

**ESCUELA POLITÉCNICA NACIONAL**

**FACULTAD DE INGENIERÍA ELÉCTRICA**

**TESIS DE GRADO  
PREVIA A LA OBTENCIÓN DEL TÍTULO  
DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES**

**ANÁLISIS Y SIMULACIÓN DE UN SISTEMA DE TRANSMISIÓN  
DIGITAL UTILIZANDO MATLAB CON SIMULINK.**

**JOHN DAVID PAREDES VALENCIA**

**JULIO 1999**

**Certifico que, bajo mi dirección,  
la presente tesis fue realizada  
en su totalidad por el  
Sr. John David Paredes Valencia**



---

**Ing. María Soledad Jiménez  
DIRECTORA DE TESIS**

## *Dedicatoria*

*A mis Padres, de  
quienes brotó siempre  
el amor y comprensión*

# *Agradecimiento*

*A la Dng, María Soledad Jiménez,  
por su constante apoyo y su acertada  
dirección, para la realización de esta tesis*

# ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN AL MATLAB Y SIMULINK.....	1
1.1 INTRODUCCIÓN.....	1
1.2 CARACTERÍSTICAS DEL MATLAB® ( <i>MATRIX LABORATORY</i> ).....	1
1.2.1 El lenguaje MATLAB® .....	2
1.2.2 Análisis de Datos, Rastreo, Gráficos y Visualización.....	7
1.2.3 Introducción al uso del MATLAB.....	8
1.2.4 Características del SIMULINK® .....	12
1.2.5 Sistemas de Modelación a través de diagramas de bloques.....	13
1.2.6 Simulación y análisis.....	15
1.2.7 Arquitectura abierta.....	16
1.2.8 SIMULINK® y librerías adicionales.....	17
1.2.9 Construcción de un modelo simple.....	17
1.3 LIMITACIONES.....	27
1.4 REFERENCIAS.....	28
CAPÍTULO 2. INTRODUCCIÓN AL SISTEMA DE SIMULACIÓN.....	29
2.1 LIBRERÍAS DEL SISTEMA DE SIMULACIÓN.....	29
2.1.1 Librería Fuentes.....	32
2.1.1.1 Simulación de Fuentes de Datos Digitales.....	33
2.1.1.2 Simulación de Retardos de Señal.....	35
2.1.1.3 Simulación de Señal de Reloj.....	37
2.1.2 Librería Análisis.....	38
2.1.2.1 Simulación de Osciladores Controlados por Voltaje.....	39
2.1.2.2 Simulación de un circuito monoestable.....	43
2.1.2.3 Simulación de un repetidor regenerativo.....	44

2.2	DENSIDAD ESPECTRAL DE POTENCIA.....	45
2.2.1	Transformada Discreta de Fourier.....	46
2.2.2	Transformada Rápida de Fourier.....	55
2.2.3	Cálculo numérico de la densidad Espectral de Potencia.....	61
2.2.4	Bloque y Algoritmo de simulación para el análisis de la Densidad Espectral de Potencia de una señal.....	63
2.3	SIMULACIÓN DE UN CANAL DE TRANSMISIÓN CONTAMINADO CON RUIDO GAUSSIANO BLANCO.....	65
2.3.1	Transmisión sin Distorsión.....	65
2.3.2	Características del Filtro de un canal de transmisión.....	67
2.3.3	Ruido Blanco Gaussiano de banda limitada.....	69
2.3.4	Bloque de Simulación que permite el análisis del efecto de un canal con ruido Blanco Gaussiano.....	70
2.4	DIAGRAMA DEL OJO.....	71
2.4.1	Simulación para la obtención del diagrama del Ojo.....	73
2.4.2	Ejemplo de Simulación del diagrama ocular y de un canal ruidoso.....	74
2.5	REFERENCIAS.....	77
CAPÍTULO 3. SIMULACIÓN DE CÓDIGOS DE LÍNEA.....		79
3.1	CÓDIGOS DE LÍNEA.....	79
3.2	ALGORITMOS DE CODIFICACIÓN Y BLOQUES DE SIMULACIÓN.....	81
3.2.1	Código NRZ Unipolar (Non Return to Zero) .....	84
3.2.2	Código NRZ polar (Non Return to Zero).....	87
3.2.3	Código RZ unipolar (Return to Zero).....	91
3.2.4	Código RZ polar (Return to Zero).....	95
3.2.5	Código AMI (Alternate Mark Inversion).....	99
3.2.6	Código HDB3(High Density Bipolar 3).....	106
3.2.7	Código B3ZS (BINARY with 3 Zeros Substitution).....	112

3.2.8	Código Bifase L o Manchester.....	114
3.2.9	Código Bifase - M (Biphase- Mark) y Bifase - S (Biphase - Space).....	119
3.2.10	Código Miller o Código de modulación por retardo.....	122
3.2.11	Código 4B3T(4 Binary into 3 Ternary).....	125
3.2.12	Código CMI (Coded Mark Inversion).....	131
3.2.13	Código PST (Pair Selected Ternary).....	134
3.2.14	Código 2B1Q (Two Binary One Quaternary).....	137
3.2.15	Código mBnB.....	141
3.2.16	Código MS43(Monitoring State 4 3).....	145
3.3	DENSIDAD ESPECTRAL DE POTENCIA DE LOS CÓDIGOS DE LÍNEA.....	49
3.3.1	Ejemplo de Simulación de los códigos de Línea.....	149
3.3.2	Ejemplo de la obtención de la Densidad Espectral de Potencia de un Código de Línea.....	155
3.4	EJEMPLO DE SIMULACIÓN DE UN CANAL DE TRANSMISIÓN CON RUIDO GAUSSIANO BLANCO EN UNA TRANSMISIÓN EN BANDA BASE.....	158
3.5	REFERENCIAS.....	160
CAPÍTULO 4. MODULACIÓN DIGITAL Y SIMULACIÓN.....		161
4.1	MODULACIÓN ASK.....	162
4.1.1	Modulación On-Off Keying (OOK).....	164
4.1.1.1	Simulación de la modulación OOK... ..	165
4.1.2	Modulación PRK (Phase Reversal Keying) o BPSK (Binary Phase Shift Keying).....	168
4.1.2.1	Simulación PRK.....	170
4.1.3	Demoduladores ASK.....	170
4.1.3.1	Demodulación ASK no coherente.....	171
4.1.3.2	Simulación Demodulación no coherente.....	172

