UNIVERSIDAD DE SEVILLA
ESCUELA SUPERIOR DE INGENIEROS
INGENIERÍA DE TELECOMUNICACIONES

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
ÁREA DE TEORÍA DE LA SEÑAL Y COMUNICACIONES

PROYECTO FIN DE CARRERA

# DISEÑO E IMPLEMENTACIÓN DE UN MODULADOR Y DEMODULADOR 16-QAM SOBRE UN TMS320C30

AUTOR: JUAN IGNACIO ROMÁN SÁNCHEZ

DIRECTOR: RUBÉN MARTÍN CLEMENTE

JUNIO DE 2002

# ÍNDICE

# 1 INTRODUCCIÓN

Todo sistema de comunicación tiene tres elementos imprescindibles: un transmisor, dónde se genera la información que se va a enviar; un canal, que provee un medio físico por el que viajará dicha información; y un receptor, que es el destinatario. En el presente proyecto se estudiará la posibilidad de establecer una comunicación entre dos sistemas remotos y cómo intentar resolver todas las problemáticas que puedan surgir. Más concretamente, se intentará que un ordenador (transmisor) envíe información mediante un cable (canal) a otro ordenador (receptor). Estos equipos no cuentan con elementos especiales para realizar la modulación y demodulación (necesarios para realizar la comunicación) y además no se puede modificar el hardware con el que se cuenta salvo configuraciones internas de ese hardware. Por lo tanto se realizará un diseño que se intente adaptar en la medida de lo posible a los elementos disponibles, recurriendo cuando no exista otra opción a su implementación mediante software, lo que resulta más lento (y barato) pero añade mucha flexibilidad.

El objetivo final es la viabilidad de dicha comunicación dejando un poco de lado cuestiones de eficiencia; eso no quiere decir que no se hayan realizado optimizaciones en el funcionamiento ya que como se verá más adelante se hacen algunas consideraciones al respecto.

En primer lugar se realiza un estudio de varios tipos de modulación para encontrar la que mejor se ajusta a este caso concreto, haciendo especial hincapié en el apartado dedicado a la sincronización. A continuación se revisarán las herramientas disponibles, tanto hardware como software, para poder realizar un aprovechamiento mejor de ellas. Una vez metidos en el diseño se tratarán cada uno de los bloques por separado, los cuales se acompañan, además de la solución final, de un diagrama de flujo que es independiente de la plataforma elegida y que permite abstraer el algoritmo implementado.

Como apoyo y para ilustrar mejor el funcionamiento del sistema se realiza también una simulación software del sistema, de manera que se pueda en cierto modo estudiar y prever su comportamiento. Dicha simulación se ha utilizado para ilustrar los resultados parciales que se van obteniendo y permiten tener una visión del proceso que se lleva a cabo en cada uno de los bloques.

# 2 TÉCNICAS DE MODULACIÓN DIGITAL

## 2.1 INTRODUCCIÓN

Se entiende por modulación al proceso por el cual una propiedad o parámetro de una señal, denominada portadora, se hace variar sistemáticamente conforme otra señal, llamada moduladora o de información. El proceso inverso se denomina demodulación y se realiza en el receptor. La modulación permite que el espectro de la señal enviada se ajuste al ancho de banda del canal y sea menos susceptible al ruido, consiguiéndose relaciones señal a ruido mayores que con una transmisión en banda base. Otra ventaja que presenta es que señales provenientes de fuentes diferentes compartan un mismo canal (multiplexación).

Típicamente la señal usada como portadora es la senoidal, que posee tres parámetros independientes que se pueden modificar: amplitud, frecuencia y fase. La variación de uno o más de ellos da lugar a toda una gama de modulaciones diferentes, cada una con cualidades propias que la hace preferible en determinadas situaciones. Sin embargo, existe una clasificación más general originada por la propia naturaleza de la información: analógica frente a digital. Las digitales presentan ventajas como una mayor inmunidad al ruido e interferencias externas, mayor flexibilidad en el diseño de los sistemas, utiliza un formato común para diferentes tipos de mensajes (voz, vídeo, datos...) y proporción mayor seguridad en la comunicación mediante el uso de encriptación.

La mayoría de las técnicas desarrolladas admiten tanto una detección coherente como no coherente. Estas últimas son más imprecisas pero los sistemas implicados son más simples. Las coherentes, sin embargo, necesitan de una sincronización ajustada para producir resultados satisfactorios; puesto que más adelante se tratará en detalle esta problemática, se va a obviar por el momento y se supondrá que esa sincronización existe.

Existen versiones multinivel de cada tipo de modulación, consistentes en agrupar M símbolos de la fuente bajo un mismo símbolo modulado, dando lugar a $2^M$ símbolos posibles; se produce un incremento en la tasa de bits transmitida, o equivalentemente una reducción en el ancho de banda si se mantiene constante la tasa de bits. Como contrapartida se produce un empeoramiento frente al ruido.

Dos son los parámetros que servirán principalmente para comparar las prestaciones de las diversas formas de modulación: la probabilidad de error en función del cociente entre la potencia de la señal y la potencia de ruido, o SNR; y la eficiencia en el uso del ancho de banda del canal, definida como la relación entre la velocidad de transmisión de la fuente respecto al ancho de banda disponible: $\rho=R_b/B$, o equivalentemente: $\rho=(T_b*B)^{-1}$. Por supuesto se prefieren modulaciones con valores pequeños de $P_e$ y grandes de $\rho$.

## 2.2 MODULACIONES DE AMPLITUD

### 2.2.1 ASK

La modulación ASK (Amplitude Shift Keying) consiste en conmutar la amplitud de la onda portadora entre dos posibles valores, correspondientes a los símbolos binarios "0" y "1", permaneciendo la frecuencia y la fase constantes. Se puede expresar la señal de forma general [4]:

$$s_i(t) = \sqrt{\frac{2E_i}{T_b}} \cos(\omega_c\, t + \phi)\ ,\ 0 < t \le T_b,\ \ para\ i = 1\ ,\ 2$$

donde $T_b$ es el tiempo de duración de un bit; $\phi$ es una fase inicial arbitraria fija (generalmente $\pi/2$); y $E_i$ es la energía del símbolo "i", que toma los valores cero (ausencia de pulso) o $E_b$ (energía por bit o energía para un tiempo $T_b$, de valor $A^2T_b/2$).

A continuación puede verse la forma que adopta la señal ASK para la secuencia binaria 010110 y un tiempo de bit de 32 mseg.

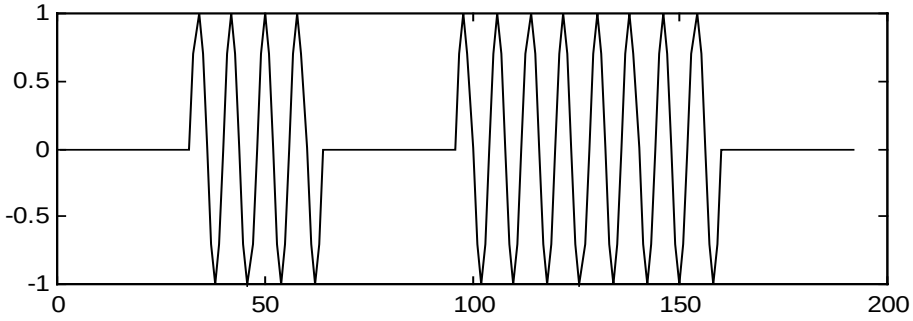


*Figura 2.1: Ejemplo Señal ASK ($T_s$=1/19200 seg).*

Esta modulación puede verse como un sistema en el que la portadora se conecta o desconecta según el caso; por eso que se conoce también a esta modulación como OOK (On-Off Keying).

Es usual representar las señales involucradas en la modulación en un espacio vectorial denominado constelación, de manera que se puedan expresar en función de un conjunto finito de funciones base al multiplicarlas por unos determinados coeficientes. En este caso resulta un espacio unidimensional, siendo la única función base:

$$\Psi(t) = \sqrt{\frac{2}{T_b}} \cos(\omega_c\, t)$$

con coeficientes ideales "0" y "$\sqrt{E_b}$ ".

Las constelaciones se suelen normalizar dividiendo por la energía de símbolo; para OOK el resultado se muestra en la figura 2.2.

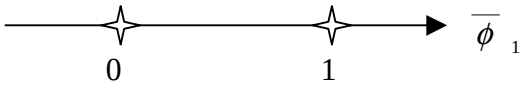


*Figura 2.2: Constelación OOK*

La energía promedio en esta modulación es:

$$E_{av} = \frac{1}{2} E_1 + \frac{1}{2} E_2 = \frac{1}{2} 0 + \frac{1}{2} E_b \Rightarrow E_{av} = \frac{A^2 T}{4}$$

y la distancia (ideal) entre las dos señales en la constelación:

$$d = \sqrt{2 E_{av}}$$

La densidad espectral de potencia de la señal ASK presenta un lóbulo principal centrado en $\omega_c$, y otros secundarios, mucho más pequeños [8]. La anchura del lóbulo principal determina el ancho de banda (más del 90% de la potencia se encuentra aquí), que es el doble que el correspondiente a la señal en banda base. Esto supone que la eficiencia espectral en ASK sea de 2 bps/Hz.

La función del receptor es la de decidir cuál fue el símbolo enviado en función de la señal recibida; puesto que la presencia de ruido en el canal es inevitable, la señal recibida será una variable aleatoria correspondiente a cada símbolo posible de la fuente. El criterio más usado es el de mínimo error (mínima distancia) a cada uno de los símbolos. Aparece por tanto una zona umbral de decisión entre símbolos adyacentes. Para ASK, y suponiendo símbolos equiprobables en la fuente, dicho umbral se establece en d/2 para que el receptor sea óptimo en el sentido de minimizar la probabilidad de error. Para un detector coherente, esta probabilidad vale:

$$P_e = Q\left(\sqrt{\frac{E_b}{N_o}}\right)$$

siendo $N_o$ la densidad espectral de ruido [4]. Para el caso de una detección no coherente y $E_b/N_0 > 1/4$, la probabilidad anterior se puede aproximar por:

$$P_e = \frac{1}{2} e^{\frac{E_b}{2N_0}}$$

cuyo valor resulta superior, esto es, el no coherente tiene un funcionamiento más pobre.

### 2.2.2 M-ASK

La expresión para las M señales posibles se obtiene generalizando la ASK:

$$s_i(t) = \sqrt{\frac{2\ (i-1)\ E_b}{T_b}} \cos(\omega_c\ t + \phi)\ ,\ 0 < t \le T_b,\ \ para\ i = 1\ ,\ M$$

En algunos lugares se hace una definición equivalente a esta con: j = i-1; en este caso j varía entre 0 y M-1 [3]. Para M=4 las amplitudes posibles son 0 (00), 1 (01), 2 (10) y 3 (11); para 01011000 se obtiene la onda de la figura 2.3.

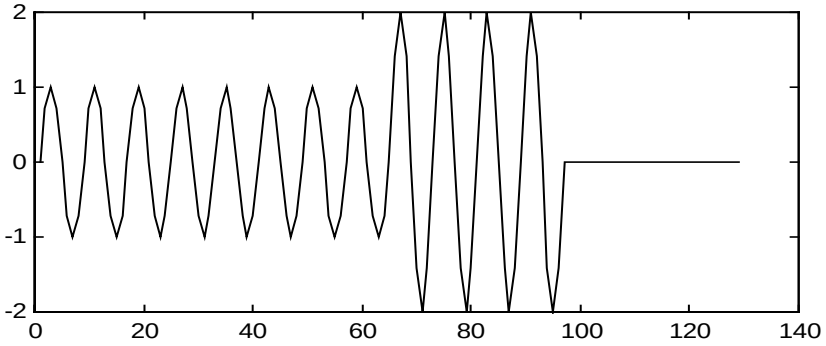


*Figura 2.3: Ejemplo Señal ASK ($T_s$=1/19200 seg).*

Con la misma función base que antes, los coeficientes son: $i\ \sqrt{E_b}$

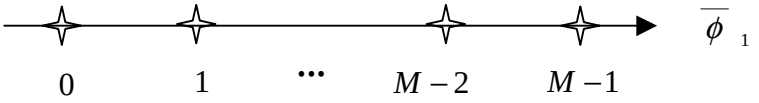


*Figura 2.4: Constelación 8-PSK*

La energía media de la constelación es: $E_{av}=E_b\ log_2 M$, lo que lleva a que la probabilidad de error sea [4]:

$$P_e = \left(1 - \frac{1}{M}\right)\ Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

### 2.2.3 QAM

Otro método para aumentar la eficiencia espectral, además de usar una modulación M-aria, es realizar una multiplexación en cuadratura (*Quadrature* AM), consistente en combinar (sumar) dos señales cuyas portadoras son ortogonales entre sí, aunque de la misma frecuencia. Una forma de expresarlas es mediante la expresión:

$$s_i(t) = \sqrt{E_b}\ \left[ I_i\ \Psi_1(t) + Q_i\ \Psi_2(t) \right]$$

Las dos señales se denominan componentes en fase y en cuadratura, y pone de manifiesto cómo se pueden mezclar datos de fuentes distintas sobre la misma señal modulada. Generalmente se tiene:

$$\Psi_1(t) = \sqrt{\frac{2}{T_b}} \cos(\omega_i\ t) \qquad \Psi_2(t) = \sqrt{\frac{2}{T_b}} \operatorname{sen}(\omega_i\ t)$$

Si los datos provienen de la misma fuente se consigue duplicar el número de bits de información enviados por segundo, a costa de una mayor complejidad de los sistemas encargados de la generación y detección.

Cualquier tipo de modulación se puede expresar en la forma anterior con unos coeficientes I y Q adecuados realizando manipulaciones algebraicas sobre la expresión general de la modulación. El caso M-ASK anterior se obtiene con Q=0 para $\phi=0$ (o I=0 si $\phi=\pi/2$); es fácil ver que el resto de modulaciones que se verán a continuación también poseen una expresión similar.

Cuando ambas portadoras tienen igual amplitud (I=Q para todos los símbolos) la constelación degenera en una 4-PSK (modulación de fase). Más adelante se definirá la QAM M-aria en la que I y Q toman varios valores independientemente, y se verá que en realidad es una modulación conjunta de amplitud y fase.

## 2.3 MODULACIONES DE FRECUENCIA

### 2.3.1 FSK

Consiste en conmutar la frecuencia instantánea de la portadora, manteniendo constantes la amplitud y la fase. La forma general de la señal FSK (*Frequency Shift Keying*) es [9]:

$$s_i(t) = \sqrt{\frac{2E_b}{T_b}} \cos(\omega_i\, t + \phi)\ ,\ 0 < t \le T_b,\ \ para\ i = 1\ ,\ 2$$

Para la secuencia 010110, FSK toma la forma de la figura 2.5.

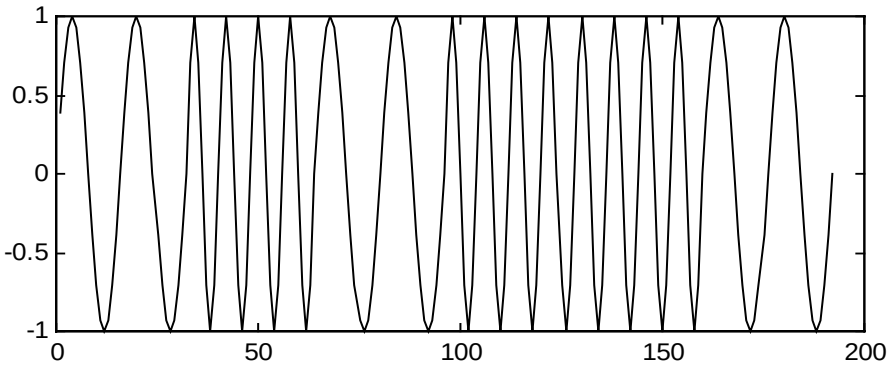


*Figura 2.5: Ejemplo Señal FSK ($T_s$=1/19200 seg).*

Una manera de ver esta modulación es como combinación de dos ASK; en efecto, la señal de la figura 2.5 puede descomponerse en dos ondas independientes, como se ilustra en la figura 2.6. Cada una de ellas corresponde a una ASK, sólo que las portadoras poseen frecuencias distintas.

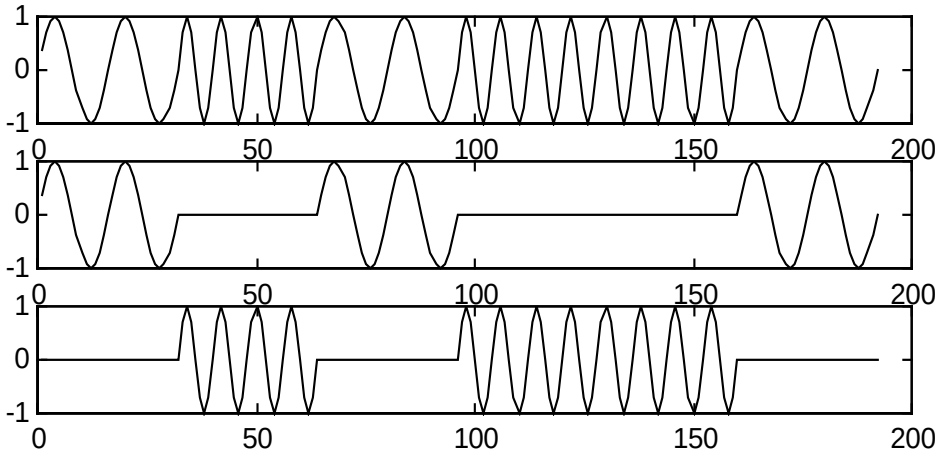


*Figura 2.6: Señal FSK como dos ASK ($T_s$=1/19200 seg).*

De esta forma se pueden identificar las dos funciones base que forman la constelación:

$$\Psi_i(t) = \sqrt{\frac{2}{T_b}}\cos(\omega_i\, t)\ ,\ \ para\ i = 1\ ,\ 2$$

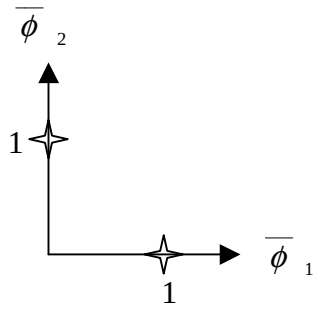


*Figura 2.7: Constelación BFSK*

Hay que hacer notar que aunque en la figura 2.7 las dos funciones base aparecen como ortogonales, esta disposición no es cierta en general y corresponde a un caso particular que se verá más adelante.

Los coeficientes valen: $a_{ij} = \begin{cases} \sqrt{E_b}, & i = j \\ 0, & i \neq j \end{cases}$, y la energía promedio: $E_{av} = E_b$.

En realidad no es necesario disponer de un oscilador para cada portadora, sino que se puede usar una única fuente que servirá como referencia para sintetizarlas, del siguiente modo:

$$\begin{cases} \omega_1 = \omega_c + \dfrac{\Delta\omega}{2} \\ \omega_2 = \omega_c - \dfrac{\Delta\omega}{2} \end{cases} \quad => \quad \begin{cases} \omega_c = \dfrac{\omega_1 + \omega_2}{2} \\ \Delta\omega = \omega_1 - \omega_2 \end{cases}$$

El espectro de la señal FSK se obtiene directamente como una ASK centrada en $\omega_c + \Delta\omega$ y otra en $\omega_c - \Delta\omega$ por lo que tiene un ancho de banda $\Delta\omega$ mayor que aquélla. El valor $\Delta\omega$ se denomina desviación en frecuencia y su valor es:

$$\Delta\omega = \frac{2\pi}{T_b}$$

Puesto que la distancia entre símbolos en FSK es la misma que en ASK, sus probabilidades de error coinciden (bajo hipótesis de equiprobabilidad y misma energía promedio de bit frente al ruido):

$$P_e = Q\left(\sqrt{\frac{E_b}{N_o}}\right)$$

Sin embargo, para unas mismas necesidades de potencia de pico, la FSK presenta una ventaja de 3 dB sobre la ASK, ya que ésta no se encuentra presente (en promedio) la mitad del tiempo.

Si se realiza una detección no coherente (usando detectores de envolvente):

$$P_e = \frac{1}{2}\ e^{-\frac{E_b}{2N_0}}$$

Al igual que ocurría en ASK, el sistema no coherente tiene una probabilidad de error mayor.

### 2.3.2 MSK

Para que la densidad espectral de potencia de una señal modulada disminuya rápidamente más allá del ancho de banda mínimo es aconsejable que ésta no presente discontinuidades. Puesto que la información reside en la desviación de frecuencia, aún queda controlar la fase para cumplir con ese objetivo. Este es el fundamento de la modulación FSK de fase continua o CPFSK (*Continuous Phase* FSK). De todas las posibilidades la más interesante es sin duda la MSK (*Minimum Shift Keying*), en la que el desplazamiento de frecuencia es mínimo, y resulta ser:

$$\Delta\omega = \frac{\pi}{T_b}$$

La expresión general de la MSK es:

$$s_i(t) = \sqrt{\frac{2E_b}{T_b}}\cos(\omega_i\ t + \phi_o \pm \frac{\pi}{2T_b})\ ,\ 0 < t \le T_b,\ \ para\ i = 1\ ,\ 2$$

donde $\phi_o$ es la fase inicial, que depende de la historia anterior del proceso de demodulación y debe elegirse de manera que se evite cualquier discontinuidad [4]. Teniendo en cuenta la expresión de $\varpi_i$ se observa que la diferencia entre las fases de dos símbolos consecutivos vale "$\pi/2$" (si el segundo es un "1") o "$-\pi/2$" (si el segundo es un "0"); de esta forma sólo son posibles fases de "$\pm\pi/2$" en los instantes múltiplos impares de $T_b$, y fases de "$\pm\pi$" en los múltiplos pares. Esto se pone de manifiesto si se pone la señal en función de sus componentes en fase y cuadratura:

$$s_i(t) = \sqrt{\frac{2E_b}{T_b}}\ \left[\cos(\phi_0)\ \cos\left(\frac{\pi\ t}{2T_b}\right)\ \cos(\omega_c t)\ -\ \text{sen}(\phi_0)\ \text{sen}\left(\frac{\pi\ t}{2T_b}\right)\ \text{sen}(\omega_c t)\right]$$

La secuencia binaria 010110 toma al modularse con MSK la forma siguiente:

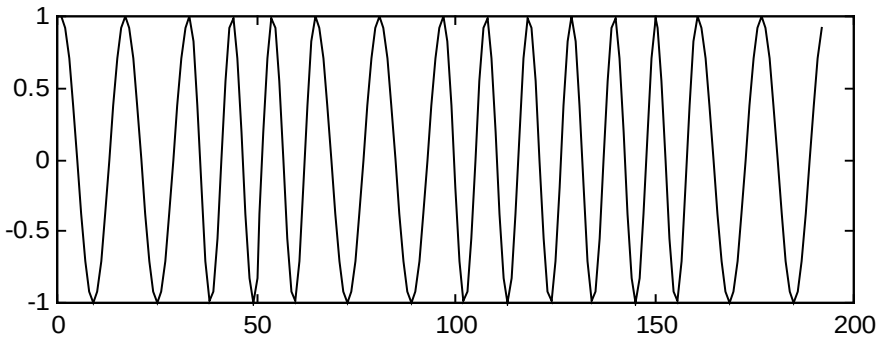


*Figura 2.8: Señal MSK ($T_s$=1/19200 seg).*

Expresada de la manera anterior, la MSK puede considerarse como un caso particular de OQPSK, que se verá más adelante. Al igual que ella las dos componentes están retrasadas $T_b$ entre sí, pero en MSK las amplitudes están ponderadas senoidalmente, mientras que en OQPSK lo están de forma rectangular (se mantienen constantes en todo el intervalo).

Existe una variante de MSK motivada por disminuir las interferencias en canales adyacentes, denominada GMSK (*Gaussian* MSK). En ella se realiza un filtrado previo a la modulación que aumenta el decaimiento de los lóbulos secundarios del espectro de potencia y estrecha el principal, a costa de introducir interferencia entre símbolos o ISI. El ancho de banda del filtro se ajusta para llegar a una solución de compromiso entre las dos magnitudes anteriores.

## 2.4 MODULACIONES DE FASE

### 2.4.1 M-PSK

Por comodidad se define en primer lugar el caso M-ario. Variando la fase de la portadora en M valores posibles se obtiene una modulación M-PSK (*Phase Shift Keying*). La expresión general es:

$$s_i(t) = \sqrt{\frac{2E_b}{T_b}} \cos(\omega_c\ t + \phi_{\ i})\ ,\ 0 < t \le T_b, \qquad \phi_i = \frac{2\pi\ i}{M},\ i = 1,\ \ldots\ ,\ M$$

Al igual que ocurría en FSK la constelación resultante es bidimensional, pero las funciones base son distintas:

$$\Psi_1(t) = \sqrt{\frac{2}{T_b}} \cos(\omega_i\ t) \qquad\qquad \Psi_2(t) = \sqrt{\frac{2}{T_b}} \operatorname{sen}(\omega_i\ t)$$

La expresión de los coeficientes correspondientes es: $\begin{cases} a_{i1} = \sqrt{E_b} \cos\dfrac{2\pi\ i}{M} \\ a_{i2} = \sqrt{E_b} \operatorname{sen}\dfrac{2\pi\ i}{M} \end{cases}$

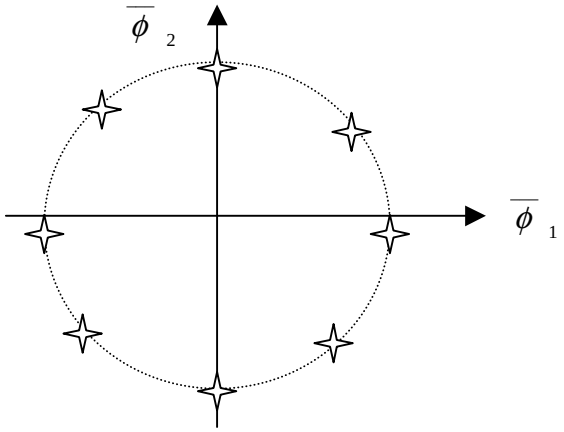


*Figura 2.9: Constelación 8-PSK*

Se observa por lo tanto que las regiones de decisión se obtienen al dividir una circunferencia en M sectores iguales. Al aumentar M se reduce el tamaño de estas regiones y, por tanto, la inmunidad al ruido. Esto puede deducirse también a partir de la expresión de la probabilidad de error:

$$P_e = 2\ Q\left(\sqrt{\frac{2E_b}{N_o}} \operatorname{sen}\left(\frac{\pi}{M}\right)\right)$$

expresión válida para M<4, calculada con el método denominado “cota de unión” y que representa una cota superior para el error [4].

### 2.4.2 BPSK

Un ejemplo particular de PSK es la PSK binaria o BPSK, mostrada en la figura 2.10 para 010110. En este caso la constelación se vuelve unidimensional y totalmente equivalente a la ASK salvo que ahora los coeficientes valen $\pm\sqrt{E_b}$ .

Esto hace que la energía media se duplique y la distancia entre símbolos aumente, con lo que la probabilidad de error toma la forma:

$$P_e = Q\left(\sqrt{\frac{2E_b}{N_o}}\right)$$

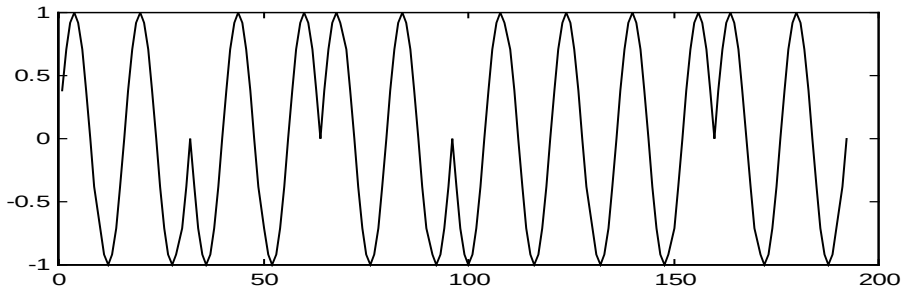


*Figura 2.10: Ejemplo Señal BPSK ($T_s$=1/19200 seg).*

### 2.4.3 DPSK

En esta variante, denominada PSK diferencial o DPSK, no se asigna la fase de forma fija, sino que se produce un cambio respecto al anterior símbolo transmitido de la siguiente manera:

- Si el siguiente símbolo es “0”, se produce un cambio de fase de 90º.

- Si el siguiente símbolo es “1”, el cambio es de 270º.

En la figura siguiente hay un ejemplo; nótese las diferencias y cambios de fase respecto a la figura 2.10.

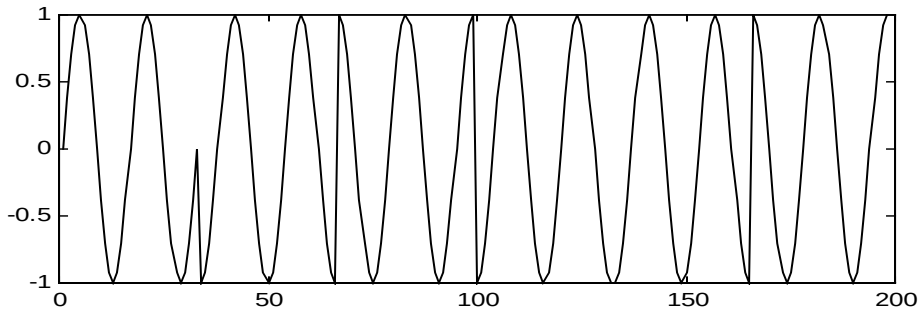


*Figura 2.11: Ejemplo Señal DPSK ($T_s$=1/19200 seg).*

De esta forma los demoduladores deben detectar cambios relativos en la fase, no valores absolutos. Las ventajas que se producen respecto a una PSK son dos: relaja las necesidades de sincronización, y presenta una menor sensibilidad al ruido.

### 2.4.4 QPSK

La modulación QPSK (*Quadrature* PSK) es en realidad una 4-PSK, y equivale a la QAM descrita anteriormente, ya que puede interpretarse de ambas formas. Sin embargo existen dos variantes que merece la pena comentar.

La primera de ellas es la denominada OQPSK (*Offset-Keyed* QPSK); en ella el flujo de bits a enviar se divide en dos secuencias: bits pares e impares; cada una de ellas se modula con una de las portadoras en cuadratura, para luego sumarlas. La señal resultante, que puede entenderse como dos BPSK ortogonales, difiere de una QPSK en la existencia de un desfase $T_b$ entre las componentes en fase y en cuadratura. Con esto se consigue que las transiciones máximas de fase sean de "$\pm\pi/2$" en lugar de los "$\pm\pi$" de la QPSK (aunque estas transiciones ocurran ahora el doble de veces). Así disminuyen las fluctuaciones de envolvente que aparecen como consecuencia del filtrado paso banda que se realiza sobre la señal resultante de la modulación, pero como consecuencia el espectro de la señal se ensancha.

La segunda de las variantes, denominada "QPSK desplazada $\pi/4$", se obtiene mediante alternancia con cada símbolo transmitido de dos constelaciones; la primera de ellas es la QPSK convencional, y la segunda la que se obtiene de rotarla $\pi/4$. Así en realidad son posibles ocho fases distintas. De esta forma las transiciones son de "$\pm\pi/4$" y "$\pm3\pi/4$" en lugar de "$\pm\pi/2$" y "$\pm\pi$" que ocurren en QPSK. De nuevo se consiguen menores variaciones de envolvente (como en OQPSK), pero además se puede realizar una demodulación no coherente.

## 2.5 MÉTODOS MIXTOS

La modulación M-PSK con más de ocho fases diferentes resulta difícil de demodular a velocidades elevadas. Para aumentar dicha velocidad en canales limitados en banda se hace necesario el uso de esquemas de modulación mixtos, de manera que se hereden las ventajas de ambos tipos y se eviten algunos de sus inconvenientes. Esto revocará en una mayor complejidad de los sistemas en la generación y detección. De todas las posibilidades la única que se verá debido a su interés es la APK, ampliamente utilizada en la práctica.

### 2.5.1 APK

Se obtiene al combinar la modulación en amplitud y de fase (*Amplitude Phase Keying*). Ofrece unos requerimientos de potencia menores que PSK para una probabilidad de error dada y alfabeto de tamaño M. La expresión general es:

$$s_i(t) = \sqrt{\frac{2E_i}{T_b}} \cos(\omega_c\ t + \phi_i)\ ,\ 0 < t \le T_b,\ \ para\ i = 1\ ,\ M$$

Nótese que varias combinaciones de amplitud ($E_i$) y fase ($\phi_i$) son posibles. Generalmente los símbolos se disponen de forma rectangular o circular.

