving se hace en dos paso, primero se carga (LOAD) una matriz acorde a la Tabla 3.32 y luego se trasnmite buscando (FETCH) los datos dentro de la matriz y serializando los bits para que entren al codificador Gray o salgan directamente.

1. **Carga del Interleaver (LOAD) (Sección 5.3.2.3.4):**
   En este paso se agrupan los bits de datos desconocidos en formato de matriz según el tamaño d einterleaver.
   Para mantener el retardo del interleave constante, el tamaño del bloque debe ser escalado por la velocidad de datos. La tabla 3.32 da una lista de las dimensiones de la matriz de interleaving (filas y columnas) para mantener el mismo retardo:

Vel. datos (bps)	Long Interleaver		Short Interleaver	
	Nro. Filas	Nro,. Columnas	Nro. Filas	Nro. Columnas
2400	40	576	40	72
1200	40	288	40	36
600	40	144	40	18
300	40	144	40	18
150	40	144	40	18
75	20	36	10	9

Table 3.32: Dimensiones de la matriz de Interleaving

   El armado se realiza d ela siguiente manera: el primer bit va a Row 0, Col 0, el próximo bit Row 9 (misma Col), siguiente, row 18 (misma Col), se suma de a 9 mod 40. Una vez llena la columna se hace el mismo procedimiento pero con la columna 1, hasta completar el número de columnas (lo anterior es válido para todos los sistemas excepto 75bps, donde se incrementan las filas de a 7 y se suma modulo 20).

2. **Busqueda en el Interleaver (FETCH)(Sección 5.3.2.3.5):**
   En este paso se buscan los bits en la matriz de armada en el paso previo y los serializa para ingresar al Codificador Gray (cuando corresponde) o directamente para su transmisión.
   Durante la busqueda se toman los datos de la matriz Load y se los serializa de la siguiente manera. Se inicia ocn la fila cero y columna cero. Los siguientes bits se toman incrementando la fila por uno y decrementando la columna por 17 (con modulo igual al número de columnas). Este proceso continúya hasta que el número de filas alcanza el máximo. En este punto el contador de filas vuleve a cero y el contador de columnas se configura para ser igual al número de columna que tenía cuando....

3.9.1.4 Código Gray Modificado

Se aplica esta codificación cuando la modulación multinivel. Las Tablas 3.33 y 3.34 describe cuando y que codificaciones se aplican:

Bit 1	Bit 2	Código Gray Modificado
0	0	00
0	1	01
1	0	11
1	1	10

Table 3.33: Código Gray Modificado para 75 bps y 1200 bps (2bits/símbolo)

Input bits			
Bit 1	Bit 2	Bit 3	Código Gray Modificado
0	0	0	000
0	0	1	001
0	1	0	011
0	1	1	010
1	0	0	111
1	0	1	110
1	1	0	100
1	1	1	101

Table 3.34: Código Gray Modificado para 2400 bps

3.9.1.5 Datos Conocidos (Sondas)

Los símbolos sonda a transmitir son simpre 0 (el tribit 000). Esto es así excepto cuando se transmite un nuevo bloque de interleaving. En es caso, los 16 primeros tribits de los probes deben ser configurados para que sean iguales a D1 y D2 de la secuencia de sincronismo. En este caso cuando el largo de la sonda es 32 se repite D1 y D2 dos veces. Si el largo de la sonda es 20, los últimos 4 tribits se hacen igual a cero.

3.9.1.6 Preambulo de sincronismo

La forma de onda de sincronización es igual para todas las velocidades.

- El preambulo debe ser de 3 o 24 segmentos de 200 mseg.
- Cada segmento de 200 mseg consta de 15 símbolos de 3 bits cada uno trasmitidos en formato patrón repetitivo.
- El tiempo de símbolo es 13.3 mseg (75 Hz).
- El tiempo de patron es 13.3mseg*15*32 = 200 mseg (segmento).

- La secuencia de símbolos debe ser: 0, 1, 3, 0, 1, 3, 1, 2, 0. Dl, D2, C1, C2, C3, 0.
- D1 indica la velocidad de bit, se utiliza en combinación con D2 (Tabla 3.35).
- D2 indica el tipo de interleaver se utiliza en combinación con D1 (Tabla 3.35).
- C1, C2, C3, indican el número de segmentos enviados (Tabla 3.36).

Bit Rate	D1 (short)	D2 (Short)	D1 (long)	D2 (long)
4800	7	6	-	-
2400	7	7	-	-
1200	4	4	4	4
600	6	6	4	5
300	6	7	4	7
150	7	4	5	4
75	7	5	5	5

Table 3.35: Asignaciones para D1 y D2

C1, C2 y C3 son de 2 bits y conforman una palabra binaria de 6 bits donde C1 es el mas significativo. Por ejemplo, la pablabra23=01011 se convierte en C1=01 C2=01 y C3=11. Pero como durante la sincronización se transmiten símbolos de tres bits las mismas se convierten segun la sigguiente Tabla 3.36:

2 bits	3 bits
00	4 (100)
10	5 (101)
01	6 (110)
11	7 (111)

Table 3.36: Conversión de C1, C2 y C3 2 bits a 3 bits

Los valores utlizados en la conversion están asociados a que el cuadrante 0,1,2,3, (fig. 3.37) se utiliza para el sincronismo y el cuadrante 4, 5, 6, 7 (fig. 3.37) se utiliza para agregar en el sincronismo información extra.

3.9.1.7 Generación del Patrón de Preambulo

La secuencia de preambulo se transmite generando un patron de señal a partir de los simbolos de la secuencia. Cada símbolo en la secuencia de preambulo es de 3bits y se debe convertir a 32 simbolos de tres bits acorde a la Tabla 7.1

Simbolo Sync (3bits)	Patron Sync (8 x 3 bits)
000	(0000 0000)x 4 veces.
001	(0404 0044)x 4 veces.
010	(0044 0044)x 4 veces.
011	(0440 0440)x 4 veces.
100	(0000 4444)x 4 veces.
101	(0404 4040)x 4 veces.
110	(0044 4400)x 4 veces.
111	(0440 4004)x 4 veces.

Table 3.37: Armado de secuencia patron a partir de símbolos sincronismo

Como se puede observar en la Tabla 3.37 y la figura 3.37 con en mapeo del patrón se logra transmitir una señal BPSK con patrones definidos que ayudan a la sincronización.

Las 6 secuencias de sincronismo (15 simobolos tribits) correspondientes al interleave 'Short' estan almacenadas en:
$'secuencias_sinc_ShortInterleave_tramaLOW.mat'$.
Los 6 patrones de secuencias de preambulo correspondientes al interleave 'Short' estan almacendas en:
$'preamble_pattern_Short1_LOW'$, $'preamble_pattern_Short2_LOW'$, $'preamble_pattern_short3_LOW'$, $'preamble_pattern_Short4_LOW'$, $'preamble_pattern_Short5_LOW'$ y $'preamble_pattern_Short6_LOW'$.
Las 6 secuencias de sincronismo (15 simobolos tribits) correspondientes al interleave 'Long' estan almacendadas en:
$'secuencias_sinc_LongInterleave_tramaLOW.mat'$.
Los 6 patrones de secuencias de preambulo correspondientes al interleave 'Long' estan almacendas en:
$'preamble_pattern_Long1_LOW'$, $'preamble_pattern_Long2_LOW'$, $'preamble_pattern_Long3_LOW'$, $'preamble_pattern_Long4_LOW'$, $'preamble_pattern_Long5_LOW'$ y $'preamble_pattern_Long6_LOW'$.
Generado con TramaLow.m.

3.9.1.8 Scrambler Sincronismo

3.9.2 Trama de alta velocidad

3.9.2.1 Preambulo de sincronismo

Generado con TramaHigh.m

3.9.2.2 Scrambler Sincronismo

3.10 *Diseño del Transmisor Multimodo*

El tansmisor se realiza en fomra modular como se describió en el Capítulo anterior. Cada

- Mapeador
- Modulador
-

3.10.1 Mapeador

Se describió parcialmente donde se describe las formas de onda.

3.10.2 Modulador Impulso

3.10.3 Filtro de Formateo de Pulso

Tal como se describió previamente los filtros de formato de pulso pueden ser RC o RRC, ambos con $\alpha = 0.33$. En función del valor de α se considera un muy buena aproximación acortar el largo del pulso (que en teoría es infinito) en $-3T_s < h(t) < 3T_s$.

El número de taps esta definido por la extensión del pulso RC o RRC respecto de T_s [17] y el oversampling M del puslo. Por ejemplo:

$$M = F_s.T_s = \frac{48000Hz}{1/2400Hz} = 20. \tag{3.45}$$

Entoces el númeor de Taps es:

$$Taps = M * D = 6 * 20 = 120 \tag{3.46}$$

El filtro se diseña a partir de su respuesta impulsiva. La misma se obtiene usando las ecuaciones que se describieron previamente o tambien se puede utilizar la herramienta FDATool. En esta última al ingresar el valor de fc se tiene que ingresar la velocidad de símbolo divida 2 $R_s/2$.

Debido a la variedad de configuraciones que prevee el estandar se diseñan seis prototipos de filtrs RC y seis prototipos de filtros RRC que cubren la vairedad de situaciones de transmisión y recepción. En el caso del modulador Del lado del receptor es necesario utilizar implementación FIR polifasicas [18] incluso a la mas baja veolicidad de muestreo (FIR=24 Taps) como por ejemplo ocurre en la implementación de Texas [19] cuyas caracteristicas som muy similares a la versión QAM del MOMIL.

Por lo tanto, a continuación de los diseños mencionados se presentan se describira la extensión polifasica con el objetivo de disponear de los lineamientos basicos de implementación para cuando sea necesario. Como se mencionó previamente la mayor restricción temporal en el DSP aparace en el demodulador [19] por ello en el siguiente Capítulo se describiran las implementaciones epecíficas para cada caso.

Especificaciones de diseño del los filtros RC (Tabla 3.38) y RRC (Tabla 3.39):

Extensión	Taps	R_{sym} (Hz)	$F_s(Hz)$	α	Sesión Prototipo
$6T_s$	24	2400	9600	0.33	$RC-1200fc-9600Fs-033roll-24taps.fda$
$6T_s$	30	2400	12000	0.33	$RC-1200fc-12000Fs-033roll-30taps.fda$
$6T_s$	48	2400	19200	0.33	$RC-1200fc-19200Fs-033roll-48taps.fda$
$6T_s$	60	2400	24000	0.33	$RC-1200fc-24000Fs-033roll-60taps.fda$
$6T_s$	96	2400	38400	0.33	$RC-1200fc-38400Fs-033roll-96taps.fda$
$6T_s$	120	2400	48000	0.33	$RC-1200fc-48000Fs-033roll-120taps.fda$

Table 3.38: Prototipos de Filtros RC

Extensión	Taps	R_{sym} (Hz)	$F_s(Hz)$	α	Sesión Prototipo
$6T_s$	24	2400	9600	0.33	$RRC-1200fc-9600Fs-033roll-24taps.fda$
$6T_s$	30	2400	12000	0.33	$RRC-1200fc-12000Fs-033roll-30taps.fda$
$6T_s$	48	2400	19200	0.33	$RRC-1200fc-19200Fs-033roll-48taps.fda$
$6T_s$	60	2400	24000	0.33	$RRC-1200fc-24000Fs-033roll-60taps.fda$
$6T_s$	96	2400	38400	0.33	$RRC-1200fc-38400Fs-033roll-96taps.fda$
$6T_s$	120	2400	48000	0.33	$RRC-1200fc-48000Fs-033roll-120taps.fda$

Table 3.39: Prototipos de Filtros RRC

3.10.3.1 Detalle Filtros Prototipo RC y RRC

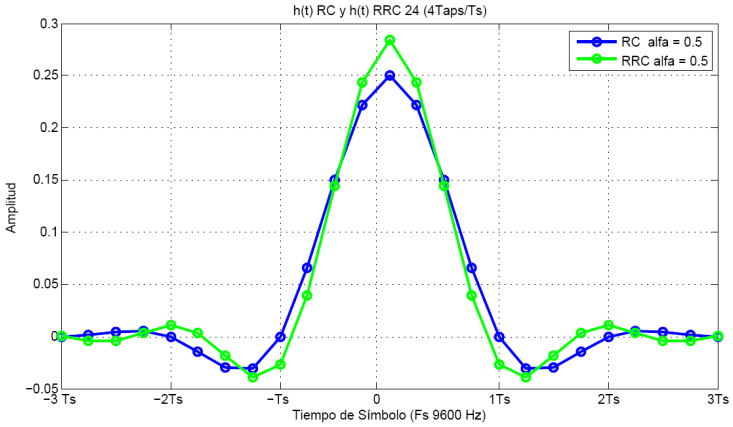

Figure 3.68: h(k) FIR 24 Taps

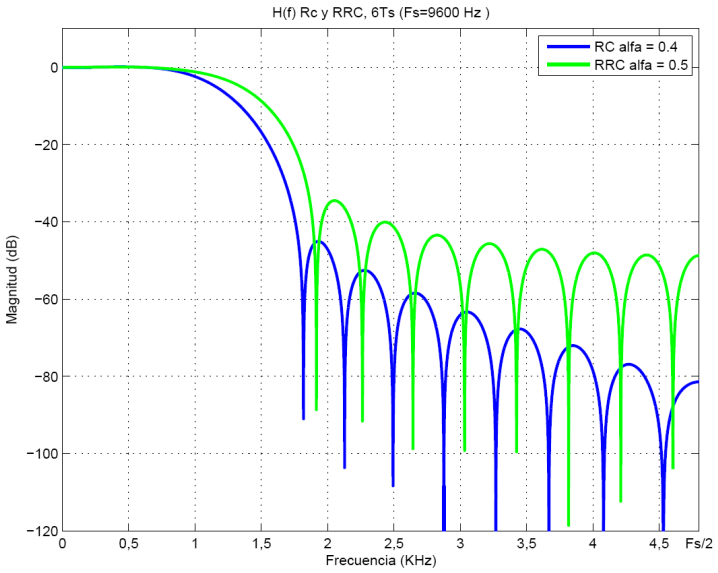

Figure 3.69: Espectro FIR 24 Taps

ARCHIVO COEFICIENTES: 'Coeff6-RRC1200fc-9600Fs-05roll-24taps' y 'Coeff-RC-1200fc-9600Fs-033roll-24taps'.

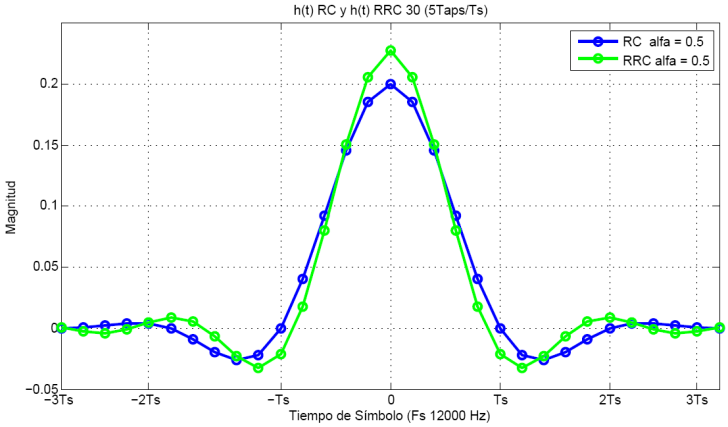

Figure 3.70: h(k) FIR 30 Taps

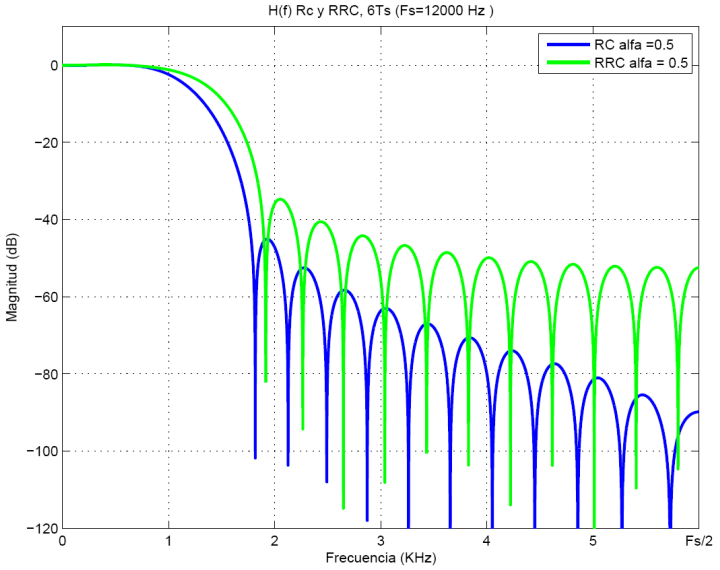

Figure 3.71: Espectro FIR 30 Taps

ARCHIVO COEFICIENTES: 'Coeff1-RC-1200fc-12000Fs-05roll-30taps' y 'Coeff8-RRC-1200fc-12000Fs-033roll-30taps'

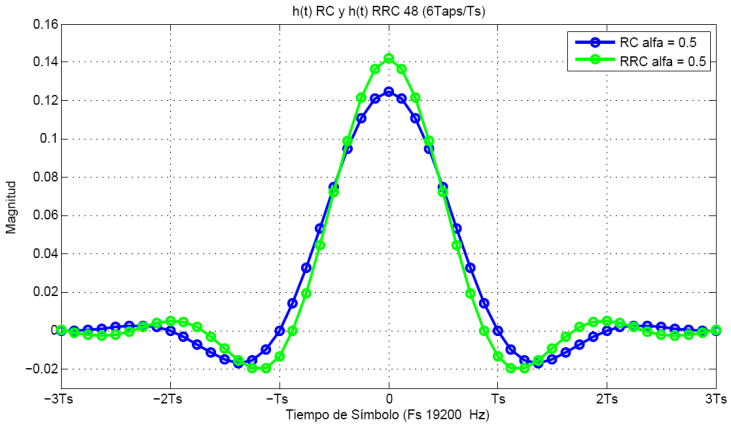

Figure 3.72: h(k) FIR 48 Taps

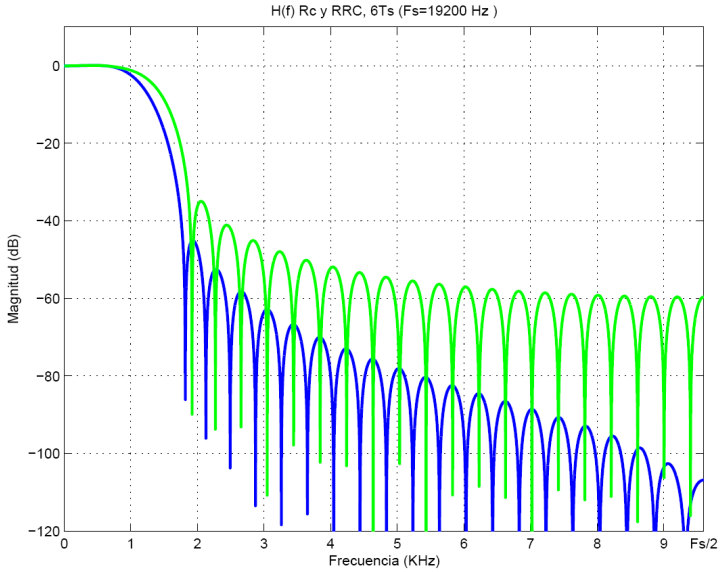

Figure 3.73: Espectro FIR 30 Taps

ARCHIVO COEFICIENTES: 'Coeff7-RRC-1200fc-19200Fs-05roll-48taps' y 'Coeff2-RC-1200fc-19200Fs-033roll-48taps'

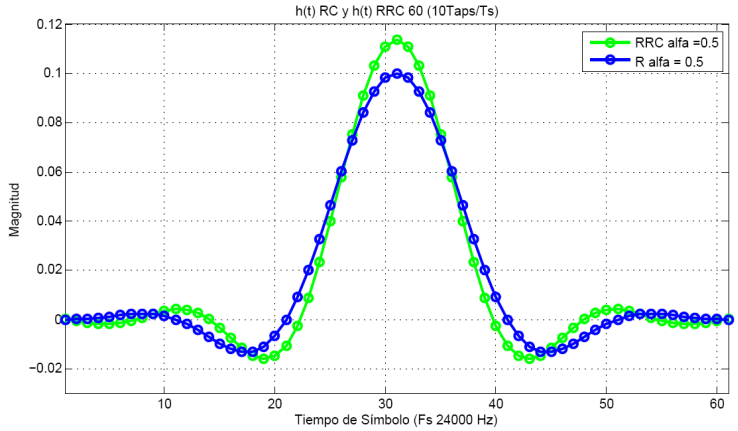

Figure 3.74: h(k) FIR 60 Taps

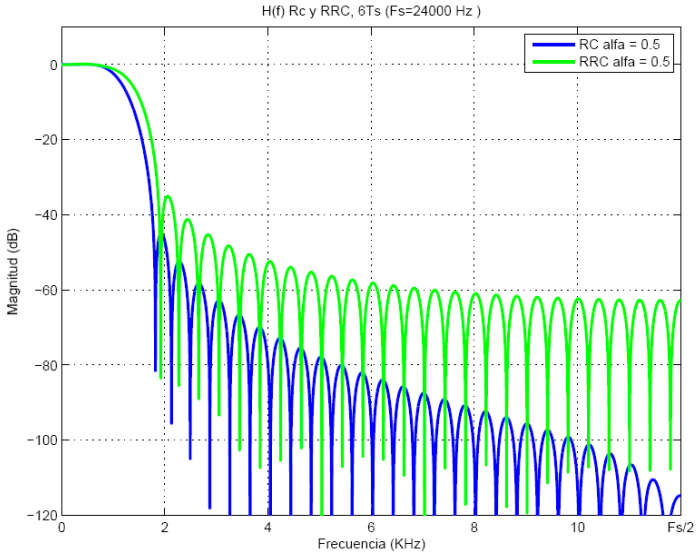

Figure 3.75: Espectro FIR 60 Taps

ARCHIVO COEFICIENTES: 'Coeff9-RRC-1200fc-24000Fs-05roll-60taps' y 'Coeff3-RC-1200fc-24000Fs-033roll-60taps'

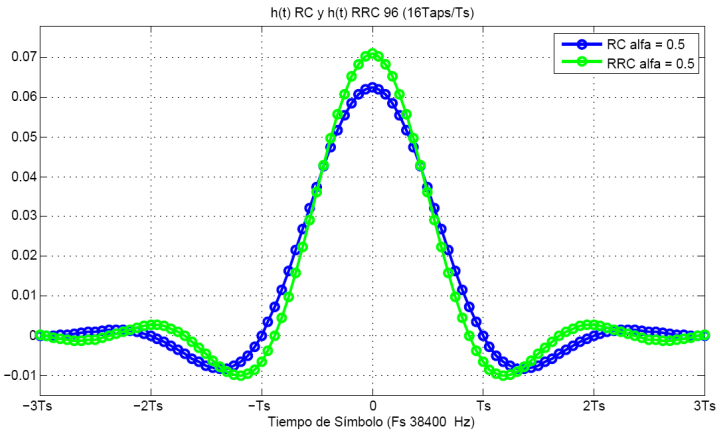

Figure 3.76: h(k) FIR 96 Taps

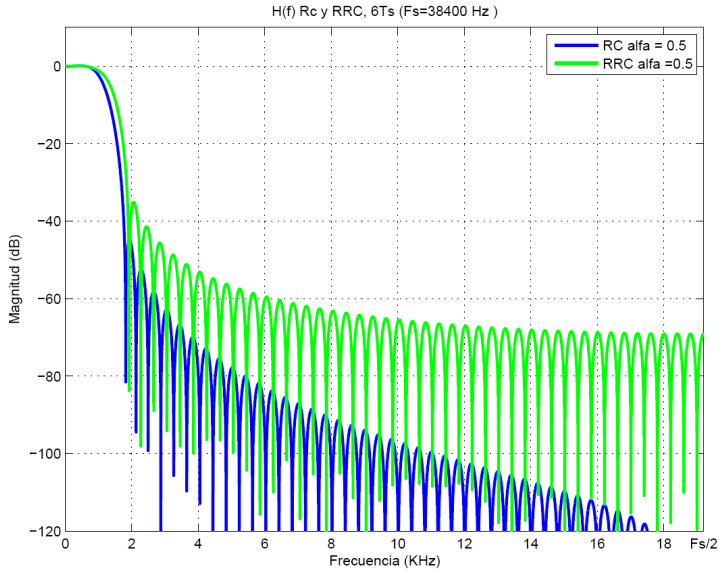

Figure 3.77: Espectro FIR 96 Taps

ARCHIVO COEFICIENTES: 'Coeff10-RRC-1200fc-38400Fs-05roll-96taps' y 'Coeff4-RC-1200fc-38400Fs-033roll-96taps' .