4.1.3.3	Demodulación ASK coherente.....	173
4.1.3.4	Simulación de la Demodulación ASK coherente.....	174
4.2	MODULACIÓN FSK ( <i>FREQUENCY SHIFT KEYING</i> ).....	175
4.2.1	Modulación FSK con discontinuidad de fase.....	175
4.2.1.1	Simulación FSK con discontinuidad de fase.....	176
4.2.2	Modulación FSK con fase continua.....	177
4.2.2.1	Simulación FSK con fase continua.....	179
4.2.3	Demodulación FSK mediante Detector de cruces por cero.....	180
4.2.3.1	Simulación del demodulador FSK mediante detección de cruces por cero.....	181
4.2.4	Demodulación FSK mediante PLL (Phase Locked Loop).....	183
4.2.4.1	Simulación del demodulador FSK mediante PLL.....	184
4.2.5	Densidad Espectral de Potencia de la señal FSK.....	186
4.2.6	Modulación MSK (Minimun Shift Keying).....	189
4.3	MODULACIÓN PSK ( <i>PHASE SHIFT KEYING</i> ).....	192
4.3.1	Moduladores PSK.....	196
4.3.1.1	Modulación PSK mediante generación de múltiples fases.....	96
4.3.1.2	Modulación PSK como combinación lineal de señales en cuadratura.....	98
4.3.1.3	Simulación del Modulador 2-PSK por red de defasaje.....	199
4.3.1.4	Simulación del modulador 2-PSK como combinación lineal de señales en cuadratura. ....	201
4.3.1.5	Simulación del Modulador 4-PSK por red de defasaje.....	203
4.3.1.6	Simulación del modulador 4-PSK como combinación lineal de señales en cuadratura... ..	205
4.3.1.7	Simulación del modulador 16-PSK como combinación lineal de señales en cuadratura.....	206
4.3.2	Demoduladores PSK.....	207
4.3.2.1	Simulación del demodulador 2-PSK.....	209



4.3.2.2 Simulación del demodulador QPSK.....	210
4.3.2.3 Simulación del demodulador 16-PSK.....	213
4.4 MODULACIÓN D-BPSK( <i>DIFERENTIAL PHASE SHIFT KEYING BINARY</i> ).....	214
4.4.1 Modulador B-DPSK.....	215
4.4.1.1 Simulación del Modulador B-DPSK.....	217
4.4.2 Demodulador B-DPSK.....	218
4.4.2.1 Simulación del Demodulador B-DPSK.....	219
4.5 MODULACIÓN QAM ( <i>QUADRATURE AMPLITUDE MODULATION</i> ).....	221
4.5.1 Modulador QAM.....	223
4.5.1.1 Simulación de Modulación 16-QAM.....	224
4.5.2 Demodulación QAM.....	225
4.5.2.1 Simulación de Demodulación 16-QAM.....	226
4.6 DENSIDAD ESPECTRAL DE POTENCIA DE LA MODULACIÓN PSK Y QAM.....	228
4.7 EJEMPLOS DE SIMULACIÓN.....	229
4.7.1 Ejemplo de Simulación de Modulación Digital.....	231
4.7.2 Ejemplo de la obtención de la Densidad. Espectral de Potencia de una señal Modulada.....	235
4.8 REFERENCIAS.....	238
CAPÍTULO 5. CONCLUSIONES.....	239
BIBLIOGRAFÍA.....	243
ANEXOS:	
ANEXO A: BLOQUE S-FUNCTION	
ANEXO B: OSCILADOR CONTROLADO POR VOLTAJE	
ANEXO C: PHASE LOCKED LOOP	
ANEXO D: INSTALACION DEL SISTEMA DE SIMULACIÓN	

## INTRODUCCIÓN

En el diseño de sistemas de comunicación la ayuda del computador se hace cada vez más útil, así se pueden mediante procesos matemáticos variar parámetros de datos y analizar resultados. Actualmente el campo de la simulación para el diseño de sistemas de comunicación es una ayuda muy versátil al mismo tiempo que poco costosa ya que no requiere de una inversión considerable con respecto al precio que tendría el construir un laboratorio real de pruebas.

Los programas *MATLAB* y *SIMULINK* de *Math Works INC.* presentan un lenguaje de programación de fácil manipulación para desarrollar modelos tanto de simulación como de análisis de datos. Es por este motivo que se ha considerado esta herramienta como una ayuda para el estudio de sistemas de comunicaciones.

En este trabajo de Tesis se utilizará principalmente el *SIMULINK 2.0*, esta herramienta es un accesorio del *MATLAB 5.0* que permite construir a partir de un ambiente gráfico un diagrama de bloques de un sistema y observar su comportamiento en el tiempo e incluso en el dominio de la frecuencia. La interfaz gráfica del *SIMULINK* es muy simple, de fácil manejo, con semejanza a un laboratorio de electrónica donde existen generadores de señal, conexiones, osciloscopios, analizadores de espectros de frecuencia, etc.

El objetivo de esta tesis es desarrollar un conjunto de librerías que contengan bloques de simulación para el estudio de códigos de línea, modulación digital, efectos de ruido blanco en una transmisión digital y análisis de la densidad espectral de potencia de señales resultantes en los estudios de codificación y modulación. Todo esto con el propósito de

teñer una herramienta didáctica y útil para el laboratorio de Comunicación Digital.

En el primer capítulo se ilustrará con ejemplos simples, como utilizar en general tanto el *MATLAB* como el *SIMULINK*, que son programas que permiten seguir desarrollando las herramientas que ayudarán a simular tanto los códigos de línea como la modulación digital. En el segundo capítulo se expone de manera general el sistema de simulación, es decir las librerías desarrolladas y su contenido.