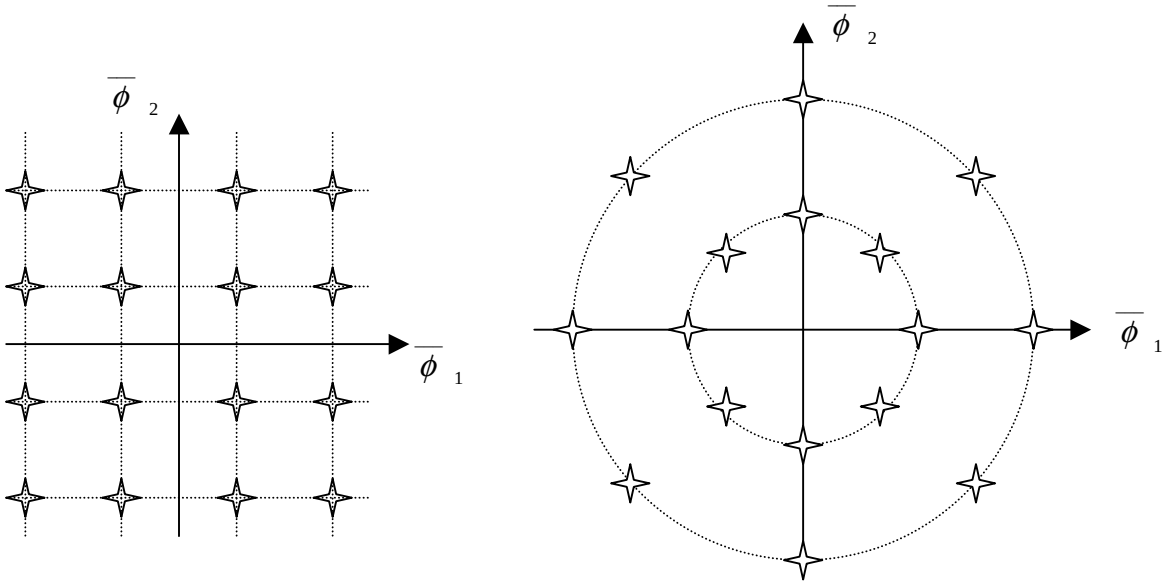


*Figura 2.12: Constelaciones 16-QAM rectangular y circular*

La señal APK puede descomponerse en dos componentes independientes con ayuda de las funciones base:

$$\Psi_1(t) = \sqrt{\frac{2}{T_b}} \cos(\omega_c\ t) \qquad \Psi_2(t) = \sqrt{\frac{2}{T_b}} \operatorname{sen}(\omega_c\ t)$$

A partir de ellas toma la expresión general:

$$s_i(t) = a_{i1}\ \sqrt{E_b}\ \Psi_1(t) + a_{i2}\ \sqrt{E_b}\ \Psi_2(t)$$

En el caso de constelación rectangular la APK puede generarse a partir de dos modulaciones L-ASK dispuestas en cuadratura, con $M=L^2$. Por eso se puede considerar la APK como una QAM M-aria o M-QAM, y se habla en general de M-QAM rectangular o M-QAM circular, según proceda, aunque se suele usar el término genérico de QAM para todos los casos de APK (ya sea rectangular o no).

Para el caso rectangular, el método de la cota de unión proporciona el resultado [6]:

$$P_e < 4\ Q\left(\sqrt{\frac{3\ \log_2 M}{2\ (M-1)}\ \frac{E_b}{N_0}}\right)$$

## 2.6 COMPARACIÓN DE SISTEMAS

La elección de una modulación depende sobre todo de la probabilidad de error ofrecida, la eficiencia del ancho de banda y la complejidad del equipo o coste.

En la figura 2.13 se representa la probabilidad de error o BER (*bit error rate*) de las modulaciones BPSK (trazo discontinuo) y ASK o BFSK (trazo continuo); de ella se desprende que con BPSK se obtiene una probabilidad de error menor que con las otras, cuando las energías medias de las modulaciones coinciden. Por otro lado para conseguir la misma probabilidad de error en ASK o BFSK que en BPSK se necesita duplicar (aumentar 3 dB) la potencia promedio.

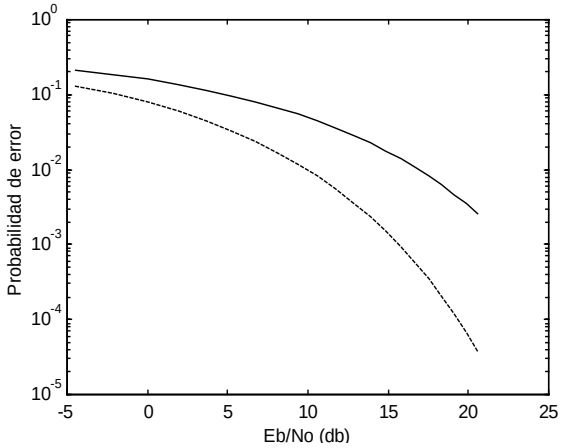


*Figura 2.13: Probabilidad de error*

En el caso de modulaciones M-arias, las prestaciones en cuanto a BER empeoran para PSK (y para ASK, aunque en menor medida) pero mejoran para FSK. Esto es así porque mientras que en PSK los símbolos se encuentran más cercanos al crecer M, en FSK se mantienen igual de espaciados (por ser señales ortogonales), pero el ancho de banda aumenta como consecuencia del aumento del número de frecuencias; esto se observa en la figura 2.14.

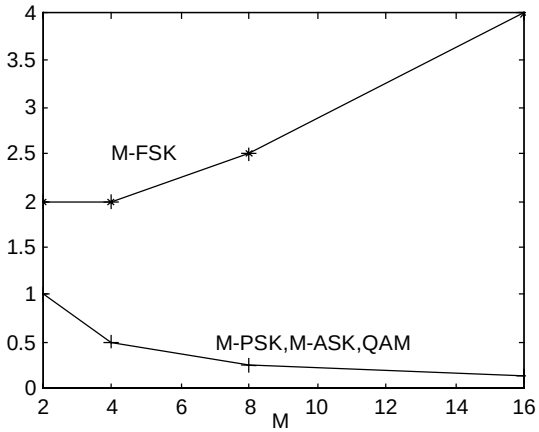


*Figura 2.14: Ancho de banda normalizado frente a M*

La comparación de las eficiencias espectrales de las diferentes modulaciones se realiza asignando un mismo valor para la probabilidad de error, que suele ser la máxima probabilidad soportable (o asumible) por el sistema. Existe un límite teórico para la capacidad que fue establecido por Shannon:

$$C = BW \log_2\left(1+\frac{S}{N}\right)$$

siendo BW el ancho de banda del canal y S/N la relación señal a ruido. Cuanto más cercano se está a este límite mejor aprovechamiento del ancho de banda del canal realizará nuestro sistema, pero a costa de una mayor complejidad para mantener fija esa probabilidad de error máxima. El mejor resultado se obtiene para 16-QAM.

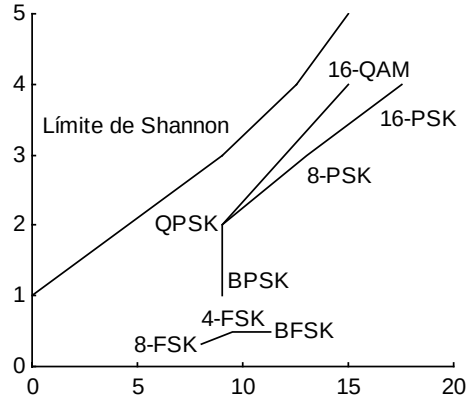


*Figura 2.15: Eficiencia en bps/Hz*

Sin embargo, existen otros factores que pueden hacer decidirse por una opción u otra. No todos los canales son lineales, ya que pueden presentarse no linealidades del tipo saturación (como por ejemplo amplificadores de ganancia no lineal). Si esto es así la elección de M-PSK y M-QAM es desaconsejable, siendo preferibles FSK, BPSK, DPSK, OQPSK o MSK.

Otro obstáculo del canal es la distorsión por retardo, que se define como la diferencia de la pendiente de la fase de la respuesta en frecuencia con la característica ideal. ASK y BPSK se comportan razonablemente bien en presencia de distorsión por retardo lineal, mientras que los métodos ortogonales (QAM, QPSK, OQPSK y MSK) se degradan en forma significativa. Para distorsión por retardo de tipo cuadrático DQPSK presenta una severa degradación, y la mejor elección es sin duda FSK.

En presencia de desvanecimientos, la característica del error de ASK y los sistemas ortogonales no se degradan con tanta rapidez como en el resto, mientras que la FSK y los sistemas diferenciales son más pobres que el promedio de dicho desvanecimiento. Respecto a las tolerancias al promedio de interferencias entre señales e ISI, BPSK y los sistemas ortogonales lo superan, mientras que M-PSK se encuentra por debajo de él.

## 2.7 SINCRONIZACIÓN

En un sistema de transmisión digital deben existir tres niveles de sincronismo: el de fase entre portadoras, de inicio de símbolo y de inicio de trama. Alguno de ellos puede no aparecer explícitamente en la onda portadora, pero debe existir dicha sincronización para que los datos puedan ser recuperados con éxito.

El sincronismo de trama es el nivel más alto de sincronización, y se hace necesario cuando la información se organiza en bloques o mensajes (tramas) con un número de símbolos uniformes. Se usan marcadores (secuencia prefijada) al inicio de la trama, y códigos especiales de sincronismo, más largos, que se detectan por correlación.

El de inicio de símbolo, también llamado recuperación de reloj o sincronización de bit, es el que determina los instantes de decisión. Puesto que la mayoría de detectores usan correladores la elección de dichos instantes debe realizarse correctamente, de manera que el máximo a la salida de ellos proporcione el valor necesario para que no haya errores. Aún en el caso de esquemas no coherentes (que no usan correladores) la precisión en los instantes de decisión es importante.

Existen dos enfoques distintos; uno emplea esquemas de lazo abierto y el otro de lazo cerrado. Los primeros trabajan directamente sobre la secuencia de datos y son más simples, mientras que los segundos son más precisos y lentos al realizar medidas comparativas entre la señal de entrada y la generada localmente en el receptor.

Por último, el sincronismo entre portadoras es necesario en una detección coherente, ya que la información en la fase es sumamente importante. Un método de obtenerlo consiste en enviar periódicamente un preámbulo conocido por el receptor antes del envío de datos; de esta forma se ajusta la fase de la portadora. Suele usarse en comunicaciones inalámbricas y vía satélite, con la finalidad de minimizar el tiempo necesario para la sincronización, pero el tiempo empleado en enviar este preámbulo reduce el tiempo efectivo disponible para enviar datos, además de consumir potencia de los equipos.

Un segundo método extrae la información necesaria para sincronizar a partir de la señal modulada. Permite una mayor eficiencia tanto en la tasa de bits como en la energía empleada, pero requieren un tiempo mayor para establecer la sincronización. Los esquemas basados en esta técnica requieren el uso de bucles o lazos enganchadores de fase (PLL o *Phase-Loop Lockback*), los cuales, dada su importancia, se describen a continuación.

### 2.7.1 PLL

Un PLL es un sistema realimentado negativamente formado por un comparador de fase, un filtro y un oscilador controlado por tensión (VCO o *Voltage-Controlled Oscilator*).

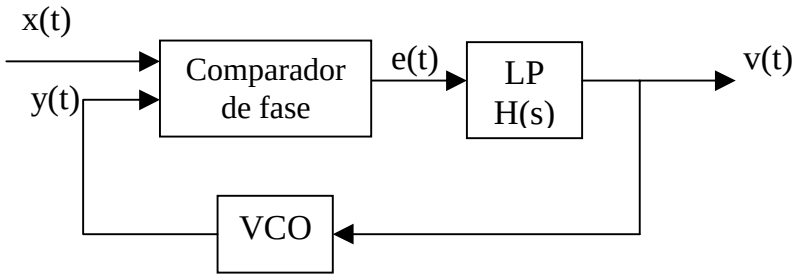


*Figura 2.16: Esquema básico PLL*

El bucle compara la fase de la señal entrante con la generada localmente en el VCO. Si la diferencia de fase de las dos señales es distinta de cero, la frecuencia de salida del VCO se ajusta en un sentido o en otro para anular esa diferencia. Si por ejemplo la frecuencia de entrada es mayor que la del VCO la diferencia de fase crece, provocando que el VCO aumente su frecuencia hasta que alcance a la de la entrada. Como la frecuencia del VCO es proporcional a la tensión a su entrada, esta tensión también es indicativa del seguimiento de frecuencia, por lo que este esquema puede usarse (y en la práctica se hace) como demodulador de frecuencia. Si se añade un integrador a la salida del PLL entonces se pueden recuperar señales moduladas en fase.

El detector de fase más simple es el formado únicamente por un multiplicador; si su ganancia es $k_m$ y se supone que las señales son senoidales:

$$\begin{cases} x(t) = A_1 \cos(w_c\, t + \phi_1) \\ y(t) = A_2 \cos(w_c\, t + \phi_2) \end{cases} \Rightarrow \; e_0(t) = \frac{k_m\, A_1\, A_2}{2}\left[\cos(\phi_1 - \phi_2) + \cos(2\omega_c\, t + \phi_1 + \phi_2)\right]$$

y el filtro (de característica paso de baja) elimina la componente en $2\omega_c$:

$$e(t) = \frac{k_m\, A_1\, A_2}{2}\cos(\phi_1 - \phi_2) = \frac{k_m\, A_1\, A_2}{2}\cos(\phi_d)$$

Esta señal es nula cuando la diferencia de fase $\phi_d$ es $\pi/2$, por lo que el bucle se dice "enganchado" cuando la entrada y la salida del VCO poseen la misma frecuencia y desfase de 90º. Si se redefine y(t) como un seno en vez de un coseno, de manera que absorba este desfase, se obtiene tras el mismo desarrollo:

$$e(t) = \frac{k_m A_1 A_2}{2} sen(\phi_d)$$

pero ahora $\phi_d$ tiende a cero para enganchar la fase.

En ocasiones se desea tener una salida al comparador que varíe de forma lineal en vez de sinusoidal con la diferencia de fase. Si las señales están saturadas o presentan una frecuencia elevada respecto al intervalo de tiempo considerado entonces pueden interpretarse como señales cuadradas. El producto de ellas es promediado en el filtro de manera que este promedio es proporcional a la fracción del tiempo que ambas son coincidentes. Si la ganancia del VCO es $k_v$ (en $seg^{-1}$) entonces:

$$\phi_2 = 2\,\pi\,k_v \int_0^t v(\tau)\,d\tau$$

Si h(t) es la respuesta impulsiva del filtro, la expresión temporal para v(t) se obtiene a partir de la integral de convolución de h(t) y e(t):

$$v(t) = \int_{-\infty}^{+\infty} e(\tau)\ h(t-\tau)\ d\tau$$

Tras derivar la primera y sustituir ésta y la anterior en $\phi_d$ se obtiene, al definir $k_0$ como la ganancia del bucle:

$$\frac{d\phi_d}{dt} = \frac{d\phi_1}{dt} - 2\,\pi\,k_0 \int_{-\infty}^{+\infty} sen(\phi_e)\ h(t-\tau)\,d\tau\ , \quad k_0 = \frac{k_v k_m A_1 A_2}{2}\ \left(seg^{-1}\right)$$

La ecuación anterior es claramente no lineal. Sin embargo, en situaciones cercanas al enganche el argumento del seno es pequeño y se puede aproximar por dicho argumento (válido en un 4% para $\phi_d$ menor a 0.5 rad) [5]; esto conduce a:

$$\phi_d(s) = \frac{1}{1+L(s)}\ \phi_1\ ,\ L(s) = \frac{2\,\pi\,k_0\,H(s)}{s}$$

L(s) se denomina función de transferencia en bucle abierto; si su ganancia es grande comparada con la unidad entonces $\phi_d$(s) tiende a cero, esto es, la fase del VCO tiende asintóticamente a la fase de la onda recibida. Sin embargo, para conocer la dinámica del PLL, o lo que es lo mismo, su capacidad para perseguir a la entrada, es necesaria la función de transferencia en bucle cerrado:

$$V(s) = \frac{k_0}{k_v}\ \frac{s\,H(s)}{s+2\,\pi\,k_0\,H(s)}\ \phi_1(s)$$

El orden del PLL viene determinado por el orden del denominador anterior, en el que juega un papel fundamental la función de transferencia del filtro H(s) empleado. El PLL más sencillo es el de primer orden y se obtiene al no usar filtro, es decir: H(s)=1 (o un valor de ganancia fijo). Así $k_0$ determina unívocamente el ancho de banda del bucle a la vez que controla la ganancia; para seguir cambios bruscos de la fase $k_0$ debe tomar valores elevados en comparación con la frecuencia de la onda recibida, pero esto aumenta el ancho de banda y con él la potencia de ruido a la salida. Si las prestaciones de este PLL no son satisfactorias entonces hay que elevar el orden:

$$H(s)=\frac{s+\frac{\omega_0}{2\pi}}{s} \Rightarrow V(s)=\frac{k_0}{k_v}\ \frac{s\left(s+\frac{\omega_0}{2\pi}\right)}{s^2+2\pi k_0+k_0\omega_0}\ \phi_1(s)$$

Y el grado de libertad introducido por $\omega_0$ permite ajustar el ancho de banda independientemente de $k_0$.

Aunque hasta ahora se han descrito PLL de naturaleza analógica no es difícil realizar el análisis para el caso discreto:

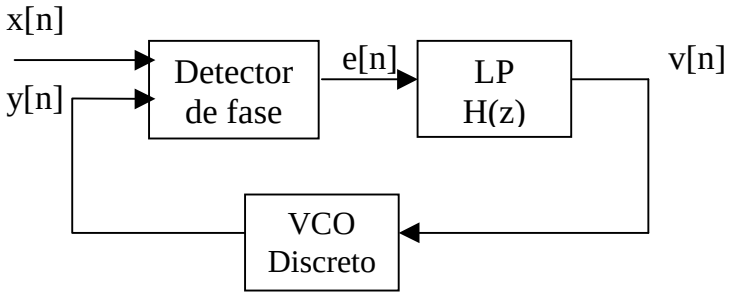


*Figura 2.17: Esquema PLL discreto*

El detector de fase y el filtro LP son versiones discretas de los equivalentes en tiempo continuo. El paso a un VCO discreto, aunque análogo a la versión continua no es tan obvio [5]. Un modelo válido es el de la figura 2.18. En él se observa que la fase del instante actual se almacena para obtener su valor en el siguiente. Además se produce tras generar la senoide una amplificación.

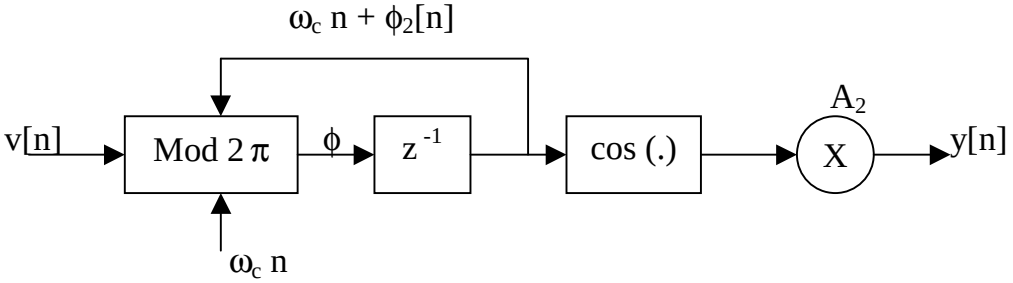


*Figura 2.18: Modelo de VCO discreto*

El modelo de PLL discreto proporciona, tras seguir el mismo desarrollo anterior:

$$\begin{cases} x[n] = A_1 \cos\left[ w_c\ n + \phi_1[n] \right] \\ y[n] = A_2 \cos\left[ w_c\ n + \phi_2[n] \right] \end{cases} \Rightarrow \quad V(z) = \frac{k_0}{k_v} \ \frac{H(z)}{H(z)+z-1} \ \Phi_1(z)$$

Para un PLL de primer orden: $L(z)=k$, y:

$$V(z) = \frac{k_0}{k_v} \ \frac{k}{k+z-1} \ \Phi_1(z)$$

Puede verse que la función de trasferencia en bucle cerrado tiene un polo en: $z=1-k$, por lo que solamente es estable sí y sólo sí: $0<k<2$. A diferencia del caso analógico hay una cota superior para la ganancia del bucle impuesta por los requerimientos de estabilidad. Además, para que la función de transferencia anterior sea de paso bajo es necesario que: $0<k<1$. Restricciones adicionales se obtienen al elevar el orden del PLL.

Para evitar estas complicaciones se usan filtros FIR *(Finite Impulse Response*, respuesta impulsiva finita) en lugar de los anteriores IIR (*Infinite Impulse Response*, respuesta impulsiva infinita), que son siempre estables. Se tiene una simulación del PLL en la figura 2.19 usando un filtro FIR paso de baja de orden 16 y poniendo a la entrada un coseno puro.

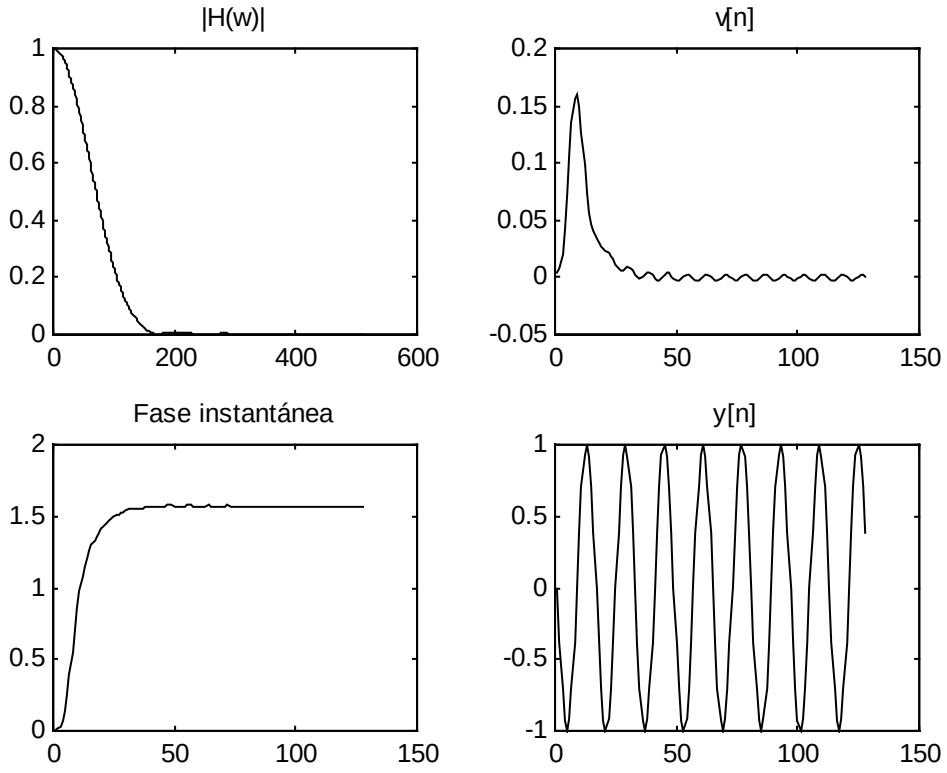


*Figura 2.19: Simulación PLL discreto ($T_s$=1/19200 seg).*

En primer lugar se tiene |H(w)|, la función de transferencia del filtro utilizado (que es de paso bajo). A continuación está la señal de error filtrada v[n] que alcanza, tras un breve (en términos relativos) transitorio, valores cercanos a cero. Nótese que su amplitud es pequeña comparada con la de x[n] (y con la de y[n]); esto se debe a que la entrada al VCO no puede ser muy grande, ya que en caso contrario se producen unas oscilaciones a su salida que impiden alcanzar un valor estable. Por esta razón con el filtrado se reduce la amplitud de v[n] de forma automática, de manera que tras generar la senoide se amplifica por medio de $A_2$ para obtener el nivel deseado, tal y como se muestra la figura 2.18.

El valor de la fase utilizada para generar la senoide en el VCO se va aproximando al valor teórico de $\pi/2$ (aproximadamente 1.57). Si se establece para la condición de enganche que el error absoluto de la diferencia entre $\phi$ y el valor en el instante anterior sea menor de una centésima de la amplitud total se consigue que dicho enganche se produzca a partir de la muestra 34, donde y[n] puede ya considerarse una versión desfasada 90º de x[n]. A partir de este momento la mejora en la precisión del enganche aumenta pero de forma casi imperceptible. Las oscilaciones que se observan en v[n] son debidas a la utilización de un esquema discreto, ya que el aumento o disminución en la salida es múltiplo de una determinada cantidad multiplicada por la ganancia, que como antes se dijo no puede ser muy grande. Para disminuir estas oscilaciones habría que aumentar la frecuencia de muestreo (lo que no siempre es posible), o disminuir la ganancia del VCO, resultando un bucle más lento y retrasando el instante de enganche.

## 2.7.2 LAZO DE COSTAS

La referencia coherente necesaria para realizar la detección no puede obtenerse con un bucle PLL ordinario si no existe un término de portadora, esto es, líneas espectrales en $\pm\omega_c$ [4]. Sin embargo se prefieren formas de modulación llamadas de portadora suprimida (SC o *Suppressed Carrier*) en las que toda la potencia se emplea en modular los datos, y no en transmitir la portadora sin modular. Es en estos casos donde se emplea el lazo de Costas.

El corazón del esquema de Costas es un PLL como el estudiado en el apartado anterior, salvo que las señales de entrada al detector de fase son un tanto especiales. El esquema completo ya en su versión discreta es:

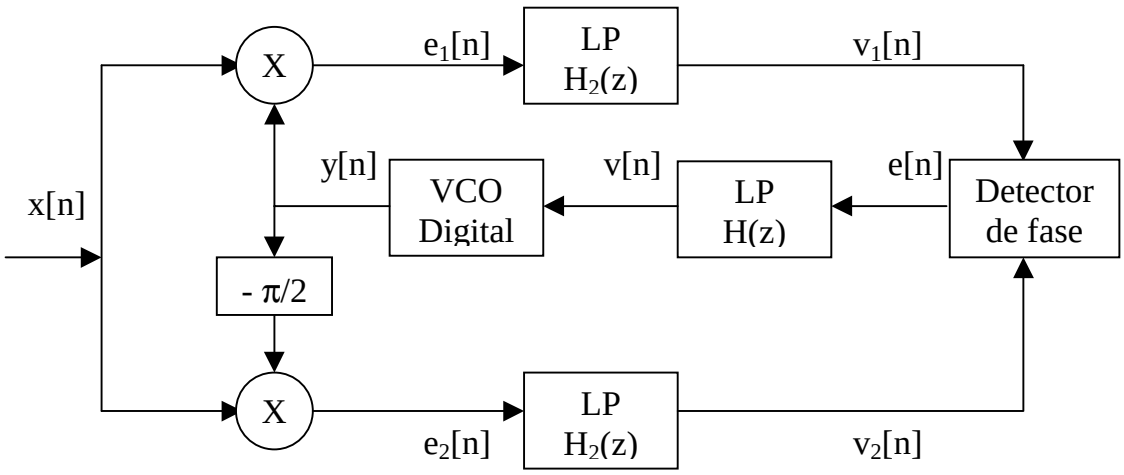


*Figura 2.20: Lazo de Costas discreto*

La señal de entrada x[n] se multiplica por la salida del VCO y por una versión retrasada de ella para dar dos señales de error, $e_1[n]$ y $e_2[n]$. Estas señales se hacen pasar por sendos filtros paso de baja para eliminar componentes de alta frecuencia y producir dos señales con alto contenido de continua, $v_1[n]$ y $v_2[n]$, que son las entradas al detector de fase (que vuelve a ser un multiplicador), generando una nueva señal de error e[n]. Tras realizarse un nuevo filtrado, v[n] es una señal aproximadamente de continua cuya amplitud genera gracias al VCO una señal senoidal que presenta un desfase $\phi$ con respecto a x[n]; la realimentación negativa provoca que este desfase tienda a cero (en realidad existe el mismo desfase de $\pi/2$ que aparecía en el PLL).

La ventaja de usar este esquema, además de no necesitar término de portadora como se comentó antes, es que presenta una mayor inmunidad al ruido, lo que en la mayoría de los casos justifica el aumento en la cantidad de proceso que hay que realizar.

# 3 HERRAMIENTAS HARDWARE

## 3.1 TMS320C30

Este dispositivo pertenece a la familia de DSP (*Digital Signal Processor* o Procesador Digital de Señales) TMS320 de Texas Instruments. A diferencia de otros dispositivos similares, su diseño se encuentra optimizado para realizar tareas de procesado de señal, lo que implica un elevado número de operaciones (multiplicaciones y sumas) sobre un gran volumen de datos. Muchas dificultades que en otros microprocesadores y microcomputadores se solventan mediante software o microcódigo en el TMS320C30 se hacen directamente por hardware.

Está basado en una arquitectura de Harvard modificada (memoria de programa y datos separada pero además permitiendo transferencias de una a otra) y la palabra tiene una longitud de 32 bits. El ciclo de instrucción es de 60 nanosegundos y se alcanzan los 16.7 millones de instrucciones por segundo y los 33.3 millones de operaciones en punto flotante por segundo. Esto se debe a que el paralelismo permite que en una misma instrucción se puedan realizar dos operaciones con operandos flotantes. Además tomando ciertas precauciones a la hora de programar se puede conseguir un funcionamiento efectivo de la segmentación (*pipeline*) incorporada en la ejecución de instrucciones, lo que eleva aún más el rendimiento que se puede conseguir. El esquema general puede verse en la figura 3.1.

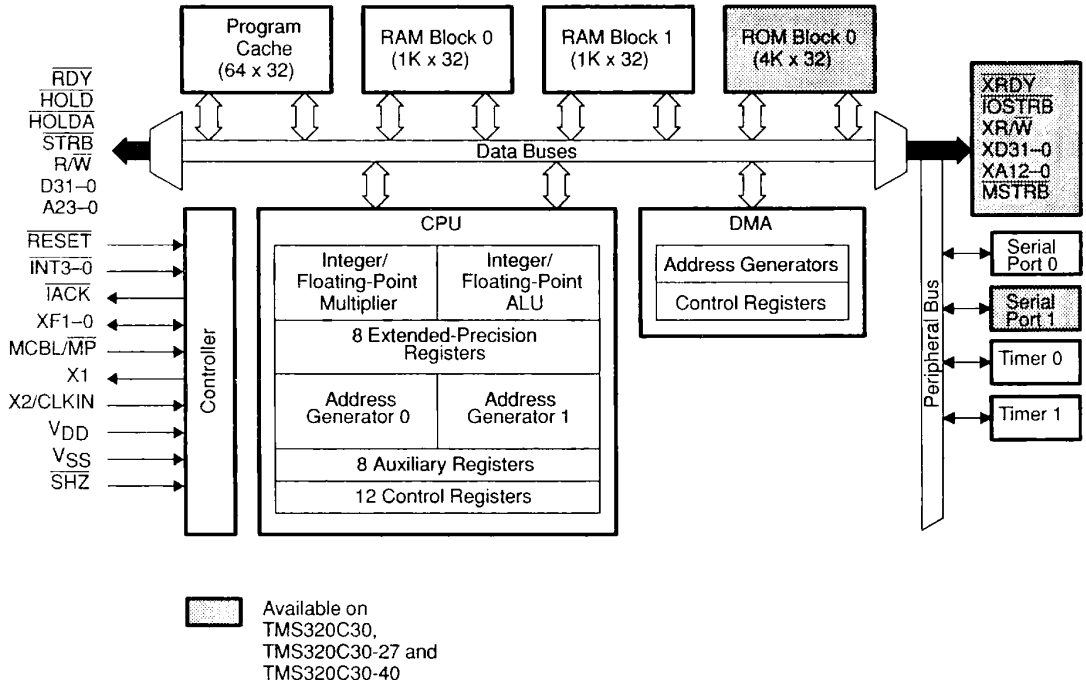


*Figura 3.1: Diagrama de bloques de la familia TMS320*

## 3.1.1 CPU (UNIDAD CENTRAL DE PROCESO)

El corazón del TMS320C30 lo compone su CPU (*Central Processing Unit* o Unidad Central de Proceso) que posee una arquitectura basada en registros. La forman varias unidades más pequeñas interconectadas mediante varios buses, los cuales además comunican con el "exterior". En la figura 3.2 se observa un detalle de la CPU y las partes que la componen, las cuales se describen a continuación.