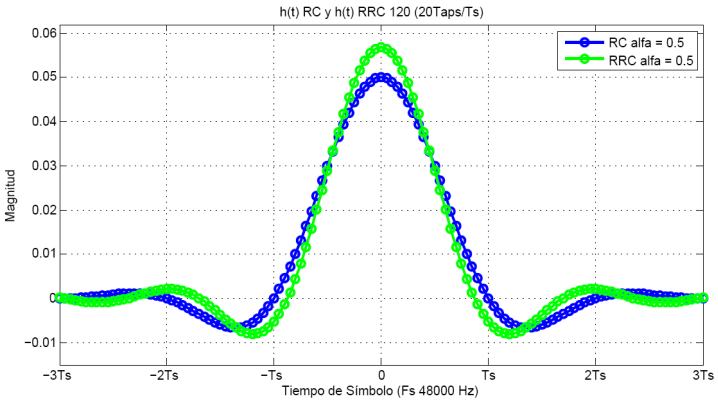

Figure 3.78: h(k) FIR 120 Taps

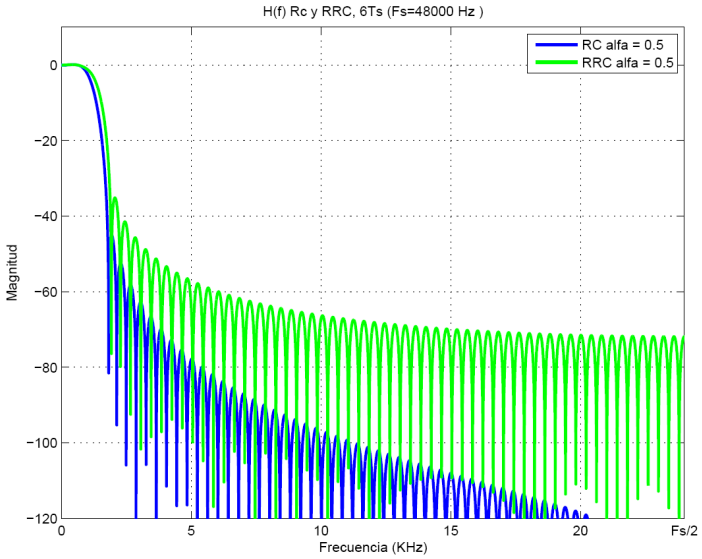

Figure 3.79: h(k) FIR 120 Taps

ARCHIVO COEFICIENTES: 'Coeff11-RRC-1200fc-48000Fs-05roll-120taps' y 'Coeff5-RC-1200fc-48000Fs-033roll-120taps'

NOTA: Todas las respuesta simpulsivas están escaldar para ue en la banda pasante la ganancia sea 1 (0dB).

3.10.4 Diagrama de Flujo Algoritmo de Transmisión

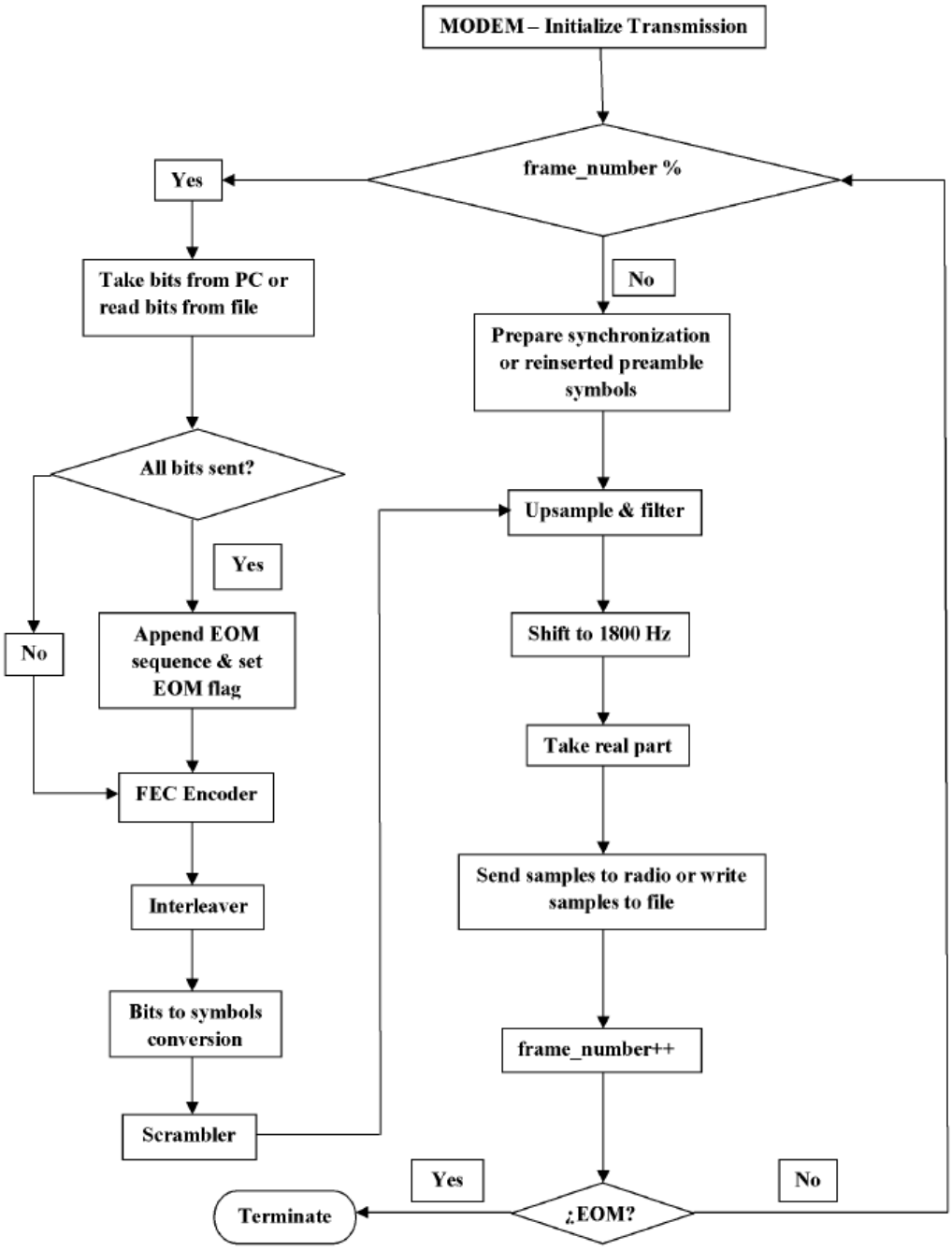

Figure 3.80: Diagrama de flujo Transmisor

El algoritmo de transmisión se refiere a la secuencia de transmisión de las formas de onda que contienen la información.Esta secuencia está formada por una estructura de datos adicionales que permiten desde

el lado del receptor las siguientes operaciones: sincronizacion, ecualización , des-entrelazado y final de mensaje. Estas operaciones definen las siguiente fases de armada de la trama de datos final (Sección 5.3.2.1 [3]).

- Fase de armado de secuencia de secuencia preambulo (S2).
- Fase de armado de secuencia de datos binarios (S1).
- Fase de armado de secuencia de secuencia Final de Mensaje (S1).
- Fase de armado de secuencia de codificanción y entrelazado secuencias anteriores (S1 y S4).

3.11 *Consideraciones de diseño Modulador para equipos de comunicaciones disponibles*

HF

3.11.1 Compatibilidad Modulaciones MIL 188-110B con HF VX1700

El Vertex VX1700 transmite en HF en los siguientes modos:

1. A1A, AM DSB, canal simple información digital, telegrafía, [20].
2. J2B, AM SSB, canal simple información digital, telegrafía, [20].
3. J2E, AM SSB, canal simple información digital, telefonía, [20].
4. A3E, AM DSB, canal simple información analógica, telefonía (incluye sonido), [20].

De los modos anteriores el mas apto pra la canalización del modem es la A3E con la cual se dispone de un canal de Audio de 3KHz confrecuencia de corte inferior 200Hz.

3.11.1.1 Analisis para modulaciones $R_b < 1200$ bps

Observando la Tabla 3.3 se puede verificar que todas la variantes pueden ser implementadas en el canal de audio dispobinble.

3.11.1.2 Analisis para modulaciones $R_b > 1200$ bps

Para este tipo de modulación, y manteniendo la compatibiilidad en portadora de 1800Hz se calculan los límites de R_s factibles. Para ello verifcamos la banda lateral de menor extension y resulta en $B_T = 3KHz - 1.8KHz = 1200Hz$. Por lo tanto asumiendo que el máximo factor de roll-off (α) en terminos practicos es 0.35, se verifica que la máxima R_s resulta:

$$R_s = B_T(1+\alpha) = 1200 * 1.33 \approx 1600 Sym/sec. \tag{3.47}$$

Por lo tanto para implementar modulaciones del Apendice C los valores propuestos en [3] (Tabla 3.30) se reducen de la siguiente manera cuando se utiliza el VX1700:

MODEM	bps	$R_s(sym/s)$	$f_c(Hz)$	$f_{Null,low}(Hz)$	$f_{Null,high}(Hz)$	Filtro	Roll-off	Modulador
QPSK	2100	1600	1800	200	3400	RRC	0.33	fig 3.54
8QPSK	3100	1600	1800	200	3400	RRC	0.33	fig 3.54
QAM16	4200	1600	1800	200	3400	RRC	0.5	fig 3.54
QAM32	5300	1600	1800	200	3400	RRC	0.5	fig 3.54
QAM64	6300	1600	1800	200	3400	RRC	0.5	fig 3.54
QAM64	8500	1600	1800	200	3400	RRC	0.5	fig 3.54

Table 3.40: Comparación parametros moduladores QPSK y QAM para B_T¡$B_{T,MIL}$ ([3])

NOTA: Los calculos fueron obtenidos del manunal del equipo.

Concluyendo, para este caso tanto el α como T_s deben ser modficados.

3.11.2 Compatibilidad Modulaciones MIL 188-110B con equipo VHF ICOM IC AC110

El equipo ICOM IC AC110 transmite en el siguiente modo

1. 6k00a3e lo que significa que se trata de modulación AM,DBSC, ancho de banda de transmisión $B_T = 6KHz$ y es un sistema de voz y telefonía [20]. Dado B_T y que es DSBSC obtenemos que el canal de audio es de 3KHz máximo.

3.11.2.1 Analisis para modulaciones $R_b < 1200$ bps

Observando la Tabla 3.3 se puede verificar que todas la variantes pueden ser implementadas en el canal de audio dispobinble.

3.11.2.2 Analisis para modulaciones $R_b > 1200$ bps

Para estas modulaciones, y manteniendo la compatibiilidad en portadora de 1800Hz se calculan los límites de R_s factibles. Para ello verifcamos la banda lateral de menor extension y resulta en $B_T = 3KHz - 1.8KHz = 1200Hz$. Por lo tanto asumiendo que el máximo factor de roll-off (α) en terminos practicos es 0.35, se verifica que la máxima R_s resulta:

$$R_s = B_T(1+\alpha) = 1200 * 1.33 \approx 1600 Sym/sec. \tag{3.48}$$

Por lo tanto para implementar modulaciones del Apendice C los valores propuestos en [3] (Tabla 3.30) se reducen de la siguiente manera cuando se utiliza el VX1700:

MODEM	bps	$R_s(sym/s)$	$f_c(Hz)$	$f_{Null,low}(Hz)$	$f_{Null,high}(Hz)$	Filtro	Roll-off	Modulador
QPSK	2100	1600	1800	200	3400	RRC	0.33	fig 3.54
8QPSK	3100	1600	1800	200	3400	RRC	0.33	fig 3.54
QAM16	4200	1600	1800	200	3400	RRC	0.5	fig 3.54
QAM32	5300	1600	1800	200	3400	RRC	0.5	fig 3.54
QAM64	6300	1600	1800	200	3400	RRC	0.5	fig 3.54
QAM64	8500	1600	1800	200	3400	RRC	0.5	fig 3.54

Table 3.41: Comparación parametros moduladores QPSK y QAM para B_T¡$B_{T,MIL}$ ([3])

3.11.3 Compatibilidad Modulaciones MIL 188-110B con equipos VHF y UHF VX3200

El Vertex 3200X transmite en VHF los siguientes modos:

1. 16k0f3e (VHF). Se trata de una modulación FM, ancho de banda 16KHz, modulación FM y es un sistema de voz y telefonía [20]. Dado que el $\Delta f = 5KHz$ se obtiene que el canal de audio es de 3KHz.

2. 11k0f3e (UHF). Se trata de una modulación FM, ancho de banda 11KHz, modulación FM y es un sistema de voz y telefonía [20].Dado que el $\Delta f = 2.5KHz$ se obtiene que el canal de audio es de 3KHz.

3.11.3.1 Analisis para modulaciones $R_b < 1200$ bps

Observando la Tabla 3.3 se puede verificar que todas la variantes pueden ser implementadas en ambos equipos para el canal de audio dispobinble.

3.11.3.2 Analisis para modulaciones $R_b > 1200$ bps

Como ambos equipos presentan el mismo ancho de banda el analisis es similar para ambos. La unica diferencia es que la versión de UHF , por tener menor desvío , va a ser mas vulnerable al ruido.
Para estas modulaciones, y manteniendo la compatibiilidad en portadora de 1800Hz se calculan los límites de R_s factibles. Para ello verifcamos la banda lateral de menor extension y resulta en $B_T = 3KHz - 1.8KHz = 1200Hz$. Por lo tanto asumiendo que el máximo factor de roll-off (α) en terminos practicos es 0.35, se verifica que la máxima R_s resulta:

$$R_s = B_T(1+\alpha) = 1200 * 1.33 \approx 1600 Sym/sec. \tag{3.49}$$

Por lo tanto para implementar modulaciones del Apendice C los valores propuestos en [3] (Tabla 3.30) se reducen de la siguiente manera cuando se utiliza el VX1700:

MODEM	bps	$R_s(sym/s)$	$f_c(Hz)$	$f_{Null,low}(Hz)$	$f_{Null,high}(Hz)$	Filtro	Roll-off	Modulador
QPSK	2100	1600	1800	200	3400	RRC	0.33	fig 7.1
8QPSK	3100	1600	1800	200	3400	RRC	0.33	fig 7.1
QAM16	4200	1600	1800	200	3400	RRC	0.5	fig 7.1
QAM32	5300	1600	1800	200	3400	RRC	0.5	fig 7.1
QAM64	6300	1600	1800	200	3400	RRC	0.5	fig 7.1
QAM64	8500	1600	1800	200	3400	RRC	0.5	fig 7.1

Table 3.42: Comparación parametros moduladores QPSK y QAM para B_T¡$B_{T,MIL}$ ([3])

3.11.4 Compatibilidad Modulaciones MIL 188-110B con equipo UHF RF-DataTech (Banco de pruebas)

Las placas RX470 y ATX470 trabajan con las siguientes caracteristicas de modulación FM.

1. Tansmite en FM, $\Delta f = 7.5KHz$ y ancho de banda de audio es 2.4KHz.

3.11.4.1 Analisis para modulaciones $R_b < 1200$ bps

Observando la Tabla 3.3 se puede verificar que todas la variantes pueden ser implementadas en el canal de audio dispobinble.

3.11.4.2 Analisis para modulaciones $R_b > 1200$ bps

3.11.5 Sobre el ancho de banda de audio

Resepcto al ancho de banda de audio tenido en cuenta en los analisis anteriores, cabe comentar que se han utlizado los anchos de banda especificados por los fabricantes. En ninguno de los equipos describen

la función transferencia de la banda pasante por lo que es factible que dicha función transferencia cumpla con la atenuación que dicta la máscara espectral propuesta en [3] pp 92. En este caso se podrían alcanzar las velocidades referidas en el estandar.

3.12 *Conclusiones*

Chapter 4

Diseño Demodulador MODEM MIL 188-110B

Este chapter .

4.1 Diagrama en bloques Receptor

4.2 Diagrama de Flujo Algoritmo Receptor

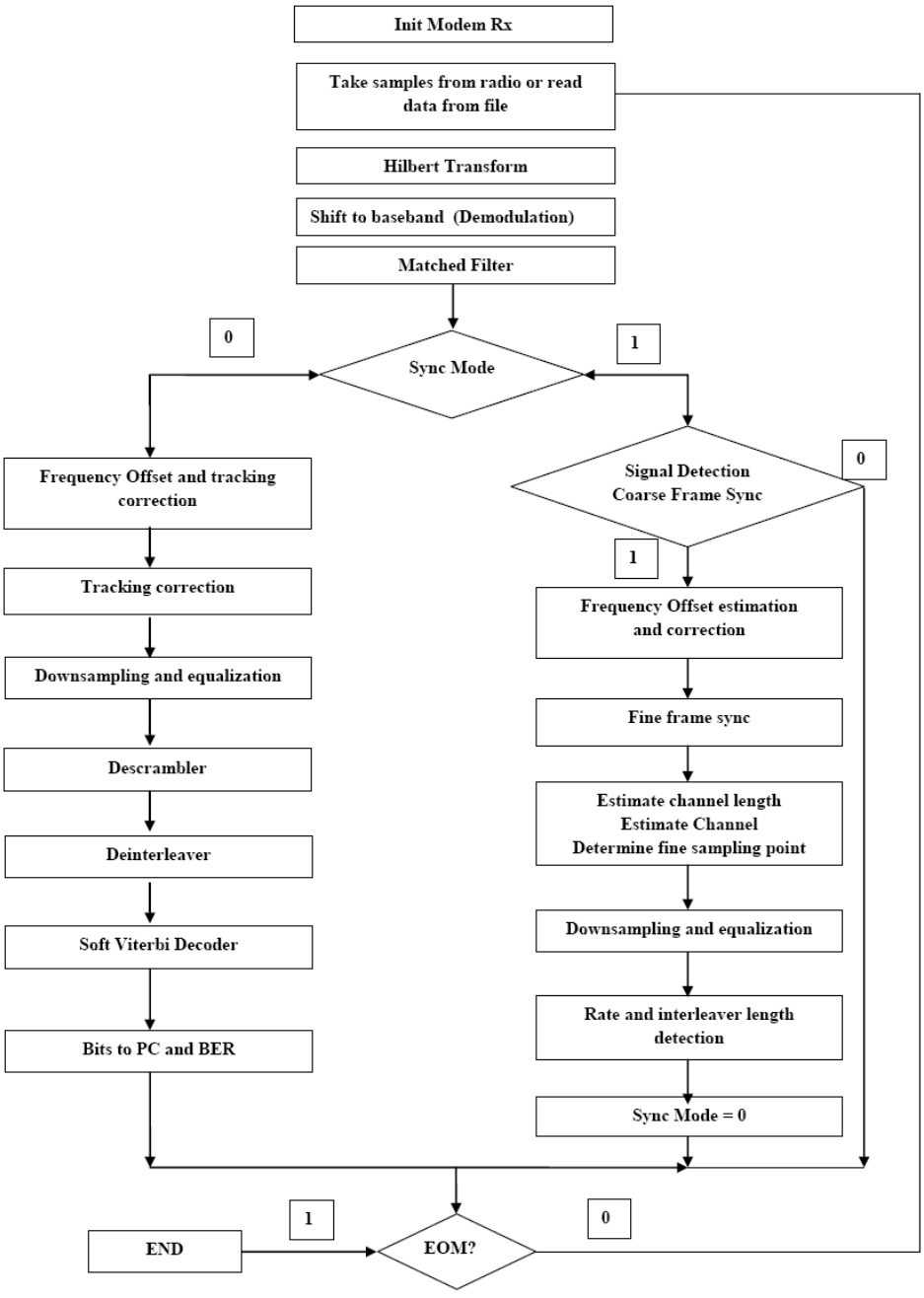

Figure 4.1: Receiver Flow Chart

Chapter 5

Simulador Canal HF y Análisis de canales VHF y UHF

En este Capítulo se presenta un análisis de los canales VHF y UHF con el objetivo de evaluar la implementación del estandard MIL 188-110B en . El objetivo es considerar a grandes rasgos

5.1 Repaso canal HF

5.1.1

5.1.2

5.1.3

5.2 Modelos Canal VHF (30 MHz - 300MHz)

5.3 Modelos Canal UHF (300 MHz - 3 GHz)

5.3.1

5.4 Channel Consideration

In order to characterize the mobile radio channel, several parameters have been developed which provide insight into the effect of the channel upon the transmitted signal. The most important of these are the *R*MS delay spread (τ_τ), *t*he Coherence Bandwidth (B_C), and the *C*oherence Time of the channel (T_C) or its inverse the Doppler Spread B_D.