El capítulo 3 abarca el diseño de los bloques de simulación para los codificadores y decodificadores de línea, además el usuario dispone de un ejemplo de utilización sobre los códigos de línea, el uso del bloque para el diagrama del ojo y del bloque para la densidad espectral de potencia.

Con un tratamiento breve sobre los esquemas de modulación digital, el capítulo 4 describe el diseño de los bloques de simulación que permitirán la modulación y demodulación digital de señales binarias.

Con el estudio y la herramienta desarrollados, a criterio del autor, se ha cumplido con el objetivo de diseñar bloques de simulación que permitan un estudio didáctico de Comunicación Digital con la ayuda del *MATLAB* con *SIMULINK*; y, además dejar desarrollado las bases necesarias para que dichos bloques se utilicen para sistemas más complejos de transmisión digital.

# **CAPITULO 1. INTRODUCCIÓN AL MATLAB Y SIMULINK**

## **1.1 INTRODUCCIÓN.**

Actualmente el campo de la simulación por computadora ha tenido un profundo desarrollo, tanto para hacer cálculos de alta complejidad como para hacer gráficos que representen datos tomados experimentalmente, todo esto constituye una ayuda muy importante para el análisis de sistemas de comunicaciones.

En este capítulo se presentarán algunas características de los dos programas que se utilizarán en esta tesis *MATLAB 5.0* y *SIMULINK 2.0* de *Math Works Inc.* Adicionalmente se cumplirá con el objetivo de generar un modelo de simulación simple, para que el usuario tenga una familiaridad con lo que se desarrollará en los siguientes capítulos.

## **1.2 CARACTERÍSTICAS DEL MATLAB® (MATRIX LABORATORY).**

A continuación se presenta la plataforma bajo la cual el programa *SIMULINK®* opera, la descripción de cada una de las características son tomadas de varios manuales y publicaciones de *Math Works INC.*, si se requiere ampliar la información se recomienda ver las referencias al final del capítulo o por *internet* revisar la pagina *Web* de *Math Works INC.* en la dirección electrónica [www.mathworks.com](http://www.mathworks.com).

### 1.2.1 El lenguaje MATLAB®.

*MATLAB* integra cálculos, visualización y programación en una herramienta flexible, con arquitectura abierta, diseñada para el manejo de proyectos a gran escala en investigación y la industria. *MATLAB* proporciona tanto a ingenieros como científicos un lenguaje intuitivo para presentar problemas y sus soluciones matemática y gráficamente; como ejemplo de ello se tiene el gráfico de la figura 1.1 donde se representa el gráfico de una función de dos variables en tres dimensiones, de la misma manera que se pueden representar funciones se pueden representar datos y realizar mediciones de magnitudes físicas.

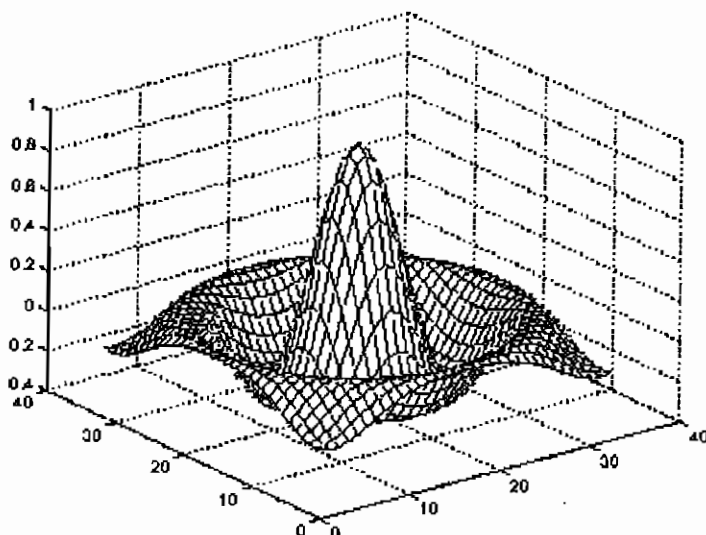


Figura 1.1 Representación en tres dimensiones de una función de dos variables.  $Z = \sin(\sqrt{x^2 + y^2}) / \sqrt{x^2 + y^2}$ .

El programa presenta ayuda en muchos campos<sup>1</sup>, entre ellos:

Cálculos matemáticos y programación.

Desarrollo de Algoritmos.

Modelación, simulación y planificación de prototipos.

Análisis de datos, exploración (análisis gráfico de una señal) y visualización.

Gráficos científicos y de ingeniería.

---

<sup>1</sup> Tomado de la referencia [2]

En cada uno de estos campos, la manera en que se presentan estas soluciones es de gran importancia para el usuario; es decir que la aplicación diseñada permita a quién la utiliza ingresar datos y obtener resultados de manera gráfica y sin confusiones. Con este fin *MATLAB* permite el desarrollo de aplicaciones, con la ayuda de una interfaz de construcción gráfica (GUI: *Graphical User Interface*: interfaz gráfica de usuario).

Con más de 500 funciones matemáticas, estadísticas y de ingeniería, el programa permite el acceso a resultados de alta calidad, las rutinas numéricas son rápidas seguras y confiables. Estas funciones son desarrolladas por expertos en matemáticas, constituyendo los cimientos del lenguaje *MATLAB*. Las funciones matemáticas tienen un desarrollo especial para operaciones matriciales, lo que permite tener un desempeño muy cercano a lenguajes de bajo nivel como el lenguaje C pero con una sintaxis menos complicada. Por ejemplo *MATLAB* contiene las siguiente operaciones matemáticas:

- Álgebra lineal y cálculos matriciales.
- Análisis de funciones estadísticas y con el método de Fourier.
- Resolución de ecuaciones diferenciales.
- Interpretación de matrices dispersas (matrices con pocos elementos diferentes de cero)<sup>1</sup>.
- Trigonometría y otras operaciones fundamentales.

El lenguaje *MATLAB* está diseñado para hacer análisis interactivo al mismo tiempo que se tiene las cualidades de una programación estructurada que facilita la construcción de algoritmos propios para cada solución deseada, es decir que el lenguaje tiene control de flujo,

---

<sup>1</sup>Más información de las matrices Dsipersas o *Sparse* en [1].

estructura de datos de programación orientada a objetos, además de herramientas gráficas ya desarrolladas y la facilidad de reconocer rutinas hechas en lenguaje C, C++ y *Fortran*.