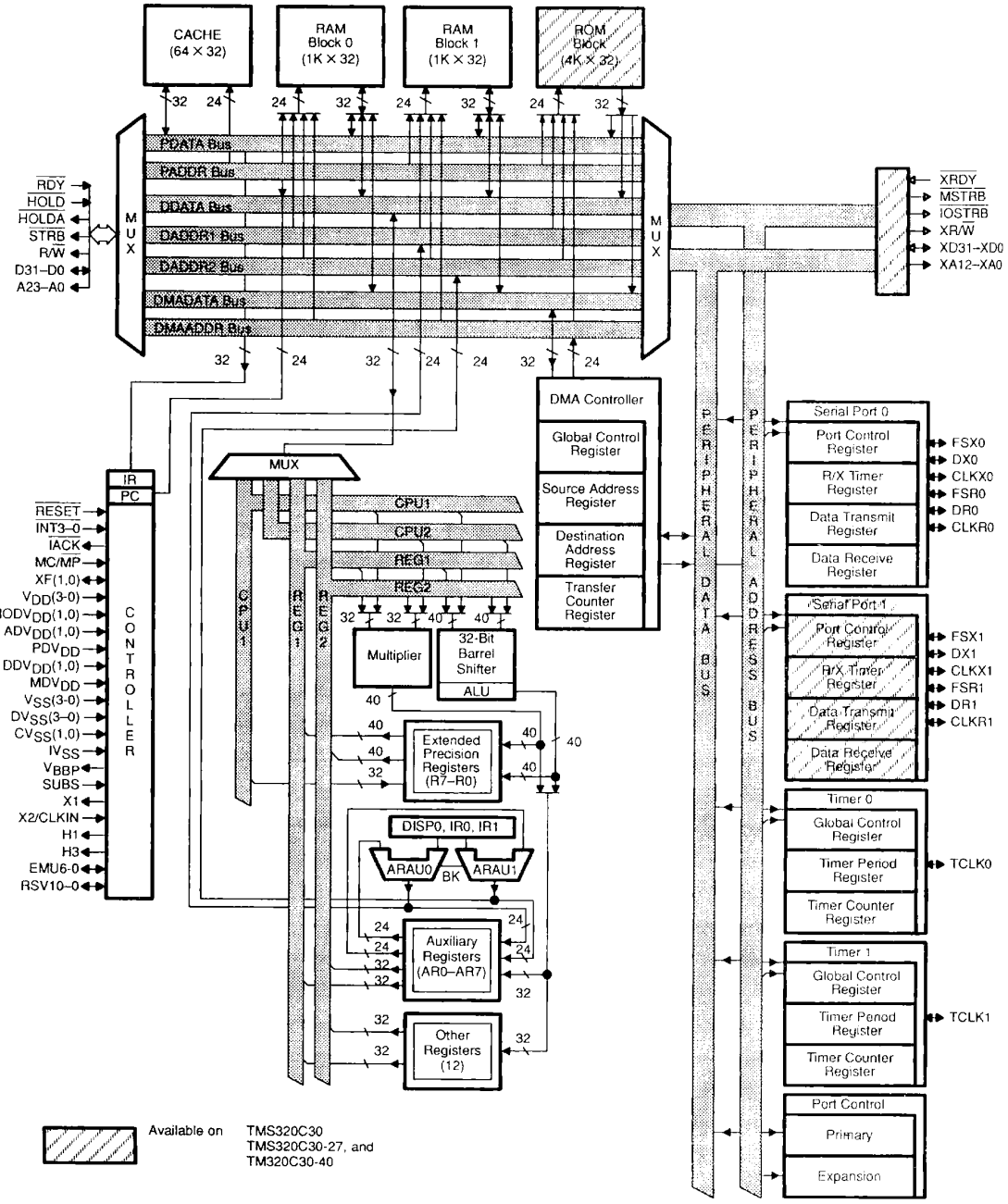


*Figura 3.2: Unidad Central de Proceso (CPU)*

#### 3.1.1.1 Multiplicador

Realiza multiplicaciones en un único ciclo sobre valores enteros de 24 bits o flotantes de 32 bits. En multiplicaciones sobre enteros el resultado es un número de 32 bits. En el caso de flotantes el resultado tiene 40. La implementación de la aritmética de punto flotante permite que las operaciones de este tipo se produzcan a velocidades equivalentes a las de punto fijo a través de un ciclo de instrucción de 50 nseg y un alto grado de paralelismo. Además mediante una normalización adecuada se consiguen reducir los errores debidos al uso de aritmética de precisión finita

#### 3.1.1.2 ALU (Unidad Aritmético-Lógica)

Ejecuta operaciones de un solo ciclo sobre datos enteros (32 bits), lógicos (32) y flotantes (40), incluyendo conversiones de entero a flotante y viceversa. En el caso de resultados de 32 bits, estos se alinean correctamente sobre la salida (de 40 bits).

#### 3.1.1.3 ARAUs (Registros Auxiliares de la Unidad Aritmética)

Son dos; cada uno puede generar una dirección en un ciclo de reloj y operar en paralelo con el multiplicador y la ALU. Además soportan varios tipos de direccionamiento haciendo uso de unos registros especiales, como se verá más adelante.

#### 3.1.1.4 Registros

Existen 28 registros asociados a la CPU. Todos pueden ser operados por el multiplicador o la ALU, y pueden usarse como registros de propósito general. Además todos ellos tienen funciones especiales:

- R0-R7: Registros de precisión extendida. Pueden almacenar y soportar operaciones sobre números enteros de 32 bits o de punto flotante de 40 bits.

- AR0-AR7: Registros auxiliares. Modificables por las ARAUs y capaces de generar direcciones de 24 bits. También pueden servir como contadores en bucles con direccionamiento indirecto.

- DP: Puntero de página de datos. Los 8 bits menos significativos se usan para direccionamiento directo como puntero a la página de datos accedida. Por lo tanto hay un total de 256 páginas ($2^8$), lo que divide la memoria total del DSP en páginas de 64K palabras de longitud.

- IR0, IR1: Registros índices. Usados por las ARAUs para realizar el direccionamiento indexado.

- BK: Registro de tamaño de bloque. Indican a las ARAUs el tamaño del bloque de datos cuando se realiza el direccionamiento circular.

- SP: Puntero a la pila del sistema. Sus 24 bits menos significativos contienen la dirección superior de la pila de sistema.

- ST: Registro de estado. Almacena información global relativa al estado de la CPU. Posee el flag de activación global de interrupciones (GIE), controla la activación, borrado y congelación de la caché, y refleja ciertos aspectos de los resultados de algunas operaciones (resultado cero, negativo, "overflow"...).

- IE: Registro de activación de interrupciones CPU/DMA. Posibilita la activación de cada una de las fuentes de interrupción de la CPU o la DMA. Para que se pueda producir una interrupción debe además estar activado GIE (en el registro ST).

- IF: Registro de flags de interrupción. Reflejan si una fuente de interrupción está activa o no. Pueden generarse interrupciones mediante software.

- IOF: Registro de flags de entrada/salida. Controla si los pines XF0 y XF1 (que son independientes) se configuran como entradas o salidas, y almacena el dato a leer o escribir en ellos.

- RC, RS, RE: Registros para repetición de bloques. RC especifica el número de veces que se repite un bloque, y RS y RE son las direcciones de comienzo y final del bloque a repetir.

- PC: Contador de programa. Contiene la dirección de la siguiente instrucción a leer de la memoria de programa.

## 3.1.2 MEMORIA

La memoria total que es capaz de direccionar es de 16 millones de palabras, pudiendo localizarse en cualquier lugar el programa o los datos. Sin embargo existen algunas direcciones reservadas, ya que el TMS320C30 posee internamente dos bloques de RAM de 1K x 32 bits cada uno y uno de ROM de 4K x 32. Cada uno de ellos es capaz de soportar dos accesos por parte de la CPU en un ciclo. La separación de los buses de programa, datos y DMA permite un gran paralelismo de operaciones; por ejemplo la CPU podría leer dos datos de un bloque RAM y coger una instrucción de programa desde memoria externa mientras el controlador de DMA escribe en el otro bloque RAM, todo en un único ciclo de instrucción.

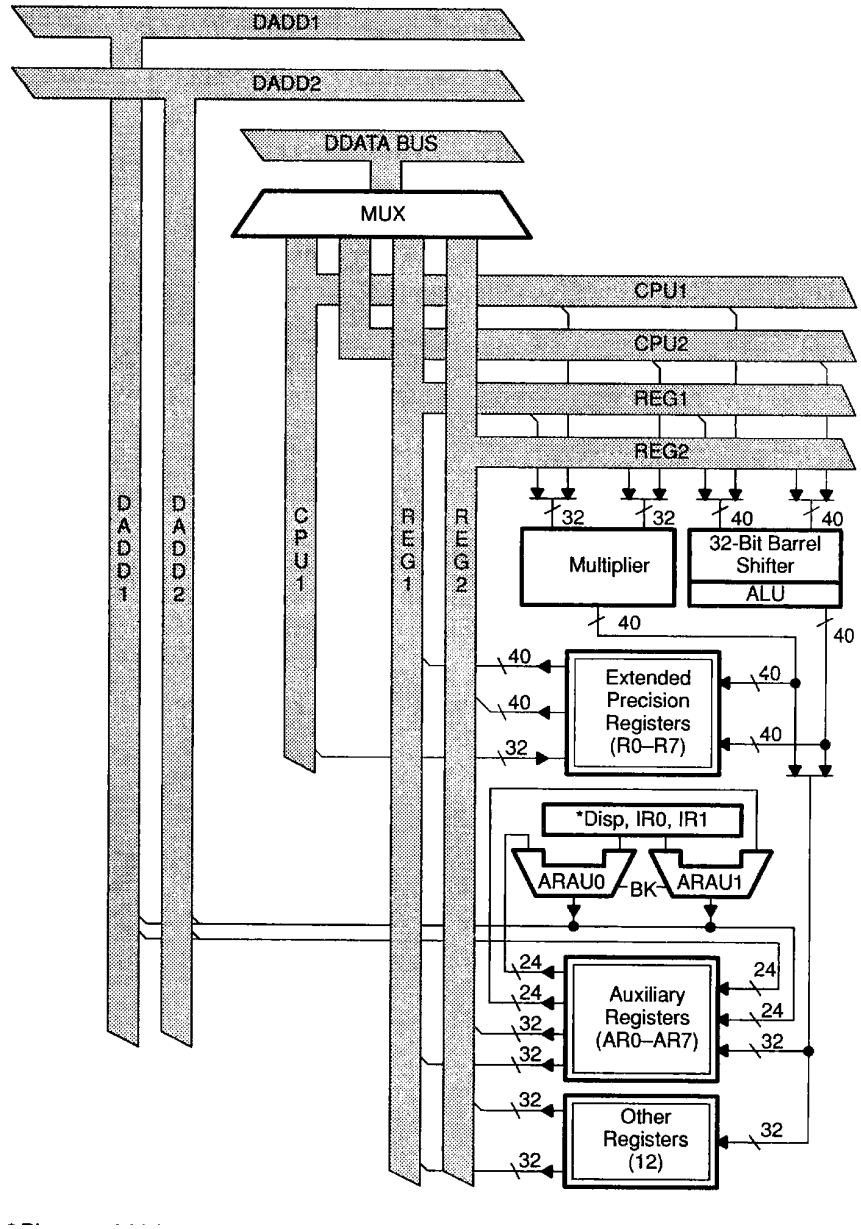


*Figura 3.3: Unidad Central de Proceso (CPU)*

Las direcciones de la ROM interna pueden mapearse hacia memoria externa si se configura el DSP como microprocesador; en caso contrario (modo microcontrolador) los accesos a los primeros 4K son dirigidos hacia la ROM interna. Las 64 primeras posiciones en ambos casos están reservadas para las direcciones de reset, vector de interrupciones y rutinas de excepciones o "traps".

#### 3.1.2.1 Caché

Una caché para instrucciones (64 palabras de RAM) distribuida en dos bloques o segmentos almacena las secciones de código repetidas frecuentemente, reduciendo el número de accesos a la memoria de programa. Esto permite almacenar el código en memoria externa, más lenta pero también más barata. Además libera los buses de estos accesos y pueden utilizarse por otros recursos, como la DMA.

El algoritmo encargado de actualizar la caché recibe el nombre de LRU (*Least Recently Used* o menos usado recientemente). Si el código es automodificable debe tenerse la precaución de limpiar la caché, ya que la copia no se modifica. Cuando se accede a memoria de datos o memoria interna de programa esta memoria se encuentra desactivada (no se usa).

#### 3.1.2.2 Direccionamiento

Existen seis tipos distintos de direccionamiento que permiten el acceso a los registros, los datos y las instrucciones. Estos se encuentran asociados en cinco grupos o modos de direccionamiento, ya que un tipo puede ser apropiado para un conjunto de instrucciones pero no para otras. Los tipos son:

- Registro: La dirección es la de un registro de la CPU.

- Directo: A los 8 bits menos significativos del registro DP se les añaden los 16 menos significativos especificados en la instrucción para formar la dirección de 24 bits.

- Indirecto: Los 24 bits menos significativos de un registro auxiliar indican la dirección del operando. Existen variantes que permiten operaciones de pre y pos incremento o decremento (de una unidad por defecto, o especificando en la instrucción un entero sin signo de 8 bits), que se pueden combinar con un índice mediante el uso de los registros de índice. Existen además dos variantes muy interesantes a la hora de implementar algunos algoritmos:

- Direccionamiento de bit inverso: mejora la velocidad de ejecución y disminuye la memoria de programa. Algoritmos como la FFT necesitan poder acceder a los datos de una forma especial y no en la forma lineal en la que están almacenados en realidad. Esa forma de acceso se convierte en natural para el DSP ya que la inversión de bits se realiza por hardware en las ARAUs; en otro caso quedaría por cuenta del programador realizarlo por software, consumiendo capacidad de cálculo y ocupando algunas instrucciones de programa.

- Direccionamiento circular: Es posible definir un bufer en memoria de manera que tras acceder al último elemento se vuelva a direccionar el primero. El registro BK almacena el tamaño del bufer (R) y su dirección de comienzo debe tener sus K bits menos significativos a cero, siendo K el menor entero que verifica: $2^K$>R. Los datos pueden accederse de esta manera como si se tratase de una ventana deslizante, donde datos más recientes van sustituyendo a otros más antiguos, ya procesados. Algoritmos como el cálculo de una correlación o convolución sacan mucho provecho de este tipo de direccionamiento indirecto.

➢ Inmediato corto: Un valor de 16 bits se especifica en la instrucción. Dependiendo del tipo de instrucción se espera un entero corto, un entero sin signo o un flotante corto.

➢ Inmediato largo: 24 bits de la instrucción especifican la dirección. Generalmente se usan para codificar las instrucciones de control del programa.

➢ Relativo al PC: El desplazamiento se almacena como un entero con signo de 16 bits, y su valor depende de si el salto es normal o retardado.

### 3.1.3 OPERACIONES DE BUS

Gran parte del elevado rendimiento que alcanza este DSP se debe a la variedad de buses internos independientes. Existen dos buses de programa, tres de datos y dos de DMA, además de dos adicionales para llevar datos desde los registros al multiplicador o la ALU. Salvo estos dos últimos, todos se encuentran conectados a los buses externos, llamados principal y de expansión, de manera que los accesos simultáneos no se limitan a memoria interna. Esto puede verse en detalle en la figura 3.4.

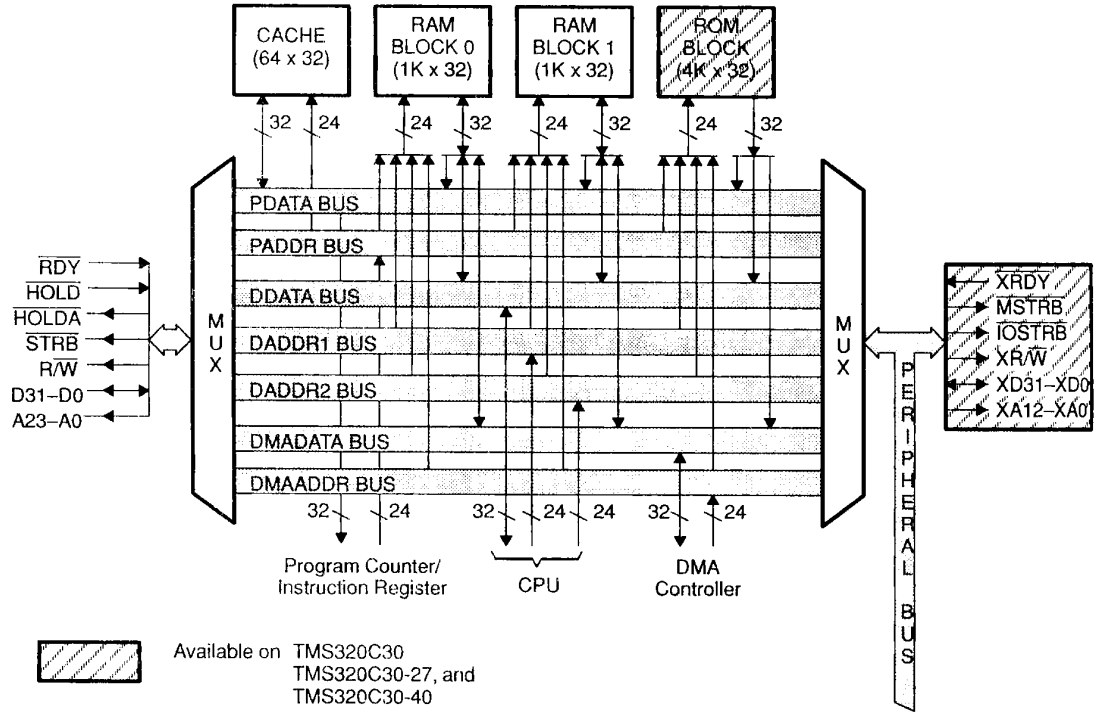


*Figura 3.4: Esquema de los buses del TMS320C30*

El hecho de que estos dos buses se puedan configurar cada uno con su propio número de estados de espera permite que periféricos lentos en el bus de expansión no ralenticen accesos a dispositivos más rápidos en el bus principal.

Existen cuatro interrupciones externas, varias internas y una de RESET externa no enmascarable, que pueden usarse para interrumpir a la DMA o a la CPU.

Se permite el uso conjunto de varios DSP mediante dos flags externos que gobiernan el procesamiento en paralelo. Estos flags pueden configurarse también como pines de entrada/salida de propósito general, en los casos en que sólo esté presente un dispositivo.

### 3.1.4 PERIFÉRICOS

Todos los elementos son controlados mediante registros mapeados en memoria en el bus de periféricos, compuesto a su vez de un bus de direcciones y otro de datos. Existe un controlador de DMA, dos timers y dos puertos serie.

El controlador DMA puede leer o escribir en cualquier posición de la memoria sin interferir en la CPU, ya que dispone de su propio generador de direcciones (registros de origen y destino) y registro contador. Esto permite acceder a memorias externas lentas o periféricos sin reducir la velocidad de proceso.

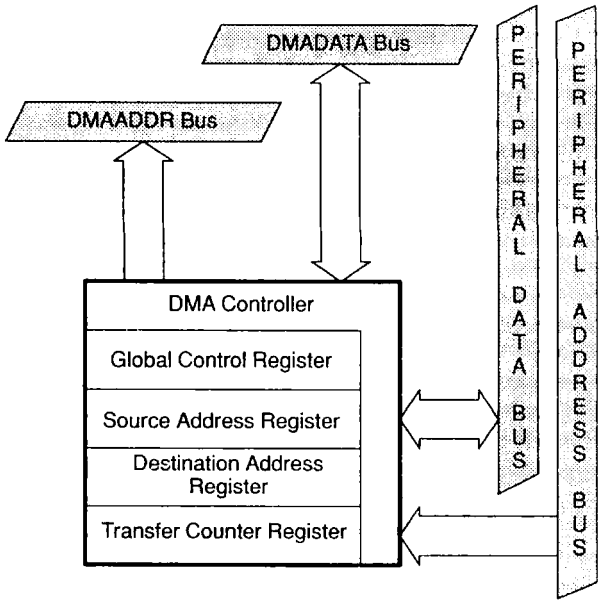


*Figura 3.5: Estructura y conexiones del dispositivo de DMA*

Los timers son relojes o contadores de eventos de 32 bits con dos modos de señalización y temporización interna o externa. Como fuente para generar la salida se usa el reloj del sistema dividido a la mitad (15 MHz); en modo reloj un valor "n" en el registro de periodo indica la frecuencia de cambio de nivel de la señal de salida (es decir, será un reloj de frecuencia mitad de 15/n MHz), mientras que en modo pulso la salida será un pulso de duración 1/30 μseg con una frecuencia de 15/n MHz. Cada uno de los Timers posee un pin de entrada/salida que puede usarse como entrada para el reloj o como salida gobernada por él. También se puede configurar como pin entrada/salida de propósito general.

Los dos puertos serie bidireccionales son totalmente independientes. Cada uno puede configurarse para enviar 8, 16, 24 ó 32 bits cada vez. La fuente de reloj para cada puerto serie puede ser interna o externa. Ambos poseen un divisor de reloj interno, caso de necesitarse. Los pines de estos puertos son configurables para entrada/salida generales o incluso como timers.

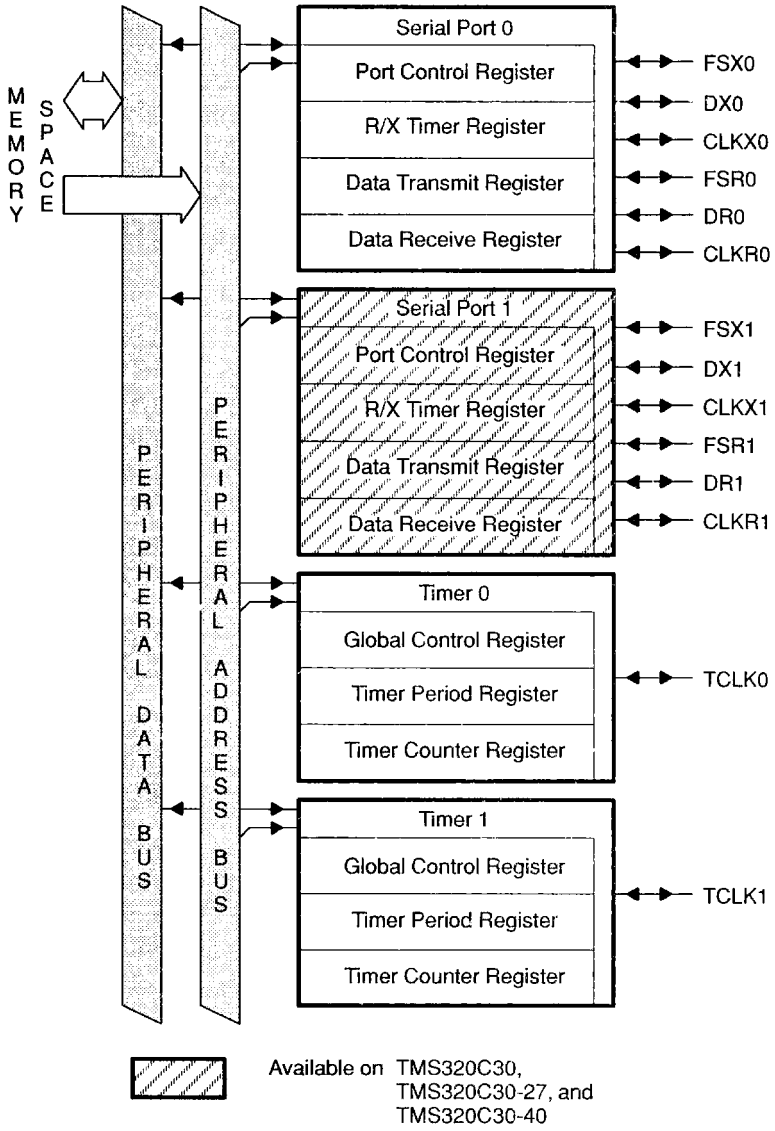


*Figura 3.6: Estructura y conexiones de los Timers y Puertos Serie*

## 3.1.5 FORMATOS NUMÉRICOS

Se admiten dos tipos de datos básicos: entero y flotante Sin embargo cada uno de ellos puede adoptar varios formatos distintos.

Todos los formatos flotantes comparten una estructura común. Un bit de signo (s) y el complemento a dos de la fracción (f) se combinan para formar la mantisa, que representa un número también en complemento a dos, pero que al estar normalizado proporciona un bit más de precisión. Un exponente (e) completa el número. Los distintos formatos varían en el número de bits que se asignan a "e" y "f". El número (x) se forma de la siguiente manera:

$$\begin{cases} x = 01.f \times 2^e, \text{ si s=0 o el primer cero es el bit de signo y el bit implícito es "1".} \\ x = 10.f \times 2^e, \text{ si s=1 o el primer uno es el bit de signo y el bit implícito es "0".} \\ x = 0, \text{ si e = mayor valor negativo en complemento a dos representable en "e".} \end{cases}$$

La lista completa de formatos es la siguiente:

- Entero corto: Complemento a dos de 16 bits de operandos inmediatos enteros, con extensión del signo a 32 bits. El rango es $-2^{15}$ a $2^{15}-1$ (= 32767), ambos inclusive.

- Entero de precisión simple: Representado en complemento a dos de 32 bits. El número es: $-2^{31} \leq x \leq 2^{31}-1$ (= $2.147483647 \times 10^9$).

- Entero sin signo corto: Comprende los enteros de 0 a $2^{16}-1$ (= 65535). Los 16 bits más significativos se rellenan con ceros.

- Entero de precisión simple sin signo: Entre 0 y $2^{32}-1$ (= $4.29496 \times 10^9$).

- Flotante corto: 16 bits, reservando 4 para el exponente y 12 para la mantisa, con extensión del signo a 32 bits. Los valores límite son:

| | |
|---|---|
| Mayor positivo = $(2-2^{-11}) \times 2^7$ | = $2.5594 \times 10^2$ |
| Menor positivo = $1 \times 2^{-7}$ | = $7.8125 \times 10^{-3}$ |
| Menor negativo = $(-1-2^{-11}) \times 2^{-7}$ | = $-7.8163 \times 10^{-3}$ |
| Mayor negativo = $-2 \times 2^7$ | = $-2.5600 \times 10^2$ |

- Flotante de precisión simple: De los 32 bits se usan 8 para el exponente y 24 para la mantisa.

| | |
|---|---|
| Mayor positivo = $(2-2^{-23}) \times 2^{127}$ | = $3.4028234 \times 10^{38}$ |
| Menor positivo = $1 \times 2^{-127}$ | = $5.87747717 \times 10^{-39}$ |
| Menor negativo = $(-1-2^{-23}) \times 2^{-127}$ | = $-5.8774724 \times 10^{-39}$ |
| Mayor negativo = $-2 \times 2^{127}$ | = $-3.4028236 \times 10^{38}$ |

➢ Flotante de precisión extendida: Puesto que se necesitan 40 bits, es necesario tomar algunas precauciones si no se quiere perder resolución, ya que los registros (salvo los R0-R7) y las posiciones de memoria sólo tienen 32. Se reservan 8 para el exponente y 32 la mantisa.

Mayor positivo = $(2-2^{-31}) \times 2^{127} = 3.4028236683 \times 10^{38}$
Menor positivo = $1 \times 2^{-127} = 5.87747717541 \times 10^{-39}$
Menor negativo = $(-1-2^{-31}) \times 2^{-127} = -5.8774717569 \times 10^{-39}$
Mayor negativo = $-2 \times 2^{127} = -3.4028236691 \times 10^{38}$

Todas las multiplicaciones de flotantes trabajan con formato de precisión simple, (aunque el resultado se exprese en formato extendido). Si los operandos no lo son se realiza una extensión o truncamiento en el caso de que estén en formato corto o extendido, respectivamente. Todas estas conversiones se realizan por hardware y no consumen capacidad de cálculo.

Adicionalmente existen instrucciones software para cambiar de entero a flotante y viceversa, normalizar un flotante de precisión extendida y pasar flotantes de precisión extendida a simple, que pueden ser usadas en cualquier momento del programa.

## 3.2 MÓDULO DE EVALUACIÓN

El módulo de evaluación o "EVM" es un dispositivo hardware montado en una tarjeta que se inserta en una bahía libre tipo "ISA" de un ordenador convencional. Posee un puerto serie, una entrada analógica y una salida, también analógica. Su diseño se ha optimizado para el procesamiento digital de señales en tiempo real, generalmente para el tratamiento de la voz, por lo que los elementos que posee están orientados a ese fin. En la figura 3.7 se observa el esquema de la tarjeta de evaluación con los bloques que la forman; algunos de ellos trabajan con señales analógicas y otros con señales digitales.

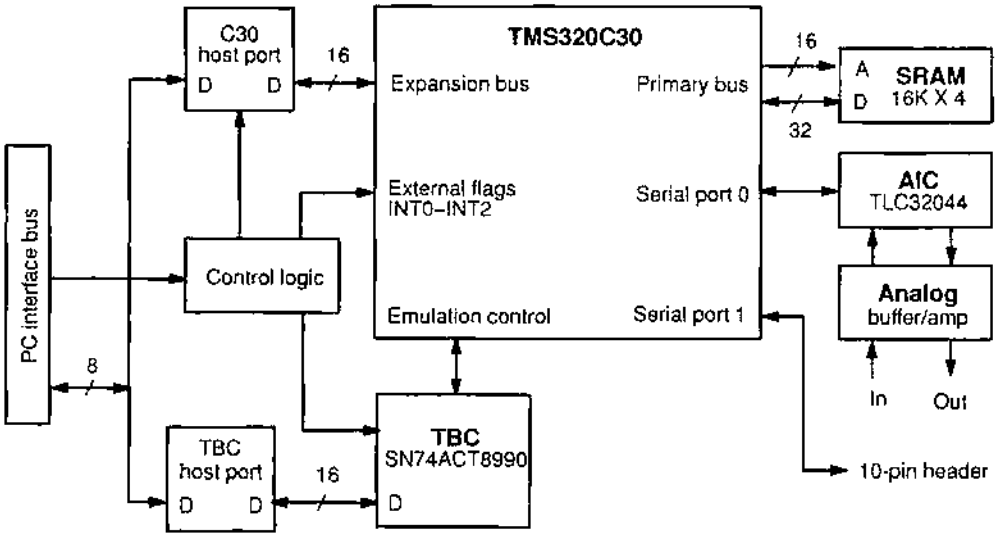


*Figura 3.7: Diagrama de bloques EVM*

### 3.2.1 PROCESAMIENTO DIGITAL

El corazón de la tarjeta lo forma un TMS320C30 que es el que realmente ejecuta el código del programa y gobierna la tarjeta. A él se conecta directamente a través del bus primario una memoria de 16K x 32 bits de tipo SRAM y 35 nseg de tiempo de acceso para almacenar datos o programa. Soporta ser accedida sin ciclos de espera pero no dispone de doble puerto, por lo que no puede ser accedida de forma directa desde el lado del ordenador.

Todo el código de programa para la tarjeta es introducido mediante un controlador de bus denominado TBC (*Test Bus Controller*), el cual se conecta con el puerto de emulación del TMS320C30. Una vez que el código se ha cargado, el ordenador y el DSP se comunican mediante un registro compartido bidireccional de 16 bits, localizado físicamente en la tarjeta, que provee una transferencia con un ancho de banda de aproximadamente 200K bytes por segundo. Ha sido diseñado para beneficiarse del canal DMA del DSP y por lo

tanto puede ser gobernado mediante interrupciones (INT0-INT2). Este registro bidireccional conecta los 8 bits inferiores de los bits de datos del bus de entrada/salida del ordenador con los 16 bits inferiores del bus de expansión del DSP, de manera que leyendo y escribiendo en dicho registro se pueden comunicar ambos dispositivos.

La tarjeta provee al puerto serie 1 del DSP de un conector exterior para que pueda ser usado por las aplicaciones que así lo requieran.

### 3.2.2 PROCESAMIENTO ANALÓGICO

La parte analógica se fundamenta en un controlador de interfaz analógica denominado AIC (*Analog Interface Circuit* o Circuito de Interfaz Analógico); en concreto se trata de un TLC32044, el cual posee incorporados dentro del propio chip un convertidor D/A (Digital/Analógico) y otro A/D (Analógico/Digital) con un rango dinámico de 14 bits y frecuencia de muestreo variable, filtros antialiasing a la entrada y la salida, un filtro de entrada de paso alto, un filtro corrector a la salida, y un nivel de entrada variable. Dos conectores tipo RCA proporcionan una entrada y una salida analógica, cuyas ganancias se encuentran fijadas en los niveles estándar de señales de audio en línea.