- *R*MS Delay Spread τ_τ: The RMS delay spread of a mobile radio channel characterizes the time dispersive nature of the channel. The RMS delay spread,τ_τ,is found from the impulse response function of the channel according to the equations 32, 33 and 34 from [21] or equations 3.37, 3.38 and 3.39 [22] where *m*ean excess delay and *R*MS delay are defined respectively.

- *C*oherence Bandwidth B_C: The coherence bandwidth is the statistical measure of the frequency range over which two frequency signals are strongly amplitude-correlated. Depending of the degree of amplitude correlation of two frequency-separated signals,there are different definitions of this parameter.
  *F*irst Definition: According to the first definition, the coherence bandwidth, B_C, is a bandwidth over which the frequency correlation function is above 0.9% or 90%, then it equals:

$$B_C \approx \frac{1}{50.\tau_{rms}} = \frac{0.02}{\tau_{rms}} \tag{5.1}$$

  *S*econd Definition:According to the second definition, the coherence bandwidth, B_C,is a bandwidth over which the frequency correlation function is above 0.5% or 50%, then it equals:

$$B_C \approx \frac{1}{5.\tau_{rms}} = \frac{0.2}{\tau_{rms}} \tag{5.2}$$

- *C*oherence Time: Coherence time T_C is the time-domain dual of Doppler spread and is used to characterize the time-varying nature of the frequency-dispersive properties of the channel in time coordinates. There is a simple relationship between these two channel characteristics:

$$T_C \approx \frac{1}{f_{Dmax}} \approx \frac{\lambda}{v} \tag{5.3}$$

- *D*oppler Spread: is a measure that is defined as a range of frequencies over which the received Doppler spectrum is essentially nonzero. It shows the spectral spreading caused by the time rate of change of the dynamic radio channel due to relative motions of vehicles (or scatters(see HF ITU 520 for example)) with respect to the receiver. According to the Doppler effect, the Doppler spread B_D depends on the maximal Doppler shift, $f_{Dm} = v/\lambda$, and on the angle ϕ between the direction of motion of moving vehicle and direction of arrival of the reflected and scattered waves:

$$B_D = 2.f_{Dm}cos\phi = 2(v/\lambda)cos\phi \tag{5.4}$$

5.4.1 Channel Behavior in digital systems: relationship between the transmitted signal symbol time and channel characteristics

According to the relationship between Symbol Time T_s, (or its inverse, the Symbol Bandwidth B_S), the Channel Coherence Bandwidth B_C, the Channel Delay Spread τ and the Channel Doppler Spread B_D (or its inverse, the Channel Coherence Time T_c); the channel behavior can show FADING effects.
There are two type of **fading effects due to TIME DISPERSION** and **two type of fading effects due to DOPPLER SPREAD**.
*I*mportant NOTE:
*T*IME DISPERSION is due to multipath delays in the channel (has NOTHING to do with the motion of the mobile or channel). TIME DISPERSION causes 'small-scale (fast)' fading, either FLAT or FREQEUNCY SELECTIVE.
*D*OPPPLER SPREAD has to do with the motion of the mobile or the channel (has NOTHING to do with the time delay spread of the channel). DOPPLER SPREAD causes 'large-scale (slow)' fading, either FLAT or FREQUENCY SELECTIVE.

Fading can be classified as a 'slow' or 'fast'. Fast fading occurs when $T_C < T_S$, slow fading occurs when $T_C > T_S$.

Figure 5.1 (From [22]) shows a resume of the relationship among symbol time (or bandwidth) and channel characteristics:

Type of Fading within the Channel	*Relations between Signal and Channel Parameters*
(A1): Flat fast fading (FFF)	(A.1.1): $B_S \ll B_c$, $B_S < B_D$
	(A.1.2): $T_S \gg \sigma_\tau$, $T_S > T_c$
(A.2): Frequency-selective fast fading (FSFF)	(A.2.1): $B_S > B_c$, $B_S < B_D$
	(A.2.2): $T_S < \sigma_\tau$, $T_S > T_c$
(B1): Flat slow fading (FSF)	(B.1.1): $B_S \ll B_c$, $B_S \gg B_D$
	(B.1.2): $T_S \gg \sigma_\tau$, $T_S \ll T_c$
(B.2): Frequency-selective slow fading (FSSF)	(B.2.1): $B_S > B_c$, $B_S \gg B_D$
	(B.2.2): $T_S < \sigma_\tau$, $T_S \ll T_c$

Figure 5.1: Types of Fading Depending on Relations between the Signal and the Channel Parameters

Figure 5.2 (From [23]) resumes a behaviour of the channels according to the relationship between transmission channel and channel charcaterisics:

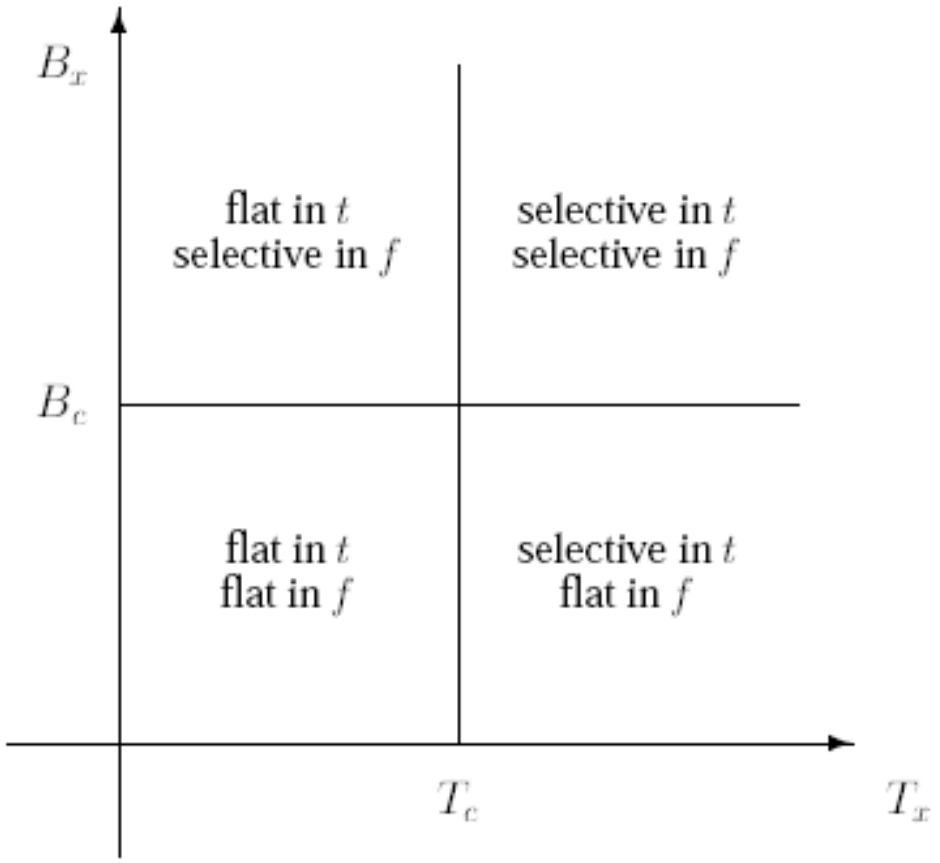

Figure 5.2: Channel classification. B_S, T_S symbol Bandwidth and Time, T_C Coherence Time ($\approx 1/B_D$) and B_C Coherence Bandwidth ($\approx 1/\tau_{rms}$)

*N*OTES on DOPPPLER SPREAD: The Doppler Spread describes the time rate of change of the state of the channel. For example if the channel has a 100Hz of B_D, then the radio channel changes its 'state' at a rate of 100 times per second and a particular trained equalizer may be assumed to be static for 0.01 seconds [21]. Also the Doppler spread yield insight into the proper length for block codes that may be used to protect data in wireless link.

5.4.2 Channel scenarios under consideration

For the MODEM design we consider the following scenarios at HF, VHF and UHF frequency ranges:

1. Land-Land (Fixed Slow speed) Scenario (HF > 1000Km, VHF & UHF up to 160km)
2. Land-Sea (max 50Km) Scenario (HF only)

3. Land-Air (200Km, max 1000Km) Scenario (VHF & UHF up to 160 Km)

4. Air-Air (200Km, max 1000Km) Scenario (VHF & UHF up to 160 Km)

5.4.2.1 RMS Delay Spread Considerations τ_{rms}

The following rms delay spread were considered for different scenarios:
*S*cenario 1:

1. For HF (long range) ITU 520.
   $\tau_{rms} = 2ms$. (10 ms at most beyond the arctic and antarctic circles)

2. For VHF and UHF (short range) we adopt Erceg model [24], [25] .
   RMS Delay Spread is then calculated as follow:

$$\tau_{rms} = T_1 d^{\epsilon} y \tag{5.5}$$

   Where, d=distance in Km, T_1 is the median value of τ_{rms} at d=1Km, ϵ lies between 0.5-1.0 and y is a lognormal variate. See next figure for channel values [**?**]:

TABLE III
MODEL PARAMETERS AND THEIR QUANTIFICATION

Parameter	Proposed Value, Range of Values† or Method of Quantification
G_1	Apply Hata's model [31] at d = 1 km.
T_1	0.4 μs (urban microcells) 0.4-1.0 μs (urban macrocells) 0.3 μs (suburban areas) 0.1 μs (rural areas) ≥ 0.5 μs (mountainous areas)*
σ_x	6-12 dB
σ_y	2-6 dB
γ	3.0-4.0
ε	0.5 (urban, suburban, rural areas) 1.0 (mountainous areas)
ρ	-0.75

Figure 5.3: Channel Model Parameter and their Quantification

Terrain	Distance Km	τ_{rms} ns
Urban	1.6	1265
Suburban	1.6	480
Rural	1.6	379
Urban	16	4000
Suburban	16	1200
Rural	16	400
Urban	160	12600
Suburban	160	3794
Rural	160	1264

Table 5.1: RMS Delay Spread for Fixed Channel

Freq MHz	Distance Km	τ_{rms} ns
100	1.6	1.26
100	16	4.13
100	160	46.8
500	1.6	1.26
500	16	4.13
500	160	46.8

Table 5.2: RMS Delay Spread for Air-Land Channel

So, for a fixed link the rms delay spread adopt the following values assuming operation at urban, suburban and rural areas:

The Erceg model was adopted by IEEE 802.16d standard. This satandard also describes that if a directional antenna with specific radiation pattern is used [**?**], then, the rms delay spread can be reduced by a factor of 2.3 [25].

*S*cenario 2: We use ITU 520 for HF (long range only).
$\tau_{rms} = 2ms$.
*S*cenario 3: We use Bello channel model (VHF and UHF) short range only.
According to [26] the Land-Air channel has the following rms delay spread values(worst case surface fluctuations):
For more details see [27] and [28].
*S*cenario 4: We use Bello channel model (VHF and UHF) short range only.
According to [26] the Land-Air channel has the following rms delay spread values (worst case surface fluctuations):

Freq MHz	Distance Km	τ_{rms} ns
100	1.6	1162
100	16	279
100	160	3483
500	1.6	1162
500	16	279
500	160	3483

Table 5.3: RMS Delay Spread for Air-Air Channel

For more details see [27] and [28].

5.4.2.2 Equalizer Requirement

Analysis of the equalizer requirements related to the channel characteristics and symbol transmission characteristics.
In previous section we described the rms multipath delay spread τ_{rms}. To avoid the use of the equalizer the designer must select the symbol time duration such as $T_S >> \tau_{rms}$. As a consequence the inter symbol interference (ISI) is negligible [21], [29]. This means that the echoes of the original signal incoming to the receiver, arrives before the reception of the next symbol causing no ISI.
Another parameter that affects the effectiveness of the equalizer is the Doppler Spread B_D or its reciprocal the Coherence Time T_C. Since the use of an equalizer implies the need to measure the channel characteristics, the channel time variations must be relatively slow compared to the transmitted symbol time T_S and, more generally compared to the rms multipath delay spread τ_{rms}. Consequently, $1/B_D = T_C >>$ or, equivalently, the spread factor must satisfy the condition $\tau_{rms}.B_D << 1$, that is, the channel must be underspread.

*R*ESUME: Equalizer Calculations

- Use Equalizer if $T_S << \tau_{rms}$ and $T_C >> \tau_{rms}$ o better said must satisfy the condition $\tau_{rms}.B_d$¡¡1 [29].

- Equalizer adaptation rate $T_{adapt} > T_C$ [21].

*S*cenario 1: Require Equalizer in HF (See Figure 5.4). Do not Require Equalizer at VHF & UHF (See Figure 5.5).
Rate of change of the channel:
For HF B_D 1Hz, so the rate of change of the channel is about 1sec.
The training sequence is in worst case 16 symbols of 83 *usec*. The worst case adaptation time is then $\approx 133msec$. The condition $\tau_{rms}.B_d$¡¡1 is satisfied since $2e-3/1 << 1$.

For fixed VHF or UHF equalizer is not required but anyway we can mention that the $B_D < 0.5Hz$, this change is due to the scatter caused by vehicles and trees movement mainly.

*S*cenario 2: Require Equalizer (See Figure 5.4).
Rate of change of the channel:
For HF B_D 1Hz, so the rate of change of the channel is about 1sec.
The training sequence is in worst case 16 symbols of 83 *usec*. The worst case adaptation time is then

$\approx 133msec$. The condition $\tau_{rms}.B_d$¡¡1 is satisfied since $2e-3/1 << 1$.

*S*cenario 3: Do not require equalizer (See Figure 5.6).

Rate of change of the channel:

For VHF & UHF the maximum B_D is around 200Hz, so if we would increase the data rate to the point were the equalizer is mandatory, then it may be actualized at a speed higher than $1/200Hz$.

*N*OTE: With this Doppler problems could arise at synchronism stage.

*S*cenario 4: Do not require equalizer (See Figure 5.7).

Rate of change of the channel:

For VHF & UHF the maximum B_D is around 600Hz, so if we would increase the data rate to the point were the equalizer is mandatory, then it may be actualized at a speed higher than $1/600Hz$.

*N*OTE: With this Doppler problems could arise at synchronism stage.

Below in figures 5.4, 5.5, 5.6, & 5.7 we can see the main feature of the chanel scenarios in terms of $B_D, T_C, \tau_{rms} and T_S$.