Esta herramienta posee una variedad de soluciones para aplicaciones específicas, denominadas *toolboxes*, que son una colección de archivos escritos en lenguaje de *MATLAB* (archivos.m)<sup>1</sup> los cuales realizan operaciones orientadas a solucionar problemas específicos. Las áreas que cubren estos archivos.m son por ejemplo: procesamiento digital de señales, sistemas de control, redes neuronales, simulación y muchos otros.

Entre los *toolboxes* más conocidos están:

*Chemometrics* (Estequiometría).

Medidas de sustancias químicas, más de 40 comandos para análisis cualitativo y cuantitativo, usando métodos de diseño, incluyendo técnicas de regresión lineal, etc.

*Communication* (Comunicaciones).

Comprende un conjunto de herramientas para el diseño, análisis y simulación de sistemas modernos de comunicaciones; combina el lenguaje *MATLAB* con la facilidad de uso de los bloques del *SIMULINK*, presentando aplicaciones de diseño muy importantes y versátiles.

*Control Systems* (Sistemas de Control).

Es fundamental para el diseño de sistemas de control, tiene funciones para crear modelos de análisis y diseño de sistemas de control

---

<sup>1</sup> Estos archivos también suelen denominarse *m-files*, se adoptará en esta tesis la denominación archivos.m.

automático. La aplicación del control automático crece cada día más con el desarrollo de la tecnología de las computadoras y su costo cada vez más bajo.

#### *Financial Computation* (Finanzas).

Basándose en *MATLAB* se han creado todo tipo de funciones de análisis financiero, esta aplicación incluye cálculos de interés, costos, análisis de cuentas y proyecciones óptimas. Es necesario disponer de los *toolboxes* de *Statistics & Optimization* (Estadística y Optimización).

#### *Frequency Domain Systems Identification* (Identificación de Sistemas en el Dominio de la Frecuencia).

Desarrollado en la Universidad de Bruselas, es una serie de archivos.m para encontrar modelos de estudio para sistemas lineales basándose en mediciones de sistemas de respuesta de frecuencia.

#### *Image Processing* (Procesamiento Digital de Imágenes).

Contiene algoritmos desarrollados para el procesamiento digital de imágenes, comprende diseño de filtros y restauración de imágenes, ampliación de imágenes, análisis y estadística, color, geometría y operaciones morfológicas además de transformaciones en dos dimensiones (2-D).

#### *Linear Matrix Inequalities*. (Sistemas de Inecuaciones de Matrices Lineales).

Este *toolbox*, creado por los investigadores Pascal Gahiner, Arkadi Nemivoriski y Alan Laub, permite resolver desigualdades de Matrices lineales, las cuales son problemas de optimización con convección especial.



### *Neural Networks* (Redes Neuronales).

Es una colección de funciones *MATLAB*, para el diseño y simulación de redes neuronales.

### *Optimization* (Optimización).

Contiene comandos para la optimización en general de funciones lineales y no lineales. Un problema de optimización puede ser la búsqueda de un punto máximo o mínimo en una función compleja, límites de altura y profundidad en un plano topográfico, etc.

### *Partial Differential Equations* (Ecuaciones en Derivadas parciales).

Se utiliza para el estudio de soluciones de ecuaciones en derivadas parciales en dos dimensiones.

### *Robust Control* (Control Robusto).

Aquí se presentan elementos para el estudio de control optimizado.

### *Signal Processing* (Procesamiento de Señales).

Es una serie de archivos.m que permiten el análisis de procesamiento de señales, incluye aplicaciones de audio y vídeo digital, por ejemplo discos compactos, HDTV (*High Definition Television*, televisión de alta definición) o en el área de medicina, lo relacionado con resonancia magnética.

### *Statistics* (Estadística).

Se presenta una serie de funciones estadísticas y de probabilidad para el análisis de datos y modelación con el método de Monte Carlo.

### *Symbolic Mathematics* (Matemáticas Simbólicas).

Esta herramienta permite el cálculo con variables literales, basado en el programa *Maple V*<sup>1</sup>.

### *System Identification* (Identificación de Sistemas).

Permite la representación de modelos matemáticos para un sistema dado, conocidas las variables de entrada y de salida, se utiliza en estudios físicos y de ingeniería.

## 1.2.2 Análisis de Datos, Rastreo, Gráficos y Visualización.

*MATLAB* está diseñado para un completo análisis de datos de un proceso, con una buena presentación de resultados. El programa puede armonizar cálculos complejos con varias herramientas de manipulación de datos, funciones y gráficos de estudio.

Los datos a analizar pueden ser el resultado de funciones matemáticas o de trabajos experimentales, estos resultados pueden ser luego sometidos a distintos tipos de gráficos con propósito de ser estudiados. Los gráficos pueden ser en dos o tres dimensiones dependiendo de las necesidades y también de las herramientas (*toolboxes*) de las que se disponga.

Con la interfaz gráfica se han desarrollado algunas herramientas que pueden indicar el alcance que tiene el programa. A continuación se presenta en la figura 1.2 un ejemplo basado en el *Signal Processing Toolbox*, esta aplicación es útil para el análisis de funciones. Este ejemplo es una aplicación gráfica interactiva, en la cual una vez que se tiene

---

<sup>1</sup> Es un programa que realiza algoritmos para cálculos algebraicos literales.

almacenados en memoria los datos de la variable independiente  $x$  y los de la variable dependiente  $y$ , en un archivo `*.mat`, permite el análisis de los datos de la función mediante el gráfico, como por ejemplo encontrar máximos, cruces de la función por cero, rastreo de puntos requeridos, etc.; solo se necesita especificar al inicio el archivo que contiene los datos de la función<sup>1</sup> y manipular los botones de control de acuerdo a su función asignada.