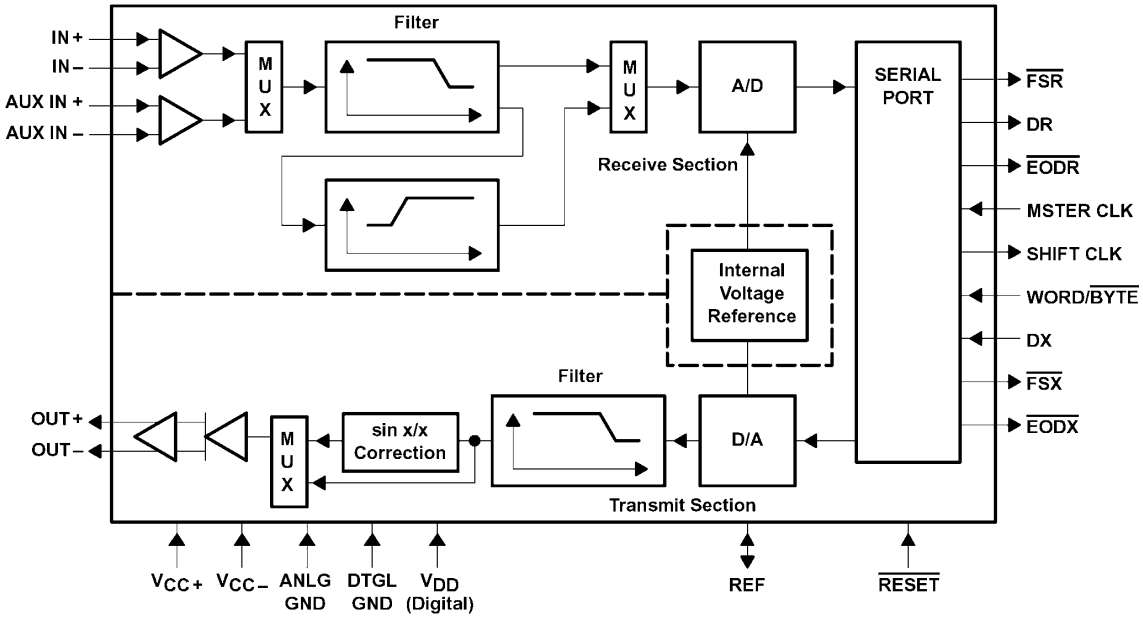


*Figura 3.8: Diagrama de bloques funcional del AIC*

La etapa de entrada consiste en un seguidor de tensión seguida de una etapa de amplificación, cada una formada por un TL072. El primero aísla la entrada del amplificador, mientras que el segundo proporciona una amplificación adecuada para maximizar el rango dinámico.

El filtro antialiasing lo componen un filtro de paso bajo transicional Chebyshev-Elíptico de orden ocho cuya frecuencia de corte es configurable (dentro de unos límites), y uno de paso alto del mismo tipo pero de cuarto orden que elimina las frecuencias por debajo de 100 Hz, siendo este último opcional. En cuanto a la etapa de salida el filtro reconstructor está formado por un filtro de paso bajo idéntico al anterior, seguido por un filtro corrector sen x/x, que puede ser usado o no (caso de no usarse debe realizarse la corrección mediante software). Por último se encuentra un amplificador de potencia LM386, que es capaz de alimentar cargas de baja impedancia a frecuencias de audio.

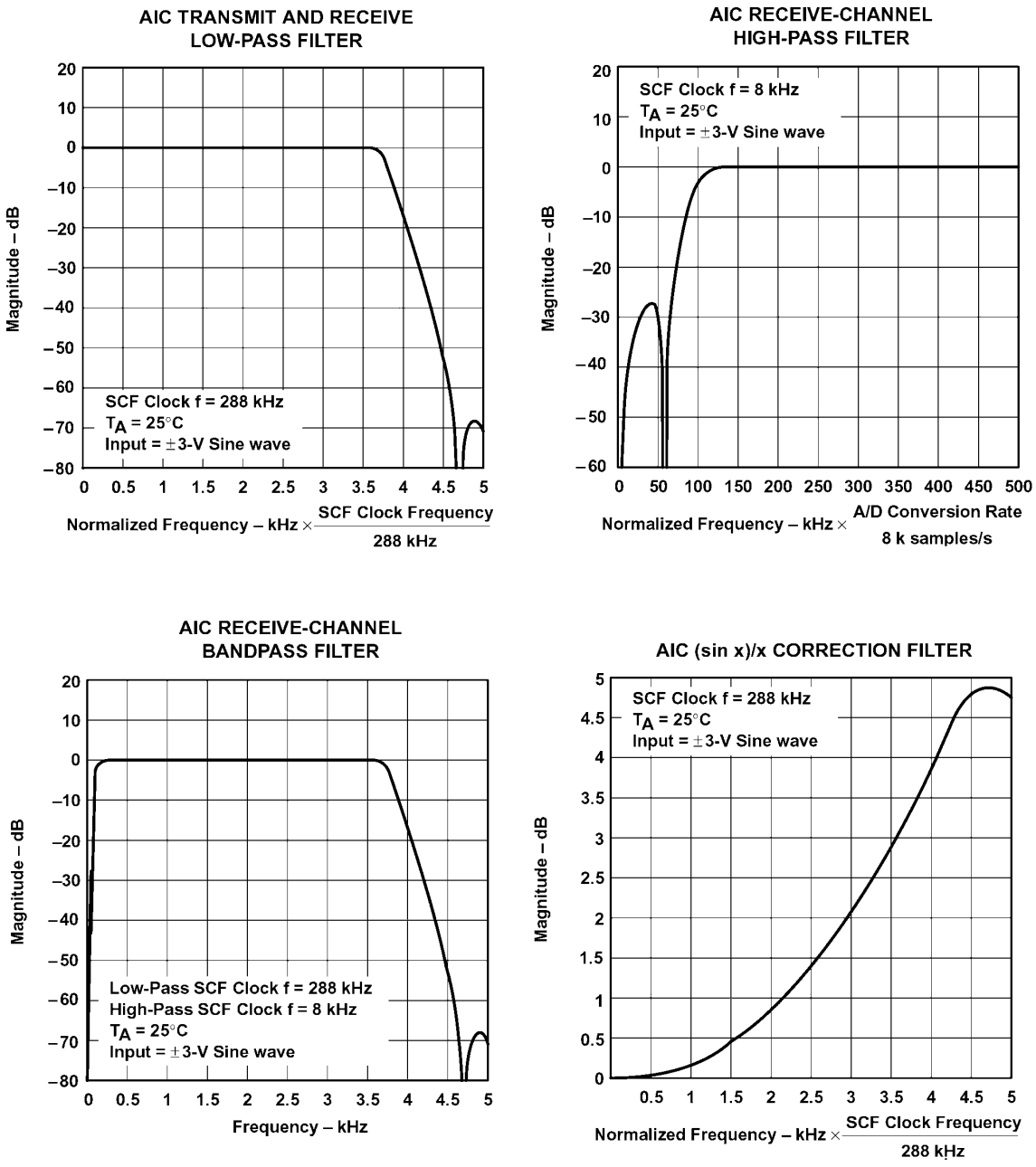


*Figura 3.9: Funciones de transferencia de los filtros LP, HP, BP y sen x/x del AIC*

El AIC está conectado físicamente con el DSP mediante el puerto serie 0. El DSP gobierna el reset del AIC mediante el pin XF0 y le proporciona una señal de reloj a través del timer 0. Ambos se comunican a través de dos registros mapeados en memoria: el registro de transmisión (localizado en 0x808048h) y el registro de recepción (en 0x80804Ch), que en funcionamiento normal almacenan los valores de entrada y de salida. El registro de transmisión permite que el DSP pueda configurar el AIC, lo que recibe el nombre de "transmisión secundaria".

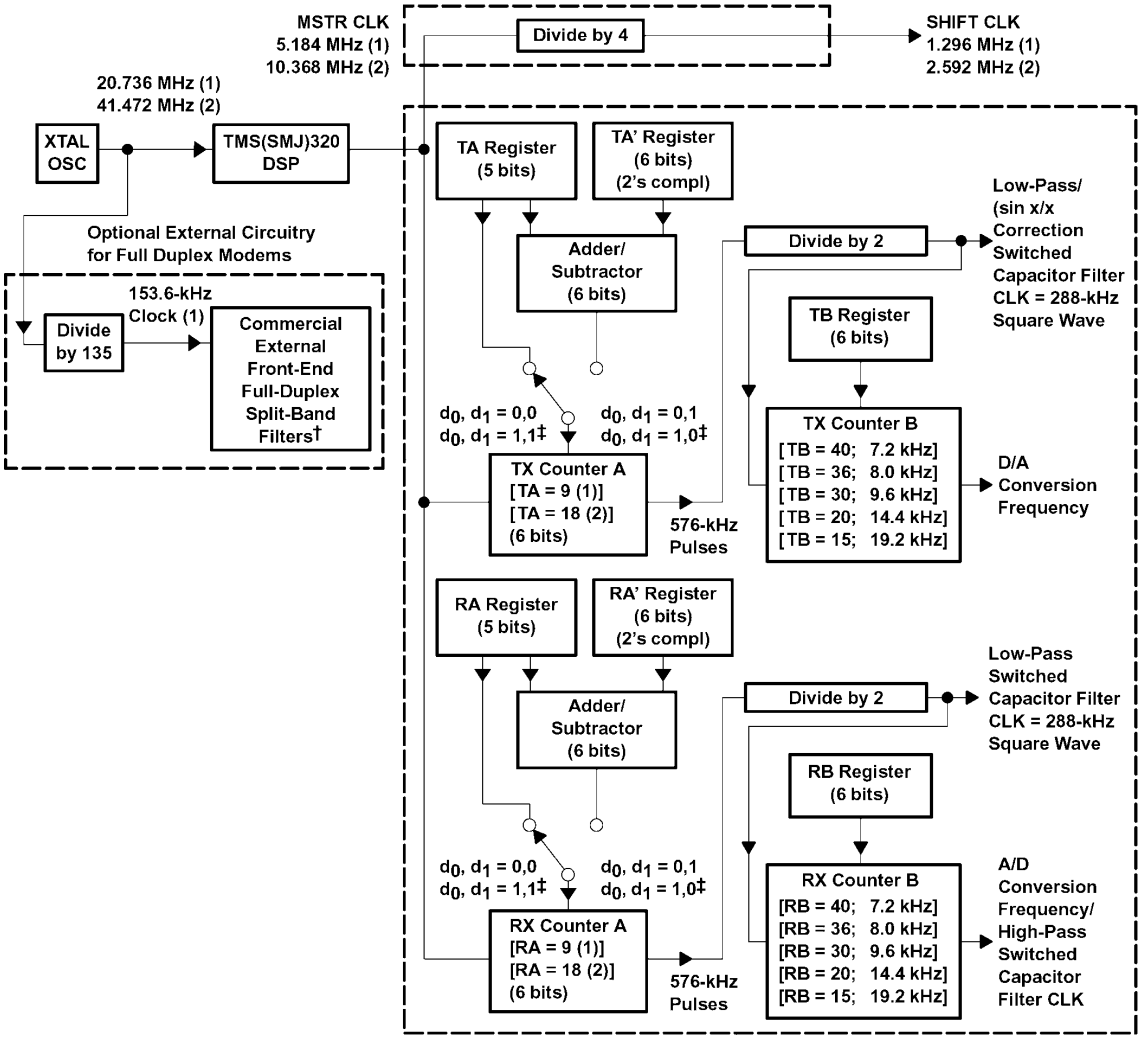


*Figura 3.10: Esquema de temporización interna del AIC*

El tamaño de la palabra para el AIC es de 16 bits (para el DSP es de 32); los 14 bits más significativos contienen el complemento a 2 de 14 bits del dato, mientras que los 2 bits menos significativos valen cero para los datos recibidos desde el AIC. Es en el envío de datos al AIC o transmisiones cuando estos 2 bits adquieren un significado de control. Cuando estos bits toman el valor 11b se está indicando al AIC que la transmisión siguiente será una transmisión secundaria, con lo que el dato siguiente que se envíe deberá ser reconocido como un comando de configuración, siendo los dos bits menos significativos de la transmisión secundaria los que determinan el destino de los otros 14.

El AIC posee otros dos registros más, llamados A y B, que se utilizan para determinar la frecuencia de muestreo $F_S$ y la frecuencia de corte $F_{LP}$ del filtro antialiasing de paso bajo. Los valores de A y B se generan de acuerdo al siguiente par de ecuaciones:

$$F_{LP} = \frac{F_{CLK}}{160A} \quad , \quad F_S = \frac{F_{CLK}}{2AB}$$

El valor de $F_{CLK}$ es la frecuencia principal de reloj derivada del timer 0 del TMS320C30. Generalmente el proceso que se realiza parte fijando en primer lugar el valor de B, que suele elegirse como 36; con ello se consigue una relación fija (y adecuada) entre la frecuencia de muestreo y la frecuencia de corte del filtro, de manera que $F_{LP} = 0.45\ F_s$. Una vez hecho esto sólo queda elegir la $F_s$ deseada para obtener el valor de A, ya que $F_{LP}$ quedará ajustada de forma automática.

Existe un último registro en el AIC que debe ser configurado, denominado registro de control de estado, que determina varios modos de funcionamiento. A modo de ejemplo la configuración resultante se muestra en la tabla 3.11 para el valor 0x02A7h.

| Bit | Estado | Efecto |
|---|---|---|
| 2 | 1 | Habilitar filtro paso alto |
| 3 | 0 | Deshabilitar modo lazo |
| 4 | 0 | Usar entrada analógica primaria |
| 5 | 1 | Transmisión sincronizada con recepción |
| 7,6 | 1,0 | ± 1.5V de entrada |
| 8 | 1 | Insertar filtro corrector sin(x)/x en D/A |

*Tabla 3.11: Registro de control de estado del AIC*

# 4 HERRAMIENTAS SOFTWARE

## 4.1 MATLAB

Matlab (*MATrix LABoratry* o Laboratorio de Matrices) es un programa de ordenador disponible para varias plataformas (PC, Macintosh, Unix,...) que contiene un lenguaje de programación propio y un conjunto de aplicaciones y herramientas gráficas para la resolución de todo tipo de problemas de ingeniería, científicos y matemáticos.

Todos los datos en Matlab se almacenan como matrices, pudiendo realizarse sobre ellas directamente las operaciones algebraicas comunes, así como ciertas operaciones especiales que permiten manipular conjuntos de datos rápidamente en diferentes formas.

Su sintaxis es muy similar a otros lenguajes de programación (como el lenguaje "C"), y ofrece una interfaz gráfica de usuario o GUI (*Graphical User Interface*) que permite usar Matlab como una herramienta de desarrollo de aplicaciones. Además el entorno "Simulink" permite realizar cualquier simulación de forma gráfica mediante la creación de bloques que realizan una determinada función y se interconectan entre sí. Se complementa con "Stateflow", un entorno de simulación gráfico para el modelado y desarrollo de sistemas basados en sucesos.

Matlab utiliza funciones para dividir operaciones complejas en otras más simples; éstas se asocian en paquetes por su afinidad, como por ejemplo "graph2d" engloba todas las funciones de tratamiento gráfico en dos dimensiones y "timefun" las correspondientes a la hora y la fecha. Complementando a estas se encuentran los llamados "Toolboxes", que son funciones altamente optimizadas escritas en el propio lenguaje de Matlab y agrupadas de materias más específicas, como por ejemplo "signal" para el procesamiento digital de señales. El usuario puede modificar cualquiera de ellas e incluso crear las suyas propias. Esto permite que una misma función pueda usarse para problemas de distinta naturaleza.

Existen varias versiones del programa y la compatibilidad "hacia atrás" está garantizada salvo pequeñas modificaciones, es decir, un programa que funciona en la versión 4 lo hará en la 5 pero al revés puede no ser cierto. Esto es lógico ya que las funciones proporcionadas se van modificando y mejorando, al tiempo que se añaden nuevas funcionalidades. Actualmente se encuentra en la versión 6.1

Las versiones anteriores a la 4 funcionaban sobre la interfaz de comandos de Ms-Dos, con las limitaciones que ello suponía. Con la 4.0 el programa se beneficia de las ventajas de los entornos de ventanas haciendo el trabajo mucho más cómodo desde el punto de vista del usuario. Además fue la primera que proporcionaba el entorno Simulink para realizar simulaciones de forma gráfica.

Las principales mejoras introducidas con la versión 5 son:

- Características de programación orientada a objetos (creación de objetos, clases y métodos), incluyendo sobrecarga de funciones y herencia.
- Matrices extensibles a cualquier número de dimensiones.
- Varias mejoras en el apartado gráfico (imágenes de 24 bits, modo de renderizado basado en bufer Z,...)
- Generación de números aleatorios por medio de semilla múltiple.
- Debugger y editor propio (¡al fin!).

La versión 6 supone entre otras cosas una gran mejora en la interfaz de usuario y permite visualizar simultáneamente otras ventanas a parte de la de trabajo, como un histórico de las órdenes introducidas o un listado de las variables existentes en memoria, permitiendo además una rápida modificación y visualización gráfica de ellas. También mejora las herramientas para el desarrollo de aplicaciones, entre las que se incluyen un compilador de Matlab, librerías gráficas y matemáticas para C/C++, y los servidores “web” y de ejecución de Matlab, que permiten distribuir versiones de las aplicaciones realizadas. Con estas herramientas puede generarse código muy eficiente para C o Ada95 a partir de diagramas Simulink o Stateflow.

## 4.2 HERRAMIENTAS DE DESARROLLO TMS320C30

Como todos los microprocesadores, el TMS320 toma sus instrucciones en forma de números binarios, llamado lenguaje máquina, y que es propio de cada arquitectura. Sin embargo su programación sería una tarea muy tediosa si no existieran una serie de herramientas que facilitasen de alguna forma esa programación. Algunas de ellas son el ensamblador, el enlazador o "linker", convertidores a formato objeto y hexadecimal, compilador de C... Aunque la utilidad de todas ellas está fuera de toda duda, sólo se van a describir brevemente las más importantes a la hora de realizar el presente proyecto.

### 4.2.1 ENSAMBLADOR

El código fuente en lenguaje ensamblador puede escribirse con cualquier editor de texto que permita almacenarlo como texto sin formato (formato de texto plano). El programa ensamblador se encarga de traducir este fichero fuente en un fichero llamado objeto, escrito en lenguaje máquina.

Los ficheros fuente contienen instrucciones entendibles por el DSP y directivas de ensamblador. Las instrucciones consisten en una línea que consta de cuatro partes o campos separados por al menos un espacio blanco o tabulación:

- Etiqueta: Puede contener hasta 32 caracteres alfanuméricos. Se utilizan para asociar una dirección simbólica con una línea, pudiendo referenciarse por ese nombre. Son opcionales, y si se usan deben empezar en la primera columna.

- Nemónico: No puede empezar en la primera columna (porque sería interpretado como una etiqueta). Son instrucciones o directivas de ensamblador.

- Operando: Son los argumentos para el nemónico. Es una lista de constantes (binarias, hexadecimales, decimales, enteros, flotantes...) y símbolos (etiquetas, registros, otros símbolos,...).

- Comentario: También son opcionales. Son notas que el programador incluye pero que son ignorados a la hora de realizar el ensamblado. Deben comenzar con el carácter ";". Una línea se considera entera de comentario si en la primera columna aparece el carácter "*".

En el fichero fuente además de instrucciones pueden aparecer también directivas de ensamblador, que se utilizan para controlar varios aspectos del proceso de ensamblado, como introducir datos (constantes o valores iniciales) en el programa, alineamiento de esos datos, y definir las llamadas secciones, que no son más que asociaciones de instrucciones (programa o datos). Las directivas van precedidas del carácter "." y las principales son:

- Directivas de sección: Sirven para crear y nombrar secciones, según el uso que se le vaya a dar. Las más importantes son:

TEXT: Código ejecutable.

DATA: Datos o variables preinicializadas.

SECT: Crea particiones lógicas en programas largos.

- Directivas de inicialización de memoria: Introducen datos en la memoria asignada a una determinada sección. Pueden usarse símbolos definidos anteriormente. Las más usadas son:

WORD: Enteros de 32 bits.

FLOAT: Constantes de punto flotante en precisión simple (32 bits).

SET: Asigna valores a símbolos (constantes en tiempo de ensamblado).

SPACE: Reserva el número de palabras especificado (rellena con ceros).

- Otras directivas:

END: Indica el final del programa. Líneas posteriores se ignoran.

GLOBAL: Convierte un símbolo en externo de manera que sea accesible por otros módulos a la hora del enlazado.

La forma de invocar al ensamblador es desde la línea de comandos:

Asm30 [fichero_fuente] [fichero_objeto] [fichero_de_listado]

Los corchetes indican que su aparición es opcional (si no aparecen el programa los solicita). El fichero de listado contiene una lista con la ubicación del programa y variables dentro de cada una de las secciones. Si el fichero fuente se nombra con la extensión “.asm”, la orden “asm30 nombre” genera la salida con el mismo nombre y extensión “.obj”.

### 4.2.2 ENLAZADOR

El enlazador combina diferentes ficheros objeto para crear un único programa ejecutable. Conforme lo va creando realiza una relocalización y resuelve todas las referencias externas. Admite una sintaxis parecida a la del lenguaje "C" para la configuración de la memoria y la definición de secciones. Dos son las directivas principales que admite el enlazador, denominadas "MEMORY" y "SECTION" (siempre deben aparecer en mayúsculas), con las que se puede definir el modelo de memoria del sistema, combinar secciones objeto, colocar secciones en áreas específicas de la memoria, definir o redefinir símbolos globales... La forma de invocar al enlazador es:

```
LNK30        nombre.cmd
```

"Nombre.cmd" es un fichero de ordenes que contiene el nombre de los ficheros objetos, el nombre del fichero de salida, y el conjunto de directivas.

La directiva "MEMORY" identifica zonas de memoria que están presentes físicamente en el sistema y pueden ser usados por el programa. Cada zona de memoria tiene un nombre, una dirección de comienzo y una longitud. La forma de utilizarla es:

```
MEMORY
{
        nombre:          origin=valor,        length=valor
        ...
}
```

Los nombres son etiquetas predefinidas, como VECS, ROM o RAM0, "origin" indica el origen y "length" su longitud. Si no se usa esta directiva, el enlazador propone uno por defecto (correspondiente al TMS320C30).

La otra directiva, "SECTIONS", indica cómo combinar las secciones definidas en los ficheros fuente y dónde colocarlas en la memoria. La sintaxis es:

```
SECTIONS
{
        sección:         nombre_i
        ...
}
```

"Sección" es cada una de las secciones definidas en los ficheros fuente y "nombre_i" es cualquiera de las etiquetas usadas en la directiva "MEMORY". Alternativamente puede sustituirse "nombre_i" por "load = valor", donde "valor" es una dirección de memoria válida que indicará el comienzo de esa sección.

### 4.2.3 SIMULADOR

Es un programa que simula el funcionamiento de la tarjeta de evaluación y el TMS320C30 mediante software, lo que permite ejecutar programas sin necesidad de disponer de ellos físicamente. Como el programa supone que no existe hardware la emulación no es completa, ya que por ejemplo no se dispone de interrupciones, por lo que una aplicación que haga uso de ellas debe ser modificada si se quiere probar su funcionamiento en el simulador. A pesar de esto, las facilidades que presenta suplen con creces esta carencia, como se verá a continuación.

La interfaz principal del programa se divide en varias áreas; en la superior e inferior se encuentran los comandos principales, que se activan con tan sólo pulsar su inicial (que se encuentra resaltada), los cuales dan paso a otros comandos o a la posibilidad de introducir argumentos.

A la izquierda se muestra el trozo de código que se está ejecutando, mostrando la dirección de las instrucciones, su código de operación y las instrucciones tal y como aparecía en el código fuente original, con lo que se puede reconocer en cada momento qué se está ejecutando. El programa incluso indica sobre qué instrucción se encuentra cada uno de los estados de la línea de ejecución o “pipeline”.

En el área de la derecha puede verse el estado de todos los registros, incluidos los registros de precisión extendida que se presentan tanto en formato entero como en punto flotante. Los registros cambiar de color si como consecuencia de ejecutar una instrucción su contenido es modificado, por lo que es fácil seguir saltos condicionales, incrementos o decrementos de direcciones,...

Una última zona enseña información seleccionable mediante uno de los comandos, de manera que puede mostrar el contenido de una zona (seleccionable) de la memoria.

Entre las operaciones que se pueden realizar se encuentran la modificación de direcciones de memoria y registros, llenado y salvado de zonas de memoria, ejecución paso a paso, creación de puntos de ruptura en la ejecución (puntos donde el programa se detiene a la espera de una acción por parte del programador),...

# 5 DISEÑO DEL SISTEMA

Una vez justificado el uso de la modulación 16-QAM y conocidas las herramientas de que se dispone se procede a diseñar e implementar el sistema. Aunque el objetivo es realizar un modulador y un demodulador, necesariamente deben ir acompañados de más bloques para que la comunicación se pueda realizar; por eso a partir de ahora se renombran como transmisor y receptor, y pasarán a denominarse modulador y demodulador sólo a una parte de ellos.

Para proceder con el diseño se ha realizado una simulación previa en Matlab. Como se dijo anteriormente, el entorno Simulink se presenta como un medio rápido para realizar el proceso, pero no se va a utilizar dado que no sólo es importante el funcionamiento global, sino también algunos resultados intermedios que ayuden a comprender cómo funciona el sistema y que permitan modificar ciertas partes para añadir cierta versatilidad y pueda usarse en otras aplicaciones. Además sólo se ha hecho uso de funciones muy básicas para ilustrar mejor el proceso real que se lleva a cabo, ya que de otro modo Matlab realizaría la mayor parte de proceso sin que se pueda observar lo que sucede internamente. Sólo en casos muy puntuales como “rand” y “randn” (paquete “elmat”, generación de números aleatorios) y “filter” (toolbox “signal”, para realizar filtrados) se ha echado mano de las funciones que provee Matlab, en el primer caso por ser necesario su uso únicamente en la simulación, y en el segundo porque la operación se describe suficientemente a la hora de realizar la implementación real.

No es conveniente cuando se trabaja con Matlab abusar de estructuras “for” y “while” desde el punto de vista de velocidad proceso [22], ya que Matlab dispone de formas más convenientes de ejecución (la mayoría de estas estructuras puede desaparecer con el uso de matrices). Sin embargo para realizar la simulación se ha optado por renunciar a la velocidad ejecución frente a la fidelidad con respecto a la programación en un lenguaje ensamblador, de manera que sea más fácil la programación final, sin perder de vista el aspecto gráfico.

Se ha intentado que la simulación sea lo más flexible posible mediante la definición de numerosas constantes; de esta manera se pueden variar algunos parámetros (como la frecuencia de muestreo o el número de ciclos de portadora por símbolo) sin tener que modificar los algoritmos de modulación o demodulación, pudiendo observar su comportamiento en distintas condiciones.

Tanto en el caso de la simulación como en el del programa ensamblador se supone conocido, tanto en el transmisor como en el receptor, el número de símbolos a enviar, lo que supone una abstracción. Puesto que esta constante está perfectamente identificada, es fácil su modificación para que se ajuste a cualquier tamaño de datos y pueda ser modificada por un usuario o una aplicación que funcione en un nivel superior.

## 5.1 CONFIGURACIÓN

La configuración del AIC se realiza en varios pasos, los cuales se ejecutan en una subrutina separada del código de la aplicación denominada “aicreset”. En primer lugar se debe configurar el timer 0 del DSP, que el AIC utilizará para generar la frecuencia de muestreo, la velocidad de desplazamiento para el envío de datos serie y las características de respuesta del filtro antialiasing. La frecuencia que va a generar es el máximo posible (7.5 MHz), lo que se consigue con un valor de “1” en el registro periodo y un funcionamiento en modo pulso (puede obtenerse el mismo resultado mediante una configuración especial del modo reloj, en el que el registro periodo se carga con el valor “0”):

```
ldi     @t0_ctladdr, ar0    ; dirección registro de control del timer
ldi     1,r1
sti     r1, *+ar0(8)        ; registro de periodo a “1”
ldi     @t0_ctlinit, r1     ; cargar palabra de configuración
sti     r1, *ar0            ; almacenar configuración
```

El valor de “t0_ctlinit” se ha establecido previamente con el valor “02C1h” (1011000001b), lo que significa que la salida se usará como un timer (y no como una salida digital de propósito general), funcionará en modo pulso, se realizará un reset del timer y comenzará a contar después realizar la configuración, y se usará como fuente el reloj interno del DSP.

A continuación se debe resetear el AIC bajando el pin XF0, lo que se produce al cargar un valor de “2” (10b) en el registro IOF, de manera que XF0 queda configurado como pin de salida y se fuerza a que esa salida sea “0”; el AIC debe estar así al menos 2 ciclos de reloj, lo que se garantizará más adelante:

```
ldi     2, iof              ; XF0 como salida, valor de salida “0”
```

Puesto que el AIC y el DSP se comunican por medio del puerto serie 0 de éste último, se configuran ahora los registros de control de los puertos de transmisión y recepción; ambos se cargan con el valor 0111h (100010001b) que se corresponde con la selección de los pines correspondientes como puerto serie (y no como pines de entrada o salida de propósito general). Además hay que inicializar el registro de control global del puerto para empezar su operación almacenando el valor 00e970300h (11101001011100000011000000000b) en dicho registro (lo que se realiza a través de la etiqueta “p0_global”); este valor selecciona el formato adecuado para la comunicación serie (modo “handshake” desactivado, señales activas a nivel bajo, datos de 16 bits,...). Una explicación más detallada se encuentra en manual correspondiente. Por último se realiza una limpieza del registro de transmisión de datos para empezar la transmisión:

```
ldi     @p0_addr, ar0       ; cargar dirección del puerto serie 0
ldi     111h,r1
sti     r1, *+ar0(2)        ; inicializar control de transmisión
sti     r1, *+ar0(3)        ; inicializar control de recepción
ldi     @p0_global, r1      ; cargar palabra de configuración
sti     r1, *ar0            ; inicializar control global del puerto 0
xor     r1, r1
sti     r1, *+ar0(8)        ; poner "0" como dato a transmitir
```

El paso siguiente consiste en eliminar la condición de "reset" impuesta anteriormente, subiendo el nivel del pin XF0; esto se consigue almacenando un "06h" (110b) en IOF. El bucle de espera se utiliza para asegurar que el proceso de "reset" se ha llevado a cabo correctamente y proporciona tiempo suficiente.

```
rpts    99
nop                         ; esperar 99 ciclos de reloj
ldi     6, iof              ; poner XF0 a 1
```

Ahora ya se pueden establecer los parámetros de funcionamiento para los dispositivos que componen el AIC. Aunque ya se puede realizar la configuración mediante una rutina de interrupción no se va realizar así sino mediante una espera activa; de esta forma se evita que otra interrupción interfiera en el proceso de configuración (además así se evita tener que definir dos rutinas de interrupción, una para este proceso y otra para cuando esté la aplicación funcionando). La espera activa consiste en mantenerse en un bucle hasta que se genere una interrupción, pero al no estar activados los flags correspondientes no se va direccionar mediante el vector de interrupciones, sino que simplemente se sale del bucle y se continúa. La rutina que permite hacer esto es muy simple:

```
wait_transmit_0     xor     if, if      ; esperar interrupción de transmisión

wloop               tstb    10h, if     ; comprobar flag de interrupción
                    bz      wloop       ; si todavía es cero se vuelve
                    rets
```

La frecuencia de muestreo se establecerá al máximo posible para el AIC, esto es, 19.2 KHz. Si se elige B=36 se obtiene directamente que para esa frecuencia de muestreo se necesita un valor de A = 6, lo que proporciona exactamente un valor de $F_{LP}$ = 7.8125 KHz. En el registro de control de estado se almacenarán los valores que aparecen en la tabla 3.11, los cuales son adecuados para esta aplicación. Esta configuración es válida tanto para el convertidor D/A (transmisor) como para el A/D (receptor), por lo que las palabras a enviar mediante transmisiones secundarias son: 0x0c18h y 0x4892h para establecer las frecuencias de muestreo y de corte, y 0x02a7h para seleccionar el modo de funcionamiento (registro de control de estado). El proceso completo es el siguiente:

```
call    wait_transmit_0     ; muestrear interrupción de transmisión
ldi     3, r1               ; 3 = 11b
sti     r1, *+ar0(8)        ; transmisión secundaria
call    wait_transmit_0
ldi     0c18h,r1            ; establecer valor para A
sti     r1, *+ar0(8)
ldi     *+ar0(12), r1       ;lectura para limpiar flag de interrupción

call    wait_transmit_0
ldi     3, r1
sti     r1, *+ar0(8)
call    wait_transmit_0
ldi     04892h, r1          ; establecer valor de B
sti     r1, *+ar0(8)
ldi     *+ar0(12), r1

call    wait_transmit_0
ldi     3, r1
sti     r1, *+ar0(8)
call    wait_transmit_0
ldi     2a7h, r1            ; registro de control de estado
sti     r1, *+ar0(8)
ldi     *+ar0(12), r1
```

A continuación se deben limpiar los flags de interrupción del DSP para eliminar cualquier interrupción pendiente:

```
xor     if, if              ; limpiar todos los flags de interrupción
```

Por último, para que el AIC quede preparado para realizar su función, hay que habilitar la interrupción del puerto serie del DSP, ya que es el lugar por el que pasan los datos para los convertidores. Además se debe activar el flag global de interrupciones o GIE, ya que de otro modo no se atendería ni ésta ni ninguna otra solicitud de interrupción.

```
or      @enbl_sp0_r, ie     ; habilitar interrupción puerto serie 0
or      ENBL_GIE, st        ; habilitar interrupciones
rets
```

Los valores “enbl_sp0_r” y “enbl_gie” corresponden a constantes que permiten enmascarar los bits adecuados; la primera se define como una verdadera constante almacenada en memoria, mientras que la segunda no es más que una etiqueta que se sustituye por el valor real en tiempo de compilación y no ocupa espacio en memoria. El realizar esto así se debe a que existe un límite en tamaño para estas últimas.