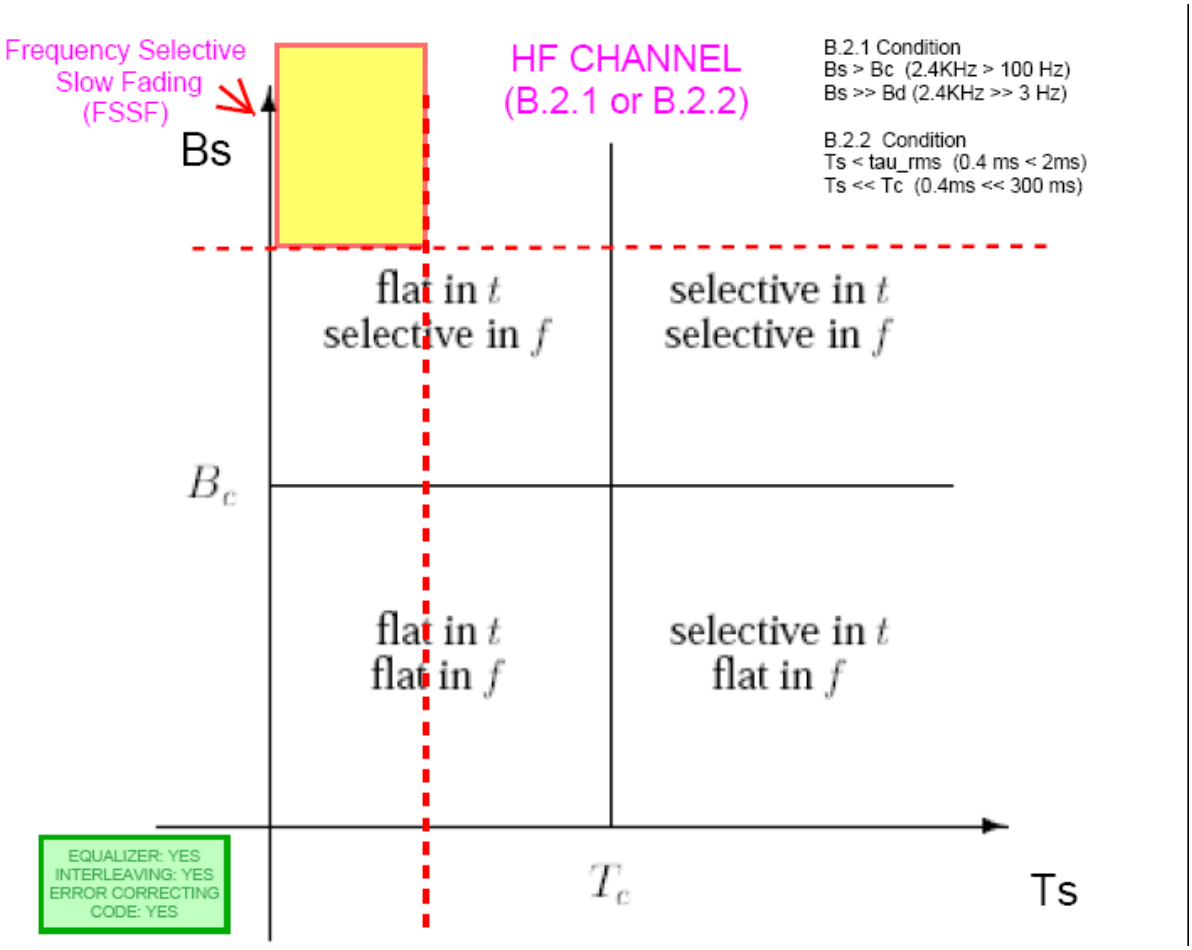

Figure 5.4: HF Channel Behavior

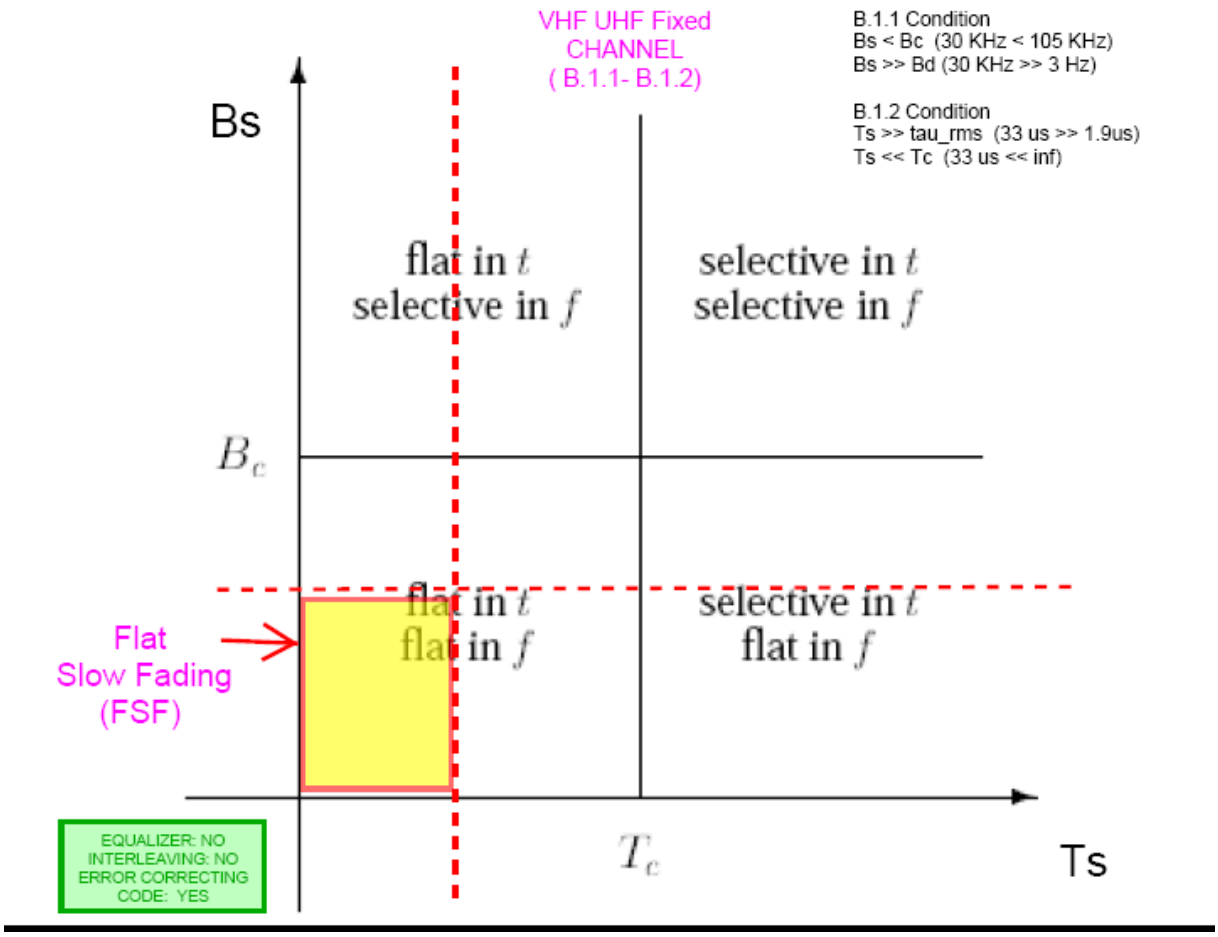

Figure 5.5: VHF UHF Fixed Channel Behavior

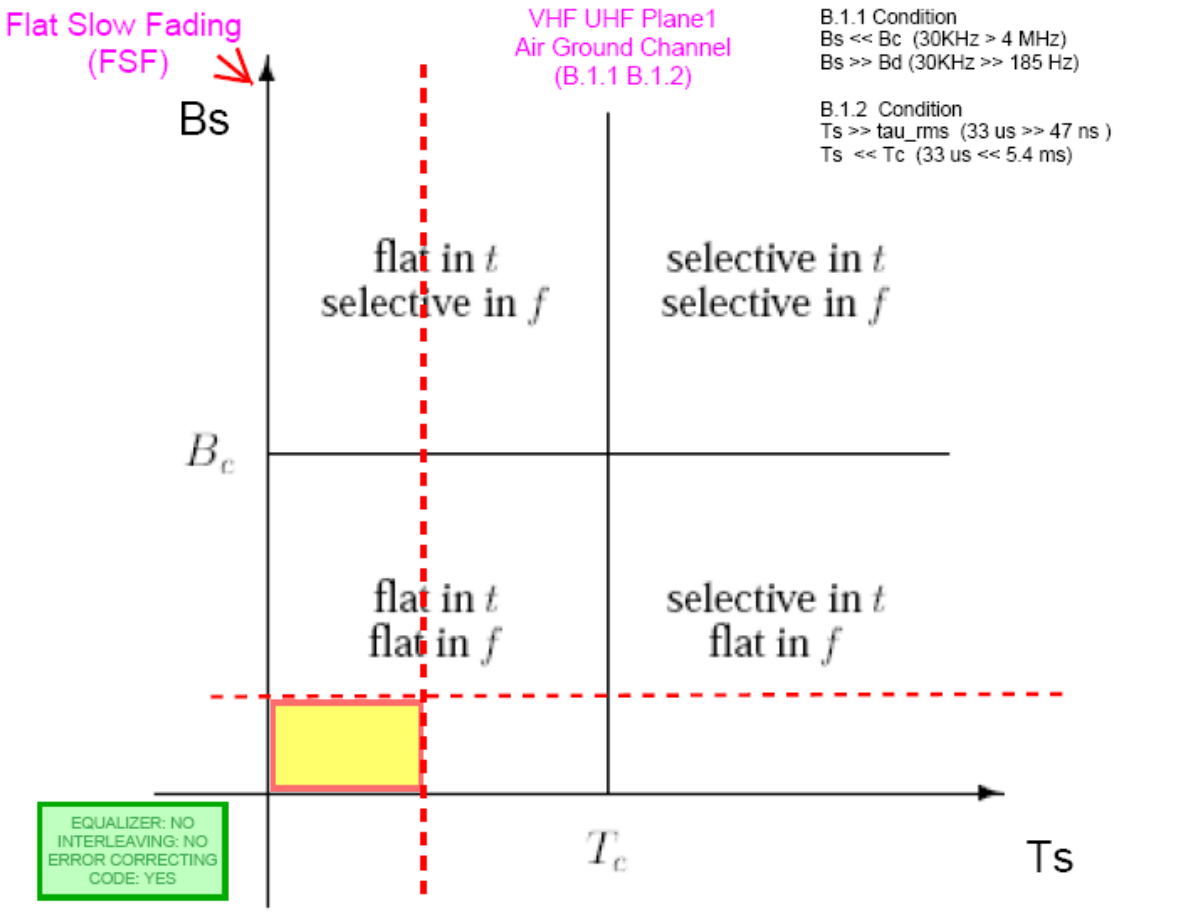

Figure 5.6: VHF UHF Plane 1 Channel Behavior

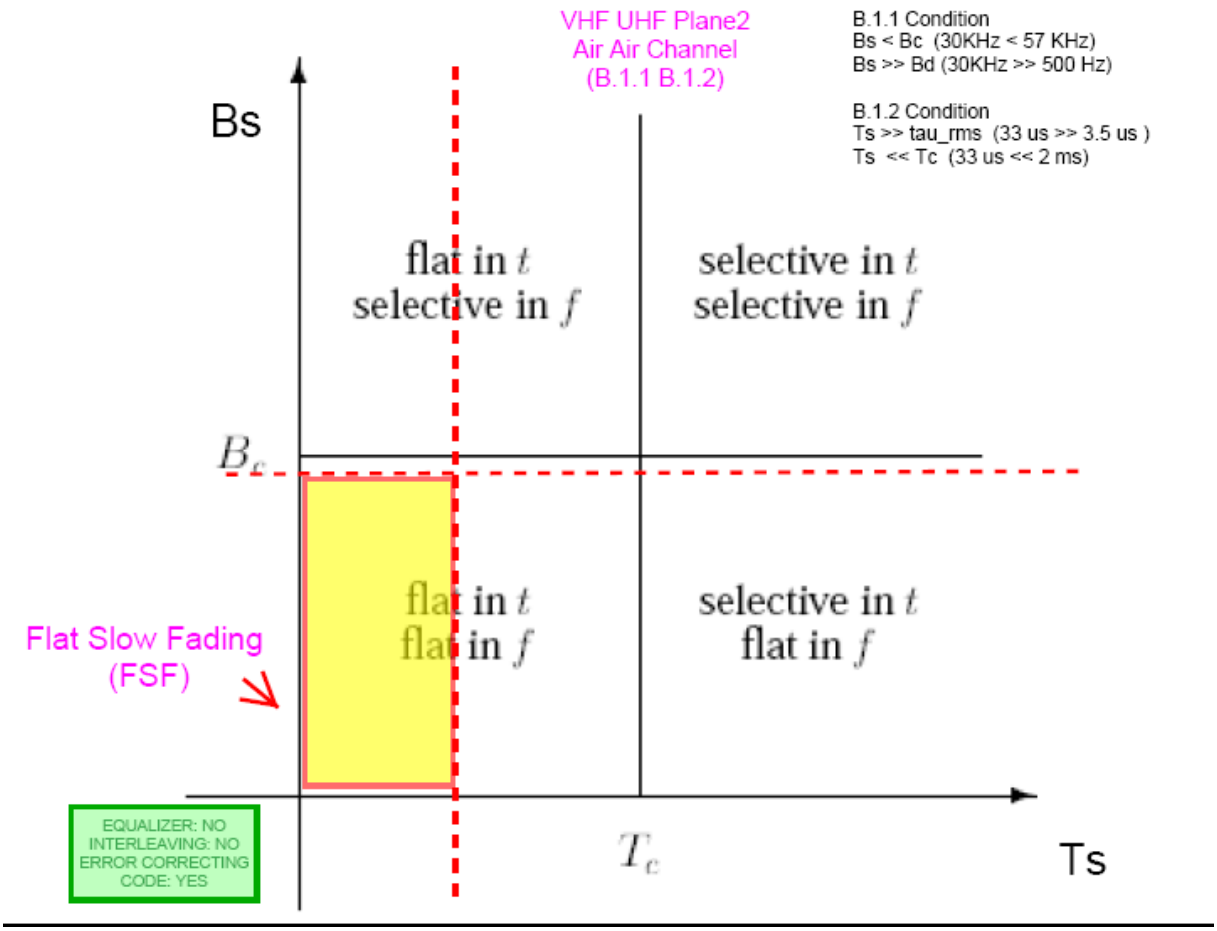

Figure 5.7: VHF UHF Plane 2 Channel Behavior

5.4.2.3 Interleaving and Interleaving latency calculation

The Interleaving calculation must complies with the following approximated expression:

$$T_{IL}/T_C \geq 10 \tag{5.6}$$

Where T_{IL} is the interleaving span time and T_C is the coherence time of the channel. Indeed when greater the previous relationship better the behavior of the interleaving but at cost of increasing the memory and latency of the total system.

Careful must be take in the calculation of optimum interleaving, for example the latency or time delay T_{delay} of the interleaver must be calculated as follow [30] Eqs. 7.7-27,7.7-28 :

$$T_{delay} = 2rNT_S \geq 2NT_C \tag{5.7}$$

where

$$r \geq \frac{T_C}{T_S} \tag{5.8}$$

As a rule of thumb $r \geq 10$ to ensure time diversity is acepted [30] (See [31], [32] for interleaver design examples). The T_{delay} is equal to the decoding delay so the factor 2 in equation 5.7 account for the process of transmission interleaver delay an reception interleaver delay.

The interleaving performs better when more fast are the fadings, since the fast fading are caused by fast movement of on o both of the antennas, then the interleaver works better when faster moves the antennas. Next figures shows how after some speed (60Km) the interleaver improves the SNR [31].

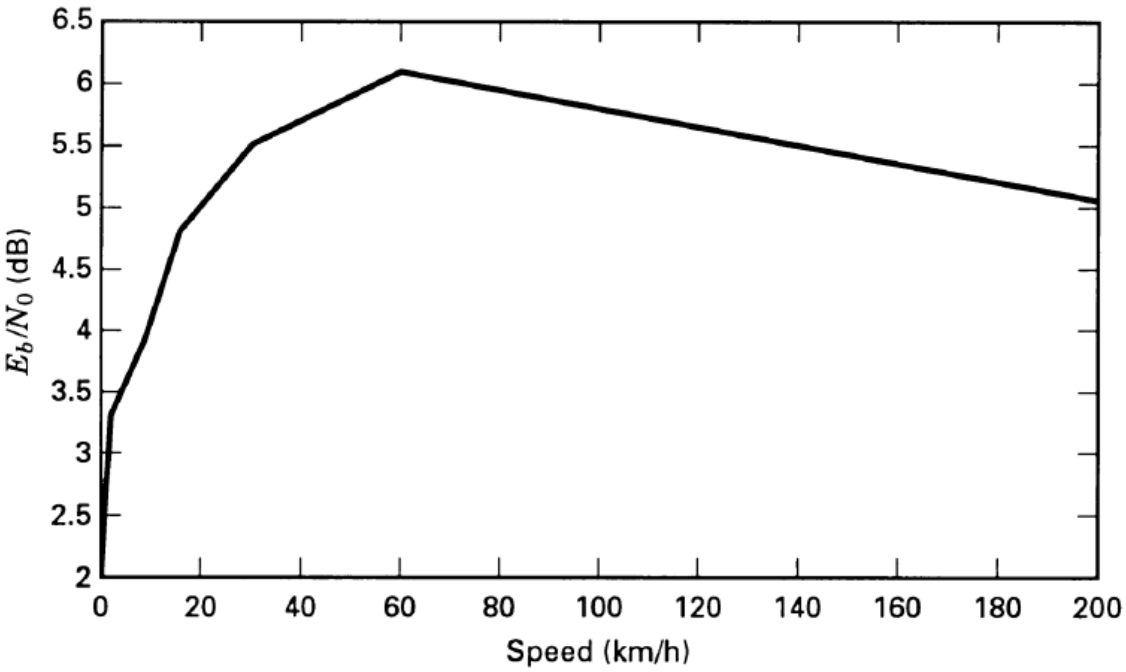

Figure 5.8: SNR performance versus speed

*S*cenario 1: In HF the maximum interleaving time defined by the standard is 6sec, complying in this way with the requirements. So the full interleaving delay (transmitter+receiver is 12sec).
Plane 1 Channel at HF (air ground 200km speed):

$$T_{C_{max}} \approx \frac{0.5}{F_{d_{min}}} \approx 1.69sec \tag{5.9}$$

at VHF so the rule of thumb minimum interleaver delay spread is approx:

$$10.T_C.2 = 33.8sec \tag{5.10}$$

Plane 2 Channel at HF (air air 500km speed):

$$T_{C_{max}} \approx \frac{0.5}{F_{d_{min}}} \approx .37sec \tag{5.11}$$

at UHF so the rule of thumb minimum interleaver delay spread is approx:

$$10.T_C.2 = 7.4sec \tag{5.12}$$

*S*cenario 2: The same as for HF MIL standard 12sec.

*S*cenario 3: Following the Eq 5.8 the minimum interleaving delay for this system is:

$$T_{C_{max}} \approx \frac{0.5}{F_{d_{min}}} \approx 10ms \tag{5.13}$$

at VHF so the rule of thumb minimum interleaver delay spread is approx:

$$10.T_C.2 = 200ms \tag{5.14}$$

$$T_{C_{max}} \approx \frac{0.5}{F_{d_{min}}} \approx 3.2ms \tag{5.15}$$

at UHF so the rule of thumb minimum interleaver delay spread is approx:

$$10.T_C.2 = 64ms \tag{5.16}$$

*S*cenario 4: Following the Eq 5.8 the minimum interleaving delay for this system is:

$$T_{C_{max}} \approx \frac{0.5}{F_{d_{min}}} \approx 4.4ms \tag{5.17}$$

at vHF so the rule of thumb minimum interleaver delay spread is approx:

$$10.T_C.2 = 88ms \tag{5.18}$$

$$T_{C_{max}} \approx \frac{0.5}{F_{d_{min}}} \approx 1.25ms \tag{5.19}$$

at UHF so the rule of thumb minimum interleaver delay spread is approx:

$$10.T_C.2 = 25ms \tag{5.20}$$

Figure 5.9 resumes the B_S, B_D, T_C and τ_{rams} for different secenarios:

System	Fmin (MHz)	Fmax (MHz)	Bandwidth KHz	Max Symbol Time (msec)	Max Delay Spread	Coherence Bandwidth Bc (50%) Hz	Coherence Time Tc Max Sec (1/2Bd)	Coherence Time Tc Min Sec (1/2Bd)	Doopler Spread Bd Min (Hz)	Doppler Spread Bd Max (Hz)
HF VX1700	1.6	30	3	0.41						
Fixed	1.6	30	3	0.41	2ms	100	.5	.5	1+0	1+0
Ship	1.6	30	3	0.41	2ms	100	0.89	0.3125	1+0.118	1+2.2
Plane1	1.6	30	3	0.41	2ms	100	1.69	0.09	0.001+0.59	0.001+11.11
Plane2	1.6	30	3	0.41	2ms	100	0.37	0.0345	1.2+ 1.48	1.2 + 27.77
IC-A1110	116	136	8.33/25	0.12/0.04						
Fixed	116	136	8.33/25	0.12/0.04	1890 ns	105 KHz	.5	.5	1+0	1+0
Ship	116	136	8.33/25	0.12/0.04		4.27 MHz	0.85	0.048	0+8.5	0+10.5
Plane1	116	136	8.33/25	0.12/0.04	46.8 ns	4.27 MHz	0.0115	0.010	0.01+42.9	0.01+50
Plane2	116	136	8.33/25	0.12/0.04	3483 ns	57.4KHz	0.0044	0.0038	6.05+107.4	6.05+125.9
VX3200V	134	174	12.5/15/25/30	0.08/0.066 0.04/0.033						
Fixed	134	174	12.5/15/25/30	0.08/0.066 0.04/0.033	1890 ns	105 KHz	.5	.5	1+0	1+0
Ship	134	174	12.5/15/25/30	0.08/0.066 0.04/0.033		4.27 MHz	0.1005	0.038	0+9.9	0+12.88
Plane1	134	174	12.5/15/25/30	0.08/0.066 0.04/0.033	46.8 ns	4.27 MHz	0.01	0.0075	0.01+49.6	0.01+64.44
Plane2	134	174	12.5/15/25/30	0.08/0.066 0.04/0.033	3483 ns	57.4KHz	0.0038	0.0030	6.05+124	6.05+161.1
VX3200VU	400	512	12.5/25	0.08/0.04						
Fixed	400	512	12.5/25	0.08/0.04	1890 ns	105 KHz	.5	.5	1+0	1+0
Ship	400	512	12.5/25	0.08/0.04		4.27 MHz	0.0165	0.0135	0+29.62	0+37.03
Plane1	400	512	12.5/25	0.08/0.04	46.8 ns	4.27 MHz	0.0032	0.0027	0.052+148.14	0.052+185.18
Plane2	400	512	12.5/25	0.08/0.04	3483 ns	57.4KHz	0.00125	0.0010	30.2+370	30.2+462.96

Fixed (HF long range, VHF UHF short range), 40 Km distance for VHF and UHF. ***Ship*** 40km speed (HF land-sea) > 1000 km distance. ***Plane1*** 200km speed (HF, VHF UHF air-ground), 1600 Km for HF, 160 km for VHF UHF. ***Plane2*** 500km speed (HF, VHF UHF air-air), 1600 Km for HF, 160 km for VHF UHF.

For 116 MHz to 174 MHz equipments range we used 100 MHz air channel data (Bello).

For 400 MHz to 512 MHz equipments range we used 500 MHz air channel data (Bello).

Doppler Spread max and min were calculated taking into account the Doppler due to the movement and the Doppler due to the Channel.

DopplerSpread=Doppler Channel + Doppler due to Movement.

Figure 5.9: Channels Parameters

VHF/UHF Air-Air, Table 8.6, page 82 [26] (includes satellite channel).
VHF/UHF Air-Land, Table 9.2, page 89 [26].

5.4.3 Communication Equipments under consideration

The previous scenarios considered for MODEM design were based on the following communications equipments owned by SIAG.

5.4.3.1 HF

VX 1700, 1.6MHz to 30MHz, bandwidth (channel 3KHz SSB, 6KHz AM)
Audio output impedance (4-16Ohms)
Audio Input impedance (600Ohms)

5.4.3.2 VHF

IC-A1110, 118-136MHZ, bandwidth (channel 8.33KHz or 25KHz).
Audio output impedance (8 Ohms) .
Audio Input Impedance (600 Ohms).

5.4.3.3 UHF

VX 3200, 134-174MHz, 400-512MHz, (channel 12.5-25KHz)
Audio output impedance (4 Ohms) .
Audio Input Impedance (600 Ohms).

Chapter 6

Conclusiones

This chapter describes

6.1 1

Chapter 7

Apéndice II- Código C- DSP

7.1 Introducción

7.2 Bloques Código C Modulador

7.2.1 Modulador IQ

ModIQ.c

7.2.1.1 Tablas Asociadas

Tablas Componentes IQ.

Modulación	Tabla	Archivo
QPSK	C-III	TablasQPSK.m
8QPSK	C-I	TablasQPSK.m
16QAM	C-V	TablasQAM.m
32QAM	C-VI	TablasQAM.m
64QAM	C-VII	TablasQAM.m

Table 7.1: Valores IQ para modulaciones QPSK y QAM

Tablas Fase Tablas Sin(DatosQ), Cos(DatosI)

7.2.2 Modulador Impulso

ModImpulso.c

7.2.3 Integrador

Integrador.c

7.2.4 Mapeador

Mapper.c

7.2.5 FEC

7.2.5.1 Codigo C

begincode

int encoder[N]; /* FEC encoder array */ int $coded_output[2 * D]$; /* Output array */

int data[D]= 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1;

int main() int i; int datacount, Ta, Tb;

/* Initial state of the encoder (all 0s) */ for (i=0; i¡N; i++) encoder[i]=0;

/* FEC coding */ for (datacount=0; datacount¡D; datacount++) /* Update encoder (take 1 bit from data and discard the oldest */ for (i=N-2; i¿=0; i–) encoder[i+1]= encoder[i]; encoder[0]= data[datacount];

/* Coding */ $Ta = encoder[0]^encoder[2]^encoder[3]^encoder[5]^encoder[6]$; $coded_output$[2*datacount]= Ta;

$Tb = encoder[0]^encoder[1]^encoder[2]^encoder[3]^encoder[6]$; $coded_output$ [2*datacount+1]= Tb; ; ;

endcode

7.2.5.2 Codigo Matlab

begincode

clear all; close all; clc;

data = [1 0 0 1 1 1 1 0 0 0 1 0 1 0 0 0 1 1 0 0 1 0 1]; encoder = []; $coded_output$ = [];

N = 7;

for i = 1:N encoder(i) = 0; end

for datacount = 1:length(data)

for i = (N-1):-1:1 encoder(i+1) = encoder(i); end encoder(1) = data(datacount);

Ta = xor(encoder(7),xor(encoder(6),xor(encoder(4),xor(encoder(3), encoder(1))))); $coded_output$(2*datacount-1) = Ta;

Tb = xor(encoder(7),xor(encoder(4),xor(encoder(3),xor(encoder(2), encoder(1))))); $coded_output$(2*datacount) = Tb;

end
endcode

7.2.6 Interleaver

7.2.6.1 C

Interleave load

begincode

/* Constants and arrays/matrices for the program */

int interleaver[R][C]; /* Interleaver matrix */

int data[D]= 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1;

int main() int i, j; int col_count, row_count, total, datacount;

/* Initial state of the interleaver */ for (j=0; j¡C; j++) for (i=0; i¡R; i++) interleaver[i][j]=0;

col_count = 0; row_count = 0; total=0;

for (datacount=0; datacount¡D; datacount++) interleaver[row_count][col_count]= data[datacount];

row_count= row_count + 9; if (row_count¿R-1) row_count= row_count-R;

total= total+1; if (total==R) col_count= col_count+1; total= 0; ; ; ;

endcode

Interleave fetch

begincode

fila= 0; col= 0; $primer_col$= 0;

for (i=0; i¡R*C-1; i++) output[i]= interleaver[fila][col]; fila= fila+1; col= col-17;

if (col¡0) col= col+C;

if (fila¿R-1) fila=0; $primer_col$= $primer_col$+1; col= $primer_col$; ; ;
endcode

7.2.6.2 Matlab

Interleave load

begincode

clear all; close all; clc;

data = 0:1:4*40-1;

interleaver = []; for j = 1:4 for i = 1:40 interleaver(i,j) = 0; end end

count = 0; $data_count$ = 0;

for j = 1:4 for i = 1:40 $data_count$ = $data_count$ + 1; interleaver(count+1,j) = data($data_count$);

count = count + 9; if count¿39 count = count-40; end end end
endcode

Interleave fetch

begincode

clear all; close all; clc;

N = 40; M = 72; data = rand(N,M)¿0.5;

output = zeros(1,N*M);

fila = 1; col = 1; $primer_col$ = 1;

for i = 1:M*N output(i) = data(fila,col); fila = fila + 1; col = col -17;

if col ¡ 1 col = col + M; end

if fila ¿ N fila = 1; $primer_col$ = $primer_col$ + 1; col = $primer_col$; end end
endcode

7.2.7 Gray

7.2.8 Filtro

7.2.9 Scrambler

7.2.9.1 C

begincode

int init[N] = 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1;

int main() int i, j; int temp, times, count; int MSB, MidB, LSB; int bitgen[N];

temp = 0; count = 0; times = 0;

for (i=0; i¡N; i++) bitgen[i] = init[i];

for (i=0; i¡(8*322); i++) temp = bitgen[0];

for (j=1; j¡N; j++) bitgen[j-1] = bitgen[j]; bitgen[N-1] = temp;

bitgen[5] = $bitgen[0]^b itgen[6]$; bitgen[7] = $bitgen[0]^b itgen[8]$; bitgen[10] = $bitgen[0]^b itgen[11]$;

count = count + 1; times = times + 1;

if (times==8) MSB = bitgen[9]; /* Scrambler Most Significant Bit (MSB) */ MidB = bitgen[10]; /* Scrambler Middle Bit (MidB) */ LSB = bitgen[11]; /* Scrambler Least Singificant Bit (LSB) */ times = 0; ;

if (count==(8*160)) for (j=0; j¡N; j++) bitgen[j] = init[j];

count = 0; ;