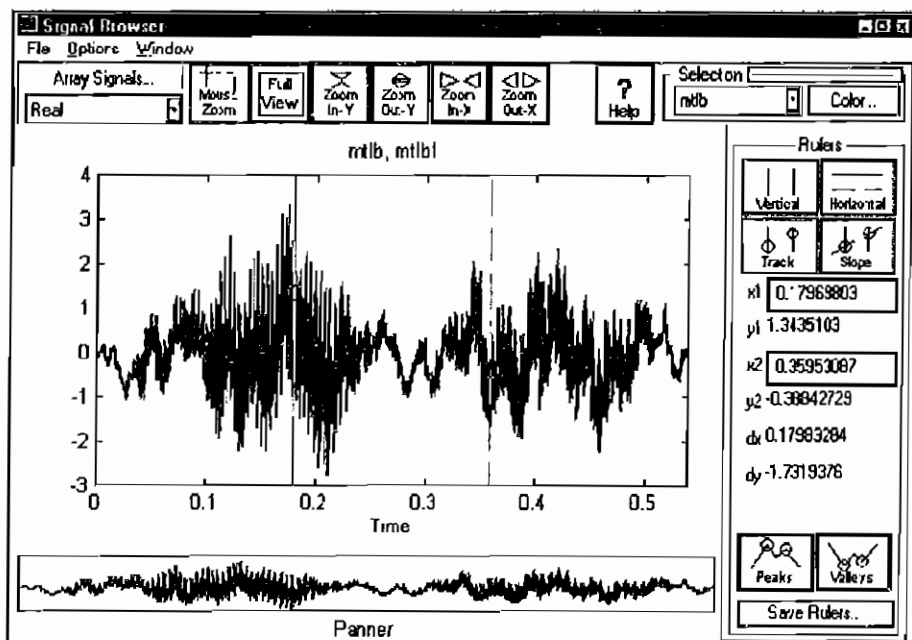


Figura 1.2 Análisis de una señal a través de una aplicación desarrollada en el *toolbox* de procesamiento de señales.

### 1.2.3 Introducción al uso del MATLAB.

Una vez instalado<sup>2</sup> apropiadamente el programa se puede observar en la figura 1.3 la ventana de trabajo del *MATLAB*. El programa consiste básicamente, de una área de memoria de datos y una línea de comandos.

<sup>1</sup> Se pueden generar aplicaciones gráficas interactivas, sin embargo se requiere de más conocimientos para indicar todo el proceso de utilización y creación de esta facilidad que se activa al escribir en la línea de comandos la palabra *sptool*.

<sup>2</sup> Sobre el proceso de instalación ver referencia [6].

Se almacena en la memoria con variables literales cualquier valor<sup>1</sup> en forma de matriz, para luego ser utilizado en funciones que pueden ser matemáticas o de un fin específico, como por ejemplo el comando *plot*, el cual presenta un gráfico de los datos ingresados.

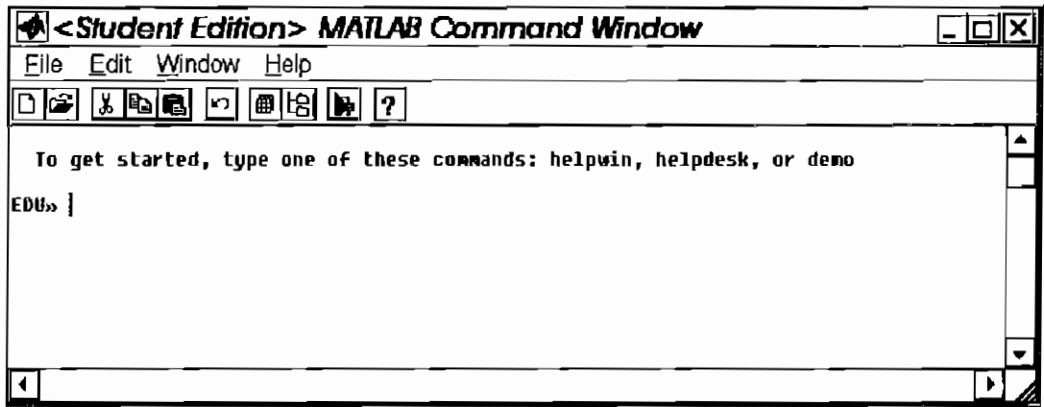


Figura 1.3 *MATLAB* y su línea de comandos cuyo inicio es `EDU>>`

A continuación se presenta una breve introducción de la manera como se utiliza la línea de comandos. Este ejemplo obtendrá un gráfico de la función seno a partir de un listado de valores de cero a  $2\pi$ . El primer paso es crear una variable que se llamará *X* con valores de cero a  $2\pi$  en pasos de 0.1 con los siguientes comandos.

```
EDU >> X = 0: 0.1: 2* pi; % ingreso variable independiente X1
```

una vez que se define la variable *X* se calcula el valor correspondiente al seno de cada elemento y se almacena en la variable *Y* de la siguiente manera:

```
EDU >> Y = sin (X); % cálculo de variable dependiente Y
```

luego de haber definido las variables, se procede a presentar un gráfico de *Y* en función de *X* con la siguiente expresión.

---

<sup>1</sup> Valor numérico o también literal

```
EDU >> plot(X,Y);      % comando para graficar Y vs X
```

por último con el comando *grid on* se hacen visibles las líneas cuadriculares que se pueden observar junto con el gráfico resultado de la función en la figura 1.4.

```
EDU >> grid on      % comando que activa las líneas de puntos
```