## 5.2 TRANSMISOR

El transmisor propuesto tiene la estructura de un transmisor en cuadratura como el de la figura 5.1 [2]. Su funcionamiento se basa en la separación de los datos en una componente en fase y otra en cuadratura, por lo que se hace necesario el uso de un desmultiplexor. Este esquema fue concebido para ser realizado mediante dispositivos electrónicos (hardware), pero puede servir de base para el diseño que se va a realizar. Los bloques lógicos determinados por las líneas punteadas corresponden a una agrupación en unidades funcionales, ya que al ser un diseño software hay operaciones que son más convenientes realizar de determinada manera, como se justificará más adelante.

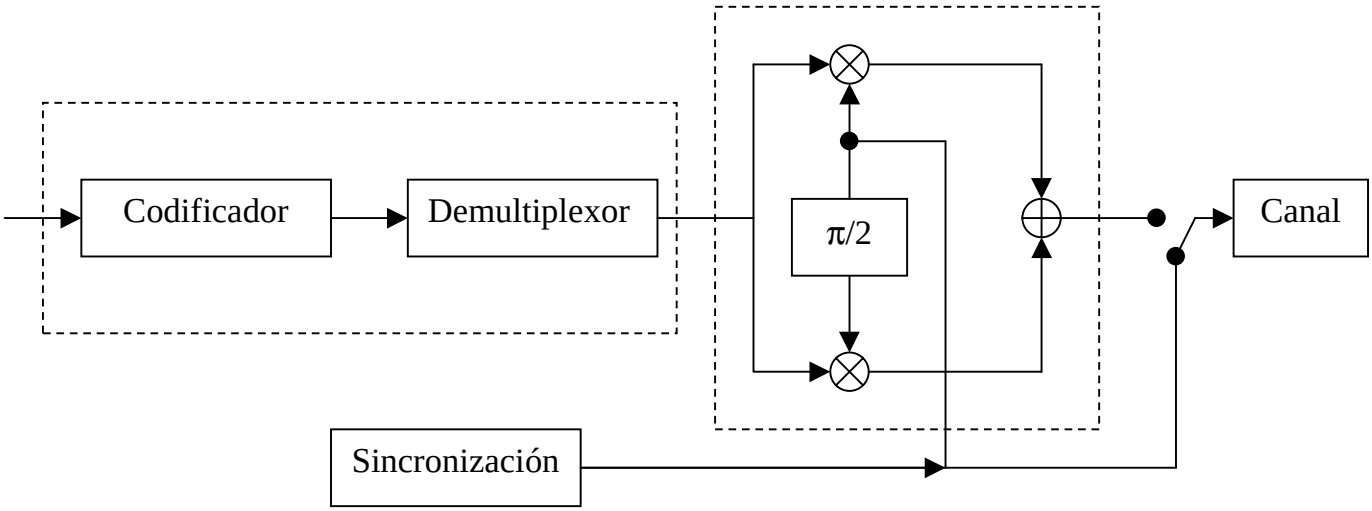


Figura 5.1: Esquema del transmisor

En la figura 5.2 se observa el diagrama de flujo del proceso que se lleva a cabo; este diagrama, al igual que el resto que aparecen más adelante, aunque concebidos para ser implementados en este DSP pueden utilizarse en otros entornos debido a su generalidad, a pesar de que describen perfectamente las operaciones a realizar.

Primero debe hacerse la inicialización de DSP y EVM para que funcionen de la forma que se necesita, además de almacenar ciertos valores en algunos registros y posiciones de memoria. El proceso es gobernado por dos bucles; uno principal, que se ejecuta una vez por cada palabra de 32 bits a enviar y depende por lo tanto de la constante “npal”, y otro que lleva la cuenta de cuántos símbolos se han enviado de cada palabra, dependiendo del valor de “nsimb”. Se ha dispuesto de dos posiciones de memoria para almacenar estos contadores a medida que se van decrementando para evitar que se modifiquen si se dejan en los registros.

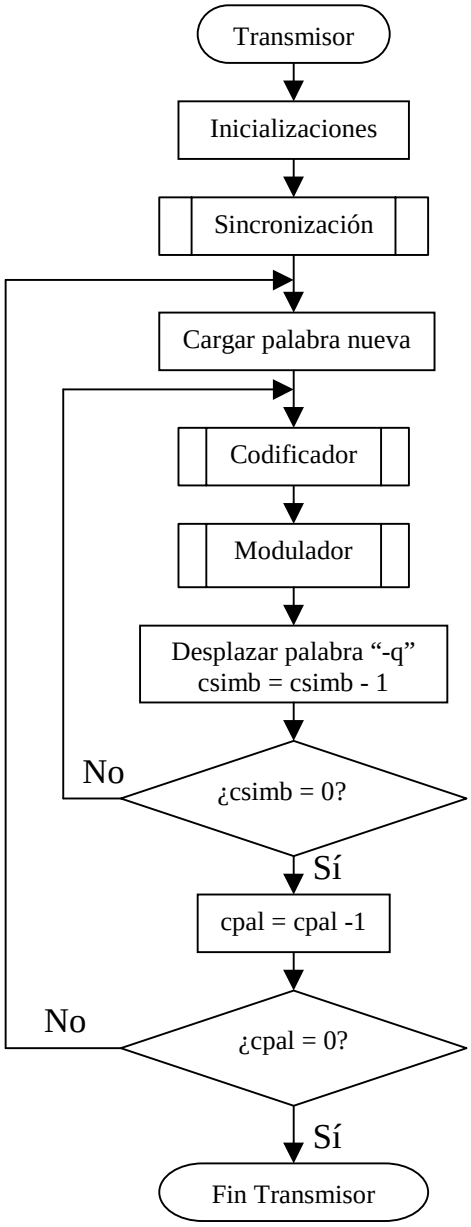


Figura 5.2: Diagrama de flujo del modulador

El código representa con fidelidad el diagrama de flujo anterior. En primer lugar se crean unas constantes que van a necesitarse más adelante, como son un desplazamiento de valor “-q” y un contador llamado “crpts”. El desplazamiento se usa para eliminar los bits ya codificados de la palabra de 32 bits, de manera que en cada pasada se trabaje sobre “q” bits; de esta forma siempre se actúa sobre los “q” bits menos significativos de la palabra de 32. Por otro lado se utiliza la constante “nsin” (número de muestras del seno almacenadas) para dar un valor inicial a un contador que se usará para los comandos de repetición de instrucciones. Se ha optado por hacerlo así para no crear una constate adicional cuando su valor no es independiente, sino que depende de otros. El valor que debe tomar es: $\frac{nsin}{2} - 1$

```
aplication  ldi   @pq, r0
            sti   r0, @offset      ; crear offset
            negi  r0, r0
            sti   r0, @despl       ; crear desplazamiento en -q

            ldi   @nsin, r0        ; carga tamaño senoide
            ror   r0               ; divide entre 2
            subi  1, r0            ; resta 1
            sti   r0, @crpts       ; contador de repeticiones listo

            ldi   @npal, r0        ; inicializar contador de palabras
            sti   r0, @cpal
```

Las operaciones anteriores se realizan aquí y no en la rutina “sysinit“ para que exista una separación entre lo que es inicialización del hardware (filtros, timer, puertos...) de las inicializaciones necesarias para ejecutar el código. En el receptor esto se hará más patente. A partir de ahora es cuando comienza realmente el proceso de transmisión.

```
            call  ssincr           ; sincronizador
newpal      ldi   *ar7++, r7       ; nueva palabra de 32 bits a enviar
            ldi   @nsimb, r0       ; cargar contador de símbolos
            sti   r0, @csimb
newsim      call  codif            ; codificador
            call  mod              ; modulador
```

La palabra de 32 bits que se quiere transmitir, que se mantiene en r7, se va desplazando cuatro bits a la derecha en cada pasada, de manera que tras 4 pasadas se almacene una nueva palabra para ser enviada

```
            lsh   @despl, r7       ; desplazamiento de -q
            ldi   @csimb, r0       ; actualizar contador de símbolos
            subi  1, r0
            sti   r0, @csimb
            bnz   newsim
```

Cuando se termina la palabra de 32 bits, se decrementa el contador correspondiente hasta que se transmitan todas:

```
            ldi   @cpal, r0        ; actualizar contador de palabras
            subi  1, r0
            sti   r0, @cpal
            bnz   newpal
fin         rets
```

### 5.2.1 SINCRONIZACIÓN

Dos son las funciones de las que se encarga este bloque. En primer lugar se encarga de proporcionar al modulador una referencia senoidal válida, necesaria para llevar a cabo el proceso de modulación. Por otro lado envía al canal una señal consistente en varios periodos de una senoide de manera que sirva como indicatriz del inicio de transmisión y el receptor pueda así prepararse para la llegada de datos. Esta señal puede además usarse para monitorizar el canal y que el receptor pueda modificar en cierto sentido su comportamiento y de esta forma la detección sea más fiable (esto se verá más adelante).

Ambos objetivos se consiguen almacenando un número determinado de muestras de un periodo completo de una senoide. El número elegido ha sido el de ocho, de tal manera que al ser la frecuencia de muestreo de 19.2 KHz resulta una frecuencia para la portadora de 2.4 KHz. A partir de un contador de periodos se controla el número de ciclos que se envían, llamado "m" y que será el mismo para la señal de sincronismo y para los datos modulados; se ha utilizado m=4, con lo que la frecuencia de los símbolos es de tan sólo 600 Hz; pero se está utilizando un código M-ario en el que cada símbolo lleva cuatro bits de información, de manera que la tasa real de información vuelve a ser de 2.4 KHz.

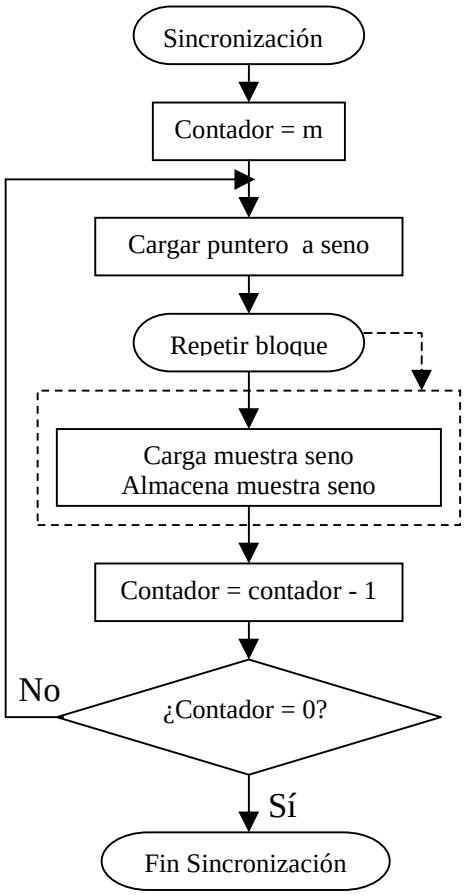


Figura 5.3: Diagrama de flujo del sincronizador

La utilización de la función seno o coseno para este fin puede parecer a priori un tanto arbitraria. Sin embargo hay que hacer un par de matizaciones. En primer lugar, si se tiene almacenado un periodo de una automáticamente se tendrá el de la otra, ya que la única diferencia entre ellas es un desfase en el número de muestras. En segundo lugar, aunque teóricamente ambas pueden ser utilizadas, el uso del coseno presenta dificultades. El medio físico o canal se encuentra en ausencia de transmisión en un estado de reposo, que a efectos del receptor equivale a una entrada nula; puesto que la primera muestra del coseno es un uno se fuerza a que el canal pase de este estado de reposo a presentar el máximo valor que se envía, pueden aparecer errores en la dinámica de esta transición. Además el transmisor puede tener problemas de saturación, ya que los dispositivos reales no pueden cambiar muy bruscamente de valor. El uso del seno, sin embargo, produce una transición más suave ya que su primera muestra es nula.

De las dos funciones que se comentaron anteriormente sólo va a realizarse aquí la segunda: enviar una señal de sincronismo al canal para indicar el comienzo de la transmisión. La otra se dejará para el bloque modulador, ya que es allí donde se necesita como fuente; de esta forma no se ocupan registros que pueden necesitarse.

Aunque el código es suficientemente rápido y compacto lo hubiera sido aún más con la utilización del direccionamiento circular para el seno; sin embargo esto generaría un problema debido a las propias exigencias de ese tipo de direccionamiento, que obliga a colocar los datos en determinadas zonas en la memoria. Como el número de muestras del seno puede ampliarse en cualquier momento (porque se desee mayor exactitud, sobre todo en el receptor) esto supondría encontrar en la memoria una zona adecuada de ese tamaño para reubicar los datos, mientras que de la forma elegida lo único que hay que cambiar es el contador “crpts” (que habría que cambiar de cualquier forma). El valor de “crpts” es uno menos que el número de muestras que se desean generar por ciclo, como requiere la instrucción “rptb”. El incremento de “2” se debe a que sólo se usan la mitad de las muestras, y es en el receptor donde se usarán todas (debido a una interpolación). El código es el siguiente:

```
ssincr      ldi     @pm, r0             ; cargar número de ciclos

ssincr2     ldi     @seno, ar0          ; cargar dirección seno
            ldi     @crpts, rc          ; cargar contador de repetición
            rptb    ssincr3
            ldf     *ar0++(2), r7       ; cargar muestra seno

ssincr3     call    ftx                 ; formatización
            subi    1, r0               ; actualizar contador
            bnz     ssincr2             ; comenzar otro ciclo
            rets
```

## 5.2.2 CODIFICADOR + DEMUX

Estos dos bloques se agrupan por comodidad, ya que se puede ir desmultiplexando al mismo tiempo que se realiza la codificación. El proceso es simple y puede realizarse directamente mediante una tabla, en la que para cada cuaterna de bits se obtengan los valores correspondientes de las amplitudes de las componentes en fase y en cuadratura.

Existen 16! maneras distintas de numerar los símbolos de una constelación 16-QAM. De todas ellas las más interesantes son aquellas que utilizan una codificación Gray, en la que cada símbolo se diferencia de los adyacentes en un único bit; de esta forma si el símbolo recibido cae en una región colindante sólo habrá un bit erróneo. Una posibilidad se muestra en la figura 5.2, donde entre paréntesis se muestra el valor binario. A partir de ésta se pueden obtener otras posibilidades mediante rotación de bits, pero la elegida asigna valores numéricos menores para símbolos más cercanos al origen.

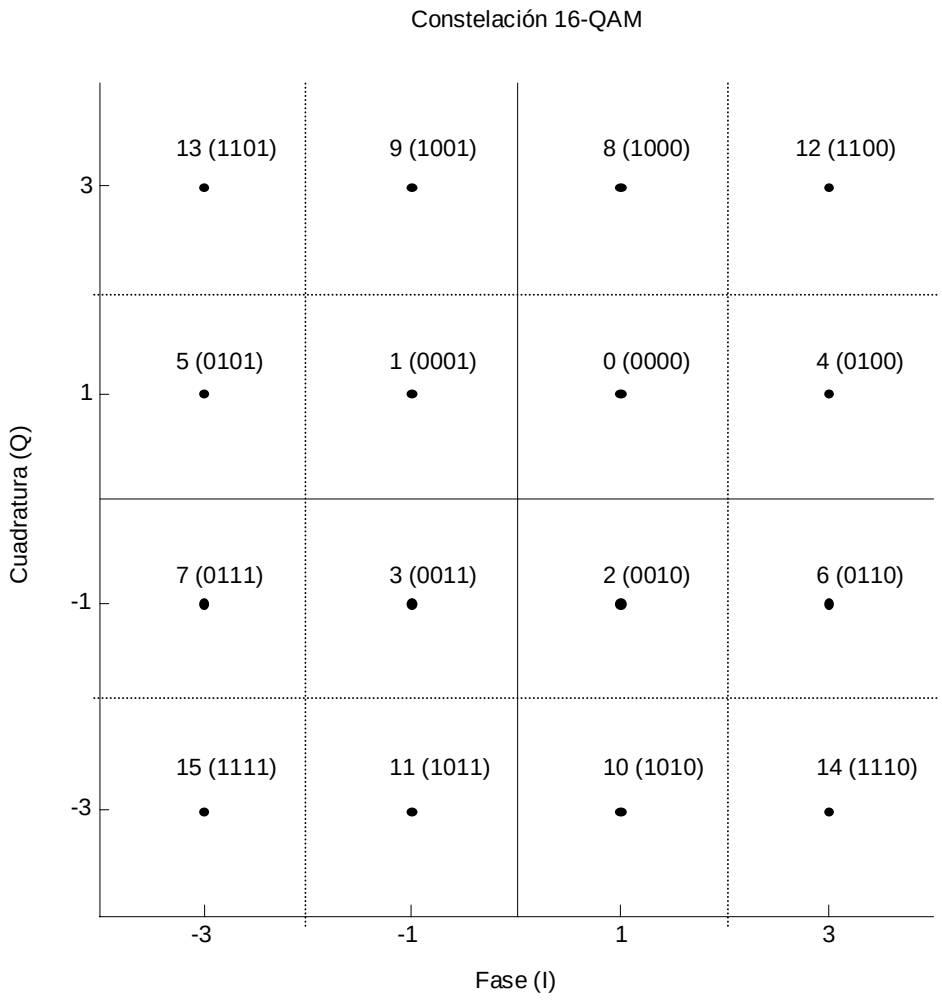


Figura 5.4: Constelación 16-QAM con codificación Gray

Para proceder a la codificación sólo hay que usar una tabla donde se asocie cada uno de los símbolos posibles con sus valores correspondientes de las componentes en fase (denominada “I”) y cuadratura (denominada “Q”). Sin embargo se puede realizar la misma operación de un modo mucho más eficiente dadas las características de la codificación elegida. Si se mira con detenimiento la figura 5.2 por filas se observa que hay bits que permanecen invariables, y lo mismo ocurre si se consideran las columnas; por lo tanto a partir de la pareja de bits (llamada dibit) de cada componente se pueden obtener directamente las amplitudes, como se muestras en la siguiente tabla:

| Dibit | Amplitud |
|---|---|
| 10 | 3 |
| 00 | 1 |
| 01 | -1 |
| 11 | -3 |

Figura 5.5: Relación bits codificados - amplitud

De esta forma no se tiene que almacenar y recorrer una tabla de 16 entradas sino sólo dos tablas idénticas de 4 entradas cada una, realizándose la operación en la mitad de tiempo. Aún se puede ir un paso más allá a partir de la tabla anterior si se observa que uno de los bits (el primero) indica el valor de la amplitud (en valor absoluto) y el otro (el segundo) señala su signo. De esta forma se reduce aún más la velocidad de proceso, de nuevo a la mitad, ya que los valores a comparar son tan sólo 2 para cada componente, con lo que el número total de comparaciones por símbolo es tan sólo de 4 (frente al los 16 que se tenían originalmente), siendo además esta última una media determinista.

La forma entonces a proceder es la siguiente: en primer lugar se supone que el símbolo es el “0000”, por lo que se asocia tanto a I como a Q el valor “1”. Como no siempre se envía ese símbolo, se deben modificar I y Q como corresponda, de la siguiente manera:

- Si el bit 3 es 1, la amplitud de Q cambia a 3.
- Si el bit 2 es 1, la amplitud de I cambia a 3.
- Si el bit 1 es 1, Q cambia de signo (ahora es negativo).
- Si el bit 0 es 1, I cambia de signo (ahora es negativo).

Los bits se han numerado de la forma usual, es decir, del menos significativo (bit 0) al más significativo (bit 3). Aunque el proceso se repita para I y Q no se obtiene ventaja ni en velocidad ni en longitud de código usando un bucle y una tabla para las máscaras ya que el bucle se ejecutaría sólo dos veces. Para programar el algoritmo se han definido las constantes siguientes, que permiten enmascarar los bits de interés:

qbig = 1000 ibig = 0100
qneg = 0010 ineg = 0001

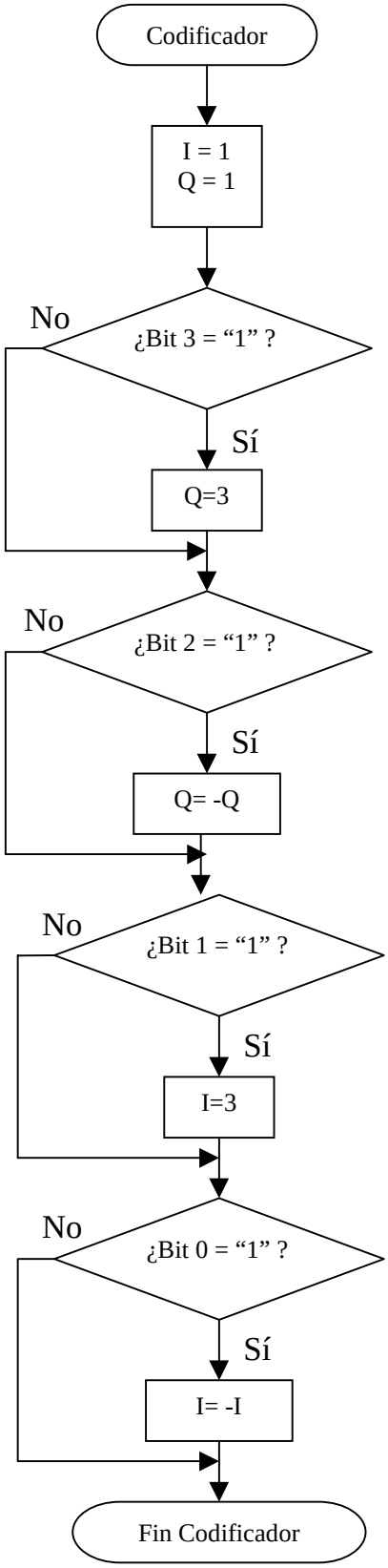


Figura 5.6: Diagrama de flujo del codificador

Nótese cómo las instrucciones “Tstb” (comprobar bit) y “LDcond” (carga condicional) del DSP facilitan la programación del algoritmo anterior.

```
codif   ldf    1, r5          ; amplitud = 1 (valor por defecto)
        ldf    1, r6          ; signo = 1 (valor por defecto)
        tstb   @ibig, r7      ; comprobar bit de amplitud
        ldfnz  3, r5          ; si no es cero modificarlo a “3”
        tstb   @ineg, r7      ; comprobar bit de signo
        ldfnz  -1, r6         ; si no es cero cambiar el signo
        mpyf   r5, r6         ; r6 = I
        stf    r6, @valI      ; almacena I

        ldf    1, r5          ; análogo para Q
        ldf    1, r6          ; el único cambio son las máscaras
        tstb   @qbig, r7
        ldfnz  3, r5
        tstb   @qneg, r7
        ldfnz  -1, r6
        mpyf   r5, r6         ; r6 = Q
        stf    r6, @valQ      ; almacena Q
        rets                  ; retorno
```

A modo de ejemplo, la cadena de datos: 0000 1101 1011 0110 proporciona las amplitudes: I = 1, -3, -1, 3 y Q = 1, 3, -3, -1.

### 5.2.3 MODULADOR

Está formado por el resto del esquema de la figura 5.1. A partir de los valores de I y Q obtenidos en el codificador hay que multiplicar un cierto número de periodos del coseno por la amplitud I, hacer lo mismo con el seno y Q, y sumar ambos resultados para formar la señal modulada. El número de periodos debe ser adecuado para que se puedan recuperar los datos y será por lo tanto en el receptor donde se realizarán consideraciones sobre dicho número. Puede observarse que los diagramas de flujo de sincronizador y modulador son iguales salvo las operaciones que se realizan dentro del bloque de repetición, puesto que la modulación requiere de dos productos y una suma.

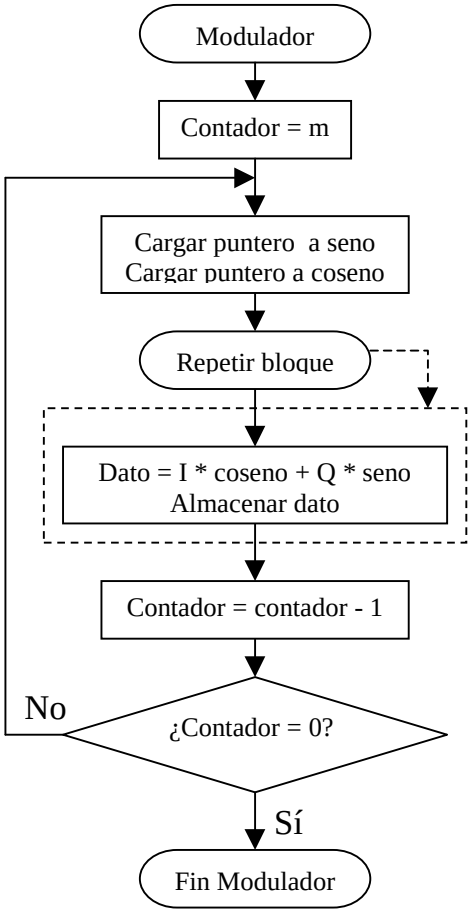


Figura 5.7: Diagrama de flujo del modulador

Al igual que ocurría en la sincronización aparece el salto de “2” muestras, pero ahora se hace necesario usar un registro de índice debido a las exigencias de direccionamiento de la instrucción “mpyf3”. Los valores de “I” y “Q” se vuelven a cargar para asegurar que no hay errores por sobreescritura de registros. El coseno se ha obtenido simplemente con la suma de un offset que es un cuarto del número de muestras almacenado para el seno.

```
mod     ldf   @valI, r5
        ldf   @valQ, r6
        ldi   @pm, r0                   ; número de ciclos

mod2    ldi   @seno, ar1                ; carga dirección seno
        ldi   ar1, ar0                  ; copia la dirección a ar0
        addi  @offset, ar0              ; sumar offset (apunta al coseno)
        ldi   @crpts, rc
        rptb  mod3
        mpyf3 r5, *ar0++(ir0), r1       ; componente en fase
        mpyf3 r6, *ar1++(ir0), r2       ; componente en cuadratura
        addf3 r1, r2, r3                ; dato modulado (suma I+Q)

mod3    call  ftx                       ; formatización
        subi  1, r0                     ; actualizar contador
        bnz   mod2                      ; empezar otro ciclo

        rets
```

Para el ejemplo del apartado anterior, las formas de onda generadas en el modulador pueden verse en la siguiente figura, donde a simple vista puede reconocerse cada uno de los símbolos y cómo se corresponden con los valores del apartado anterior.

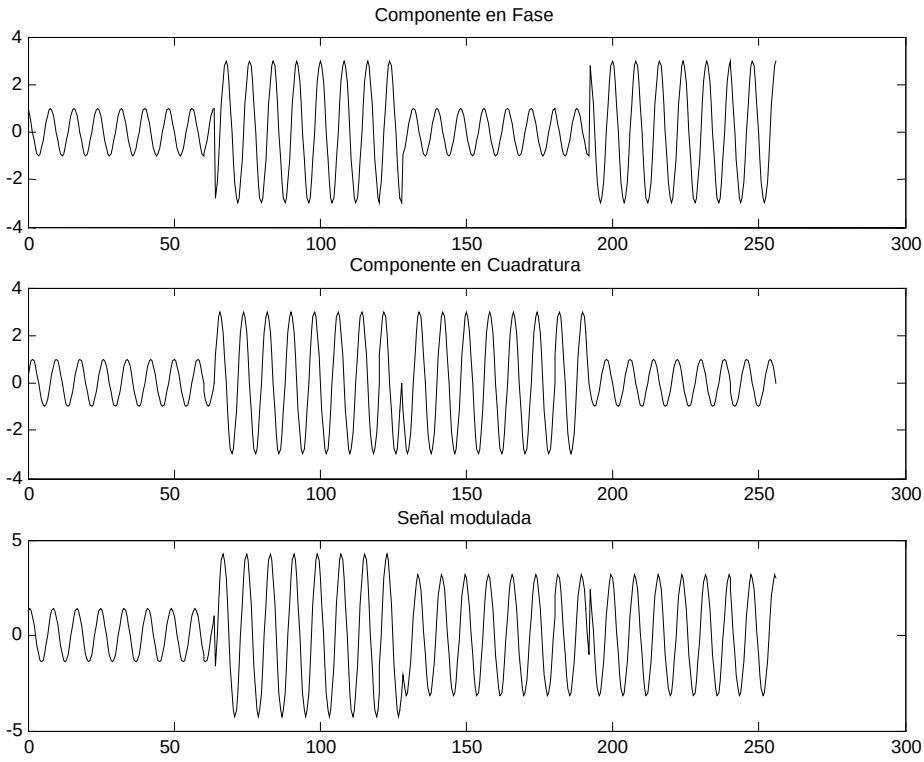


Figura 5.8: Ejemplo de señal modulada *($T_s$=1/19200 seg).*

## 5.3 INTERFAZ E/S

Tal y como se ha realizado la operación de modulación en el apartado anterior es totalmente imposible que se pueda realizar la comunicación. Esto es debido a que por el momento sólo se ha considerado que el dato modulado sale por el puerto serie del DSP, pero para poner dicho dato en el canal es necesario que atraviese el AIC y acomodarlo en cierta forma al canal. Esto debe hacerse con todas las muestras a enviar, tanto las resultantes de la sincronización como de la modulación.

Como ya se ha visto el AIC admite sólo 16 bits de datos, siendo los dos bits menos significativos igual a cero. Por lo tanto el dato hay que convertirlo a un valor de 14 bits en complemento a dos. Puesto que con 14 bits hay $2^{14}$ (=16386) números disponibles, se tiene uno menos de la mitad para valores positivos y uno menos de la otra mitad para los negativos (esto es debido a la representación del cero), o sea, se pueden distinguir números en el rango $\pm 2^{13}$-1 (= ± 8191). Se consiguen disminuir los errores por pérdida de precisión dividiendo el dato a enviar por este valor, o equivalentemente multiplicándolo por el valor inverso, ya que se puede calcular a priori al ser fijo. Después de este proceso se debe acomodar el número al formato que espera el AIC, lo que se consigue con la conversión a entero del valor resultante de la operación anterior y después con un desplazamiento hacia la izquierda de dos bits. Puesto que al realizar el desplazamiento se rellena con ceros no habría que hacer nada más.