;

;

endcode

7.2.9.2 Matlab

begincode

clear all; close all; clc;

N = 12;

init = [1 0 1 1 1 0 1 0 1 1 0 1]; bitgen = []; temp = 0; bitgen = init; times = 0; count = 0;

for i = 1:(8*322) temp = bitgen(1);

for j = 2:N bitgen(j-1) = bitgen(j); end bitgen(N) = temp;

bitgen(6) = xor(bitgen(1),bitgen(7)); bitgen(8) = xor(bitgen(1),bitgen(9)); bitgen(11) = xor(bitgen(1),bitgen(12));

count = count + 1; times = times + 1;

if times==8 MSB = bitgen(10); MidB = bitgen(11); LSB = bitgen(12); times = 0; end

if count==(8*160) bitgen = init; count = 0; end end

endcode

7.3 Bloques Código C Demodulador

7.4 CODEC TMS320C6416

Chapter 8

Appendix III- C- CODE for Accessory Hardware

8.1 Introduction

8.2 Programming Transceiver Control Board (PIC32)

8.2.1 Main Program

8.2.2 Keyboard Board

8.2.3 Menu interrupts or Polling?

8.2.4 Display Programming

8.3 Configuring TX UHF Board with Control Board SPI

8.3.1 Initialization Sequence and Program

8.3.2 Control Sequence and Program

8.3.3 Monitoring Sequence and Program

8.4 Configuring RX UHF Board with Control Board SPI

8.4.1 Initialization Sequence and Program

8.4.2 Control Sequence and Program

8.4.3 Monitoring Sequence and Program

9.1. Configuración del DSP

9.1.1. Registros del CODEC AIC23

```
//-----------Codec configuration settings------------
DSK6416_AIC23_Config config = {
    0x0017, // 0 DSK6416_AIC23_LEFTINVOL  Left line input channel volume
    0x0017, // 1 DSK6416_AIC23_RIGHTINVOL Right line input channel volume
    0x00d8, // 2 DSK6416_AIC23_LEFTHPVOL  Left channel headphone volume
    0x00d8, // 3 DSK6416_AIC23_RIGHTHPVOL Right channel headphone volume
    0x0010, // 4 DSK6416_AIC23_ANAPATH    Analog audio path control
    0x0000, // 5 DSK6416_AIC23_DIGPATH    Digital audio path control
    0x0002, // 6 DSK6416_AIC23_POWERDOWN  Power down control
    0x0043, // 7 DSK6416_AIC23_DIGIF      Digital audio interface format
    0x0000, // 8 DSK6416_AIC23_SAMPLERATE Sample rate control 48KHz para que la portadora funcione en 1.8KHz
    0x0001  // 9 DSK6416_AIC23_DIGACT     Digital interface activation
};

DSK6416_AIC23_CodecHandle hCodec;
DSK6416_init(); // Initialize the board support library, must be called first.
hCodec = DSK6416_AIC23_openCodec(0, &config); // Start the codec.
DSK6416_AIC23_setFreq(hCodec, DSK6416_AIC23_FREQ_48KHZ); // Seteo de la frecuencia de muestreo.
DSK6416_AIC23_rset(hCodec, DSK6416_AIC23_LEFTHPVOL, 0x00ca);
DSK6416_AIC23_rset(hCodec, DSK6416_AIC23_RIGHTHPVOL, 0x00ca);
```

9.1.2. Algoritmo de convolución directa

```
//************************************************************
void convol(int inicio, int fin, int flag){ //Convoluci\'{o}n para la Rama Q.
//Para la rama I es equivalente.
n=0;
for(m=inicio;m<fin;m++)
{
x=n;mp=m;
while(mp>=inicio)
{
Spointer=&sinc[mp];
if(flag==1) Qpointer=&uno[inicio+x-mp];
else Qpointer=&cero[inicio+x-mp];
Qout[n] = Qout[n] + *Qpointer * *Spointer;
mp=mp-1;
```

```
}
n=n+1;
}
}

//***********************************************************
```

9.1.3. Configuracion e Inicialización de EDMA

```
EDMA_Handle hEdma;
EDMA_Handle hEdmaReload;
short gXmtTCC;

EDMA_Config gEdmaConfig = {
    EDMA_OPT_RMK(
        EDMA_OPT_PRI_LOW,// Priority
        EDMA_OPT_ESIZE_16BIT,// Element size
        EDMA_OPT_2DS_NO,// 2 dimensional source
        EDMA_OPT_SUM_INC,// Src update mode
        EDMA_OPT_2DD_NO,// 2 dimensional dest
        EDMA_OPT_DUM_INC,// Dest update mode
        EDMA_OPT_TCINT_YES,// Cause EDMA interrupt
        EDMA_OPT_TCC_OF(0),// Transfer Complete Code
        EDMA_OPT_TCCM_DEFAULT,// Transfer Complete Code Upper Bits (c64x only)
        EDMA_OPT_ATCINT_DEFAULT,// Alternate TCC Interrupt (c64x only)
        EDMA_OPT_ATCC_DEFAULT,// Alternate Transfer Complete Code (c64x only)
        EDMA_OPT_PDTS_DEFAULT,// Peripheral Device Transfer Source (c64x only)
        EDMA_OPT_PDTD_DEFAULT,// Peripheral Device Transfer Dest (c64x only)
        EDMA_OPT_LINK_YES,// Enable link parameters
        EDMA_OPT_FS_YES // Use frame sync
    ),
    EDMA_SRC_OF(gBuf0),// src address
    EDMA_CNT_OF(BUFFSALIDA),// Count = buffer size
    EDMA_DST_OF(gBuf1),// dest address
    EDMA_IDX_OF(0),// frame/element index value
    EDMA_RLD_OF(0) // reload
};
/*--------------------------------------*/

/* ----------- initEdma ----------------*/
void initEdma(void)
```

```
{
hEdma = EDMA_open(EDMA_CHA_XEVT2, EDMA_OPEN_RESET); //abro el canal edma.
gXmtTCC = EDMA_intAlloc(-1);
gEdmaConfig.opt |= EDMA_FMK(OPT, TCC, gXmtTCC);
EDMA_config(hEdma, &gEdmaConfig);     // configuro el canal.

EDMA_intHook(gXmtTCC, edmaHwi);

EDMA_intClear(gXmtTCC); //clear any spurious interrupt
EDMA_intEnable(gXmtTCC);// enable the EDMA interrupt

hEdmaReload = EDMA_allocTable(-1);
EDMA_config(hEdmaReload, &gEdmaConfig);

EDMA_link(hEdma, hEdmaReload); //le digo donde apuntar en la prox transferencia.
EDMA_link(hEdmaReload, hEdmaReload);
}
```

9.2. Códigos Fuente

A continuación se detallan los archivos con extención .c utilizados en el DSP. En la sección Archivos en Común, se encuentran los cuatro archivos que son comunes a todas las modulaciones. Estos son edma.c, qam.c, transferencia.c y transmision.c. Estos archivos pueden considerarse como módulos, puesto que se invocan de igual forma para cada tipo de modulación, siendo totalmente ajenos a las distintas propiedades de modulación.

9.2.1. Archivos en Común

9.2.2. EDMA.C

```
/* ============== edma.c ================*/

//This is the code needed to setup the edma. The "EDMA Config" type data structure
//holds the parameters to be programmed into a EDMA channel.Register Make (RMK)
//macros build a 32-bit unsigned int; below it is used to build the Options (OPT)
//register. The OF macros provide the proper typecasting needed for the EDMA Config
//data structure.

/* -----------Include files------------ */
#include <csl.h>
```

```
#include <csl_edma.h>
#include "transferencia.h"
#include "transmision.h"
#include "dsk6416.h"
/*---------------------------------------*/

/*-------------Declaraciones------------*/
#define BUFFSIZE 26
#define BUFFSALIDA 39
#define TABLA_BIT 78
#define DATA_TABLE_SIZE 16
/*---------------------------------------*/

/*--------------Prototypes--------------*/
void initEdma(void);
void edmaHwi(int tcc);
/*---------------------------------------*/

/*---------------Variables--------------*/
short gBuf0[BUFFSALIDA];
short gBuf1[BUFFSALIDA];
extern int sample;
extern int bit1;
extern int bit2;
extern int cont;
extern int down;
int contT=0;

short Qout[TABLA_BIT];
short Iout[TABLA_BIT];

short processData[((DATA_TABLE_SIZE)/2)*TABLA_BIT];

extern int finTx;
int terminar=0;
extern int start;
/*---------------------------------------*/

/* ---------- Global Variables -------- */
EDMA_Handle hEdma;
EDMA_Handle hEdmaReload;
```

```
short gXmtTCC;

EDMA_Config gEdmaConfig = {
    EDMA_OPT_RMK(
        EDMA_OPT_PRI_LOW,// Priority
        EDMA_OPT_ESIZE_16BIT,// Element size
        EDMA_OPT_2DS_NO,// 2 dimensional source
        EDMA_OPT_SUM_INC,// Src update mode
        EDMA_OPT_2DD_NO,// 2 dimensional dest
        EDMA_OPT_DUM_INC,// Dest update mode
        EDMA_OPT_TCINT_YES,// Cause EDMA interrupt
        EDMA_OPT_TCC_OF(0),// Transfer Complete Code
        EDMA_OPT_TCCM_DEFAULT,// Transfer Complete Code Upper Bits (c64x only)
        EDMA_OPT_ATCINT_DEFAULT,// Alternate TCC Interrupt (c64x only)
        EDMA_OPT_ATCC_DEFAULT,// Alternate Transfer Complete Code (c64x only)
        EDMA_OPT_PDTS_DEFAULT,// Peripheral Device Transfer Source (c64x only)
        EDMA_OPT_PDTD_DEFAULT,// Peripheral Device Transfer Dest (c64x only)
        EDMA_OPT_LINK_YES,// Enable link parameters
        EDMA_OPT_FS_YES // Use frame sync
    ),
    EDMA_SRC_OF(gBuf0),// src address
    EDMA_CNT_OF(BUFFSALIDA),// Count = buffer size
    EDMA_DST_OF(gBuf1),// dest address
    EDMA_IDX_OF(0),// frame/element index value
    EDMA_RLD_OF(0) // reload
};
/*-------------------------------------*/
//*********************************************************
//*********************************************************
/* ----------- initEdma ----------------*/
void initEdma(void)
{
hEdma = EDMA_open(EDMA_CHA_XEVT2, EDMA_OPEN_RESET); //abro el canal edma.
gXmtTCC = EDMA_intAlloc(-1);
gEdmaConfig.opt |= EDMA_FMK(OPT, TCC, gXmtTCC);
EDMA_config(hEdma, &gEdmaConfig);      //configuro el canal.

EDMA_intHook(gXmtTCC, edmaHwi);

EDMA_intClear(gXmtTCC);  //clear any spurious interrupt
EDMA_intEnable(gXmtTCC); // enable the EDMA interrupt
```

```
hEdmaReload = EDMA_allocTable(-1);
EDMA_config(hEdmaReload, &gEdmaConfig);

EDMA_link(hEdma, hEdmaReload); //le digo donde apuntar para la prox transferencia.
EDMA_link(hEdmaReload, hEdmaReload);
}
//************************************************************
//************************************************************
void edmaHwi(int tcc) //ISR: Interrupt Service Routine
{
if (start==1){
transferir();
EDMA_setChannel(hEdma);
transmitir();
if(finTx==1) terminar=1;
}

//esto es para verlo en el osciloscopio (se repite).
if(terminar==1) { //basta con sacar este m\'{o}dulo if.
contT=0;
finTx=0;
terminar=0;
}
}
//************************************************************
//************************************************************
```

9.2.3. QAM.C

```
/* --------------------MODULADOR--------------------------*/
/* ---------DIEC, Universidad Nacional del Sur------------*/

/*----------------Includes Files----------------------*/
#include <csl.h>
#include <csl_edma.h>
#include <csl_irq.h>
#include "edma.h"
#include "modulacion.h"
#include "qamcfg.h"
#include "dsk6416.h"
```

```
#include "dsk6416_aic23.h"
#include "transmision.h"
/*-----------------------------------------------------*/

/*------------------Definiciones----------------------*/
#define BUFFSIZE 26
#define TABLA_BIT 78
/*-----------------------------------------------------*/

/* ------------------Prototipos----------------------*/
void initHwi(void);
/* -----------------------------------------------------*/

/* --------------------Variables----------------------*/
int bit1;
int bit2;
int delete;
extern int terminar;
extern short Qout[TABLA_BIT];
extern short Iout[TABLA_BIT];
/* -----------------------------------------------------*/
//*********************************************************
//*********************************************************
void main()
{
initHcodec(); // Se abre el Codec de audio.
initEdma(); // Se inicializa Enhanced Direct Memory Access.
initHwi(); // Se inicializan las interrupciones (Hardware Interrupts).

for(delete=0;delete<78;delete++){ //Pongo en cero los dos arreglos, para que no haya problema cuando
//sume los primeros 39 de Q con los segundos 39 de I, que en el 1er
Qout[delete]=0; //caso, no hay nada aun en Iout.
Iout[delete]=0;
}

modular(); //Modular

EDMA_setChannel(hEdma); //Transmitir

while(1){} // Lazo infinito, durante el cual se modula y transmite.
// while(terminar==0){} //Hasta que no termine de transmitirse todo se queda aca.
```

```
// closeHcodec(); // Se cierra el Codec de audio.
}

//*********************************************************
//*********************************************************
void initHwi(){
IRQ_enable(IRQ_EVT_EDMAINT); //enable EDMA interrupt to CPU(IER)
IRQ_globalEnable();    //turn on global interrupts(GIE)
}
//*********************************************************
//*********************************************************
```

9.2.4. TRANSFERENCIA.C

```
/* ======== transferencia.c ======== */

//Este archivo toma el arreglo processData, donde estan todos los datos modulados,
//y va transfiriendo la informacion al buffer gBuf0 de a tramos de 39 valores.

/*-----------Includes-----------*/
#include "transferencia.h"
#include <std.h>
#include "dsk6416.h"
/*------------------------------*/

/*---------Definitions----------*/
#define BUFFSIZE 26
#define DATA_TABLE_SIZE 16
#define TABLA_BIT 78
#define BUFFSALIDA 39
/*------------------------------*/

/*---------Variables------------*/
extern short gBuf0[BUFFSALIDA];
extern short processData[((DATA_TABLE_SIZE)/2)*TABLA_BIT];
extern int contT;
int tr;
int finTx=0;
/*------------------------------*/
```

```
/*----------Prototypes----------*/
void transferir();
/*------------------------------*/
//*********************************************************
//*********************************************************
void transferir(){
for (tr = 0; tr<BUFFSALIDA; tr++)
{
if(contT<((DATA_TABLE_SIZE)/2)*TABLA_BIT){
gBuf0[tr] = processData[contT]; //Voy transfiriendo bloques de 39 a gBuf0.
contT=contT+1;
}
else{
gBuf0[tr]=0;  //Se da cuando se termina processData, pero no aun gBuf0.
}
}

if(contT>=(((DATA_TABLE_SIZE)/2)*TABLA_BIT)){
finTx=1; //Ultimo tramo transferido. Con el termina la transmision.
}
}
//*********************************************************
//*********************************************************
```

9.2.5. TRANSMISION.C

```
/* ================== transmision.c ====================== */

//Este archivo se encarga de transmitir por el codec la modulacion total.
//Lo hace de a bloques, y con el buffer gBuf1, al cual se transfieren los
//datos de gBuf0.

/* --------------------Includes-----------------------*/
#include "transmision.h"
#include "dsk6416.h"
#include "dsk6416_aic23.h"
/*---------------------------------------------------*/

/* -------------------Definitions---------------------*/
#define BUFFSIZE 26
#define BUFFSALIDA 39
```

```
/*-------------------------------------------------*/

/*--------------------Variables--------------------*/
int contador;
Int16 *px;
Int16 out;
DSK6416_AIC23_CodecHandle hCodec; //Defino el Handle del codec ac\'{a} afuera, para que las otras
//funciones (transmitir y close) puedan usarla.
extern short gBuf1[BUFFSALIDA];
/*-------------------------------------------------*/

/*--------------------Prototypes-------------------*/
void initHcodec();
void transmitir();
void closeHcodec();
/*-------------------------------------------------*/

//*********************************************************
//*********************************************************
void initHcodec(){

//-----------Codec configuration settings------------
DSK6416_AIC23_Config config = {
    0x0017, // 0 DSK6416_AIC23_LEFTINVOL  Left line input channel volume
    0x0017, // 1 DSK6416_AIC23_RIGHTINVOL Right line input channel volume
    0x00d8, // 2 DSK6416_AIC23_LEFTHPVOL  Left channel headphone volume
    0x00d8, // 3 DSK6416_AIC23_RIGHTHPVOL Right channel headphone volume
    0x0010, // 4 DSK6416_AIC23_ANAPATH    Analog audio path control
    0x0000, // 5 DSK6416_AIC23_DIGPATH    Digital audio path control
    0x0002, // 6 DSK6416_AIC23_POWERDOWN  Power down control
    0x0043, // 7 DSK6416_AIC23_DIGIF      Digital audio interface format
    0x0000, // 8 DSK6416_AIC23_SAMPLERATE Sample rate control, 48KHz para que la portadora funcion
    0x0001  // 9 DSK6416_AIC23_DIGACT     Digital interface activation
};

//DSK6416_AIC23_CodecHandle hCodec;
DSK6416_init(); // Initialize the board support library, must be called first.
hCodec = DSK6416_AIC23_openCodec(0, &config); // Start the codec.
DSK6416_AIC23_setFreq(hCodec, DSK6416_AIC23_FREQ_48KHZ); // Seteo de la frecuencia de muestreo.
DSK6416_AIC23_rset(hCodec, DSK6416_AIC23_LEFTHPVOL, 0x00ca);
DSK6416_AIC23_rset(hCodec, DSK6416_AIC23_RIGHTHPVOL, 0x00ca);
```

```
}
//*********************************************************
//*********************************************************
void transmitir(){
for (contador=0;contador<BUFFSALIDA;contador++){
px = &gBuf1[contador];
// Send a sample to the left channel
while (!DSK6416_AIC23_write(hCodec, *px));
// Send a sample to the right channel
//while (!DSK6416_AIC23_write(hCodec, *px)); //No es necesario el canal derecho.
}
}
//*********************************************************
//*********************************************************
void closeHcodec(){
   DSK6416_AIC23_closeCodec(hCodec); // Close the codec
}
//*********************************************************
//*********************************************************
```

9.3. Archivos MSK

Estos archivos son los necesarios para ejecutar esta modulación, junto con los cuatro archivos antes descriptos (archivos en común).

9.3.1. MODULACION.C

```
/* ============= modulacion.c ================ */

//Este archivo se encarga de recibir los datos procesados (filtrados,
//formateados, etc), y modularlos; es decir, multiplicarlos por la
//portadora de 1.8KHz (Seno o coseno segun corresponda).

/*------------------Includes-------------------*/
#include "modulacion.h"
#include "formatear.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------------------*/
```

```
/*-----------------Definiciones----------------*/
#define BUFFSIZE 26
#define DATA_TABLE_SIZE 16
#define TABLA_BIT 78
#define BUFFSALIDA 39
//#define PROCESS_SIZE 500
/*---------------------------------------------*/

/*-----------------Variables-------------------*/
int *pa; //puntero que recorre la tabla de datos.
Int16 *pb;   //puntero que recorre una portadora.
Int16 *pc;   //puntero que recorre la otra portadora.
Int16 *pbb;     //puntero que recorre la tabla mQout.
Int16 *pcc; //puntero que recorre la tabla mIout.
extern int bit1;
extern int bit2;
int s1=0;
int s2=0;
int contQ;
int contI;
int num;
extern short gBuf0[BUFFSALIDA];
extern short gBuf1[BUFFSALIDA];

extern short mQout[TABLA_BIT];
extern short mIout[TABLA_BIT];

extern short processData[((DATA_TABLE_SIZE)/2)*TABLA_BIT+BUFFSALIDA];

int incremento=0;
int inc;
int start=0;
int sample;

int borrarAUX;
//---------------Datos de Entrada---------------
int data[DATA_TABLE_SIZE] = {
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1/*,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
```

```
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0*/
};

//------------Ciclo de Seno de 1.8KHz------------
Int16 sinetable[BUFFSIZE] = {
0,19,37,53,66,75,79,79,75,66,53,37,19,0,
-19,-37,-53,-66,-75,-79,-79,-75,-66,-53,
-37,-19
};

//-----------Ciclo de Coseno de 1.8KHz-----------
Int16 costable[BUFFSIZE] = {
    80,78,71,60,45,28,10,-10,-28,-45,-60,-71,
-78,-80,-78,-71,-60,-46,-29,-10,9,28,45,
60,71,78
};
/*---------------------------------------------*/

/*-----------------Prototypes------------------*/
void modulacion(int numFIN,int contQ,int contI);
void datos(int sample);
void modular();

/*---------------------------------------------*/

//*******************************************************
```

```
//*********************************************************
void datos(int sample){
pa = &data[sample];
bit1 = *pa;
bit2 = *(pa+1);
}

//*********************************************************
//*********************************************************
void modulacion(int numFIN,int contQ,int contI){

for(num=0;num<numFIN;num++)
{
pb=&costable[s1];
pc=&sinetable[s2];

pbb = &mQout[contQ];
pcc = &mIout[contI];

processData[incremento] =  *pb * *pcc + *pc * *pbb;// Salida Modulada   *pb * *pcc + *pc * *pbb;

s1=s1+1; if(s1==26) s1=0;
s2=s2+1; if(s2==26) s2=0;
contQ=contQ+1; contI=contI+1;

incremento = incremento + 1;
}
}
//*********************************************************
//*********************************************************
void modular(){
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
for(sample=0;sample<DATA_TABLE_SIZE;sample=sample+2)
{
datos(sample); //Busco los dos bits de la tabla de datos.
//filtrarQ(bit1);     //Filtro el bit Q primero, porque ahora hacemos OQPSK.
darFormato(bit2,1); //Doy al bit Q el formato de medio ciclo de seno. ---MSK
modulacion(39,39,0); //LLeno processData con el ciclo modulado correspondiente, con 39 valores.
//filtrarI(bit2); //Ahora filtro el bit I, que esta defasado T/2 con respecto al Q.
darFormato(bit1,0); //Doy al bit I el formato de medio ciclo de seno, y defasado 90 grados. ---MSK
modulacion(39,0,39); //LLeno processData con el ciclo modulado correspondiente, con los siguientes
```

```
}
//Esta parte es para que termine con el Ãºltimo medio ciclo del canal Q (bit2).
for(borrarAUX=0;borrarAUX<TABLA_BIT;borrarAUX++){
mIout[borrarAUX]=0;
}
modulacion(39,39,0);
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
start=1;
}
//*********************************************************
//*********************************************************
```

9.3.2. FORMATEAR.C

```
/* ======== formatear.c ======== */

/*------------Includes-------------*/
#include "formatear.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------*/

/*------------Definitions----------*/
#define TABLA_BIT 78
#define MEDIO_CICLO 78
/*---------------------------------*/

/*------------Variables------------*/
int flagQ=0; //flag=0 : primer ciclo +    flag=1 : segundo ciclo -
int flagI=0;
int s;
Int16 *mPointer;

short mQout[TABLA_BIT];
short mIout[TABLA_BIT];

short canalQ[700]; //PRUEBAAAAA
short canalI[700];
int valQ=0;
int valI=0;
int valv;
```

```
//Medio ciclo de Seno, representado por 78 muestras.
Int16 medioSeno[MEDIO_CICLO] = { //x100
0,4,8,12,16,20,24,28,32,35,39,43,46,50,53,57,60,63,66,69,72,75,77,80,82,84,87,89,90,92,
93,95,96,97,98,99,99,100,100,100,100,100,99,99,98,97,96,95,94,92,90,89,87,85,82,80,78,75,
72,69,66,63,60,57,54,50,47,43,39,36,32,28,24,20,16,12,8,4
};
/*---------------------------------*/

/* ----------Prototypes------------*/
void darFormato(int bit, int rama);
/*---------------------------------*/

//*********************************************************
//*********************************************************
void darFormato(int bit, int rama){

if (bit==0) bit=-1; //si es 1 queda como 1.

for(s=0; s<MEDIO_CICLO; s++)
{
mPointer = &medioSeno[s];

if(rama==0){ //Rama Q
if(flagQ==0){mQout[s] = bit * *mPointer;}
if(flagQ==1){mQout[s] = bit * -*mPointer;}
}
if(rama==1){ //Rama I
if(flagI==0){mIout[s] = bit * *mPointer;}
if(flagI==1){mIout[s] = bit * -*mPointer;}
}
}

if(rama==0){if(flagQ==0)flagQ=1;else flagQ=0;} //Termino un ciclo y paso al siguiente.
if(rama==1){if(flagI==0)flagI=1;else flagI=0;} //Termino un ciclo y paso al siguiente.
}
//*********************************************************
//*********************************************************
```

9.4. Archivos PCM/FM

Estos archivos son los necesarios para ejecutar esta modulación, junto con los cuatro archivos antes descriptos (archivos en común).