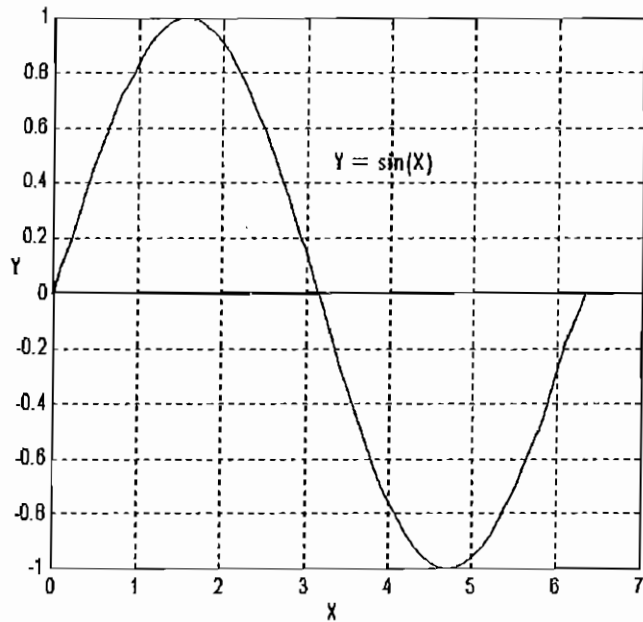


Figura 1.4 Resultado gráfico  $Y = \sin(X)$

Es necesario indicar que la mayor parte de procesos del *MATLAB* se hace en base a matrices, así las variables  $X$  y  $Y$  son matrices de una sola fila o columna. El comando *plot(X,Y)* genera una matriz que combina en orden los valores de  $X$  y  $Y$  para definir los puntos de unión del gráfico que por defecto se unen linealmente. Debido a este manejo matricial de datos se debe tener cuidado con el manejo de las dimensiones de las variables,

---

<sup>1</sup> El símbolo % en el lenguaje de programación solo se emplea para hacer un comentario, no tiene ninguna implicación.

por ejemplo al no tener las mismas dimensiones X y Y se producirá un mensaje de error.

Para no repetir un grupo de instrucciones, o con el fin de hacer funciones de usuario, se agrupan una serie de comandos en un archivo que se guarda con la extensión ".m". Los archivos.m son aquellos que representan un listado de comandos o funciones, son hechos en lenguaje de texto común, escritos en cualquier editor de texto simple como puede ser el editor "notepad.exe" propio de *Microsoft Windows* o cualquier otro. En la versión 5.0 del *MATLAB* se tiene un editor incorporado, el cual tiene una visualización de sintaxis con colores y además proporciona la facilidad directa de realizar depuraciones controladas del flujo de programa que representa el archivo. En la figura 1.5 se tiene la representación de la opción de depuración con un archivo del tipo ".m".

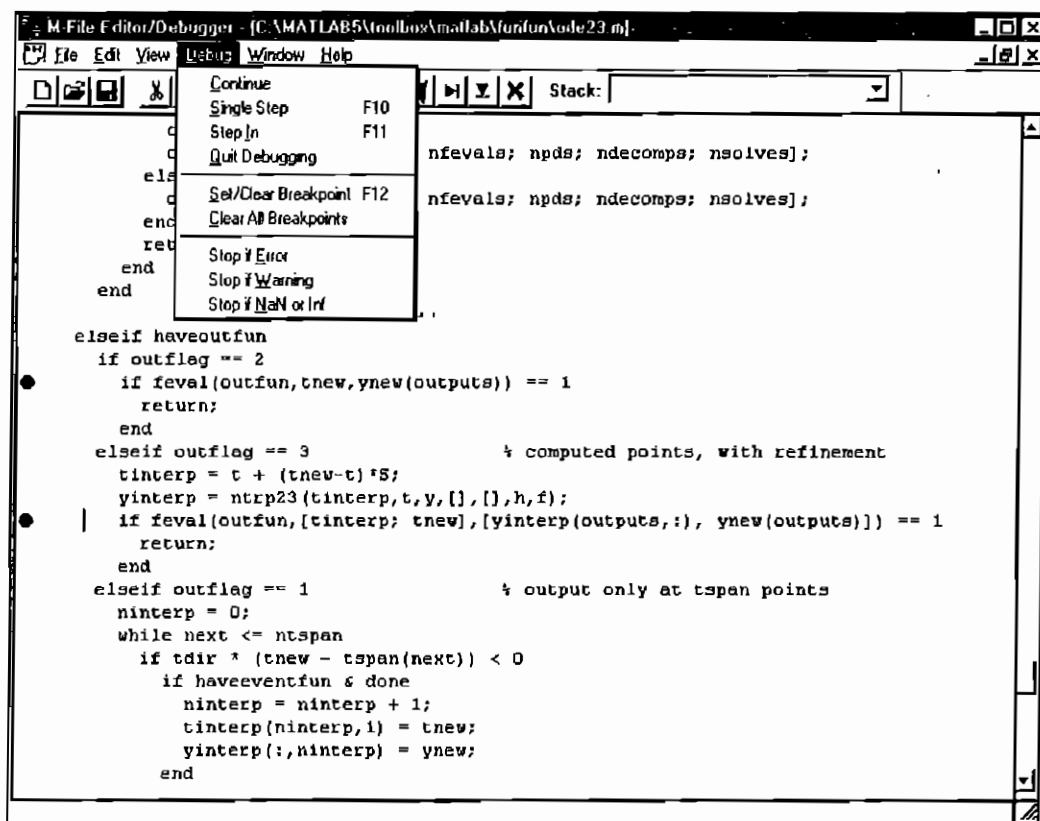


Figura 1.5 El editor de archivos ".m" indicando la opción de depurar, aquí se señala donde debe parar el programa para hacerse una evaluación.

Esta ha sido una breve introducción al ambiente de trabajo del *MATLAB*, las librerías y ejemplos de modelación desarrollados en esta tesis son también instrucciones para ser escritas en la línea de comandos, mostrándose la librería o ejemplo respectivamente, esto se explicará oportunamente en los capítulos correspondientes.

#### 1.2.4 Características del *SIMULINK*<sup>®</sup>.

En esta sección se indicarán las bases para la utilización del *SIMULINK* con el fin de familiarizar al usuario con el manejo de librerías y generación de modelos simples.

*SIMULINK* es una herramienta para modelar y simular una gran variedad de sistemas dinámicos, los cuales pueden ser lineales, no lineales, en tiempo discreto, en tiempo continuo y sistemas que son una combinación de estos<sup>1</sup>. El programa provee la facilidad de crear modelos a simular, como si se tratara de animar un diagrama de bloques de un texto, los elementos de estos diagramas de bloques se obtienen de librerías copiándolos gráficamente y modificando sus variables con ayuda del ratón, una vez generado el modelo se puede visualizar la simulación. A continuación se indican las características de los modelos a simularse según la referencia [3].