Sin embargo se puede aprovechar la formatización del dato anterior para acomodar la amplitud de la señal modulada al rango dinámico del AIC de manera que se obtenga un mejor comportamiento del convertidor D/A y la señal se degrade menos en su viaje por el canal, con la precaución de no llegar a ninguna situación de saturación, lo que produciría una degradación seria en la señal modulada. Puesto que se conoce suficientemente bien la señal modulada se puede conocer el valor máximo (o mínimo, en valor absoluto) de su amplitud, que es algo inferior a 4. Si se divide por esta cantidad todas las muestras estarán comprendidas en el rango ±1, por lo que esta normalización mejorará la representación numérica de la muestra. Esta operación se puede combinar con la anterior de manera que se realicen ambos ajustes simultáneamente. El bucle permite que se pueda realizar la sincronización por interrupción.

```
ftx     mpyf  @rango, r3     ; escalado en el rango dinámico
        fix   r3, r3         ; conversión a entero
        lsh   2, r3          ; desplazamiento de 2 bits
        idle                 ; esperar una interrupción

        rets
```

Parecería lo más lógico que el transmisor se ejecute dentro de la rutina de interrupción; sin embargo se ha optado por hacerla lo más ligera posible, y realizar la sincronización mediante la espera con la instrucción “idle” de la rutina anterior. Con esto se consigue por un lado que la rutina de interrupción se guarde completamente en la caché y se ejecute más rápidamente; por otro, el código es más simple y se puede seguir la línea de ejecución del programa más fácilmente. En el otro caso habría que introducir bucles y condiciones de salto dentro de la rutina de interrupción o programar varias de ellas y actualizar el vector de interrupciones con la dirección adecuada según el estado del programa, resultando claramente en un código de mayor longitud y complejidad (además de perderse tiempo salvando y recuperando registros cada vez que entre en la rutina de interrupción). Esto es aplicable tanto al transmisor como al receptor. En el caso del transmisor la rutina de interrupción se muestra a continuación, donde la etiqueta PARMS apunta a la página de datos donde están definidos el vector de interrupciones y las direcciones de los registros internos del DSP (como por ejemplo “p_serie”). Hay que realizar una lectura necesariamente para limpiar el flag de interrupción, lo que se puede realizar de varias maneras y una consiste simplemente leer el registro:

```
receive0    push   st              ; salvar registros
            push   ar0
            push   r0              ; r3 no se salva, es el dato a enviar
            push   dp
            ldp    PARMS           ; cargar página de datos
            ldi    @p_serie, ar0   ; dirección puerto serie
            ldi    *+ar0(12), r0   ; limpiar flag de interrupción
            sti    r3, *+ar0(8)    ; enviar dato
            pop    dp              ; restaurar registros
            pop    r0
            pop    ar0
            pop    st

            reti
```

Existe otra forma de funcionamiento que se puede utilizar: en lugar de saltar a una rutina de interrupción se puede sustituir la instrucción “idle” por un bucle de espera activa donde se compruebe si se ha activado el flag correspondiente al puerto serie en el registro IF. La diferencia con la rutina de interrupción es que de este modo no se atienden otras peticiones de interrupción, ya que este método exige que no se active el flag global de interrupciones o GIE. Sin embargo con el método utilizado se pueden ejecutar otras rutinas de interrupción, como un envío de datos DMA (hacia o desde al PC, por ejemplo); de esta forma el funcionamiento de la modulación y demodulación no se vería afectado si las otras rutinas de interrupción son poco pesadas desde el punto de vista computacional.

En el receptor hay que deshacer las operaciones de formato del dato que se realizaron en el transmisor. Ahora hay que mantenerse a la espera al principio de la rutina (y no al final como antes); una vez que llega el dato éste sólo trae 14 bits útiles (en complemento a dos), siendo cero los dos bits menos significativos; el dato es alineado haca el bit más significativo de un registro (desplazamiento lógico de 16 bits) para luego realizarse un desplazamiento aritmético hacia la derecha de 18 bits, de manera que se consiguen dos cosas: que los bits útiles del dato queden alineados al LSB y que se mantenga el signo de este dato al realizar ese desplazamiento; con esto el dato ha quedado convertido a un entero (con signo) en complemento a dos de 32 bits. Después de convertirlo a flotante hay que devolverlo a su nivel original, ya que en el transmisor se ajustó para aprovechar mejor el rango dinámico de los convertidores, por lo que se multiplica por el inverso de aquella cantidad. Por último el dato es almacenado en una posición de memoria.

```
frx     idle                ; esperar una interrupción
        lsh   16, r0        ; alinear los bits al MSB
        ash   -18, r0       ; alinear los bits al LSB
        float r0, r0        ; conversión a flotante
        mpyf  @irango, r0   ; ajuste del rango dinámico
        stf   r0, @datorig  ; almacenar dato

        rets
```

La rutina de interrupción del receptor del puerto serie es muy similar, ya que la única diferencia es que no aparece la instrucción de almacenamiento; en este caso la lectura del dato también sirve para limpiar el flag de interrupción:

```
receive0 push  st           ; salvar registros
         push  ar0
         push  r0
         push  dp
         ldp   PARMS        ; cargar página de datos
         ldi   @p_serie, ar0 ; dirección puerto serie 0
         ldi   *+ar0(12), r0 ; leer dato
         pop   dp           ; restaurar registros
         pop   r0
         pop   ar0
         pop   st

         reti
```

## 5.4 RECEPTOR

El esquema que se ha usado para el receptor es el siguiente [2]:

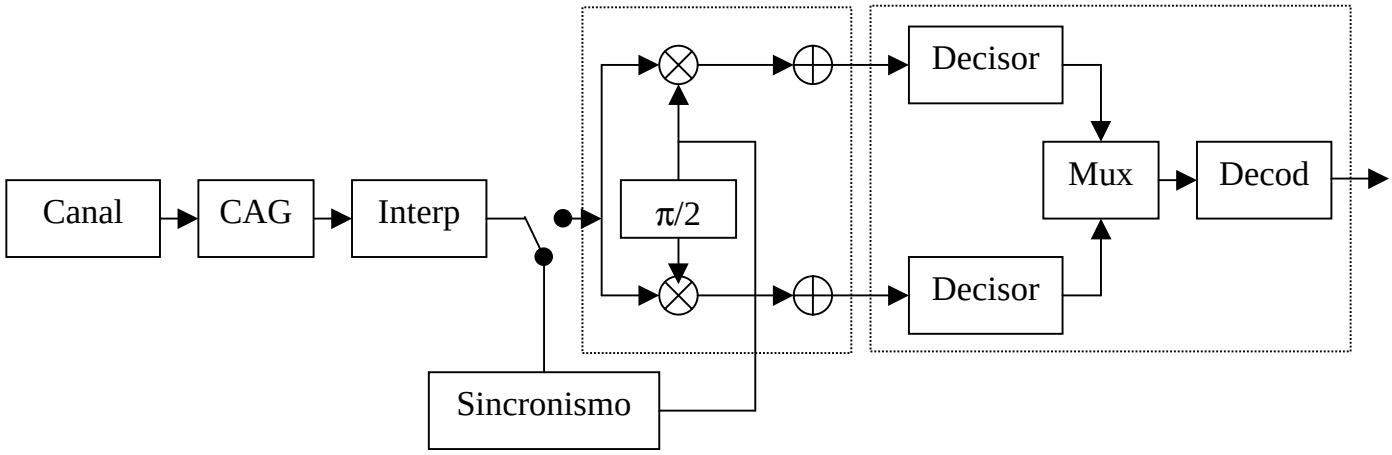


Figura 5.9: Esquema del receptor

Al igual que el del transmisor, este esquema fue concebido para su implementación mediante circuitería, aunque puede servir como referencia para un diseño de programación. De ahí que se hayan agrupado algunos bloques mediante las líneas de puntos para una mayor facilidad a la hora de programar y un mayor rendimiento a la hora de la ejecución.

En primer lugar se realiza un ajuste en la ganancia de la señal que llega, ya que la modulación QAM es muy sensible a las variaciones de amplitud que pueden aparecer en el canal. A continuación un interpolador permite doblar la cantidad de muestras útiles que el sistema puede utilizar sin que por ello deba variarse la frecuencia de muestreo o la comunicación se realice empleando un tiempo mayor. La sincronización también se beneficia de esta interpolación, de manera que se puede emplear un mecanismo avanzado de detección de fase como el lazo de Costas; de otra forma el número de muestras sería insuficiente para obtener una referencia de fase válida.

Una vez conseguida la sincronización se realiza el proceso de demodulación mediante un bloque muy similar al usado en transmisión. Tras una decisión en cada una de las ramas hay que entrelazar los valores mediante una multiplexación, para luego realizar la decodificación que permita obtener los datos enviados.

Como se dijo en el apartado anterior, la base del funcionamiento está en la rutina “frx” y la rutina de interrupción del puerto serie, de manera que tras ejecutarse ésta se obtiene un dato resultante del muestreo ya en formato flotante y en los niveles adecuados de amplitud, el cual se almacena en “datorig”.

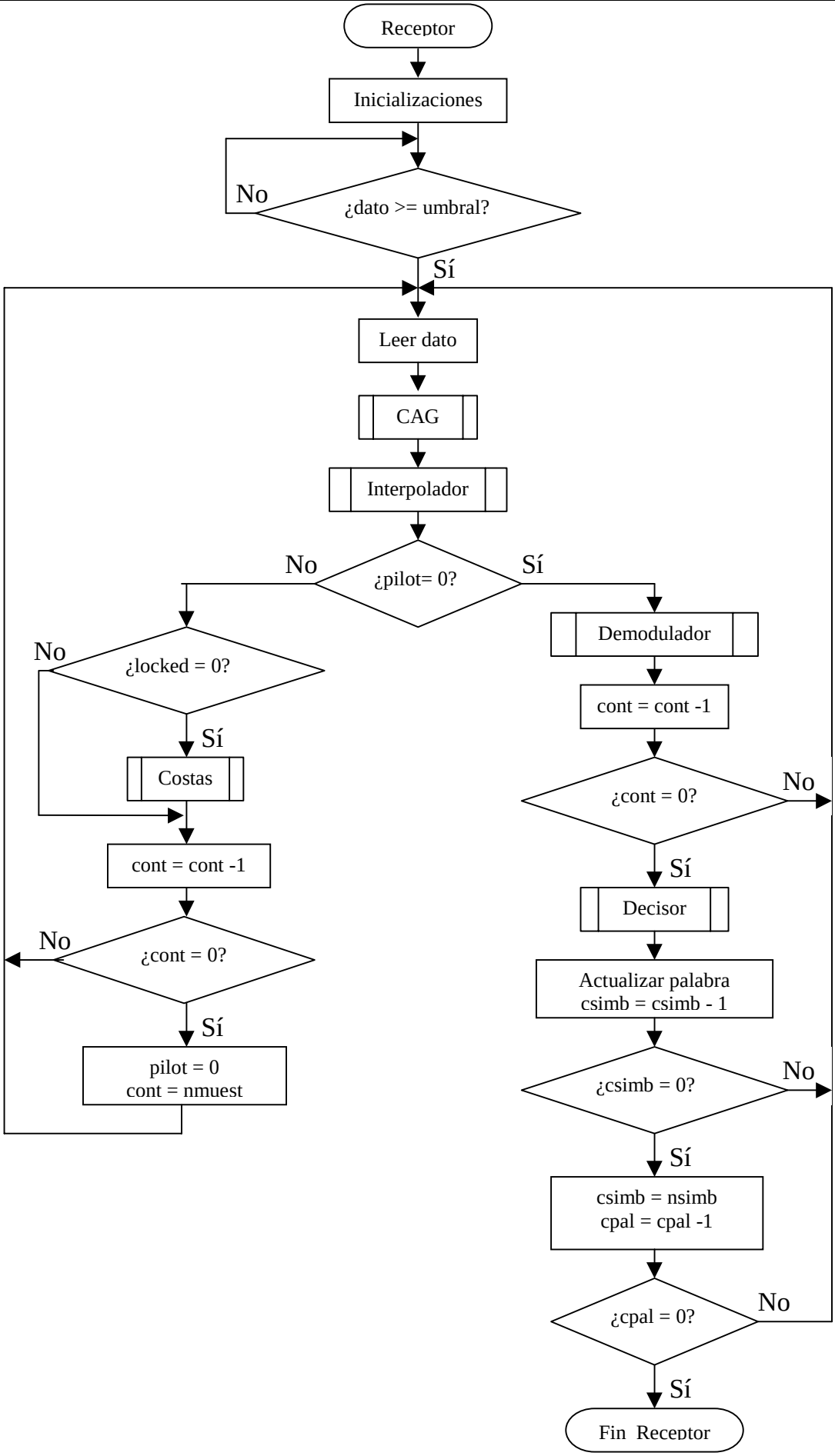


Figura 5.10: Diagrama de flujo del receptor

El receptor posee tres estados claramente diferenciados; por un lado está la fase de reposo, en la que se encuentra monitorizando el puerto serie hasta que la muestra que se recibe supera un determinado umbral, momento en el que se pasa a la fase de sincronización, en la que se intenta detectar cuál es la fase de la onda que está llegando para de esta forma la demodulación pueda realizarse correctamente; por último está la fase de demodulación, en la que tras un proceso sobre las muestras que llegan se van recuperando los datos que fueron enviados. Las dos fases activas (sincronización y demodulación) poseen algunas partes en común, como son el controlador automático de ganancia y el interpolador. En el diagrama de flujo anterior se detalla cómo interactúan los bloques entre sí y las condiciones que deben cumplirse para llamar a unas rutinas u otras. La explicación en detalle se hará a continuación mediante el código ensamblador.

Lo primero que hay que realizar es la configuración de DSP y del EVM, al igual que hay que crear ciertas constantes y almacenarlas en memoria. Esto último se realiza en la rutina llamada “initvar”, cuyo contenido es el siguiente:

```
initvar     ldi     0, r0           ; r0 = 0
            ldi     r0, ir0
            ldi     r0, ir1
            sti     r0, @psinud     ; índice de las senoides = 0
```

El flag “locked” se usa para almacenar el estado del lazo de Costas; toma el valor “0” hasta que se alcanza el enganche en el bucle, tomando el valor “1”; “fi” es el valor resultante del lazo, que se usará para modificar los valores de las senoides:

```
            sti     r0, @locked     ; inicializar flag a 0
            sti     r0, @fi         ; inicializar fi
```

Otro flag, “pilot”, es el utilizado para indicar si se está en la fase de sincronización con el valor “1” (detectando la señal piloto que envía el transmisor) o en la fase de demodulación con el valor “0”.

```
            addi    1, r0           ; r0 = 1
            sti     r0, @pilot      ; inicializar flag a 1
```

Se necesita crear una constante para indicar el número de rotaciones necesario para alinear el símbolo resultante de una nueva decisión con los símbolos ya decididos y almacenados en la palabra de 32 bits. Más adelante se verá cómo se realiza esto; baste decir ahora que si en la palabra de 32 bits caben “pq” símbolos la constante va a valer “pq-1”:

```
            ldi     @pq, r0         ; inicializar el número de rotaciones
            subi    1, r0
            sti     r0, @nrot
```

Lo siguiente es inicializar algunas variables y crear algunas constantes más; el comentario de la línea de ensamblador es suficientemente descriptiva, por lo que no se va a añadir nada más:

```
ldf     0.0, r0          ; inicializar acumulador
sti     r0, @acumul
ldf     @nivel0, r0      ; inicializar umbral
stf     r0, @nivel
ldi     @pm, r0
mpyi    @nsin, r0
sti     r0, @nmuest      ; número de muestras de un símbolo
ldi     @n, bk           ; tamaño bufer circular
ldi     @nsin, r0        ; crear offset para el coseno (nsin/4)
lsh     -2, r0
sti     r0, @offset
ldi     @n, r0           ; inicializar contador de repeticiones
subi    1, r0
sti     r0, @crpts       ; n-1
ldi     @n, r0           ; inicializar contador de repeticiones 2
lsh     -1, r0
subi    1, r0
sti     r0, @crpts2      ; n/2 - 1
ldi     @nsimb, r0       ; inicializar contador de símbolos
sti     r0, @csimb
ldi     @npal, r0        ; inicializar contador de palabras
sti     r0, @cpal
```

Para terminar sólo queda almacenar las direcciones de unos bufers en una tabla de punteros, denominada “pbuf”. Esto se consigue tratando las direcciones de comienzo de estos bufers (@xin, @xlazo...) como un dato, de manera que se cargan no en registro de direccionamiento sino en uno de precisión extendida; de esta forma se puede almacenar en “pbuf” este dato; para poder utilizarlo habrá que cargarlo entonces sí en un registro de direccionamiento y usarlo como tal. Son 8 bufers en total aunque sólo aparecen dos, ya que el proceso es repetitivo, y su utilidad y funcionamiento se verá en las rutinas de interpolación y sincronización.

```
ldi     @pbuf, ar0       ; inicializar tabla de punteros
ldi     @xin, r0
sti     r0, *ar0++
ldi     @xlazo, r0
sti     r0, *ar0++
rets
```

Esto concluye la rutina “initvar” y deja al programa receptor preparado para empezar a funcionar.

Tras la iniciación el programa debe encontrarse en la fase de reposo, monitorizando el puerto serie hasta que llegue una muestra que supere el umbral:

```
aplication   call   initvar          ; inicializaciones
             ldf    @umbral, r1      ; detectar si se supera el umbral
detect       call   frx
             cmpf   @datorig, r1
             bge    detect           ; si datorig <=umbral vuelve
```

Una vez que se detecta el inicio de la comunicación debe ajustarse el contador de muestras, ya que las muestras que no han superado el umbral se han perdido; se estima que son tres las que lo han hecho, que es el número que se pierde en ausencia de ruido. No obstante si se perdiera alguna más (o menos) la diferencia estaría en el valor resultante de la sincronización y no aparecerían errores; sin embargo se ha incluido la corrección por conocerse a priori y de esta forma no contribuir a posibles errores de decisión:

```
             ldi    @nmuest, r0
             subi   3, r0            ; se ha avanzado 3 muestras
             sti    r0, @cont        ; cont = m * nsin - 3
```

Una vez hecho esto se entra en la parte compartida de las fases de sincronización y demodulación; el dato leído desde el puerto serie se almacena en uno de los bufers circulares, en concreto “xin”, para que pueda ser usado por el interpolador; la razón de realizar esto aquí y no en la rutina obedece simplemente a una cuestión conceptual: de esta forma la rutina de interpolación se dedica únicamente a esa operación y no a otras.

```
inisinc      call   frx              ; dato nuevo en @datorig
             ldf    @datorig, r7
             ldi    @pbuf, ar0       ; cargar puntero a @xin
             ldi    *ar0, ar1
             stf    r7, *ar1++       ; almacenar muestra actual en bufer
             sti    ar1, *ar0        ; actualizar puntero
             call   cag              ; control automático de ganancia
             call   interp           ; interpolador
```

Tras realizar la interpolación hay que comprobar si se están recibiendo muestras de la señal piloto o sin embargo son datos modulados; el bufer que se usa es el mismo en un caso y en otro para de esta forma ahorrar memoria:

```
             ldi    @pilot, r0
             bz     inidata     ; flag = 1 indica sincronismo
```

Si todavía se está recibiendo la señal piloto, hay que comprobar si el lazo está enganchado a la fase o no:

```
            ldi     @locked, r0     ; comprobar si el bucle está enganchado
            bnz     nopll
            call    costas
```

Tanto si se ha realizado el enganche como si no se decrementa el contador y se comprueba si es cero; si no lo es vuelve hacia atrás y si lo es se continúa. De esta forma si no se consigue el enganche se procede a la demodulación con el último valor resultante del lazo para que el error sea el menor posible:

```
nopll       ldi     @cont, r0       ; cargar contador
            subi    1, r0           ; decrementar contador
            sti     r0, @cont       ; almacenar contador
            bnz     inisinc         ; salta si cont no es cero
```

Con el contador a cero ya se ha recibido la señal piloto completa, por lo que se desactiva el flag, se restaura el contador con su valor inicial y se vuelve al principio mediante un salto retardado:

```
            bd      inisinc         ; salto retardado
            sti     r0, @pilot      ; almacenar cero en flag
            ldi     @nmuest, r0     ; restaurar contador
            sti     r0, @cont       ; fin sincronismo
```

Ahora se pasa a la fase de recuperación de los datos: Tras ejecutar la rutina de demodulación hay que decrementar el contador y comprobar si es cero para llamar al decisor o seguir demodulando:

```
inidata     call    demod           ; demodulador
            ldi     @cont, r0       ; actualizar contador
            subi    1, r0
            sti     r0, @cont
            bnz     inisinc         ; volver al principio si no es cero
            call    decis           ; decisor
```

Una vez que se tienen los bits tras la decisión se almacenan en la posición de memoria “dataout” y siguientes; para que estos datos sean los mismos (si no han ocurrido errores) que los que se enviaron es necesario realizar un empaquetamiento de los bits en grupos de 32, que es la longitud de los datos para el DSP. El proceso a seguir es el siguiente: la palabra ya recibida se guarda en un registro, mientras que los bits nuevos se guardan en otro; sobre ambos registros se realizan q-1 rotaciones hacia la derecha, de manera que los bits nuevos (que aparecen en los LSB) quedan alineados sobre los MSB del registro; puesto que la palabra de memoria se ha inicializado con ceros, la suma (OR lógico) de ambos registros proporciona la nueva palabra. Para verlo más claro se va ilustrar con un ejemplo; los valores de los registros se muestran en formato hexadecimal, y ya se ha tenido en cuenta que en el caso 16-QAM se tiene q=4:

```
ldi     @dataout, ar7       ; cargar dirección palabra recibida
ldi     *+ar7(ir1), r6
```

Si se supone que se han recibido, por este orden, los símbolos 1, 2, 3, 4 y 5, y que el siguiente ha sido el 6; el contenido de los registros r6 y r7 será:

r6 = 54321000 r7 = 00000006

Puede comprobarse cómo r6 es en efecto una palabra resultante de un paso anterior siempre y cuando se haya inicializado con ceros. Si ahora se ejecutan:

```
rpts    @nrot       ; rotar dato antiguo
ror     r6
rpts    @nrot       ; rotar dato nuevo
ror     r7
```

Tras las rotaciones de 4 bits (nrot se inicializó como q-1) se tendrá:

r6 = 05432100 r7 = 600000000

Para finalizar la suma lógica proporciona: r7 = 65432100

```
or      r6, r7          ; añadir los bits nuevos
sti     r7, *+ar7(ir1)  ; almacenar palabra
```

Con el símbolo nuevo salvado hay que proceder a decrementar el contador de símbolos y comprobar si es cero; en caso afirmativo se restaura el valor inicial de este contador y se decrementa el contador de palabras. De nuevo hay que realizar la comprobación de si es cero o no lo es, volviendo hacia atrás en caso de que lo sea y finalizando el programa de recepción si lo es.

```
ldi     @csimb, r0      ; actualizar contador de símbolos
subi    1, r0
sti     r0, @csimb
bnz     inidata         ; si no es cero vuelve al principio
ldi     @nsimb, r0      ; restaurar contador de símbolos
sti     r0, @csimb
addi    1, ir1
ldi     @cpal, r0       ; actualizar contador de palabras
subi    1, r0
sti     r0, @cpal
bnz     inidata

rets                    ; fin de la aplicación
```

### 5.4.1 CAG

La modulación QAM es muy sensible a las variaciones de potencia media de la señal recibida debido a que gran parte del proceso de recuperación de los datos depende de que esa potencia se mantenga más o menos constante. Sin embargo la señal al atravesar el canal siempre sufre alteraciones debido al ruido, por lo que es necesario realizar alguna operación para eludir esta dificultad. La mejor opción es la de usar un amplificador de ganancia variable a la entrada del receptor, antes de la conversión A/D, de manera que se pueda ajustar y corregir de forma adecuada, pero dado que no es posible por impedirlo el hardware se utilizará una solución software, lo que plantea varias disyuntivas.

Primero hay que decidir en qué lugar colocar el bloque, cosa que como puede observarse en el esquema del receptor se realiza antes de la interpolación. La razón de esto es que si se coloca detrás hay que realizar el proceso sobre dos muestras por cada una que ha llegado, pero dado que se conocen los coeficientes del filtro, conocida la entrada sería posible determinar la salida y cómo afectarían las muestras interpoladas; dicho de otro modo, se obtiene la misma información a partir de las muestras directamente tal y como se reciben que tras realizar la interpolación, pero de la primera forma se realiza la mitad de procesamiento.

La segunda cuestión es cómo modificar la ganancia para conseguir mantener la potencia media; existen buenos métodos para realizarlo pero son computacionalmente muy costosos debido a que implican el cálculo de raíces cuadradas y divisiones, además de necesitar un gran número de muestras antes de alcanzar un valor significativo.

El método que se va a seguir es el siguiente: en primer lugar se supone que la potencia media de la señal que se recibe vale uno; a continuación se ajustan las ganancias de cada uno de los bloques a la unidad, de manera que no se alteren los niveles de potencia. Este proceso hay que realizarlo antes de poner a funcionar el programa, y se ha hecho con ayuda de la simulación Matlab. El decisor necesita definir unos umbrales para obtener el símbolo correcto, umbrales que como se verá más adelante tan sólo dependen de un único valor, denominado “nivel”; en ausencia de ruido “acumul” debería valer 64, por lo que se multiplica el obtenido por su inverso para dar un factor corrector que multiplicado a 2 (valor en ausencia de ruido) pasa a ser el “nivel” nuevo. Una vez ya en ejecución se realiza el cálculo del cuadrado de la muestra recibida y se procede a su acumulación. Dada la sencillez de la rutina no se ilustra el diagrama de flujo correspondiente.

```
cag     mpyf  r7, r7          ; r7 * r7
        addf  @acumul, r7      ; acumular
        stf   r7, @acumul      ; guardar el nuevo acumulado
        mypf  @nivel0, r7      ; valor inverso en ausencia de ruido
        stf   r7, @nivel       ; guardar nuevo nivel
        rets
```

## 5.4.2 INTERPOLADOR

Para que se pueda obtener una sincronización más precisa y el proceso de demodulación sea más exacto se realiza una interpolación de los datos recibidos; de esta forma se puede trabajar como si se recibieran el doble de muestras, lo que hubiera significado doblar la frecuencia de muestreo utilizada en el transmisor, o equivalentemente disminuir a la mitad la frecuencia de la onda portadora. De esta forma tanto transmisor como receptor (y los convertidores implicados) trabajan a la misma frecuencia, aunque internamente en recepción se haga al doble, por lo que habrá que garantizar que se pueden realizar dos pasadas por el programa del ya al parecer recargado receptor (una para el dato original y otra para el interpolado) antes de que llegue la siguiente muestra.

Un primer método para realizar la interpolación es reconstruir la señal analógica y proceder a un muestreo a la nueva frecuencia (el doble de la que había), pero de nuevo se necesitaría un hardware distinto. Sin embargo el proceso de interpolación se puede realizar por completo en el dominio discreto muy fácilmente: se insertan entre cada dos muestras recibidas una muestra nula, lo que equivale a una expansión en la escala de tiempos, para luego realizar un filtrado de paso bajo y obtener las muestras interpoladas.

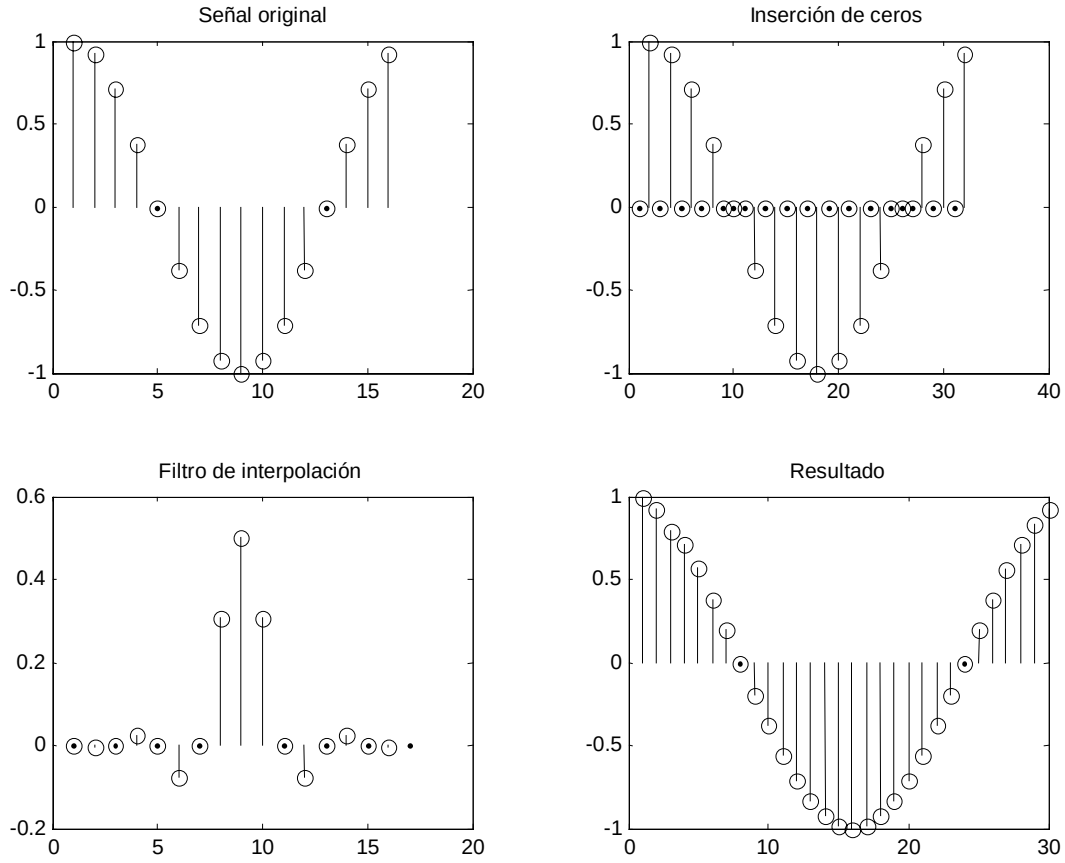


Figura 5.11: Ejemplo de interpolación *($T_s$=1/19200 seg).*

Para realizar el filtrado se puede utilizar una técnica denominada "descomposición polifásica". Aunque su uso es general y se puede aplicar tanto a la señal de entrada como al filtro, se va utilizar sobre los coeficientes del filtro, ya que es en alteraciones en la frecuencia de muestreo cuando aparecen ventajas computacionales con su implementación [13]. La técnica consiste en representar una secuencia discreta de la siguiente manera:

$$H(z) = \sum_{i=0}^{M-1} z^{-i}\ P_i\left(z^M\right)$$

Las $P_i$ se denominan componentes polifásicas y vienen dadas por:

$$P_i(z) = \sum_{k=-\infty}^{\infty} h[k\ M + \ i]\, z^{-k}$$

Sin embargo para el caso de interpolaciones es más conveniente definir lo que se llama la representación "polifásica transpuesta", que se obtiene a partir de la anterior de la siguiente manera:

$$H(z) = \sum_{i=0}^{M-1} z^{-(M-1-i)}\ Q_i\left(z^M\right) \quad , \quad siendo: \quad Q_i = P_{M-1-i}(z)$$

Para una interpolación de una muestra debe tomarse M=2, con lo que:

$$H(z) = Q_0\left(z^2\right) + z^{-1}\ Q_1\left(z^2\right) \quad , \quad siendo: \quad \begin{cases} Q_0 = \sum_{k=-\infty}^{\infty} h[2\ k + \ 1]\, z^{-k} \\ Q_1 = \sum_{k=-\infty}^{\infty} h[2\ k]\, z^{-k} \end{cases}$$

La secuencia H(z) puede descomponerse entonces de la manera anterior en las muestras en los instantes pares retrasadas, sumadas a las muestras de los instantes impares. Por lo tanto, al ser la operación de filtrado una operación lineal se puede realizar la operación a partir de la entrada y las muestras del filtro impares, y sumar la salida con las resultantes de realizar el mismo proceso con las muestras pares. En principio no parece que se haya ahorrado nada, pero debe recordarse que la señal que entra en el filtro no es cualquiera, sino la resultante de insertar ceros entre cada dos muestras recibidas; se concluye entonces que la mitad de la salida de cada una de las componentes polifásicas presenta un valor nulo (al ser la entrada nula), con lo que se puede dividir el filtrado completo en dos pero con la mitad de coeficientes. Para una mejor compresión del proceso se adjuntan dos tablas; en la primera se observan los productos resultantes de la operación de filtrado tal y como se que realizarían sin aplicar la descomposición polifásica; puede verse como la mitad de los productos son nulos, los correspondientes a posiciones impares cuando la entrada es el cero insertado (entrada 1) y las pares cuando es el dato recibido (entrada 2).

| Entrada 1 | Filtro | Productos | | Entrada 2 | Filtro | Productos |
|---|---|---|---|---|---|---|
| 0 | h[0] | 0 | | x[n] | h[0] | x[n] * h[0] |
| x[n] | h[1] | x[n] * h[1] | | 0 | h[1] | 0 |
| 0 | h[2] | 0 | | x[n-1] | h[2] | x[n-1] * h[2] |
| x[n-1] | h[3] | x[n-1] * h[3] | | 0 | h[3] | 0 |
| 0 | h[4] | 0 | | x[n-2] | h[4] | x[n-2] * h[4] |
| x[n-2] | h[5] | x[n-2] * h[5] | | 0 | h[5] | 0 |
| 0 | h[6] | 0 | | x[n-3] | h[6] | x[n-3] * h[6] |
| x[n-3] | h[7] | x[n-3] * h[7] | | 0 | h[7] | 0 |
| 0 | h[7] | 0 | | x[n-4] | h[7] | x[n-4] * h[7] |
| x[n-4] | h[6] | x[n-4] * h[6] | | 0 | h[6] | 0 |
| 0 | h[5] | 0 | | x[n-5] | h[5] | x[n-5] * h[5] |
| x[n-5] | h[4] | x[n-5] * h[4] | | 0 | h[4] | 0 |
| 0 | h[3] | 0 | | x[n-6] | h[3] | x[n-6] * h[3] |
| x[n-6] | h[2] | x[n-6] * h[2] | | 0 | h[2] | 0 |
| 0 | h[1] | 0 | | x[n-7] | h[1] | x[n-7] * h[1] |
| x[n-7] | h[0] | x[n-7] * h[0] | | 0 | h[0] | 0 |

Figura 5.12: Tabla de filtrado original

Los coeficientes se han numerado en la forma usual partiendo de h[0] pero se ha tenido en cuenta que el filtro posee simetría, lo que se traduce en que sólo únicamente 8 coeficientes distintos y no 16, de manera que se tiene: h[15-i]=h[i], i = 0, 1,... , 15.

| Entrada 1 | Filtro | Productos | | Entrada 2 | Filtro | Productos |
|---|---|---|---|---|---|---|
| x[n] | h[1] | x[n] * h[1] | | x[n] | h[0] | x[n] * h[0] |
| x[n-1] | h[3] | x[n-1] * h[3] | | x[n-1] | h[2] | X[n-1] * h[2] |
| x[n-2] | h[5] | x[n-2] * h[5] | | x[n-2] | h[4] | X[n-2] * h[4] |
| x[n-3] | h[7] | x[n-3] * h[7] | | x[n-3] | h[6] | X[n-3] * h[6] |
| x[n-4] | h[6] | x[n-4] * h[6] | | x[n-4] | h[7] | X[n-4] * h[7] |
| x[n-5] | h[4] | x[n-5] * h[4] | | x[n-5] | h[5] | X[n-5] * h[5] |
| x[n-6] | h[2] | x[n-6] * h[2] | | x[n-6] | h[3] | X[n-6] * h[3] |
| x[n-7] | h[0] | x[n-7] * h[0] | | x[n-7] | h[1] | X[n-7] * h[1] |

Figura 5.13: Tabla de filtrado final

En la tabla anterior aparecen sólo los productos válidos para realizar el filtrado. La conclusión que se obtiene es que hay que filtrar la entrada sin insertar cero alguno, pero reordenando los coeficientes del filtro como se muestra en la segunda columna, de tal forma que para la segunda entrada los coeficientes deben utilizarse en orden inverso como, se indica en la quinta columna.