9.4.1. MODULACION.C

```
/* ============= modulacion.c ================ */

//Este archivo se encarga de recibir los datos procesados (filtrados,
//formateados, etc), y modularlos; es decir, multiplicarlos por la
//portadora de 1.8KHz (Seno o coseno segun corresponda).

/*------------------Includes-------------------*/
#include "modulacion.h"
#include "integrador.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------------------*/

/*-----------------Definiciones----------------*/
#define BUFFSIZE 26
#define DATA_TABLE_SIZE 16
#define TABLA_BIT 78
#define BUFFSALIDA 39
#define TABLA_SINC 468
#define CICLO_DE 222
/*---------------------------------------------*/

/*-----------------Variables-------------------*/
int *pa; //puntero que recorre la tabla de datos.
Int16 *pb;   //puntero que recorre una portadora.
Int16 *pc;   //puntero que recorre la otra portadora.
Int16 *pbb;     //puntero que recorre la tabla mQout.
Int16 *pcc; //puntero que recorre la tabla mIout.
extern int bit1;
extern int bit2;
int s1=0;
int s2=0;
int contQ;
int contI;
```

```
int num;
extern short gBuf0[BUFFSALIDA];
extern short gBuf1[BUFFSALIDA];

extern short processData[((DATA_TABLE_SIZE)/2)*TABLA_BIT];

int incremento=0;
int inc;
int start=0;
int sample;
int sample2;

extern int fertigQ;
extern int fertigI;

extern short acumulador[DATA_TABLE_SIZE*TABLA_BIT];
Int16 *pfeil;
Int16 *pfeilSEN;
Int16 *pfeilCOS;
short teilQ[DATA_TABLE_SIZE*TABLA_BIT];
short teilI[DATA_TABLE_SIZE*TABLA_BIT];
extern int alturaMuestra; // 10000/156 = 64
int flecha;

//---------------Datos de Entrada---------------
int data[DATA_TABLE_SIZE] = {
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1/*,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
```

```
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0*/
};

//------------Ciclo de Seno de 1.8KHz------------
Int16 sinetable[BUFFSIZE] = {
0,19,37,53,66,75,79,79,75,66,53,37,19,0,
-19,-37,-53,-66,-75,-79,-79,-75,-66,-53,
-37,-19
};

//-----------Ciclo de Coseno de 1.8KHz-----------
Int16 costable[BUFFSIZE] = {
    80,78,71,60,45,28,10,-10,-28,-45,-60,-71,
-78,-80,-78,-71,-60,-46,-29,-10,9,28,45,
60,71,78
};

Int16 senDE[CICLO_DE] = {
0,11,22,34,45,56,67,78,89,100,111,121,132,143,153,163,173,184,193,203,213,222,231,240,249,258,266,275,283,
291,298,306,313,319,326,332,339,344,350,355,360,365,369,374,377,381,384,387,390,392,394,396,397,399,399,
400,400,400,399,399,398,396,394,392,390,387,384,381,378,374,370,365,361,356,350,345,339,333,326,320,313,
306,299,291,283,275,267,258,250,241,232,223,213,204,194,184,174,164,154,143,133,122,111,100,90,79,68,57,
45,34,23,12,1,-11,-22,-33,-44,-55,-66,-77,-88,-99,-110,-121,-131,-142,-152,-163,-173,-183,-193,-203,-212,
-222,-231,-240,-249,-257,-266,-274,-282,-290,-298,-305,-312,-319,-326,-332,-338,-344,-350,-355,-360,-365,
-369,-373,-377,-381,-384,-387,-390,-392,-394,-396,-397,-399,-399,-400,-400,-400,-399,-399,-398,-396,-395,
-393,-390,-388,-385,-381,-378,-374,-370,-366,-361,-356,-351,-345,-339,-333,-327,-320,-313,-306,-299,-291,
-284,-276,-267,-259,-250,-241,-232,-223,-214,-204,-195,-185,-175,-164,-154,-144,-133,-123,-112,-101,-90,
-79,-68,-57,-46,-35//,-24
};

Int16 cosDE[CICLO_DE] = {
400,400,399,399,397,396,394,392,390,387,384,381,378,374,370,365,360,355,350,345,339,333,326,320,313,
306,298,291,283,275,267,258,250,241,232,222,213,203,194,184,174,164,153,143,132,122,111,100,89,78,67,
```

```
56,45,34,23,12,0,-11,-22,-33,-44,-56,-67,-78,-89,-100,-110,-121,-132,-142,-153,-163,-173,-183,-193
-212,-222,-231,-240,-249,-258,-266,-274,-283,-290,-298,-305,-312,-319,-326,-332,-338,-344,-350,-35
-365,-369,-373,-377,-381,-384,-387,-390,-392,-394,-396,-397,-399,-399,-400,-400,-400,-399,-399,-39
-394,-392,-390,-387,-385,-381,-378,-374,-370,-365,-361,-356,-350,-345,-339,-333,-327,-320,-313,-30
-291,-283,-275,-267,-259,-250,-241,-232,-223,-214,-204,-194,-184,-174,-164,-154,-143,-133,-122,-11
-90,-79,-68,-57,-46,-35,-23,-12,-1,10,21,33,44,55,66,77,88,99,110,121,131,142,152,162,173,183,193,
221,231,240,249,257,266,274,282,290,298,305,312,319,326,332,338,344,350,355,360,365,369,373,377,38
387,390,392,394,396,397,398//,399
};

/*-----------------------------------------------*/

/*-----------------Prototypes-------------------*/
void modulacion(int numFIN,int contQ,int contI);
void datos(int sample);
void modular();

/*-----------------------------------------------*/

//*********************************************************
//*********************************************************
void datos(int sample){
pa = &data[sample];
bit1 = *pa;
bit2 = *(pa+1);
}

//*********************************************************
//*********************************************************
void modulacion(int numFIN,int contQ,int contI){

for(num=0;num<numFIN;num++)
{
pb=&costable[s1];
pc=&sinetable[s2];

pbb = &teilQ[contQ];
pcc = &teilI[contI];

processData[incremento] =  *pb * *pbb + *pc * *pcc;// Salida Modulada   *pb * *pcc + *pc * *pbb;
```

```
s1=s1+1; if(s1==26) s1=0;
s2=s2+1; if(s2==26) s2=0;
contQ=contQ+1; contI=contI+1;

incremento = incremento + 1;
}
}
//************************************************************
//************************************************************
void modular(){

for(sample=0;sample<DATA_TABLE_SIZE;sample=sample+2)
{
datos(sample);
integrar(bit1);
integrar(bit2);
}
// Hier hat ,,Acumulador" geladen. Jetzt muss ich COS und SIN von dieses machen.
for(sample2=0;sample2<(DATA_TABLE_SIZE*TABLA_BIT);sample2++)
{
pfeil = &acumulador[sample2];

flecha = *pfeil/alturaMuestra;

if(flecha>=0)
{
pfeilCOS = &cosDE[flecha]; // P.ej: pfeil = 128 ---> 128/64 = 2. cosDE[2]=399
pfeilSEN = &senDE[flecha]; // P.ej: pfeil = 128 ---> 128/64 = 2. senDE[2]=22
teilQ[sample2] = *pfeilCOS;
teilI[sample2] = *pfeilSEN;
}
else
{
pfeilCOS = &cosDE[-flecha]; // P.ej: pfeil = -128 ---> -128/64 = -2. cosDE[2]=-399
pfeilSEN = &senDE[-flecha]; // P.ej: pfeil = -128 ---> -128/64 = -2. senDE[2]=-22
teilQ[sample2] = *pfeilCOS;
teilI[sample2] = -1 * *pfeilSEN;
}
}
//Bis hier habe ich COS und SIN von das Integral gemacht. Wir kÃ´nnen jetzt beide Ableitungen modulieren.
```

```
modulacion((DATA_TABLE_SIZE/2)*TABLA_BIT,0,0);

start=1; //Zum fangen der Transmission an.
}
//*********************************************************
//*********************************************************
```

9.4.2. INTEGRADOR.C

```
/* ======== integrador.c ======== */
// Se considera que la altura de cada bit es de 10000 (o -10000 si es un -1). De esta forma, la in
// de cada bit sera 5000, es decir la mitad. Cada una de las 78 muestras del bit se representa com
// 10000/156 = 64, para que la suma de todos ellos sea 5000.

/*------------Includes-------------*/
#include "integrador.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------*/

/*------------Definitions----------*/
#define TABLA_BIT 78
#define TABLA_SINC 468
#define MEDIO_CICLO 78
#define DATA_TABLE_SIZE 16
/*---------------------------------*/

/*------------Variables------------*/
short acumulador[DATA_TABLE_SIZE*TABLA_BIT];
int acu = 0;
int a;
int alturaMuestra = 64; // 10000/156 = 64
int maximo = 14208; // se da cuando se llega a 2pi. Aqui hay que volver a 0. (X.2pi.h<=2pi ---> X<
// Para h=0.7 ---> X < 1.42, y aqui este valor equivale a 14200. (1.42 * 5000 / 0.5)
// 14208 para que de exacto, porque va de a 64 valores.
int resto;

/*---------------------------------*/


/*-----------Prototypes------------*/
```

```
void integrar(int bit);
/*--------------------------------*/


//************************************************************************
//************************************************************************
void integrar(int bit)
{
for(a=0; a < TABLA_BIT; a++)
{
if(acu==0) {acumulador[acu]=alturaMuestra;}
else{
if(bit==1){acumulador[acu] = (acumulador[acu-1] + alturaMuestra);}
if(bit==0){acumulador[acu] = (acumulador[acu-1] - alturaMuestra);}
}

if((acumulador[acu] >= maximo)||((acumulador[acu] <= -maximo)))
{
if(bit==1){resto = acumulador[acu] - maximo;}
if(bit==0){resto = acumulador[acu] + maximo;}

acumulador[acu]=resto; //idealmente, resto=0.
}
acu=acu+1;
}
}
//************************************************************************
//************************************************************************
```

9.5. Archivos PCM/FM con filtro

Estos archivos son los necesarios para ejecutar esta modulación, junto con los cuatro archivos antes descriptos (archivos en común).

9.5.1. MODULACION.C

```
/* ============= modulacion.c ================ */

//Este archivo se encarga de recibir los datos procesados (filtrados,
//formateados, etc), y modularlos; es decir, multiplicarlos por la
```

```
//portadora de 1.8KHz (Seno o coseno segun corresponda).

/*------------------Includes-------------------*/
#include "modulacion.h"
#include "filtrado.h"
#include "integrador.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------------------*/

/*-----------------Definiciones----------------*/
#define BUFFSIZE 26
#define DATA_TABLE_SIZE 16
#define TABLA_BIT 78
#define BUFFSALIDA 39
#define TABLA_SINC 468
#define CICLO_DE 355
/*---------------------------------------------*/

/*-----------------Variables-------------------*/
int *pa; //puntero que recorre la tabla de datos.
Int16 *pb;   //puntero que recorre una portadora.
Int16 *pc;   //puntero que recorre la otra portadora.
Int16 *pbb;     //puntero que recorre la tabla mQout.
Int16 *pcc; //puntero que recorre la tabla mIout.
extern int bit1;
extern int bit2;
int s1=0;
int s2=0;
int contQ;
int contI;
int num;
extern short gBuf0[BUFFSALIDA];
extern short gBuf1[BUFFSALIDA];

extern short processData[((DATA_TABLE_SIZE)/2)*TABLA_BIT];

int incremento=0;
int inc;
int start=0;
int sample0;
```

```
int sample;
int sample2;

extern int fertigQ;
extern int fertigI;

extern short acumulador[DATA_TABLE_SIZE*TABLA_BIT];
Int16 *pfeil;
Int16 *pfeilSEN;
Int16 *pfeilCOS;
short teilQ[DATA_TABLE_SIZE*TABLA_BIT];
short teilI[DATA_TABLE_SIZE*TABLA_BIT];
int flecha;
int gefunden=0;
int min=0;
int position;

//---------------Datos de Entrada---------------
int data[DATA_TABLE_SIZE] = {
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1/*,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,
1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
```

```
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
};

//------------Ciclo de Seno de 1.8KHz------------
Int16 sinetable[BUFFSIZE] = {
0,19,37,53,66,75,79,79,75,66,53,37,19,0,
-19,-37,-53,-66,-75,-79,-79,-75,-66,-53,
-37,-19
};

//-----------Ciclo de Coseno de 1.8KHz-----------
Int16 costable[BUFFSIZE] = {
    80,78,71,60,45,28,10,-10,-28,-45,-60,-71,
-78,-80,-78,-71,-60,-46,-29,-10,9,28,45,
60,71,78
};
//Ciclos de Sen y Cos para la seÃ±al integrada.
Int16 senDE[CICLO_DE] = {
0,7,14,21,28,35,42,49,56,63,70,77,84,91,98,105,112,118,125,132,139,145,152,158,165,171,178,
184,190,196,202,209,215,220,226,232,238,244,249,255,260,265,271,276,281,286,291,296,300,305,
309,314,318,322,327,331,335,338,342,346,349,353,356,359,362,365,368,371,373,376,378,380,382,
384,386,388,390,391,393,394,395,396,397,398,399,399,400,400,400,400,400,400,399,399,398,398,
397,396,395,393,392,391,389,387,386,384,382,379,377,375,372,370,367,364,361,358,355,351,348,
344,341,337,333,329,325,321,316,312,308,303,298,294,289,284,279,274,268,263,258,252,247,241,
236,230,224,218,212,206,200,194,188,181,175,169,162,156,149,142,136,129,122,116,109,102,95,
88,81,74,68,61,54,47,39,32,25,18,11,4,-3,-10,-17,-24,-31,-38,-45,-52,-59,-66,-73,-80,-87,-94,
-101,-108,-114,-121,-128,-135,-141,-148,-154,-161,-167,-174,-180,-186,-193,-199,-205,-211,
-217,-223,-229,-234,-240,-246,-251,-257,-262,-268,-273,-278,-283,-288,-293,-298,-302,-307,
-311,-316,-320,-324,-328,-332,-336,-340,-344,-347,-351,-354,-357,-360,-363,-366,-369,-372,-374,
-377,-379,-381,-383,-385,-387,-389,-390,-392,-393,-394,-396,-397,-397,-398,-399,-399,-400,
-400,-400,-400,-400,-400,-399,-399,-398,-397,-396,-395,-394,-393,-392,-390,-388,-387,-385,
-383,-381,-379,-376,-374,-371,-368,-366,-363,-360,-356,-353,-350,-346,-343,-339,-335,-331,
-327,-323,-319,-315,-310,-306,-301,-296,-292,-287,-282,-277,-272,-266,-261,-256,-250,-245,
-239,-233,-227,-222,-216,-210,-204,-197,-191,-185,-179,-172,-166,-159,-153,-146,-140,-133,
-126,-120,-113,-106,-99,-92,-86,-79,-72,-65,-58,-51,-44,-37,-30,-22,-15,-8
};

Int16 cosDE[CICLO_DE] = {
400,400,400,399,399,398,398,397,396,395,394,392,391,389,388,386,384,382,380,378,375,373,370,
367,364,362,358,355,352,349,345,341,338,334,330,326,322,317,313,309,304,299,295,290,285,280,
```

```
275,270,264,259,253,248,242,237,231,225,219,213,207,201,195,189,183,176,170,163,157,150,144,
137,131,124,117,110,103,97,90,83,76,69,62,55,48,41,34,27,20,13,6,-1,-9,-16,-23,-30,-37,-44,
-51,-58,-65,-72,-79,-86,-93,-99,-106,-113,-120,-127,-133,-140,-147,-153,-160,-166,-173,-179,
-185,-191,-198,-204,-210,-216,-222,-228,-233,-239,-245,-250,-256,-261,-266,-272,-277,-282,
-287,-292,-297,-301,-306,-310,-315,-319,-323,-327,-331,-335,-339,-343,-346,-350,-353,-357,
-360,-363,-366,-368,-371,-374,-376,-379,-381,-383,-385,-387,-388,-390,-392,-393,-394,-395,
-396,-397,-398,-399,-399,-400,-400,-400,-400,-400,-400,-399,-399,-398,-397,-397,-396,-394,
-393,-392,-390,-389,-387,-385,-383,-381,-379,-377,-374,-372,-369,-366,-363,-360,-357,-354,
-351,-347,-344,-340,-336,-332,-328,-324,-320,-316,-311,-307,-302,-297,-293,-288,-283,-278,
-273,-267,-262,-257,-251,-246,-240,-234,-229,-223,-217,-211,-205,-199,-193,-186,-180,-174,
-167,-161,-154,-148,-141,-134,-128,-121,-114,-108,-101,-94,-87,-80,-73,-66,-59,-52,-45,-38,
-31,-24,-17,-10,-3,4,11,18,26,33,40,47,54,61,68,75,82,89,95,102,109,116,123,129,136,143,149,
156,162,169,175,181,188,194,200,206,212,218,224,230,236,241,247,252,258,263,269,274,279,284,
289,294,298,303,308,312,317,321,325,329,333,337,341,344,348,351,355,358,361,364,367,370,372,
375,377,379,382,384,386,387,389,391,392,394,395,396,397,398,398,399,399,400,400

};

/*---------------------------------------------*/

/*-----------------Prototypes-------------------*/
void modulacion(int numFIN,int contQ,int contI);
void datos(int sample);
void modular();

/*---------------------------------------------*/

//*********************************************************
//*********************************************************
void datos(int sample){
pa = &data[sample];
bit1 = *pa;
bit2 = *(pa+1);
}

//*********************************************************
//*********************************************************
void modulacion(int numFIN,int contQ,int contI){

for(num=0;num<numFIN;num++)
{
```

```
pb=&costable[s1];
pc=&sinetable[s2];

pbb = &teilQ[contQ];
pcc = &teilI[contI];

processData[incremento] =  *pb * *pbb + *pc * *pcc;// Salida Modulada   *pb * *pcc + *pc * *pbb;

s1=s1+1; if(s1==26) s1=0;
s2=s2+1; if(s2==26) s2=0;
contQ=contQ+1; contI=contI+1;

incremento = incremento + 1;
}
}
//*********************************************************
//*********************************************************
void modular(){

for(sample0=0;sample0<DATA_TABLE_SIZE;sample0=sample0+2)
{
datos(sample0);
filtrar(bit1);
filtrar(bit2);
}
// Hier sind die Daten in ,,filtradoCompleto" gefiltert.
for(sample=0;sample<DATA_TABLE_SIZE;sample++)
{
integrar();
}
// Hier hat ,,Acumulador" geladen. Jetzt muss ich COS und SIN von dieses machen.
for(sample2=0;sample2<(DATA_TABLE_SIZE*TABLA_BIT);sample2++)
{
pfeil = &acumulador[sample2];

if(*pfeil>=0){flecha=*pfeil;} //Ich nemme ,,flecha" nur als Positiv.
else{flecha=-*pfeil;}

gefunden=0; // Ich rÃ»cksetze diese Daten.
min=0;
position=0;
```

```
while(gefunden==0)
{
if((flecha>=min)&&(flecha<=min+39)){gefunden=1;}
position=position+1; //Position kann sein von 0 bis 355. Diese nummer gehÃ´rt der Tabelle cosDE und senDE.
min=min+40; //HÃ´chstens wird ,,position" 355 erreichen.
}

if(position==355){position=354;} //weil das Sample 355 nicht existiert.

if(*pfeil>=0)
{
pfeilCOS = &cosDE[position]; // P.ej: pfeil = 128 --->cosDE[3]=... 128 corresponde al tramo 4 (de 120 a 139)
pfeilSEN = &senDE[position]; // P.ej: pfeil = 128 --->senDE[3]=...
teilQ[sample2] = *pfeilCOS;
teilI[sample2] = *pfeilSEN;
}
else
{
pfeilCOS = &cosDE[position]; // P.ej: pfeil = -128 ---> Cos(-128)=Cos(128). cosDE[3]=-...
pfeilSEN = &senDE[position]; // P.ej: pfeil = -128 ---> Sen(-128)=-Sen(128). senDE[3]=-...
teilQ[sample2] = *pfeilCOS;
teilI[sample2] = -1 * *pfeilSEN;
}
}
//Bis hier habe ich COS und SIN von das Integral gemacht. Wir kÃ´nnen jetzt beide Ableitungen modulieren.

modulacion((DATA_TABLE_SIZE/2)*TABLA_BIT,0,0);

start=1; //Zum fangen der Transmission an.
}
//**********************************************************
//**********************************************************
```