---

<sup>1</sup> La modelación será siempre digital, más se consideran sistemas continuos aquellos que realizan operaciones en cada instante del tiempo de simulación y representados por la transformada de Laplace, no así los discretos que tiene un tiempo de muestreo de la señal de entrada y son representados por la transformada Z.

### 1.2.5 Sistemas de Modelación a través de diagramas de bloques.

Para representar sistemas que se desean analizar se ha desarrollado una técnica ilustrativa en los textos técnicos, los denominados diagramas de bloques, en *SIMULINK* para la simulación de sistemas se parte de esta idea intuitiva de los diagramas de bloques para generar el modelo a simular.

Las características de la generación de Modelos para la simulación son las siguientes:

Modelos concatenados, es decir que se crean sistemas complejos a partir de sistemas simples, sin límites de bloques o conexiones, esto se realiza agrupando bloques para que trabajen como subsistemas (un sistema dentro otro) de esta manera se puede generar un proyecto sin perder la visión de su aplicación.

Librería de bloques, la cual contiene los elementos más usados para la simulación como *sources* (generadores de funciones), *sinks* (equipos de visualización de datos), *Discrete* (para sistemas discretos), *linear* (sistemas lineales), *no linear* (sistemas no lineales) y *connections* (conexiones, como entradas en un subsistema o interconexiones con subsistemas, etc.).

Conexiones escalares y vectoriales, los modelos pueden tener bloques de entradas y salidas de una sola señal o múltiples señales dependiendo de la definición de cada bloque.



Asignación de nombres a señales y puertos, es muy útil rotular las líneas de conexión para constancia y estudio de la misma, así como también poner nombre a los puertos de entrada y salida de un bloque o subsistema creado.

## Resumen de las librerías del *SIMULINK*:

### *Source* (fuentes)

- Generación de señales senoidales, rampas y ondas cuadradas.
- Generador de ruido.

### *Sinks* (sumideros)

- Presentación de datos (*Scope*, campo), visión numérica y bloques de gráficos.
- Transferencia de datos a archivos.
- Salidas de bloques al campo de trabajo *MATLAB*<sup>1</sup>.

### *Discrete* (Sistemas discretos)

- Funciones de Transferencia (Transformada Z), filtros, retardos y bloques descritos por ecuaciones en diferencias.

### *Linear* (Sistemas lineales)

- Funciones de transferencia (Transformada de Laplace), bloques de ganancia y sistemas descritos por ecuaciones diferenciales.
- Integradores.
- Diferenciadores.

### *Nolinear* (Sistemas no lineales)

- Limitadores, bloques con histéresis y muestreadores de señal.
- Operadores lógicos y relacionadores de señal.

### *Connections* (Conexiones)

- Bloques multiplexores y demultiplexores analógicos de señal.
- Puertos de entrada y salida de subsistemas.
- Habilitación de un subsistema por estado (*Enable*) o por cambio de estado (*trigger*).

---

<sup>1</sup> Es decir que los datos quedan activos en memoria listos para ser usados en el MATLAB, ya que el SIMULINK trabaja con variables locales es decir que no se presentan en el MATLAB.

Una información detallada sobre cada bloque de esta librería se encuentra en [4].

#### 1.2.6 Simulación y análisis.

*SIMULINK* y *MATLAB* permiten al usuario pasar sin problemas de un estado de análisis a diseño y simulación; en la simulación pueden variarse los parámetros de los bloques al mismo tiempo que se está efectuando la simulación y observar los resultados casi instantáneamente. Para la simulación se tiene una serie de algoritmos de integración cuya descripción se encuentra en [5].

Con los resultados de la simulación se pueden extraer modelos lineales, optimización de parámetros, análisis paramétrico (variación de varios parámetros al mismo tiempo) e incluso animación, ya que mediante un bloque programable denominado *S-function*<sup>1</sup> permite asociar los datos de entrada con comandos del *MATLAB* y conseguir la animación.

A continuación se indica en la figura 1.6 un ejemplo de un diagrama de bloques que utiliza resultados para realizar la simulación de un péndulo invertido. La aplicación es interactiva y permite con el cursor triangular modificar, mientras transcurre el tiempo, la posición de inclinación del péndulo. Cuando se termina la simulación la opción de *playback* de la figura 1.6 presenta una secuencia de imágenes de acuerdo a la simulación.

---

<sup>1</sup> Las *S-function* son útiles para generar nuevos bloques, el detalle de este bloque se lo describe en el Anexo A.