En el diagrama de flujo siguiente se detalla el algoritmo completo del interpolador; la única consideración a tener en cuenta sobre él es que los coeficientes g(i) corresponden a los del filtro de interpolación pero ya ordenados según se explico anteriormente, tomando el índice "i" de g valores entre 0 y 7. Con la ordenación del filtro sólo es necesario almacenar los coeficientes distintos y no los 16, ya que en el segundo caso basta con recorrerlos a la inversa.

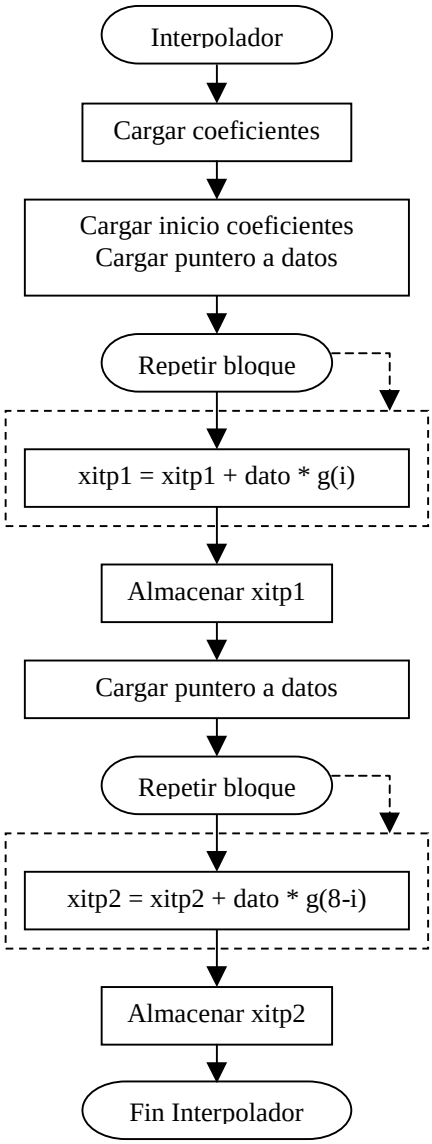


Figura 5.14: Ejemplo de interpolación

Para la programación en ensamblador no hay más que seguir el diagrama de flujo anterior tomando un par de precauciones, como se verá seguidamente:

```
interp      ldi     @hn, ar6        ; ar6 apunta al primer coeficiente

            ldf     0.0, r1         ; limpiar registros
            ldf     0.0, r2
```

Se carga la dirección de una tabla de punteros, de manera que se pueda acceder a la dirección del último valor del bufer que almacena las muestra de entrada; hay que hacer notar que el valor que devuelve la tabla de punteros no siempre es la posición inicial de este bufer, sino que es la posición de la última muestra que se guardó en ese bufer, el cuál se direcciona de forma circular:

```
        ldi     @pbuf, ar1          ; cargar dirección tabla de punteros
        ldi     *ar1, ar5           ; cargar puntero a @xin
```

Ahora se puede realizar ya la primera operación de filtrado:

```
        rpts    @crpts2
        mpyf    *ar6++, *ar5--%, r1 ; filtrado
||      addf    r1, r2, r2
        addf    r1, r2, r2          ; última suma
        stf     r2, @xintp1         ; almacenar muestra interpolada
```

De nuevo hay que limpiar los registros para que la instrucción de suma en paralelo parta de un valor inicial de cero:

```
        ldf     0.0, r1             ; limpiar registros
        ldf     0.0, r2
```

Para realizar el segundo filtrado hay que empezar a leer los datos almacenado desde el principio otra vez; la única diferencia con el anterior es que ahora los coeficientes del filtro se recorren desde el final hacia el principio; hay que tener en cuenta que tras el primer filtrado el registro apunta a la posición siguiente al último coeficiente (como consecuencia del posincremento utilizado), por lo que se hace necesario ahora el uso de un predecremento:

```
        ldi     *ar1, ar5           ; cargar puntero a @xin
        rpts    @crpts2
        mpyf    *--ar6, *ar5--%, r1 ; filtrado; ahora h(n) se recorre al revés
||      addf    r1, r2, r2
        addf    r1, r2, r2          ; última suma
        stf     r2, @xintp2         ; almacenar muestra interpolada
```

Para finalizar, teniendo en cuenta que “xin” es un bufer circular de tamaño 16 del que sólo se usan 8 en el filtrado, hay que avanzar el puntero 9 posiciones para que la muestra siguiente se encuentre en el lugar adecuado:

```
        rpts    @crpts2
        ldf     *ar5++%, r2         ; avanzar n muestras
        ldf     *ar5++%, r2         ; puntero listo para la siguiente muestra
        sti     ar5, *ar1           ; almacenar el puntero
        rets
```

### 5.4.3 SINCRONIZACIÓN

La forma en que se va a realizar la sincronización es mediante un lazo de Costas como el del apartado 2.7.2 cuyo esquema se repite aquí por comodidad; la señal piloto que se envía será utilizada como entrada a este bloque, de manera que pasado el tiempo de símbolo (que se corresponde con 4 ciclos de la portadora) se obtenga un valor de fase que permita disminuir en la medida de lo posible los errores debidos a utilizar relojes diferentes en transmisor y receptor.

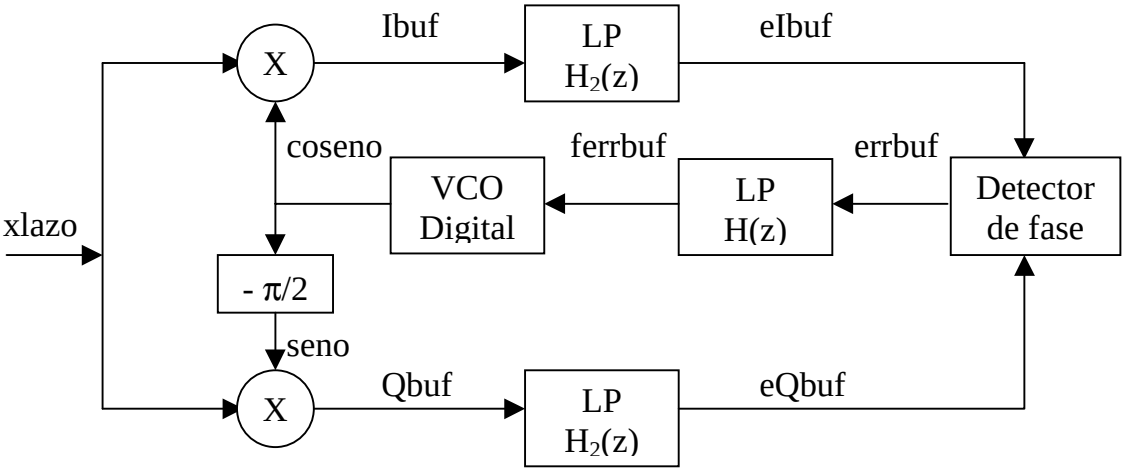


Figura 5.15: Esquema digital del lazo de Costas

Además de la señal de entrada el lazo necesita para funcionar un par de señales provenientes de un oscilador las cuales deben presentar un retraso de $\pi/2$ entre sí. La fase inicial puede ser cualquiera, pero como no se prevén variaciones grandes en la fase se ha optado por utilizar como punto de partida las mismas que en el transmisor: un seno y un coseno, ambas con fase inicial nula; el coseno será la verdadera referencia, mientras que el seno se obtendrá directamente a partir de él mediante un desplazamiento de $\pi/4$ (traducido a muestras), ya que ambos comparten las mismas posiciones de memoria, y sólo se diferencian en la posición en la que empiezan a cargarse las muestras.

Para poder programar el algoritmo se necesitan siete bufers, los cuales necesitan ser accedidos de forma circular. Puesto que sólo se dispone de ocho registros de direccionamiento y las senoides deben ocupar también alguno, se debe ir salvando el contenido de ellos de alguna forma; esto se va a realizar mediante una tabla de punteros, denominada “pbuf”, donde se irán guardando las direcciones de las posiciones de los bufers en las que se almacenarán los datos siguientes; la dirección se almacenará de forma absoluta y no en forma relativa con respecto al comienzo del bufer; de esta forma al cargar el valor de la tabla en un registro de direccionamiento se puede acceder directamente a la posición correspondiente de memoria.

Un primer vistazo al diagrama de bloques puede hacer pensar que su programación será muy complicada; sin embargo un examen más minucioso descubre que en realidad está compuesto por tres bloques iguales, en los que dos señales de entrada se multiplican y sobre el resultado se realiza una operación de filtrado, siendo la única variación los bufers que intervienen. Se puede ahorrar entonces tamaño de código si se usa una rutina auxiliar que realice estas operaciones; esta rutina se denomina “mulfin” y puesto que para las senoides no se usa el direccionamiento circular (por los mismos motivos apuntados en el transmisor) debe programarse teniendo en cuenta que uno de los bufers de entrada no se puede ser accedido de esa manera. Puesto que en el transmisor ya se explicaron detalladamente cómo realizar productos y filtrados con el DSP no se van a añadir explicaciones adicionales.

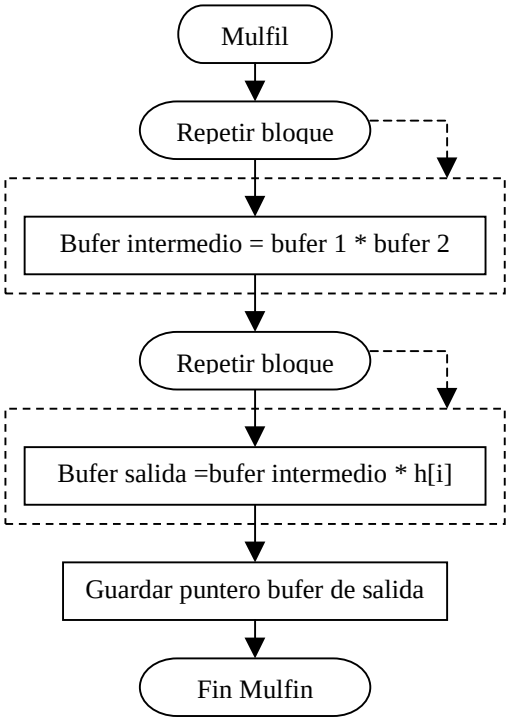


Figura 5.16: Esquema digital del lazo de Costas

```
mulfil      ldi     @crpts, rc          ; realiza un producto y un filtrado
            rptb    finmult             ; ar2 = bufer de entrada 1 (no circular)
            mpyf    *ar2++, *ar3++%, r1     ; ar3 = bufer de entrada 2
finmult     stf     r1, *ar4++          ; ar4 = bufer intermedio
            rpts    @crpts              ; ar5 = bufer de salida
            mpyf    *ar6++, *ar4++, r1  ; ar6 = coeficientes del filtro
        ||  addf    r1, r2, r2
            addf    r1, r2
            stf     r2, *ar5++%         ; guardar valor resultante
            sti     ar5, *ar1++         ; guardar posición bufer de salida

            rets
```

Con ayuda de “mulfil” la rutina del lazo de Costas sólo debe encargarse de cargar adecuadamente los registros de direccionamiento antes de cada llamada, ya que la rutina anterior ya almacena el resultado en el bufer de salida y lo prepara para la siguiente vez que vaya a utilizarse. Tras realizar la operación sobre los tres bloques (rodeados por las líneas discontinuas) sólo queda realizar una llamada al VCO.

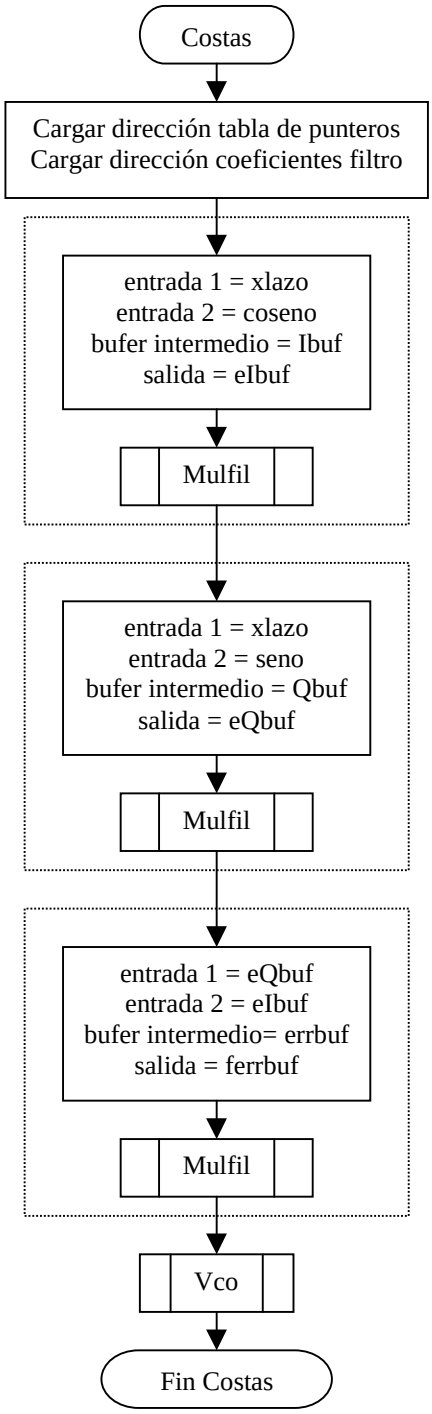


Figura 5.17: Diagrama de flujo del lazo de Costas

El código correspondiente es el siguiente:

```
costas      ldi     @pbuf, ar1          ; cargar dirección tabla de punteros
            addi    1, ar1              ; ar1 -> xlazo
            ldi     @hn, ar6            ; carga coeficientes del filtro
```

A1 sumar uno a la dirección de “pbuf” se consigue la dirección actual del bufer “xlazo” (en “pbuf” está la de “xin”, utilizada en el interpolador). También se carga la dirección de los coeficientes del filtro; aunque en el esquema aparecen tres filtros de los que sólo dos son iguales, se va utilizar el mismo para todos los casos, ya que la función que realizan todos ellos es la misma: eliminar las componentes de alta frecuencia que aparecen como consecuencia de multiplicar las dos secuencias. La respuesta en frecuencia de este filtro de paso bajo FIR de orden 16 aparece en la figura de abajo.

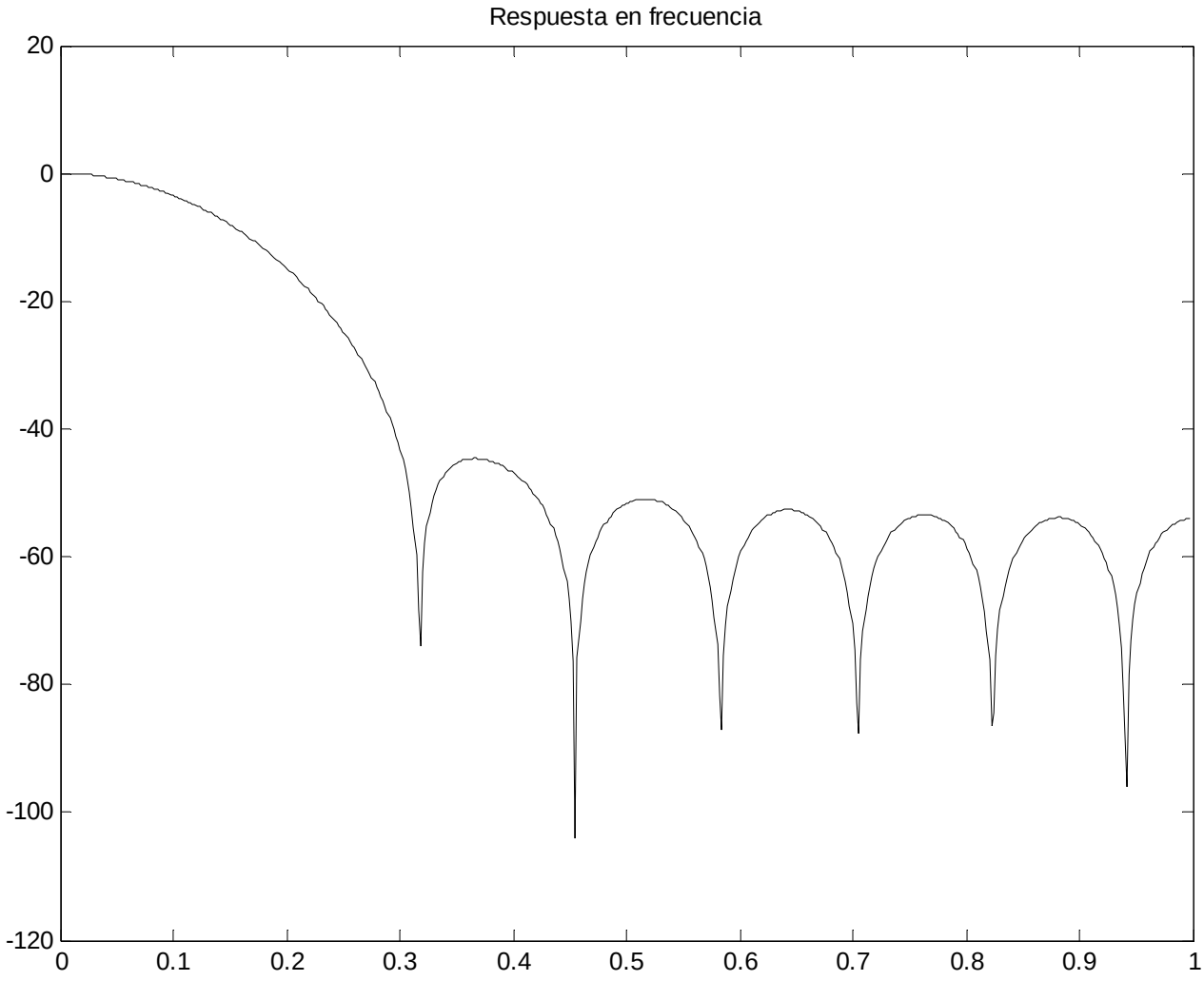


Figura 5.18: Respuesta en frecuencia del filtro del lazo de Costas

El primer bloque corresponde a la rama superior (componente en fase); la forma en la que se cargan los registros es la siguiente:

```
        ldi     @seno, ar2          ; Bloque 1
        addi    @offset, ar2        ; ar2 -> coseno
        ldi     *ar1++, ar3         ; ar3 -> xlazo
        ldi     *ar1++, ar4         ; ar4 -> Ibuf
        ldi     *ar1, ar5           ; ar5 -> eIbuf
        call    mulfil              ; ar1 -> Qbuf
```

Al finalizar el registro “ar1” apunta a “Qbuf” (por el posincremento). Para el segundo de los bloques no hay que volver a cargar la dirección de “xlazo” ya que después de ejecutar “mulfil” se ha dado una vuelta completa al bufer:

```
        ldi     @seno, ar2          ; Bloque 2; ar2 -> seno; ar3 -> xlazo
        ldi     *ar1++, ar4         ; ar4 -> Qbuf
        ldi     *ar1, ar5           ; ar5 -> eQbuf
        call    mulfil              ; ar1 -> errbuf
```

Para preparar el tercero de los bloques hay que hacer un pequeño cambio respecto a los otros dos, ya que las entradas son dos bufers circulares, cuando antes sólo lo era uno. Puesto que la dirección inicial de estos bufers va cambiando, la forma más cómoda para seguir aprovechando la rutina “mulfil” es realizar una copia de este bufer en otro cuya dirección inicial sea siempre la misma y la rutina funcione. Aunque se puede hacer esto con “eIbuf” o “eQbuf” se va a hacer sobre este último debido simplemente a que su dirección sigue guardada en el registro “ar5” y así ahorrar una instrucción:

```
        ldi     @bufaux, ar2        ; Bloque 3
        ldi     @crpts, rc          ; copia en bufer no circular
        rptb    copia
        ldi     *ar5++%, r1         ; eQbuf sigue en ar5
copia   sti     r1, *ar2++          ; ar2 -> eQbuf
```

Una vez que las entradas son las adecuadas ya se puede repetir el proceso:

```
        ldi     *ar1++, ar3         ; ar3 -> eIbuf
        ldi     *ar1++, ar4         ; ar4 -> errbuf
        ldi     *ar1, ar5           ; ar5 -> ferrbuf
        call    mulfil
```

Por último se realiza una llamada a la rutina VCO:

```
        call    vco                 ; llamada al VCO

        rets
```

Una vez que se ha hecho una pasada por el lazo hay que ver qué hacer con el valor de la fase que se obtiene; esta fase debe usarse para realimentar el lazo variando la fase de las ondas senoidales que se utilizan para realizar los productos por la señal de entrada. Para ello se va a programar un VCO digital, ya que la tarjeta no dispone de elementos hardware para realizar esta operación. Además el VCO debe ser capaz de controlar si se ha conseguido un valor para la fase más o menos estable, de manera que no se siga ejecutando el lazo si no es necesario. La condición que se va a considerar como de enganche es que el error absoluto entre el valor actual y el que se tenía cinco pasadas anteriores sea menor o igual a una centésima; esto no supone mayor inconveniente porque la salida del lazo se va almacenando, por lo que es fácil cambiar tanto la cota para este error como el procedimiento para calcularlo.

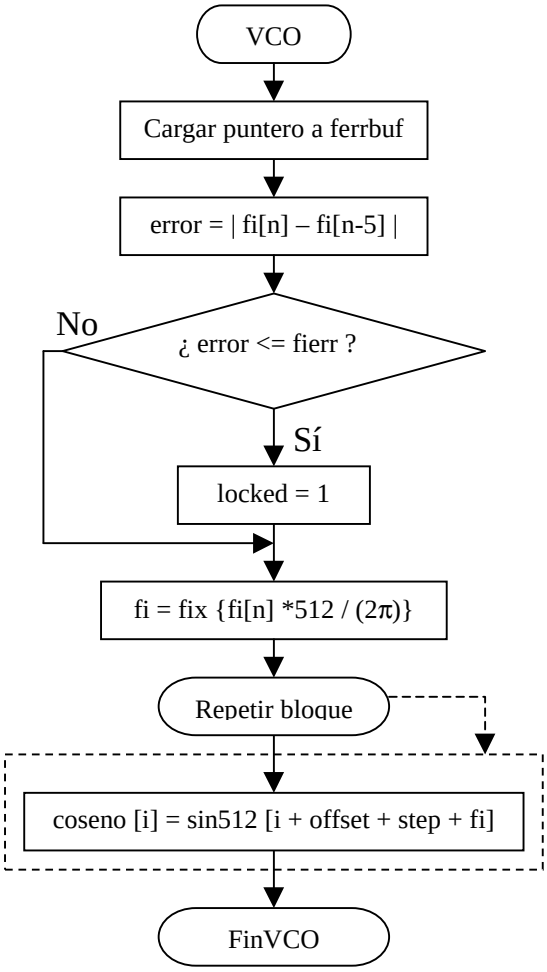


Figura 5.19: Diagrama de flujo del VCO

Para poder utilizar la fase devuelta por el lazo y ajustar las senoides hay que convertirlo en un entero para que pueda servir como índice; la operación es la siguiente:

$$fi = \left\lfloor fi[n] \ \frac{512}{2\pi} \right\rfloor$$

El redondeo se realiza conforme la instrucción “fix”, es decir, hacia “-∞”.

Lo primero que debe hacer la rutina ensamblador es cargar los valores del bufer de salida que se van a necesitar; como anteriormente se realizó un posincremento sobre el bufer para situarlo en la posición donde almacenar el siguiente, hay que resituarlo primero un lugar atrás y después cuatro más:

```
vco     subi   1, ar1          ; ar1 -> ferrbuf
        ldi    *ar1, ar5       ; ar5 -> ferrbuf[n+1]
        ldf    *ar5--%, r0     ; lectura inútil
        ldf    *ar5--%, r0     ; valor de fi[n]
        rpts   3
        ldf    *ar5--%, r1     ; ar5 -> ferrbuf[n-5]
```

Con los valores de interés ya en los registros se comprueba si se supera la cota para el error absoluto y se modifica el flag correspondiente si es necesario:

```
        ldi    @locked, r2     ; cargar flag de enganche (debe ser cero)
        subi   r0, r1          ; fi[n] - fi[n-5]
        absf   r1              ; |fi[n] - fiant[n-5]|
        cmpf   @fierr, r1      ; si es menor que el error
        ldfle  1, r2           ; locked = 1
        sti    r2, @locked     ; guardar flag de enganche
```

Ahora se transforma y escala el valor resultante del lazo:

```
        mpyf   @inv2pi, r0     ; fi[n] /(2*pi)
        mpyf   @nsin512, r0    ; 512 / (2*pi)
        fix    r0, r0          ; conversión a entero
        sti    r0, @fi         ; almacenar desfase
```

Una vez que se conoce el desfase se copian 16 muestras equiespaciadas desde la senoide de 512, lo que se consigue con la constante “step”:

```
        ldi    @seno, ar0      ; seno para demodular
        addi   @offset, ar0    ; posición del coseno
        ldi    @sin512, ar1    ; seno de 512 muestras
        addi   @fi, ar1        ; suma de la fase
        ldi    @step, ir0      ; cargar el paso entre muestras
        ldi    @crpts, rc      ; contador de repeticiones
        rptb   copy
        ldf    *ar1++(ir0), r0 ; cargar muestra siguiente de @sin512
copy    stf    r0, *ar0++      ; almacenar en @coseno

        rets
```

A modo de ejemplo se muestra la evolución de la fase de salida del lazo en ausencia de ruido; puede verse cómo se alcanza el valor teórico de π/2 (1.57 aproximadamente) y la condición de enganche se produce alrededor de la muestra 43. Si se compara con el gráfico del PLL existen claramente dos diferencias: el lazo de Costas es más lento (es decir, el tiempo de subida es mayor) y las oscilaciones son menores (la salida es más estable).