9.5.2. INTEGRADOR.C

```
/* ======== integrador.c ======== */
// Se considera que la altura de cada bit es de 10000 (o -10000 si es un -1). De esta forma, la integral
// de cada bit sera 5000, es decir la mitad. Cada una de las 78 muestras del bit se representa como
// 10000/156 = 64, para que la suma de todos ellos sea 5000.

/*------------Includes-------------*/
```

```
#include "integrador.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------*/

/*------------Definitions----------*/
#define TABLA_BIT 78
#define TABLA_SINC 468
#define MEDIO_CICLO 78
#define DATA_TABLE_SIZE 16
/*---------------------------------*/

/*------------Variables------------*/
short acumulador[DATA_TABLE_SIZE*TABLA_BIT];
int acu = 0;
int a;
int aa=0;
//int alturaMuestra = 64; // 10000/156 = 64
Int16 alturaMuestra; //Ahora la altura de la muestra depende del raised Cosine.
int maximo = 14200; // se da cuando se llega a 2pi. Aqui hay que volver a 0. (X.2pi.h<=2pi ---> X<
// Para h=0.7 ---> X < 1.42, y aqui este valor equivale a 14200. (1.42 * 5000 / 0.5)
int resto;
Int16 *pfeilAUX;  //Apunta a la tabla de valores filtrados.
extern short filtradoCompleto[DATA_TABLE_SIZE*TABLA_BIT];

/*---------------------------------*/


/*-----------Prototypes------------*/
void integrar();
/*---------------------------------*/


//*************************************************************************
//*************************************************************************
void integrar()
{
for(a=aa; a < aa+TABLA_BIT; a++)
{
pfeilAUX = &filtradoCompleto[a];
alturaMuestra = *pfeilAUX;
```

```
if(acu==0) {acumulador[acu]=alturaMuestra;}
else
{
acumulador[acu] = acumulador[acu-1] + alturaMuestra; //alturaMuestra puede ser positivo o negativo.
}

if((acumulador[acu] >= maximo)||((acumulador[acu] <= -maximo)))
{
if(acumulador[acu] >= maximo){resto = acumulador[acu] - maximo;}
if(acumulador[acu] <= maximo){resto = acumulador[acu] + maximo;}

acumulador[acu]=resto; //idealmente, resto=0.
}
acu=acu+1;
}
aa=aa+78;
}
//************************************************************************
//************************************************************************
```

9.5.3. FILTRADO.C

```
/* ======== filtrado.c ======== */
// Este filtro usa una respuesta impulsiva FIR, la cual se trunca en +- 3T, y se hace la convolucion complet
// Este mÃ©todo tiene muchos mÃ¡s cÃ¡lculos que con el metodo de tablas de QAMv4.4, pero a este le vamos a a
// un filtrado multirate, lo que va a mejorar el desempeÃ±o, aun haciendo muchas multiplicaciones por 0.

/*------------Includes-------------*/
#include "filtrado.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------*/

/*------------Definitions----------*/
#define TABLA_BIT 78
#define TABLA_SINC 468
#define MEDIO_CICLO 78
#define DATA_TABLE_SIZE 16
/*---------------------------------*/
```

```
/*------------Variables------------*/
short filtradoCompleto[DATA_TABLE_SIZE*TABLA_BIT];
int v;
int vv=0;
Int16 *pfeilR;

//Pulso coseno elevado de un tiempo de bit (78 muestras)
Int16 rCosine[TABLA_BIT] = {  // Escalado a 115 para que la suma (la integral) de 5000.
0,2,5,8,11,14,17,20,24,27,31,35,39,43,47,51,55,59,63,67,71,75,79,82,86,89,
93,96,99,101,104,106,108,110,111,113,114,114,115,115,115,114,114,113,111,
110,108,106,104,101,99,96,93,89,86,82,79,75,71,67,63,59,55,51,47,43,39,35,
31,27,24,20,17,14,11,8,5,2
};
/*---------------------------------*/

/* ----------Prototypes------------*/
void filtrar(int bit);
/*---------------------------------*/

//*********************************************************
//*********************************************************
void filtrar(int bit){

for(v=0;v<TABLA_BIT;v++)
{
pfeilR = &rCosine[v];

if(bit==1){filtradoCompleto[vv] = *pfeilR;}
if(bit==0){filtradoCompleto[vv] = -*pfeilR;}

vv=vv+1;
}
}

// Aus diesem Fall ist ,,filtradoCompleto" ganz voll und dann kann ich dieses eine Integration mac

//*********************************************************
//*********************************************************
```

9.6. Archivos SOQPSK-TG (FSK)

Estos archivos son los necesarios para ejecutar esta modulación, junto con los cuatro archivos antes descriptos (archivos en común).

9.6.1. MODULACION.C

```
/* ============= modulacion.c ================ */

//Este archivo se encarga de recibir los datos procesados (filtrados,
//formateados, etc), y modularlos; es decir, multiplicarlos por la
//portadora de 1.8KHz (Seno o coseno segun corresponda).

/*------------------Includes-------------------*/
#include "modulacion.h"
#include "filtrado.h"
#include "integrador.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------------------*/

/*-----------------Definiciones----------------*/
#define BUFFSIZE 26
#define DATA_TABLE_SIZE 16
#define TABLA_BIT 78
#define BUFFSALIDA 39

#define CICLO_DE 355
/*---------------------------------------------*/

/*-----------------Variables-------------------*/
int *pa; //puntero que recorre la tabla de datos.
Int16 *pb;   //puntero que recorre una portadora.
Int16 *pc;   //puntero que recorre la otra portadora.
Int16 *pbb;   //puntero que recorre la tabla mQout.
Int16 *pcc; //puntero que recorre la tabla mIout.
extern int bit1;
extern int bit2;
int s1=0;
int s2=0;
int contQ;
```

```
int contI;
int num;
extern short gBuf0[BUFFSALIDA];
extern short gBuf1[BUFFSALIDA];

extern short processData[(DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT];

int incremento=0;
int inc;
int start=0;
int sample0;
int sample;
int sample2;

extern int fertigQ;
extern int fertigI;

extern short acumulador[(DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT];
Int16 *pfeil;
Int16 *pfeilSEN;
Int16 *pfeilCOS;
short teilQ[(DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT];
short teilI[(DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT];
int flecha;
int gefunden=0;
int min=0;
int position;
extern int contFinFiltro;

//---------------Datos de Entrada---------------
int data[DATA_TABLE_SIZE] = {
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1/*,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,1,0,1,
```

```
*/
};

//------------Ciclo de Seno de 1.8KHz------------
Int16 sinetable[BUFFSIZE] = {
0,19,37,53,66,75,79,79,75,66,53,37,19,0,
-19,-37,-53,-66,-75,-79,-79,-75,-66,-53,
-37,-19
};

//-----------Ciclo de Coseno de 1.8KHz-----------
Int16 costable[BUFFSIZE] = {
    80,78,71,60,45,28,10,-10,-28,-45,-60,-71,
-78,-80,-78,-71,-60,-46,-29,-10,9,28,45,
60,71,78
};
//Ciclos de Sen y Cos para la seÃ±al integrada.
Int16 senDE[CICLO_DE] = {
0,7,14,21,28,35,42,49,56,63,70,77,84,91,98,105,112,118,125,132,139,145,152,158,165,171,178,
184,190,196,202,209,215,220,226,232,238,244,249,255,260,265,271,276,281,286,291,296,300,305,
309,314,318,322,327,331,335,338,342,346,349,353,356,359,362,365,368,371,373,376,378,380,382,
384,386,388,390,391,393,394,395,396,397,398,399,399,400,400,400,400,400,400,399,399,398,398,
397,396,395,393,392,391,389,387,386,384,382,379,377,375,372,370,367,364,361,358,355,351,348,
344,341,337,333,329,325,321,316,312,308,303,298,294,289,284,279,274,268,263,258,252,247,241,
236,230,224,218,212,206,200,194,188,181,175,169,162,156,149,142,136,129,122,116,109,102,95,
88,81,74,68,61,54,47,39,32,25,18,11,4,-3,-10,-17,-24,-31,-38,-45,-52,-59,-66,-73,-80,-87,-94,
-101,-108,-114,-121,-128,-135,-141,-148,-154,-161,-167,-174,-180,-186,-193,-199,-205,-211,
-217,-223,-229,-234,-240,-246,-251,-257,-262,-268,-273,-278,-283,-288,-293,-298,-302,-307,
-311,-316,-320,-324,-328,-332,-336,-340,-344,-347,-351,-354,-357,-360,-363,-366,-369,-372,-374,
-377,-379,-381,-383,-385,-387,-389,-390,-392,-393,-394,-396,-397,-397,-398,-399,-399,-400,
-400,-400,-400,-400,-400,-399,-399,-398,-397,-396,-395,-394,-393,-392,-390,-388,-387,-385,
-383,-381,-379,-376,-374,-371,-368,-366,-363,-360,-356,-353,-350,-346,-343,-339,-335,-331,
-327,-323,-319,-315,-310,-306,-301,-296,-292,-287,-282,-277,-272,-266,-261,-256,-250,-245,
-239,-233,-227,-222,-216,-210,-204,-197,-191,-185,-179,-172,-166,-159,-153,-146,-140,-133,
-126,-120,-113,-106,-99,-92,-86,-79,-72,-65,-58,-51,-44,-37,-30,-22,-15,-8
};

Int16 cosDE[CICLO_DE] = {
400,400,400,399,399,398,398,397,396,395,394,392,391,389,388,386,384,382,380,378,375,373,370,
367,364,362,358,355,352,349,345,341,338,334,330,326,322,317,313,309,304,299,295,290,285,280,
275,270,264,259,253,248,242,237,231,225,219,213,207,201,195,189,183,176,170,163,157,150,144,
```

```
137,131,124,117,110,103,97,90,83,76,69,62,55,48,41,34,27,20,13,6,-1,-9,-16,-23,-30,-37,-44,
-51,-58,-65,-72,-79,-86,-93,-99,-106,-113,-120,-127,-133,-140,-147,-153,-160,-166,-173,-179,
-185,-191,-198,-204,-210,-216,-222,-228,-233,-239,-245,-250,-256,-261,-266,-272,-277,-282,
-287,-292,-297,-301,-306,-310,-315,-319,-323,-327,-331,-335,-339,-343,-346,-350,-353,-357,
-360,-363,-366,-368,-371,-374,-376,-379,-381,-383,-385,-387,-388,-390,-392,-393,-394,-395,
-396,-397,-398,-399,-399,-400,-400,-400,-400,-400,-400,-399,-399,-398,-397,-397,-396,-394,
-393,-392,-390,-389,-387,-385,-383,-381,-379,-377,-374,-372,-369,-366,-363,-360,-357,-354,
-351,-347,-344,-340,-336,-332,-328,-324,-320,-316,-311,-307,-302,-297,-293,-288,-283,-278,
-273,-267,-262,-257,-251,-246,-240,-234,-229,-223,-217,-211,-205,-199,-193,-186,-180,-174,
-167,-161,-154,-148,-141,-134,-128,-121,-114,-108,-101,-94,-87,-80,-73,-66,-59,-52,-45,-38,
-31,-24,-17,-10,-3,4,11,18,26,33,40,47,54,61,68,75,82,89,95,102,109,116,123,129,136,143,149,
156,162,169,175,181,188,194,200,206,212,218,224,230,236,241,247,252,258,263,269,274,279,284,
289,294,298,303,308,312,317,321,325,329,333,337,341,344,348,351,355,358,361,364,367,370,372,
375,377,379,382,384,386,387,389,391,392,394,395,396,397,398,398,399,399,400,400
};

/*-----------------------------------------------*/

/*-----------------Prototypes-------------------*/
void modulacion(int numFIN,int contQ,int contI);
void datos(int sample);
void modular();

/*-----------------------------------------------*/

//*********************************************************
//*********************************************************
void datos(int sample){
pa = &data[sample];
bit1 = *pa;
bit2 = *(pa+1);
}

//*********************************************************
//*********************************************************
void modulacion(int numFIN,int contQ,int contI){

for(num=0;num<numFIN;num++)
{
pb=&costable[s1];
pc=&sinetable[s2];
```

```
pbb = &teilQ[contQ];
pcc = &teilI[contI];

processData[incremento] =  *pb * *pbb + *pc * *pcc;// Salida Modulada   *pb * *pcc + *pc * *pbb;

s1=s1+1; if(s1==26) s1=0;
s2=s2+1; if(s2==26) s2=0;
contQ=contQ+1; contI=contI+1;

incremento = incremento + 1;
}
}
//*********************************************************
//*********************************************************
void modular(){

for(sample0=0;sample0<DATA_TABLE_SIZE;sample0=sample0+2)
{
datos(sample0);
filtrar(bit1);
filtrar(bit2);
}
terminarFiltrado();
// Hasta aqui estan los datos filtrados.
for(sample=0;sample<(DATA_TABLE_SIZE+contFinFiltro);sample++)
{
integrar();
}
// Hasta aqui se cargo acumulador.
for(sample2=0;sample2<((DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT);sample2++)
{
pfeil = &acumulador[sample2];

if(*pfeil>=0){flecha=*pfeil;} //Tomo a flecha solo como positivo.
else{flecha=-*pfeil;}

gefunden=0; // Ich rÃ»cksetze diese Daten.
min=0;
position=0;
while(gefunden==0)
```

```
{
if((flecha>=min)&&(flecha<=min+39)){gefunden=1;}
position=position+1; //Position va de 0 a 500 (valores de la tabla).
min=min+40; //El valor m\'{a}ximo de position es 500.
}

if(position==355){position=354;} //weil das Sample 355 nicht existiert.

if(*pfeil>=0)
{
pfeilCOS = &cosDE[position]; // P.ej: pfeil = 128 --->cosDE[3]=... 128 corresponde al tramo 4 (de
pfeilSEN = &senDE[position]; // P.ej: pfeil = 128 --->senDE[3]=...
teilQ[sample2] = *pfeilCOS;
teilI[sample2] = *pfeilSEN;
}
else
{
pfeilCOS = &cosDE[position]; // P.ej: pfeil = -128 ---> Cos(-128)=Cos(128). cosDE[3]=-...
pfeilSEN = &senDE[position]; // P.ej: pfeil = -128 ---> Sen(-128)=-Sen(128). senDE[3]=-...
teilQ[sample2] = *pfeilCOS;
teilI[sample2] = -1 * *pfeilSEN;
}
}
//Hasta aqui hice Cos y Sen de la se\~{a}l integrada.

modulacion((DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT,0,0);

start=1; //Zum fangen der Transmission an.
}
//*********************************************************
//*********************************************************
```

9.6.2. INTEGRADOR.C

```
/* ======== integrador.c ======== */
// Se considera que la altura de cada bit es de 10000 (o -10000 si es un -1). De esta forma, la in
// de cada bit sera 5000, es decir la mitad. Cada una de las 78 muestras del bit se representa com
// 10000/156 = 64, para que la suma de todos ellos sea 5000.

/*------------Includes-------------*/
#include "integrador.h"
```

```
#include <std.h>
#include "dsk6416.h"
/*---------------------------------*/

/*------------Definitions----------*/
#define TABLA_BIT 78
#define TABLA_SINC 78
#define MEDIO_CICLO 78
#define DATA_TABLE_SIZE 16
/*---------------------------------*/

/*------------Variables------------*/
short acumulador[(DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT];
int acu = 0;
int a=0;
int aa;

Int16 alturaMuestra; //Ahora la altura de la muestra depende del raised Cosine.
int maximo = 14200; // se da cuando se llega a 2pi. Aqui hay que volver a 0. (X.2pi.h<=2pi ---> X<1/h)
// Para h=0.7 ---> X < 1.42, y aqui este valor equivale a 14200. (2 * 5000 / 0.7)
int resto;
Int16 *pfeilAUX;  //Apunta a la tabla de valores filtrados.
extern short filtradoCompleto[(DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT];

/*---------------------------------*/


/*-----------Prototypes------------*/
void integrar();
/*---------------------------------*/


//*************************************************************************
//*************************************************************************
void integrar()
{
for(aa=a; aa < a+TABLA_BIT; aa++)
{
pfeilAUX = &filtradoCompleto[aa];
alturaMuestra = *pfeilAUX;
```

```
if(acu==0) {acumulador[acu]=alturaMuestra;}
else
{
acumulador[acu] = acumulador[acu-1] + alturaMuestra; //alturaMuestra puede ser positivo o negativo
}

if((acumulador[acu] >= maximo)||((acumulador[acu] <= -maximo)))
{
if(acumulador[acu] >= maximo){resto = acumulador[acu] - maximo;}
if(acumulador[acu] <= maximo){resto = acumulador[acu] + maximo;}

acumulador[acu]=resto; //idealmente, resto=0.
}
acu=acu+1;
}
a=a+78;
}
//*************************************************************************
//*************************************************************************
```

9.6.3. FILTRADO.C

```
/* ======== filtrado.c ======== */
// Este filtro usa una respuesta impulsiva FIR, la cual se trunca en +- 3T, y se hace la convoluci
// Este mÃ©todo tiene muchos mÃ¡s cÃ¡lculos que con el metodo de tablas de QAMv4.4, pero a este l
// un filtrado multirate, lo que va a mejorar el desempeÃ±o, aun haciendo muchas multiplicaciones p

/*------------Includes------------*/
#include "filtrado.h"
#include <std.h>
#include "dsk6416.h"
/*--------------------------------*/

/*------------Definitions----------*/
#define TABLA_BIT 78
#define TABLA_SINC 78
#define MEDIO_CICLO 78
#define DATA_TABLE_SIZE 16
/*--------------------------------*/

/*------------Variables------------*/
```

```
short filtradoCompleto[(DATA_TABLE_SIZE*TABLA_BIT)+TABLA_BIT];
int v;
int vv=0;
Int16 *pfeilQ;

int tramo1=1; //estas se van modificando segun que tramo se este calculando.
int tramo2=0;
int tramo3=0;
int tramo4=0;
int comienzo1=0; //estas se modifican una vez, y nunca mas.
int comienzo2=0;
int comienzo3=0;

extern int bit1;
extern int bit2;
int flag11;
int flag12;
int flag21;
int flag22;
int flag31;
int flag32;
int flag41;
int flag42;

int i;
Int16 *Qpointer;
Int16 *Ipointer;
extern short Qout[TABLA_SINC];
extern short Iout[TABLA_SINC];
int finFiltro=0;
int contFinFiltro=0;

/*---------------------------------*/

/* --------Prototypes--------------*/
void filtrar(int bit);
void terminarFiltrado();
/*---------------------------------*/

Int16 tabla1[TABLA_SINC] = { //x64 para que la suma de todo un ciclo de 5000=0.5 (integral). Roll Off=0.7
0,0,1,1,2,3,3,4,4,5,6,7,7,8,9,10,10,11,12,13,14,15,16,17,18,19,20,21,
```

```
22,23,24,25,26,27,28,29,30,31,32,33,34,36,37,38,39,40,41,42,43,44,45,
45,46,47,48,49,50,50,51,52,53,53,54,55,55,56,56,57,57,57,58,58,58,58,
59,59,59,59
};

Int16 tabla2[TABLA_SINC] = {
59,59,59,59,59,58,58,58,58,57,57,57,56,56,55,55,54,53,53,52,51,50,50,
49,48,47,46,45,45,44,43,42,41,40,39,38,37,36,34,33,32,31,30,29,28,27,
26,25,24,23,22,21,20,19,18,17,16,15,14,13,12,11,10,10,9,8,7,7,6,5,4,
4,3,3,2,1,1,0
};

Int16 tabla1N[TABLA_SINC] = {
0,0,-1,-1,-2,-3,-3,-4,-4,-5,-6,-7,-7,-8,-9,-10,-10,-11,-12,-13,-14,-15,
-16,-17,-18,-19,-20,-21,-22,-23,-24,-25,-26,-27,-28,-29,-30,-31,-32,-33,
-34,-36,-37,-38,-39,-40,-41,-42,-43,-44,-45,-45,-46,-47,-48,-49,-50,-50,
-51,-52,-53,-53,-54,-55,-55,-56,-56,-57,-57,-57,-58,-58,-58,-58,-59,-59,
-59,-59
};

Int16 tabla2N[TABLA_SINC] = {
-59,-59,-59,-59,-59,-58,-58,-58,-58,-57,-57,-57,-56,-56,-55,-55,-54,-53,
-53,-52,-51,-50,-50,-49,-48,-47,-46,-45,-45,-44,-43,-42,-41,-40,-39,-38,
-37,-36,-34,-33,-32,-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,
-18,-17,-16,-15,-14,-13,-12,-11,-10,-10,-9,-8,-7,-7,-6,-5,-4,-4,-3,-3,-2,
-1,-1,0
};


void filtrar(int bit){

if(tramo1!=0) // <-----------------------------------
{
if(bit==1) flag11=1;//flag:positivo
else flag11=0;//flag:negativo

for(i=0;i<TABLA_SINC;i++)
{
if(flag11==1) Qpointer=&tabla1[i];
else Qpointer=&(tabla1N[i]);
Qout[i]=*Qpointer;
```

```
if(comienzo1!=0)
{
if(flag21==1) Qpointer=&tabla2[i];
else Qpointer=&tabla2N[i];
Qout[i]=Qout[i]+*Qpointer;
}
}
comienzo1=1;
}

if(tramo2!=0) // <---------------------------------
{
if(bit==1) flag21=1;//flag:positivo
else flag21=0;//flag:negativo

for(i=0;i<TABLA_SINC;i++)
{
if(flag11==1) Qpointer=&tabla2[i];
else Qpointer=&tabla2N[i];
Qout[i]=*Qpointer;

if(flag21==1) Qpointer=&tabla1[i];
else Qpointer=&tabla1N[i];
Qout[i]=Qout[i]+*Qpointer;
}
}

if(tramo1==1){tramo1=1;tramo2=1;}
if((tramo2==1)&&(tramo1==0)){tramo1=1;tramo2=0;}

if((tramo1==1)&&(tramo2==1)){tramo1=0;tramo2=1;}


for(v=0; v<TABLA_BIT; v++)
{
pfeilQ = &Qout[v];
filtradoCompleto[vv] = *pfeilQ;
vv=vv+1;
}
```

```
}

void terminarFiltrado(){

if(tramo1==1){flag11=3;}

while(finFiltro==0)
{
if(tramo1!=0) // <-----------------------------------
{

for(i=0;i<TABLA_SINC;i++)
{
if(flag11==1) Qpointer=&tabla1[i];
else Qpointer=&(tabla1N[i]);
Qout[i]=*Qpointer;

if(flag21==1) Qpointer=&tabla2[i];
else Qpointer=&tabla2N[i];
Qout[i]=Qout[i]+*Qpointer;
}

flag21=3;
}

if(tramo2!=0) // <---------------------------------
{

for(i=0;i<TABLA_SINC;i++)
{
if(flag11==1) Qpointer=&tabla2[i];
else Qpointer=&tabla2N[i];
Qout[i]=*Qpointer;

if(flag21==1) Qpointer=&tabla1[i];
else Qpointer=&tabla1N[i];
Qout[i]=Qout[i]+*Qpointer;
}
flag11=3;
}
```

```
if(tramo1==1){tramo1=1;tramo2=1;}
if((tramo2==1)&&(tramo1==0)){tramo1=1;tramo2=0;}

if((tramo1==1)&&(tramo2==1)){tramo1=0;tramo2=1;}


for(v=0; v<TABLA_BIT; v++)
{
pfeilQ = &Qout[v];
filtradoCompleto[vv] = *pfeilQ;
vv=vv+1;
}

contFinFiltro=contFinFiltro+1;
if((flag11==3)&&(flag21==3)){finFiltro=1;}
}
}
```

9.7. Archivos SOQPSK-TG (QAM)

Estos archivos son los necesarios para ejecutar esta modulación, junto con los cuatro archivos antes descriptos (archivos en común).

9.7.1. MODULACION.C

```
/* ============= modulacion.c ================ */

//Este archivo se encarga de recibir los datos procesados (filtrados,
//formateados, etc), y modularlos; es decir, multiplicarlos por la
//portadora de 1.8KHz (Seno o coseno segun corresponda).

/*------------------Includes-------------------*/
#include "modulacion.h"
#include <std.h>
#include "dsk6416.h"
/*---------------------------------------------*/

/*-----------------Definiciones----------------*/
#define BUFFSIZE 26
#define DATA_TABLE_SIZE 16
```

```
#define TABLA_BIT 78
#define BUFFSALIDA 39
//#define PROCESS_SIZE 500
/*---------------------------------------------*/

/*-----------------Variables-------------------*/
int *pa; //puntero que recorre la tabla de datos.
Int16 *pb;   //puntero que recorre una portadora.
Int16 *pc;   //puntero que recorre la otra portadora.
Int16 *pbb;   //puntero que recorre la tabla mQout.
Int16 *pcc; //puntero que recorre la tabla mIout.
extern int bit1;
extern int bit2;
int s1=0;
int s2=0;
int contQ;
int contI;
int num;
extern short gBuf0[BUFFSALIDA];
extern short gBuf1[BUFFSALIDA];

extern short Qout[TABLA_BIT];
extern short Iout[TABLA_BIT];

extern short processData[((DATA_TABLE_SIZE)/2)*TABLA_BIT+BUFFSIZE];

int incremento=0;
int inc;
int start=0;
int sample;
//---------------Datos de Entrada---------------
int data[DATA_TABLE_SIZE] = {
1,1,0,1,1,0,0,1,0,1,0,0,1,1,0,1/*,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,
```

```
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0,
0,0,0,1,1,0,1,1,0,0,1,1,1,1,1,1,1,0,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,0,1,1,0,0,0,1,1,0,0*/
};

//------------Ciclo de Seno de 1.8KHz------------
Int16 sinetable[BUFFSIZE] = {
0,19,37,53,66,75,79,79,75,66,53,37,19,0,
-19,-37,-53,-66,-75,-79,-79,-75,-66,-53,
-37,-19
};

//-----------Ciclo de Coseno de 1.8KHz-----------
Int16 costable[BUFFSIZE] = {
    80,78,71,60,45,28,10,-10,-28,-45,-60,-71,
-78,-80,-78,-71,-60,-46,-29,-10,9,28,45,
60,71,78
};
/*-----------------------------------------------*/

/*-----------------Prototypes-------------------*/
void modulacion(int numFIN,int s1,int s2, int contQ,int contI);
void datos(int sample);
void modular();

/*-----------------------------------------------*/

//*******************************************************
//*******************************************************
void datos(int sample){
pa = &data[sample];
bit1 = *pa;
bit2 = *(pa+1);
```

```
}

//*********************************************************
//*********************************************************
void modulacion(int numFIN,int s1,int s2,int contQ,int contI){

for(num=0;num<numFIN;num++)
{
pb=&costable[s1];
pc=&sinetable[s2];

pbb = &Qout[contQ];
pcc = &Iout[contI];

processData[incremento] =  *pb * *pcc + *pc * *pbb;

s1=s1+1; if(s1==26) s1=0;
s2=s2+1; if(s2==26) s2=0;
contQ=contQ+1; contI=contI+1;

incremento = incremento + 1;
}
}
//*********************************************************
//*********************************************************
void modular(){
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
for(sample=0;sample<DATA_TABLE_SIZE;sample=sample+2)
{
datos(sample); //Busco los dos bits de la tabla de datos.
filtrarQ(bit1);     //Filtro el bit Q primero, porque ahora hacemos OQPSK.
modulacion(39,0,13,0,39); //LLeno processData con el ciclo modulado correspondiente, con 39 valore
filtrarI(bit2); //Ahora filtro el bit I, que esta defasado T/2 con respecto al Q.
modulacion(39,13,0,39,0); //LLeno processData con el ciclo modulado correspondiente, con los sigui
}
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
start=1;
}
//*********************************************************
//*********************************************************
```

9.7.2. FILTRADO.C

```
/* ======== filtrado.c ======== */
// Este filtro usa una respuesta impulsiva FIR, la cual se trunca en +- 3T, y se hace la convolucion complet
// Este mÃ©todo tiene muchos mÃ¡s c\'{a}lculos que con el metodo de tablas de QAMv4.4, pero a este le vamos
// un filtrado multirate, lo que va a mejorar el desempeÃ±o, aun haciendo muchas multiplicaciones por 0.

/*------------Includes-------------*/
#include "filtrado.h"
#include <std.h>
#include "dsk6416.h"
/*--------------------------------*/

/*------------Definitions----------*/
#define TABLA_BIT 78
#define TABLA_SINC 468
#define MEDIO_CICLO 78
/*--------------------------------*/

/*------------Variables------------*/
int tramo1Q=1; //estas se van modificando segun que tramo se este calculando.
int tramo2Q=0;
int tramo3Q=0;
int tramo4Q=0;
int tramo5Q=0;
int tramo6Q=0;
int comienzo1Q=0; //estas se modifican una vez, y nunca mas.
int comienzo2Q=0;
int comienzo3Q=0;
int comienzo4Q=0;
int comienzo5Q=0;
int tramo1I=1; //estas se van modificando segun que tramo se este calculando.
int tramo2I=0;
int tramo3I=0;
int tramo4I=0;
int tramo5I=0;
int tramo6I=0;
int comienzo1I=0; //estas se modifican una vez, y nunca mas.
int comienzo2I=0;
int comienzo3I=0;
int comienzo4I=0;
```

```
int comienzo5I=0;

int flag11;
int flag12;
int flag21;
int flag22;
int flag31;
int flag32;
int flag41;
int flag42;
int flag51;
int flag52;
int flag61;
int flag62;
int i;
int x;
int m;
int mp;
int n;
int borrar;

extern int bit1; //Son los bits de entrada (los datos a transmitir).
extern int bit2;

int i;
Int16 *Qpointer; //Estos punteros apuntan a las tablas de la SINC Raised Cosine.
Int16 *Ipointer;
Int16 *Spointer;

extern short Qout[TABLA_BIT]; //Arreglos donde se guarda el filtrado.
extern short Iout[TABLA_BIT];


//Sinc de 468 muestras.
Int16 sinc[TABLA_SINC] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,
2,3,3,3,3,3,3,3,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,4,4,4,4,3,3,3,3,2,2,1,1,0,0,-1,-1,-2,-2,-3,-3,-4,-5,
-5,-6,-7,-7,-8,-9,-10,-10,-11,-12,-13,-14,-14,-15,-16,-17,-18,-19,
-19,-20,-21,-22,-22,-23,-24,-25,-25,-26,-26,-27,-28,-28,-29,
-29,-29,-30,-30,-30,-30,-31,-31,-31,-31,-31,-30,-30,-30,-29,
```

```
-29,-28,-28,-27,-26,-26,-25,-24,-22,-21,-20,-19,-17,-16,-14,
-12,-10,-9,-7,-4,-2,0,2,5,7,10,13,15,18,21,24,27,31,34,37,41,44,48,51,55,59,62,
66,70,74,78,82,86,90,94,98,102,106,110,114,118,122,126,130,134,138,
142,146,150,154,157,161,165,168,172,175,179,182,185,188,192,
194,197,200,203,205,208,210,212,214,216,218,220,222,223,224,
225,227,227,228,229,229,230,230,230,230,230,229,229,228,227,
227,225,224,223,222,220,218,216,214,212,210,208,205,203,200,
197,194,192,188,185,182,179,175,172,168,165,161,157,154,150,
146,142,138,134,130,126,122,118,114,110,106,102,98,94,90,86,82,78,
74,70,66,62,59,55,51,48,44,41,37,34,31,27,24,21,18,15,13,10,7,5,2,0,-2,-4,-7,-9,-10,
-12,-14,-16,-17,-19,-20,-21,-22,-24,-25,-26,-26,-27,-28,-28,
-29,-29,-30,-30,-30,-31,-31,-31,-31,-31,-30,-30,-30,-30,-29,
-29,-29,-28,-28,-27,-26,-26,-25,-25,-24,-23,-22,-22,-21,-20,
-19,-19,-18,-17,-16,-15,-14,-14,-13,-12,-11,-10,-10,-9,-8,-7,
-7,-6,-5,-5,-4,-3,-3,-2,-2,-1,-1,0,0,1,1,2,2,3,3,3,3,4,4,4,4,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,4,4,4,4,4,4,4,3,3,3,3,3,3,
3,3,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
};

//Bit representado por 78 muestras.
Int16 uno[TABLA_BIT]={
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};
//Bit representado por 78 muestras.
Int16 cero[TABLA_BIT]={
-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};
/*---------------------------------*/

/* ----------Prototypes------------*/
void filtrarQ(int bit1);
void filtrarI(int bit2);
void convol(int inicio, int fin, int flag);
void convol2(int inicio, int fin, int flag);
/*---------------------------------*/

//*****************************************************
//*****************************************************
void filtrarQ(int bit1){
```

```
for(borrar=0;borrar<78;borrar++){ //Â¡Â¡Â¡Â¡VER SI HAY ALGO MAS EFICIENTE!!!!
Qout[borrar]=0;
}

if (tramo1Q!=0){
if (bit1==1) flag11=1; else flag11=0;
convol(0,78,flag11);
if(comienzo1Q==1){
convol(390,468,flag21);
convol(312,390,flag31);
convol(234,312,flag41);
convol(156,234,flag51);
convol(78,156,flag61);
}
comienzo1Q=1;
}

if (tramo2Q!=0){
if (bit1==1) flag21=1; else flag21=0;
convol(78,156,flag11);
convol(0,78,flag21);
if(comienzo2Q==1){
convol(390,468,flag31);
convol(312,390,flag41);
convol(234,312,flag51);
convol(156,234,flag61);
}
comienzo2Q=1;
}

if (tramo3Q!=0){
if (bit1==1) flag31=1; else flag31=0;
convol(156,234,flag11);
convol(78,156,flag21);
convol(0,78,flag31);
if(comienzo3Q==1){
convol(390,468,flag41);
convol(312,390,flag51);
convol(234,312,flag61);
}
```

```
comienzo3Q=1;
}

if (tramo4Q!=0){
if (bit1==1) flag41=1; else flag41=0;
convol(234,312,flag11);
convol(156,234,flag21);
convol(78,156,flag31);
convol(0,78,flag41);
if(comienzo4Q==1){
convol(390,468,flag51);
convol(312,390,flag61);
}
comienzo4Q=1;
}

if (tramo5Q!=0){
if (bit1==1) flag51=1; else flag51=0;
convol(312,390,flag11);
convol(234,312,flag21);
convol(156,234,flag31);
convol(78,156,flag41);
convol(0,78,flag51);
if(comienzo5Q==1){
convol(390,468,flag61);
}
comienzo5Q=1;
}

if (tramo6Q!=0){
if (bit1==1) flag61=1; else flag61=0;
convol(390,468,flag11);
convol(312,390,flag21);
convol(234,312,flag31);
convol(156,234,flag41);
convol(78,156,flag51);
convol(0,78,flag61);
}
```

```
if(tramo6Q==1){tramo1Q=1;tramo2Q=0;tramo3Q=0;tramo4Q=0;tramo5Q=0;}
if(tramo5Q==1){tramo1Q=0;tramo2Q=0;tramo3Q=0;tramo4Q=0;tramo5Q=0;tramo6Q=1;}
if(tramo4Q==1){tramo1Q=0;tramo2Q=0;tramo3Q=0;tramo4Q=0;tramo5Q=1;tramo6Q=0;}
if(tramo3Q==1){tramo1Q=0;tramo2Q=0;tramo3Q=0;tramo4Q=1;tramo5Q=0;tramo6Q=0;}
if(tramo2Q==1){tramo1Q=0;tramo2Q=0;tramo3Q=1;tramo4Q=0;tramo5Q=0;tramo6Q=0;}

if((tramo1Q==1)&&(tramo6Q==0)){tramo1Q=0;tramo2Q=1;tramo3Q=0;tramo4Q=0;tramo5Q=0;tramo6Q=0;}
if((tramo1Q==1)&&(tramo6Q==1)){tramo1Q=1;tramo2Q=0;tramo3Q=0;tramo4Q=0;tramo5Q=0;tramo6Q=0;}
}

//*********************************************************
//*********************************************************
void filtrarI(int bit2){

for(borrar=0;borrar<78;borrar++){ //Â¡Â¡Â¡Â¡VER SI HAY ALGO MAS EFICIENTE!!!!
Iout[borrar]=0;
}

if (tramo1I!=0){
if (bit2==1) flag12=1; else flag12=0;
convol2(0,78,flag12);
if(comienzo1I==1){
convol2(390,468,flag22);
convol2(312,390,flag32);
convol2(234,312,flag42);
convol2(156,234,flag52);
convol2(78,156,flag62);
}
comienzo1I=1;
}

if (tramo2I!=0){
if (bit2==1) flag22=1; else flag22=0;
convol2(78,156,flag12);
convol2(0,78,flag22);
if(comienzo2I==1){
convol2(390,468,flag32);
convol2(312,390,flag42);
convol2(234,312,flag52);
convol2(156,234,flag62);
}
```

```
comienzo2I=1;
}

if (tramo3I!=0){
if (bit2==1) flag32=1; else flag32=0;
convol2(156,234,flag12);
convol2(78,156,flag22);
convol2(0,78,flag32);
if(comienzo3I==1){
convol2(390,468,flag42);
convol2(312,390,flag52);
convol2(234,312,flag62);
}
comienzo3I=1;
}

if (tramo4I!=0){
if (bit2==1) flag42=1; else flag42=0;
convol2(234,312,flag12);
convol2(156,234,flag22);
convol2(78,156,flag32);
convol2(0,78,flag42);
if(comienzo4I==1){
convol2(390,468,flag52);
convol2(312,390,flag62);
}
comienzo4I=1;
}

if (tramo5I!=0){
if (bit2==1) flag52=1; else flag52=0;
convol2(312,390,flag12);
convol2(234,312,flag22);
convol2(156,234,flag32);
convol2(78,156,flag42);
convol2(0,78,flag52);
if(comienzo5I==1){
convol2(390,468,flag62);
}
comienzo5I=1;
}
```

```
if (tramo6I!=0){
if (bit2==1) flag62=1; else flag62=0;
convol2(390,468,flag12);
convol2(312,390,flag22);
convol2(234,312,flag32);
convol2(156,234,flag42);
convol2(78,156,flag52);
convol2(0,78,flag62);
}

if(tramo6I==1){tramo1I=1;tramo2I=0;tramo3I=0;tramo4I=0;tramo5I=0;}
if(tramo5I==1){tramo1I=0;tramo2I=0;tramo3I=0;tramo4I=0;tramo5I=0;tramo6I=1;}
if(tramo4I==1){tramo1I=0;tramo2I=0;tramo3I=0;tramo4I=0;tramo5I=1;tramo6I=0;}
if(tramo3I==1){tramo1I=0;tramo2I=0;tramo3I=0;tramo4I=1;tramo5I=0;tramo6I=0;}
if(tramo2I==1){tramo1I=0;tramo2I=0;tramo3I=1;tramo4I=0;tramo5I=0;tramo6I=0;}

if((tramo1I==1)&&(tramo6I==0)){tramo1I=0;tramo2I=1;tramo3I=0;tramo4I=0;tramo5I=0;tramo6I=0;}
if((tramo1I==1)&&(tramo6I==1)){tramo1I=1;tramo2I=0;tramo3I=0;tramo4I=0;tramo5I=0;tramo6I=0;}
}

//*********************************************************
//*********************************************************
void convol(int inicio, int fin, int flag){ //Convolucion para la Rama Q.

n=0;
for(m=inicio;m<fin;m++)
{
x=n;mp=m;
while(mp>=inicio)
{
Spointer=&sinc[mp];
if(flag==1) Qpointer=&uno[inicio+x-mp];
else Qpointer=&cero[inicio+x-mp];
Qout[n] = Qout[n] + *Qpointer * *Spointer;
mp=mp-1;
}
n=n+1;
}
}
```

```
//**********************************************************
//**********************************************************
void convol2(int inicio, int fin, int flag){ //Convolucion para la Rama I.

n=0;
for(m=inicio;m<fin;m++)
{
x=n;mp=m;
while(mp>=inicio)
{
Spointer=&sinc[mp];
if(flag==1) Ipointer=&uno[inicio+x-mp];
else Ipointer=&cero[inicio+x-mp];
Iout[n] = Iout[n] + *Ipointer * *Spointer;
mp=mp-1;
}
n=n+1;
}
}
//**********************************************************
//**********************************************************
```

Bibliography

[1] E. Venosa, *Software Radio. Sampling Rate Selection, Design and Syncheonization.* Springer, 2012.

[2] R. Hosking, "Digital receiver handbook: Basics of software radio 5th. theory of operation applications products," *Pentek Inc.*

[3] "Mil 188-110b standard."

[4] J. Nieto, "On air results of spatial diversity for the mil-std-188-110b appendix c (stanag 4539) 9600 bps waveform," *IEEE*, pp. 432–436, 2001.

[5] W. F. . J. Nieto, "Understanding hf channel simulator requirements in order to reduce hf modem performance measurement variability."

[6] M. Simon, *Communicatiolns over Fading Channel. 2nd Ed.* Wiley, 2005.

[7] RapidM, "Rm-tc4-lh-hf module," *www.rapidm.com.*

[8] R. Hosking, "Putting fpgas to work in software radio systems 7th ed. technology, resources products, applications," *Pentek Inc.*, 2013.

[9] . e. a. Clark Watterson, "Experimental confirmation of an hf channel model," *1970.*

[10] ITU, "Recomendación itu f-1487. testing of hf modems with bandwidths of up to about 12 khz using ionospheric channel simulators," *ITU-R*, 2000.

[11] Matworks, "http://www.mathworks.com/help/comm/examples/hf-ionospheric-channel-models.html#zmw57dd0e1743 dirección." *Matworks*, 2010.

[12] ITU, "Recomendación itu 520.2. empleo de simuladores de canales ionosfericos en ondas decametricas," *ITU-R*, 1992.

[13] E. J. . et al., *Third Generation and Wideband HF radio Communications.* Artech House, 2012.

[14] . J. N. W. Furman, "Understanding hf channel simulator requirements inorder to reduce hf modem performance measurement," *Harris Corporation.*

[15] T. Ha, *Theory and Design of Digital Communication Systems.* Cambridge Press, 2011.

[16] J. Proakis and D. Manolakis, *Digital Signal Processing. Principles, Alorithms, and Applications.* MacMilian Publishing, 1992.

[17] A. Devices, "Digital pulse shaping g basics application note 922," 1978.

[18] P. Vaidyanathan, *Multirate Filter Banks.* Prentice Hall, 1993.

[19] T. Instruments, "V.34 transmitter and receiver implementation on the tms320c50 dsp," *Texas Application Report*, 1997.

[20] COMSEARCH, "Guide tom emission designators," *www.comsearch.com.*

[21] H. Theodore S.Rappaport, Keith Blankenship, "Propagation and radio system design issues in mobile radio systems for the glomo project," *GloMo Program*, 1997.

[22] E. P. Nathan Blaunstein, *IONOSPHERE AND APPLIED ASPECTS OF RADIO COMMUNICATION AND RADAR.* CRC Press, 2008.

[23] E. Biglieri, G. Caire, and G. Tarico, "Coding for fading channel: A survey," *Signal Processing for Multimedia*, 1999.

[24] V. Erceg, "Channel models for fixed wireless systems," *IOSPAN Wireless Company*, 2001.

[25] S. G. L. G. Vinko Erceg, David Michelson, "A model for the multipath delay profile of fixed wireless channels," *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS,*, vol. 17, no. 3, 1999.

[26] P. Bello, "Generic channel simulator," *repared under Contract MDA904-95-C-2078*, 1997.

[27] B. C. Benoit ROTURIER, "A general model for vhf aeronauticalmultipath propagation channel," 1999.

[28] P. Bello, "Aeronautical channel characterization," *IEEE Trans on Comm*, vol. 21, no. 5, pp. 548–563, May 1973.

[29] S. S. Ezio Biglieri, John Proakis, "Fading channels: Information-theoretic and communications aspects," *IEEE TRANSACTIONS ON INFORMATION THEORY,*, vol. 44, no. 6, October 1998.

[30] A. S. John B. Anderson, *Coded Modulation Systems.* Kluwer, 2002.

[31] B. Sklar, "Mitigating the degradation effects of fading channels."

[32] ——, "Fading channels chapter four," *Buscar de que libro es.*