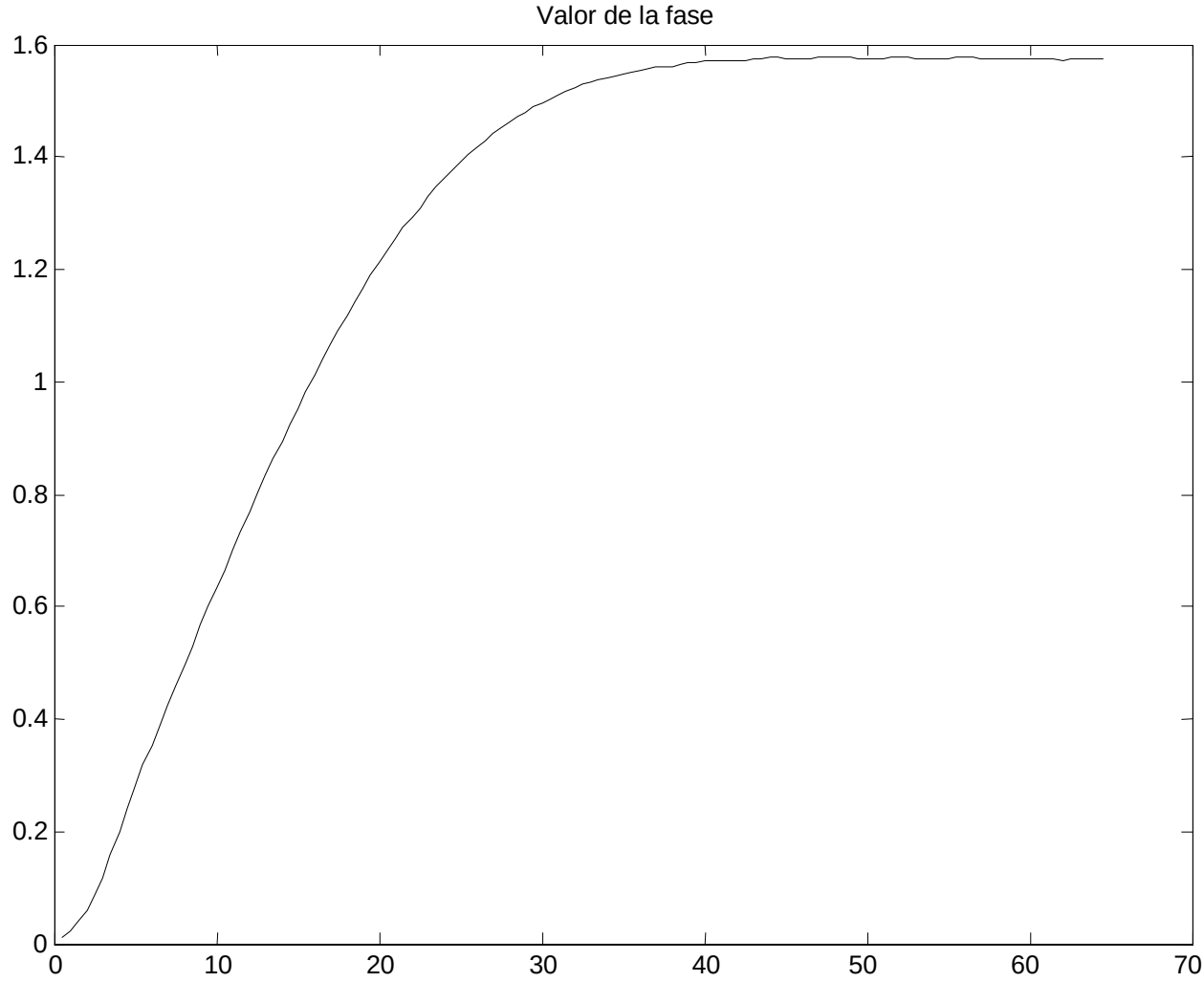


Figura 5.20: Evolución de la fase del lazo

### 5.4.4 DEMODULADOR

El proceso que realiza es muy simple, tan sólo el producto de las muestras por la señal de reloj que se tiene almacenada. La única precaución es la de ir salvando el índice que se utiliza como puntero a la tabla de la senoide, ya que el registro de índice se utiliza en otras rutinas. Los valores resultantes de la acumulación de los productos se almacenan en dos variables de memoria, "valI" para la componente en fase y "valQ" para la de cuadratura; el diagrama de flujo correspondiente puede verse a continuación:

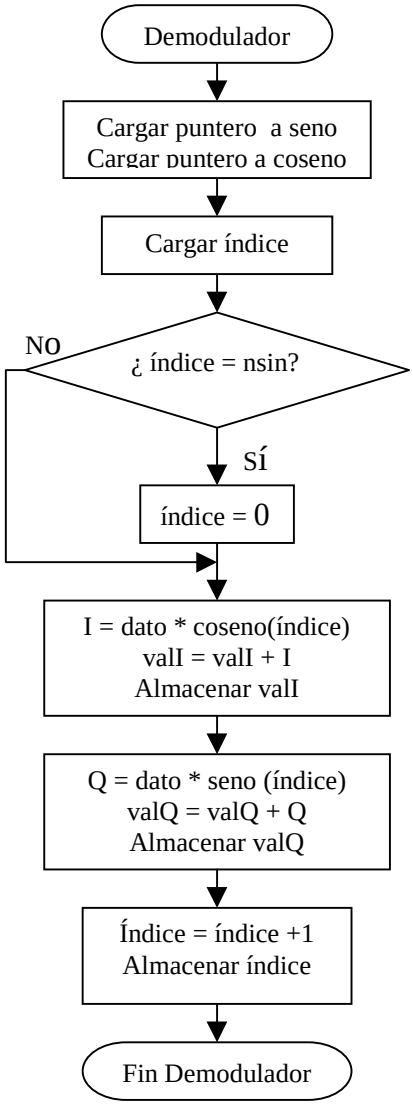


Figura 5.21: Diagrama de flujo del demodulador

Cargar las direcciones del seno y del coseno se realiza de igual forma que en el transmisor; la única modificación está en el valor del offset, que aquí es el doble debido a la interpolación ya que ahora el número de muestras de cada periodo es el doble que antes.

```
demod   ldi   @seno, ar1        ; cargar inicio seno
        ldi   @seno, ar0
        addi  @offset, ar0      ; ajustar inicio coseno
```

Puesto que para las senoides no se usa el direccionamiento circular, hay que comprobar si se ha llegado al límite de la tabla, y si es así comenzar a recorrerla desde el principio otra vez:

```
        ldi   @psinud, ir0      ; cargar índice
        cmpi  @nsin, ir0        ; si se alcanza el tope de @nsin
        ldiz  0, ir0            ; restaurarlo a cero
```

La multiplicación y acumulación se realiza con un par de instrucciones por componente; las otras dos van guardando los resultados parciales:

```
        mpyf  *+ar0(ir0), r7, r1    ; componente en fase
        addf  @valI, r1
        stf   r1, @valI
        mpyf  *+ar1(ir0), r7, r2    ; componente en cuadratura
        addf  @valQ, r2
        stf   r2, @valQ
```

Una vez que se ha utilizado la muestra de la senoide, se incrementa el índice y se guarda para ser usado la próxima vez que entre en la rutina:

```
        addi  1, ir0            ; incrementar índice
        sti   ir0, @psinud      ; almacenarlo

        rets
```

### 5.4.5 DECISOR+MULTIPLEXOR+DESCODIFICADOR

De nuevo por comodidad se agrupan estos tres bloques, ya que se obtiene mayor eficiencia de esta forma que si se trataran por separado. El funcionamiento del conjunto es el siguiente: en primer lugar se debe realizar una decisión para la componente en fase y otra para la componente en cuadratura, ya que ambas pueden tomar uno de entre cuatro valores distintos (-3, -1, 1 ó 3). Una vez detectado el valor más cercano a estos se realiza una multiplexación, formándose una cadena en la que los lugares impares los ocupan los valores provenientes de la decisión para la componente en cuadratura y los pares de fase. Entonces el descodificador toma de su entrada un par de valores y por medio de una tabla (inversa a la del codificador del transmisor) obtiene los cuatro bits del símbolo correspondientes a ese par de amplitudes. Pero esta forma de proceder es bastante ineficiente desde el punto de vista de su ejecución, por lo que razonando igual que en el codificador del transmisor se puede ganar en velocidad, ya que se puede realizar la descodificación directamente tras la decisión, y multiplexar bits y no amplitudes.

| Amplitud | Dibit |
|---|---|
| 3 | 10 |
| 1 | 00 |
| -1 | 01 |
| -3 | 11 |

Figura 5.22: Relación amplitud - bits codificados

La función de transferencia del decisor tiene entonces forma de escalera; si la entrada no supera el umbral se decide un dibit y si lo supera se compara con el siguiente umbral, procediendo de la misma forma hasta llegar al último. Fijadas las amplitudes anteriores las regiones de decisión quedan determinadas por los umbrales x=0 y x=±2, como se muestra en la figura 5.23

El número medio de comparaciones que hay que realizar se calcula como la suma del número de comparaciones necesarias para decidir cada símbolo, ponderadas por la probabilidad de que dicho símbolo se envíe:

$$N = 1*P("11") + 2*P("01") + 3*P("00") + 3*P("10")$$

ya que para decidir el último término no se necesita ninguna comparación adicional. Suponiendo símbolos equiprobables se tiene que todas las probabilidades son iguales a 0.25; entonces:

$$N = 0.25*(1+2+3+3) = 2.25$$

Como hay que hacerlo para ambas componentes, el número medio total de comparaciones necesarias para cada símbolo es de 4.5 comparaciones.

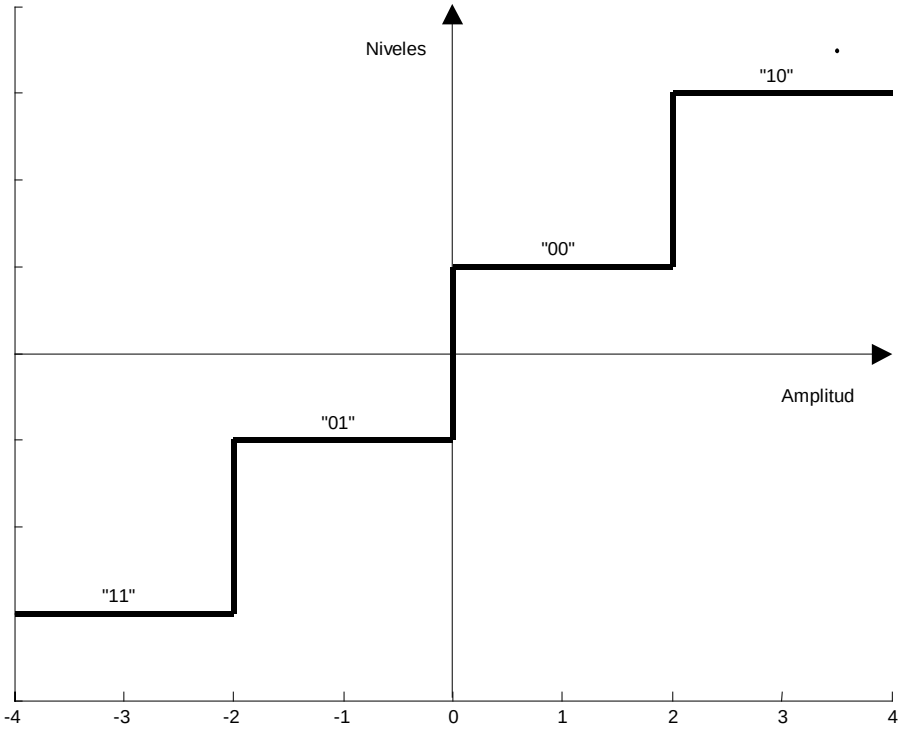


Figura 5.23: Decisor de 4 niveles

Aún se puede acelerar más la ejecución mediante un algoritmo similar al usado en el codificador. Se supone un símbolo, el correspondiente a I=1 y Q=1 (el 0000 en la constelación), y se hace lo siguiente:

- Si |Q| >2, el bit 1 vale 1.

- Si |I| >2, el bit 2 vale 1.

- Si Q es negativo, el bit 3 vale 1.

- Si I es negativo, el bit 4 vale 1.

Con el método anterior se baja a una media determinista de 4 comparaciones para obtener el símbolo correcto en la constelación, frente a las algo menos de 8.5 comparaciones necesarias con la tabla directa de 16 símbolos y las 4.5 del método anterior. Aunque la mejora obtenida en este último paso pueda parecer escasa para justificar su programación, presenta como ventaja añadida que el código generado es también más corto y no necesita tablas auxiliares, lo que redunda en una mayor velocidad.

Usando el mismo ejemplo que en el transmisor pero a la inversa, las series de amplitudes I = 1, -3, -1, 3 y Q = 1, 3, -3, -1 proporcionan la cadena de datos binarios: 0000 1101 1011 0110.

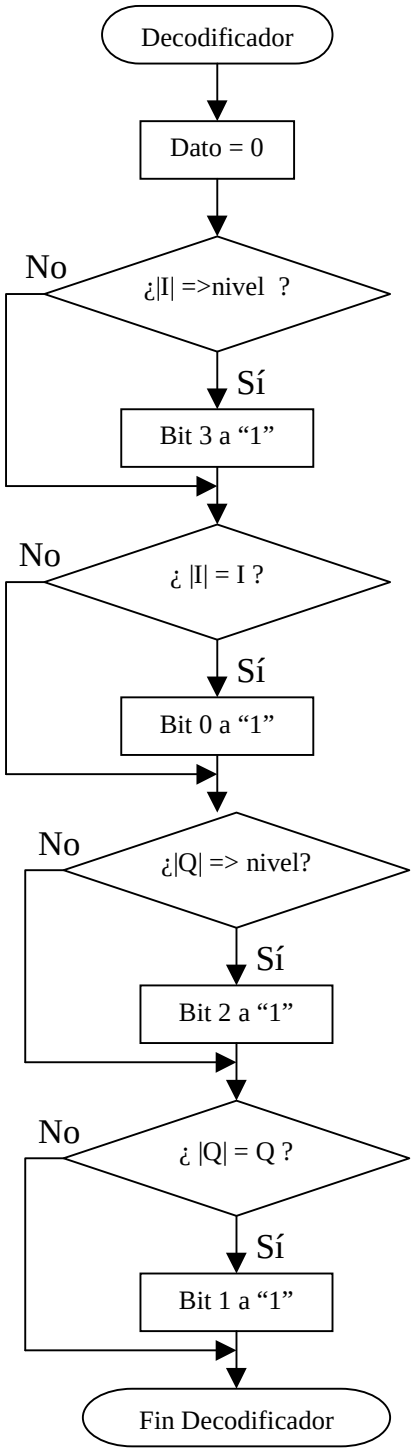


Figura 5.24: Diagrama de flujo del Decodificador

La programación del algoritmo en ensamblador es la siguiente:

```
decis   ldi     0, r7           ; valor por defecto
        ldi     0, r5           ; componente en fase
        absf    @valI, r6       ; r6 = |I|
        cmpf    @nivel, r6      ; ¿|I| => nivel?
        ldige   @ibig, r5       ; bit a 3 a "1"
        addi    r5, r7
        ldi     0, r5
        cmpf    @valI, r6       ; ¿I = |r6|?
        ldinz   @ineg, r5       ; bit 0 a "1"
        addi    r5, r7

        ldi     0, r5           ; componente en cuadratura
        absf    @valQ, r6       ; r6 = |Q|
        cmpf    @nivel, r6      ; ¿|Q| => nivel?
        ldige   @qbig, r5       ; bit a 2 a "1"
        addi    r5, r7
        ldi     0, r5
        cmpf    @valQ, r6       ; ¿Q = |r6|?
        ldinz   @qneg, r5       ; bit 1 a "1"
        addi    r5, r7          ; el dato está en r7

        rets
```

# 6 CONCLUSIONES

Una vez expuesto todo lo anterior y comprobado su funcionamiento sólo resta indicar partes mejorables en el sistema y esbozar alguna futura línea de desarrollo.

La primera y principal contrariedad para desarrollar el sistema de comunicación reside en la tarjeta de evaluación seleccionada, ya que está pensada para trabajar con señales de voz y dispone de elementos optimizados para ese fin. De esta forma las frecuencias implicadas están muy limitadas ya que como se ha visto la velocidad máxima de muestreo es de 19.2 KHz; si a esto se une que cada periodo de la portadora debe tener al menos ocho muestras y que deben enviarse al menos cuatro periodos completos de dicha portadora, se tiene que la tasa de símbolos enviados es de 600 bps, aunque la modulación 16-QAM utilizada provoca que la tasa efectiva de datos sea de 2.4 KHz (porque cada símbolo modulado lleva 4 bits de información). Por lo tanto el sistema se vería muy beneficiado en cuanto a rendimiento con la inclusión de convertidores más rápidos; además algunas partes críticas del código deberían sustituirse por un dispositivo hardware, como por ejemplo el VCO (si puede ser el lazo de Costas completo o el PLL al menos mucho mejor), lo que sin duda también facilita que se eleve la frecuencia a la que trabaja el sistema; también cambiar los amplificadores de entrada por unos con ganancia ajustable (para el CAG); o incluir un oscilador independiente del que posee internamente el DSP para ajustar mejor la frecuencia de muestreo, ya que el uso de un timer puede producir errores cuando se utilizan como fuente de sincronización.

En cuestión de cambios en el diseño la mejora fundamental estaría en utilizar una modulación TCM (*Trellis-Coded Modulation*), que combina la codificación de canal y la modulación en un único bloque, de forma que estas operaciones no se optimicen por separado (lo que lleva a resultados subóptimos) [13]. El uso de una codificación convolucional introduce dependencias entre señales consecutivas y se hace necesario aumentar el número de puntos de la constelación para acomodar los bits de redundancia sin necesidad de incrementar el ancho de banda de transmisión. Aunque el proceso de decisión que debe llevar a cabo el receptor presenta un coste computacional mayor que los esquemas clásicos (debido al algoritmo de Viterbi), se produce una mejora neta en la probabilidad de error.

Entre los cambios que pueden realizarse en el interior de los bloques está la modificación del control automático de ganancia, ya que el empleado es excesivamente simple y no funciona adecuadamente cuando el canal presenta demasiado ruido, por lo que se puede sustituir por algún algoritmo de cálculo de potencia que no requiera demasiado número de operaciones, si ello es posible.

Respeto a la sincronización hay que comentar que la condición impuesta para el enganche de fase en el lazo de Costas puede ser poco restrictiva si el canal presenta demasiado ruido de fase, de manera que no sea suficiente para que aparezcan errores en la recepción con el valor considerado en la condición de enganche, haciéndose necesario variar la cota asumible para el error o modificar el método de cálculo de ese error (por la media de un cierto número de valores o minimizando alguna función de error). Por otro lado, puesto que el canal varía con el tiempo, en transmisiones cortas el valor obtenido con el lazo es válido para un determinado intervalo; si la cantidad de datos a enviar es muy grande, superado ese intervalo empiezan a recibirse datos incorrectos por pérdida de sincronización, pudiendo resolverse reenviando la señal piloto una vez mandados cierto número de símbolos. Todo esto lleva a pensar en intentar otro esquema de demodulación distinto: ya que se tiene un lazo de Costas puede usarse para detectar la fase de cada símbolo, y una vez que se ha hecho esto proceder a detectar su amplitud (posiblemente se hiciera necesario una codificación diferencial para los símbolos), para lo que sería más conveniente emplear una modulación 16-QAM rectangular (habría 4 fases distintas, cada una con 4 símbolos posibles) ya que como se dijo con anterioridad la QAM equivale a una APK.

Por otro lado los filtros usados en el lazo de Costas realizan su función pero no son los mejores que se pueden elegir; aunque los utilizados se han diseñado por el método de las ventanas, existen métodos avanzados de diseño que permiten ajustar mejor la banda pasante y la frecuencia de corte (puede cambiarse el tipo de ventana, o diseñarse un filtro IIR, con todo lo que ello supone). Además se pueden usar filtros distintos dentro del lazo, ya que cada uno actúa sobre señales distintas conocidas a priori, por lo que se puede diseñar cada filtro a medida. Por otro lado se puede ampliar el orden de esos filtros a 31 sin modificar el esquema de memoria ya que las exigencias del direccionamiento circular obligaron a reservar 32 palabras en memoria para cada filtro cuando el orden es 16.

En último lugar el programa ha sido concebido suponiendo que no existe nada más, ni otro programa ejecutándose ni entidades de nivel superior que lo controlen, por lo que conoce el número de datos a enviar y cuáles son esos datos. Para que el sistema sea más real debe diseñarse una aplicación que, trabajando sobre esta (por ejemplo funcionando desde el PC), le comunique cuándo comenzar la transmisión y cuándo ponerse a la espera para recibir, y que mediante una trama (de formato fijo y que se envíe en primer lugar) pueda configurar el extremo remoto para que se inicie el envío de datos. Además esta aplicación podría llevar un control de errores básico (basado en códigos CRC, por ejemplo) y, con alguna ligera modificación de la rutina de configuración para permitir ejecutar modulador y demodulador en la misma máquina, poder incluso llevar un control sobre el flujo de datos.

# 7 APÉNDICES

## 7.1 COMPATIBILIDAD DE PROGRAMAS

Se ha comprobado que los listados propuestos funcionan correctamente en los siguientes programas y versiones. La ejecución sobre versiones anteriores no se puede garantizar completamente debido al uso de técnicas y herramientas propias de dichas versiones:

TMS320C30 Development Tools 4.50

TMS320C30 Simulator 1.30

TMS320C30 High Level Language Debugger - Release 4.60

(los anteriores funcionando bajo Ms-Dos 6.22)

Matlab 6.0.0.88 - Release 12

(funcionando bajo Windows XP)

## 7.2 COSTE COMPUTACIONAL

A continuación aparece una estimación más o menos precisa del coste computacional en número de ciclos de las rutinas programadas en ensamblador; aparecen las constantes: m=4, nsin = 8, nsimb = 8 y npal =2. Para el transmisor:

| | | | |
|---|---|---|---|
| receive0: | | | 16 ciclos |
| codif: | | | 20 ciclos |
| ftx: | 8 + recevive0 | = | 24 ciclos |
| mod: | 13 + nsin (7 + ftx) | = | 261 ciclos |
| sincr: | 8 + nsin (5 + ftx) | = | 240 ciclos |

Con estos datos se puede calcular el número de ciclos del conjunto:

19 + sincr + npal (7 + nsimb (13 + codif + mod)) = 4977 ciclos

Procediendo de igual manera para el receptor:

| | | | |
|---|---|---|---|
| receive0: | | | 12 ciclos |
| decis: | | | 26 ciclos |
| demod: | | | 18 ciclos |
| frx: | 10 + receive0 | = | 22 ciclos |
| vco: | 33 + 4 * nsin | = | 97 ciclos |
| mulfil: | 16 + 4 * nsin | = | 48 ciclos |
| interp: | 30 + nsin | = | 38 ciclos |
| costas: | 40 + 2 * nsin + 3 * mulfil | = | 200 ciclos |

El cálculo para el receptor completo se realiza ahora como:

npal * (16 + m*nsimb*(21+frx+interp + 2*demod)+decis)+
+ 61 + m * nsin (24 + frx + interp + 2 * costas) = 46065 ciclos

El receptor ejecuta más instrucciones debido principalmente al uso de la interpolación (que duplica el procesamiento) y el lazo de costas.

Más importancia tiene el número de instrucciones ejecutadas durante el tiempo que pasa entre dos ejecuciones seguidas de la rutina de interrupción, que vale unos 50 μseg. (1/19200). El caso más crítico es el receptor recibiendo la señal de sincronismo; en este caso se tiene:

40 + frx + interp + 2 * costas = 500 ciclos

Si el DSP tarda 60 nseg. en ejecutar cada instrucción (reloj de 15 MHz) se tardarían unos 30 μseg., es decir, el micro queda libre el 40% del tiempo.

## 7.3 SIMULACIÓN

A continuación se encuentra el listado de la simulación del sistema completa para Matlab (versión 6). Posee numerosos comentarios para que pueda seguirse su desarrollo:

```
clear;rand('state',0);randn('state',0);
q=4;                          % Bits por símbolo (constelación 16-QAM)
fc=2.4e3;
fs=19.2e3;                    % Frecuencias de portadora y de muestreo
n=fs/fc;                      % Número de muestras por símbolo
nsimb=4*q;                    % Número de DATOS a tx (múltiplo de q)
m=8;                          % Ciclos de portadora por símbolo
L=2*n;                        % Longitud filtro interpolación (0:L-1)
var=0.05;mean=0;              % Varianza y media del canal
ltrama0=(1+nsimb/q)*n*m;      % Longitud de la trama TOTAL
fasetx=pi/8;                  % Fase oscilador local transmisor
faserx=0;                     % Fase oscilador local receptor
ts=(0:m*2*(fs/fc)-1)'/fs;     % Base de tiempos
Is0=cos(2*pi*fs*ts/L+fasetx);
Qs0=sin(2*pi*fs*ts/L+fasetx);       % Senoides, m ciclos
h=fir1(L-1,0.5)';                   % Filtro LP de interpolación
g=fir1(L-1,0.1)';                   % Filtro LP

% Varios búfers y variables inicializados todos a cero
buf=zeros(L,1); fil=buf; vc=buf;
mI=0; Irx=0; mulI=buf; filI=buf;    fi0=buf;
mQ=0; Qrx=0; mulQ=buf; filQ=buf;    fi=buf;

itemp=1;                % Índice temporal
isen=1;                 % Puntero a senoides
cont=2*m*n;             % Variable contador
ncont=0;                % Número de muestras válidas para demodular
pilot=1;                % Señal piloto
maxi=0.5;               % Mitad valor máximo admisible para entrada VCO
gain=0.6;               % Ganancia VCO
err=1e-3;               % Error admitido para el enganche de fase
eng=0;                  % Marca de enganche
Is=cos(2*pi*fs*ts/L);
Qs=sin(2*pi*fs*ts/L);   % Aproximación senoides

% Vectores dummy (se necesita crearlos antes de usarlos)
sincr=[];         % Señal de sincronismo piloto
xmod=[];          % Señal modulada
xout=[];          % Vector recibido

% Varias señales de entrada binaria; debe quitarse el comentario
% según se desee una u otra. Por defecto se modula la denominada
% “señal patrón”; que tiene un símbolo de cada fila y uno de cada
% columna. Nótese que es necesario modificar “nsimb”.
nsimb=4*q;xin=[0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0]';    % Señal Patrón

% Todos los símbolos, en orden según numeración
%nsimb=16*q;xin=[0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 ...
                 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 ...
                  1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1]';
% Datos binarios aleatorios (distribución uniforme)
%nsimb=128*q;xin=round(rand(nsimb,1));         % Datos aleatorios
```

```
ltrama0=(1+nsimb/q)*n*m;      % Si cambia “nsimb” hay que recalcularla

% Modulador
for ind=1:m
   sincr=[sincr;Qs0(1:2:L)];       % m ciclos, señal piloto
end

for ind=1:q:nsimb             % Se agrupan q bits
z=xin(ind:ind+q-1);           % Codificador + Desmultiplexor
   I=1;Q=1;                   % Se supone 0000; se modifica según z(i)
   if z(1)==1 Q=3; end        % MSB
   if z(2)==1 I=3; end
   if z(3)==1 Q=-Q; end
   if z(4)==1 I=-I; end       % LSB
   fase=I*Is0(1:2:length(Is0));     % Componente en fase
   quad=Q*Qs0(1:2:length(Qs0));     % Componente en cuadratura
   xmod=[xmod; fase+quad];          % Suma
end

canal=[sincr;xmod];                 % Señal que es enviada
ruido=canal+sqrt(var)*randn(size(canal))-mean;       % Canal ruidoso

% Demodulador
rxin=filter(hcn,hcd,ruido);         % Filtrado BP
it=[interp(rxin,2)];                % Interpolación
ltrama=ltrama0*2;                   % Nueva longitud de trama
ind=1;                              % Variable índice
acum=0;                             % Acumulador de potencia
while ltrama>0
   for incr=0:1                     % 2 pasadas
      dato=it(itemp+incr);
      ltrama=ltrama-1;              % Decrementar el contador de trama
      cont=cont-1;                  % Decrementar el contador
      if pilot==1                   % cag
         acum=acum+dato^2;
         ncont=ncont+1;
         if ncont==2*m*n acum=acum/ncont;end
      else
         dato=dato*sqrt(0.5/acum);
      end
      if pilot==1                   % Señal piloto
         % Lazo de Costas
         if abs(dato)>maxi maxi=abs(dato);end        % Limitador
         mulI(L)=dato*Is(isen)/maxi;
         mulQ(L)=dato*Qs(isen)/maxi;
         filI=filter(g,1,mulI);
         filQ=filter(g,1,mulQ);
         fil=filI.*filQ;                        % Detector de fase
         vc=filter(g,1,fil);                    % Filtro del bucle
         fi0(L)=fi0(L-1)+vc(L);                 % VCO digital
         fi=filter(g,1,fi0);                    % Salida filtrada
         Is=gain*cos(2*pi*fs*ts/L+fi0(L));
         Qs=gain*sin(2*pi*fs*ts/L+fi0(L));      % Generación senoide
         for j=2:L                            % Desplazamiento bufers
            mulI(j-1)=mulI(j);
            fil(j-1)=fil(j);
            vc(j-1)=vc(j);
            mulQ(j-1)=mulQ(j);
            fi0(j-1)=fi0(j);
         end
         isen=isen+1;               % Incrementar índice a senoides
```

```
         if (abs(fi(L-3)-fi(L))<=err & eng==0)
            eng=1;                  % Marca de enganche de fase
         end
         if cont==0
            faserx=fi(L);           % Fase resultante del lazo
            fil=zeros(L,1);
            mulI=fil;
            filI=fil;
            fi0=fil;                % Limpiado de bufers
            vc=fil;
            mulQ=fil;
            filQ=fil;
            fi=fil;
            Is=Is/gain;Qs=Qs/gain;  % Reinicio y escalado senoides
            isen=1;                 % Reinicio puntero a senoides
            maxi=0.5;               % Restaurar el valor inicial
            cont=2*m*n;             % Reinicio contador
            pilot=0;                % Señal piloto recibida
            ncont=0;                % Reiniciar número de muestras
            eng=0;                  % Limpiar marca de enganche
         end
      else       % Dato (pilot==0)
         mI=dato*Is(ind+incr);      % Multiplicar componente en fase
         Irx=Irx+mI;                % Se acumula
         mQ=dato*Qs(ind+incr);      % Multiplicar cuadratura
         Qrx=Qrx+mQ;                % Se acumula
         ncont=ncont+1;             % Se incrementa el contador
         if cont==0
            cont=2*m*n;             % Reiniciar contador
            div=ncont/2;
            Irx=Irx/div;Qrx=Qrx/div;
            I=round(Irx);Q=round(Qrx);    % Decisor
            z=[0 0 0 0]';                 % Multiplexor+ Descodificador
            if (abs(Q)>=2) z(1)=1;end     % MSB
            if (abs(I)>=2) z(2)=1;end
            if (abs(Q)~=Q) z(3)=1;end
            if (abs(I)~=I) z(4)=1;end     % LSB
            xout=[xout;z];
            Irx=0;
            Qrx=0;
            ncont=0;                      % Reiniciar contador
            Irx=0;
            Qrx=0;                        % Limpiar acumuladores
         end
      end
 end
   itemp=itemp+2;                   % Incrementar el índice temporal
   ind=ind+2;                       % Incrementar índice en 2
   if ind>=(2*m*n+1)                % Si se excede el tamaño de las
      ind=1;                        % senoides, se vuelve a apuntar
   end                              % al principio
end
```

# 8 REFERENCIAS

[1] AN INTODUCTION TO ANALOG AND DIGITAL COMMUNICATIONS, Simon Haykin, Ed. John Wiley & Sons, 1989.

[2] COMMUNICATION SYSTEM, Simon Haykin, Ed. John Wiley & Sons, 1994 (3ª edición).

[3] DIGITAL COMMUNICATIONS, Edward A. Lee - Ed. Kluwer Academic Publishers, 1997 (2ª edición).

[4] DIGITAL COMMUNICATIONS, John G. Proakis, Ed. Mc Graw - Hill, 1995 (3ª edición).

[5] DIGITAL PHASE MODULATION, John B. Anderson, Ed. Plenum Press, 1986.

[6] DIGITAL TRANSMISSION ENGINEERING, John B. Anderson, Ed Prentice-Hall.

[7] SIMULATION OF COMMUNICATION SYSTEMS, Jeruchim - Balaban - Shanmugan. Ed. Plenum Press, 1992.

[8] DIGITAL PROCESSING SYSTEMS, A COMPUTER APROACH, Mitra.

[9] INTRODUCTION TO COMMUNICATION SYSTEMS, F.G. Stremler, Ed. Addison-Wesley, 1990 (3ª edición).

[10] DIGITAL AND ANALOG COMMUNICATON SYSTEMS, L.W. Couch II, Ed. Prentice-Hall, 1997 (5ª edición).

[11] BLIND CARRIER PHASE ACQUISITION FOR QAM CONSTELLATIONS, Costas N. Georghiades, IEEE Transactions on Communications, vol. 45, nº 11, Noviembre 1997.

[12] INTRODUCTION TO SIGNAL PROCESSING, S.J. Orfanidis, Ed. Prentice-Hall, 1996.

[13] ADVANCED DIGITAL SIGNAL PROCESSING, THEORY AND APPLICATIONS. G. Zelniker - F.J. Taylor, Ed Marcel Dekker, 1994.

[14] ARQUITECTURA DE COMPUTADORES, UN ENFOQUE PRÁCTICO, J.L. Hennessy – D.A. Patterson, Ed. McGraw-Hill, 1995.

[15] DIGITAL SIGNAL PROCESSING APPLICATIONS WITH THE TMS320 FAMILY, VOL 1, K. Lin, Ed. Prentice-Hall, 1987.

[16] TMS320C3x, User's Guide, Digital Signal Processing Products, revisión F, Texas Instruments, 1992.

[17] DIGITAL SIGNAL PROCESSING APPLICATIONS WITH THE TMS320C30 EVALUATION MODULE, Selected Aplication Notes, Digital Signal Processing Products, revisión A, Texas Instruments, 1991.

[18] TMS320C30x EVALUATION MODULE, Technical Reference, Microprocessor Development Systems, revisión B, Texas Instruments, 1990.

[19] TMS320 FLOATING-POINT DSP ASSEMBLY LANGUAJE TOOLS, User's Guide, Microprocessor Development Systems, revisión B, Texas Instruments, 1995.

[20] TMS320C30x C SOURCE DEBUGGER, User's Guide, Microprocessor Development Systems, revisión F, Texas Instruments, 1991.

[21] A DIGITAL SIGNAL PROCESSING LABORATORY USING THE TMS320C30, H.V. Sorensen, Ed. Prentice-Hall, 1997.

[22] MASTERING MATLAB 5, A COMPREHENSIVE TUTORIAL AND REFERENCE, D.Hanselman - B. Littlefield, Ed. Prentice-Hall, 1998.

[23] Apuntes de la asignatura TRANSMISIÓN DE DATOS, 3º Ingeniería de Telecomunicaciones (Plan del 91).

[24] Apuntes de la asignatura LABORATORIO DE TRANSMISIÓN DE DATOS, 3º Ingeniería de Telecomunicaciones (Plan del 91).

[25] Apuntes de la asignatura SISTEMAS ELECTRÓNICOS DIGITALES, 3º Ingeniería de Telecomunicaciones (Plan del 91).

[26] Apuntes de la asignatura TEORÍA DEL CONTROL AUTOMÁTICO, 3º Ingeniería de Telecomunicaciones (Plan del 91).

[27] Apuntes de la asignatura TRATAMIENTO DIGITAL DE SEÑALES, 4º Ingeniería de Telecomunicaciones (Plan del 91).

[28] Apuntes de la asignatura LABORATORIO TRATAMIENTO DIGITAL DE SEÑALES, 4º Ingeniería de Telecomunicaciones (Plan del 91).

[29] Apuntes de la asignatura TRATAMIENTO DIGITAL DE SEÑALES EN COMUNICACIONES, 5º Ingeniería Telecomunicaciones (Plan del 91).

**Agradecimientos**

A mis padres por su ayuda y apoyo incondicional.

A mi hermano por esas pizzas+peli y su “ambiente” musical.

A Miguel Angel y su experiencia, su escaner y su (mi) grabadora.

Al artista conocido como Antonio Jesús, y su dominio sobre Photoshop.

Al jefe y sus años de amistad, a su campo y a su tía Patro.

A mi vecinita de arriba, la más “mala” y a la vez más “guapa”.

A los perdigoncitos y a mamá perdigona, por esas juerguecitas gitanas.

Al “Agu” y su apoyo logístico.

A mis “churris”, que de vez en cuando se acuerdan de mi.

A mis “chochis”, que nunca me dejan en paz.

A cierta enfermera que anda por ahí, por sus cuidados.

A Haykin, Proakis & Friends, por mantener mi nivel de inglés.

A los creadores del “Blues” y el “Rock ’N’ Roll”.

A todo el clan de los McLoud y en especial a J.D.

A mi portátil, por ser un campeón tanto con Linux como con XP.

Sin vosotros este proyecto no hubiera sido posible. Gracias a todos.

*Juan Ignacio*