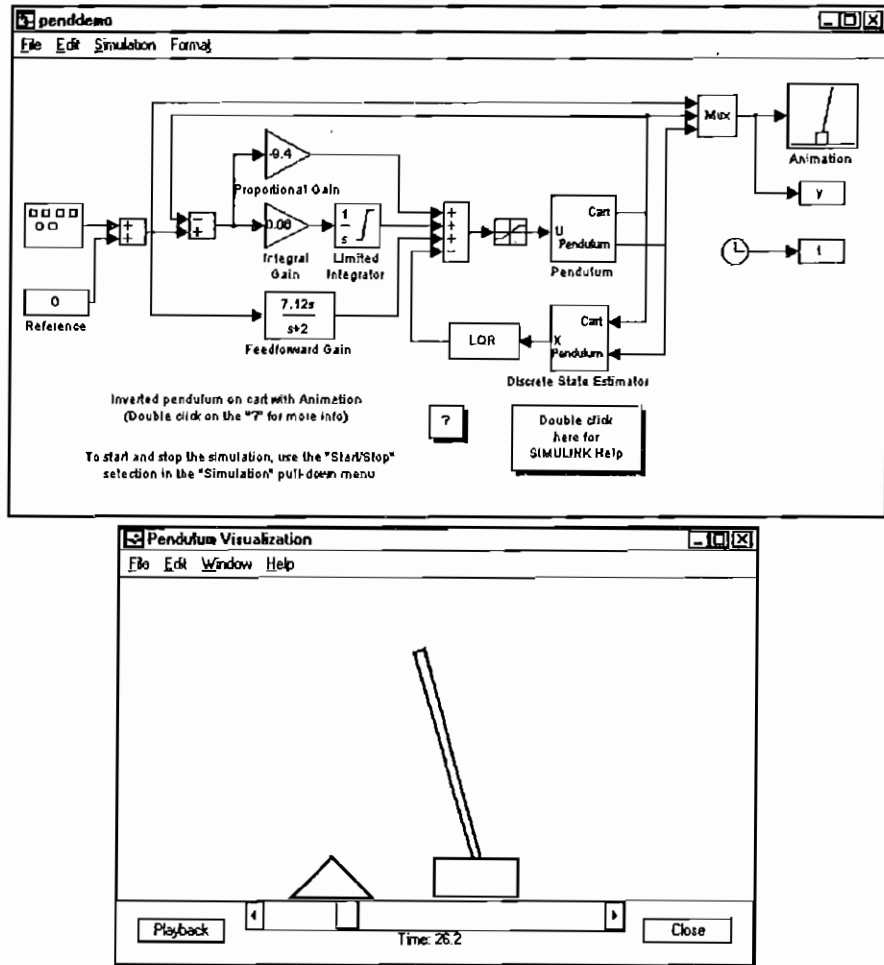


Figura 1.6 Ejemplo de Simulación 'penddemo', se trata de la simulación de un péndulo

### 1.2.7 Arquitectura abierta.

*SIMULINK* facilita ampliar los modelos de simulación, personalizar los bloques ya existentes y crear librerías. Esto último es muy importante ya que gracias a esta facilidad se desarrolló las librerías de Comunicación Digital necesarias en esta tesis.

Existen cuatro maneras de añadir bloques especializados a la librería de bloques:

- Agrupando bloques en un subsistema.
- Con la ayuda de la interfaz de personalizar un bloque<sup>1</sup>.
- Mediante algoritmos en C y *Fortran*.
- Empleando los archivos.m de *MATLAB*.

*SIMULINK* permite personalizar un bloque mediante un interfaz denominado *MASK*, el cual crea una ventana de dialogo con la facilidad de poner nombres a las variables, iconos y comentarios de ayudas, además de realizar ciertos comandos previos en lenguaje *MATLAB*, necesarios para dicho bloque.

### 1.2.8 SIMULINK® y librerías adicionales.

Los *toolboxes* presentados anteriormente, también presentan funciones y bloques para ser utilizados en *SIMULINK*, existe un *toolbox* adicional "*Real time Workshop*", el cual genera código C directamente de los diagramas de *SIMULINK* para conseguir una simulación en tiempo real tanto de sistemas lineales, discretos o híbridos. Esta herramienta es compatible con tarjetas *DSP* (*Digital Signal Processing* procesamiento digital de señales), con controladores y con una variedad de hardware comercial.

### 1.2.9 Construcción de un modelo simple.

Por razones didácticas se presenta un ejemplo de la creación de un modelo básico para simulación, más adelante se tendrán aclaraciones específicas de las herramientas empleadas según su necesidad, cualquier

---

<sup>1</sup> Esto se refiere a los bloques tipo *MASK* descritos en [5] completamente.

otra ampliación de esta información se encuentra en las referencias [4] y [5] de la bibliografía.

El modelo a construir consta de un generador de señal senoidal y del cálculo de su integral, el diagrama a crearse se verá así:

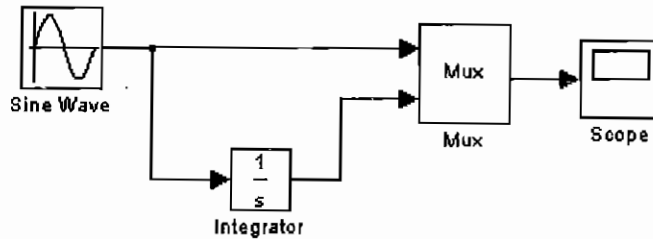


Figura 1.7 Diagrama de bloques para simular una señal senoidal y su integral.

En la ventana de comandos del *MATLAB* al escribir la palabra *SIMULINK*, se puede tener acceso a la librería de bloques que corresponde a la figura 1.8:

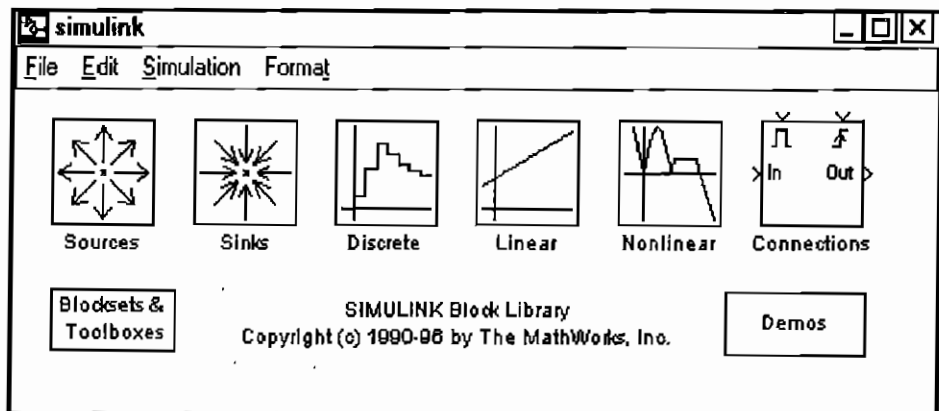


Figura 1.8 Librerías del *SIMULINK*

Del menú *File* de esta ventana se selecciona *new*, para crear un nuevo modelo. Los bloques a utilizarse se obtienen de esta ventana de la siguiente forma:

El bloque Sine Wave block de la librería *Sources*.

El bloque Scope de la librería *Sinks*.

El bloque Integrator de la librería *Linear*.

El bloque Mux de la librería *Connections*.

Al abrir la librería *Sources* se tiene acceso al bloque seno, para esto se debe ejecutar un doble *click* (el presionar el botón activo del ratón, se suele denominar *click*) en el ícono (o gráfico del bloque) respectivo. Todos los bloques en esta librería son generadores de señal, es así como se ve en la figura 1.9:

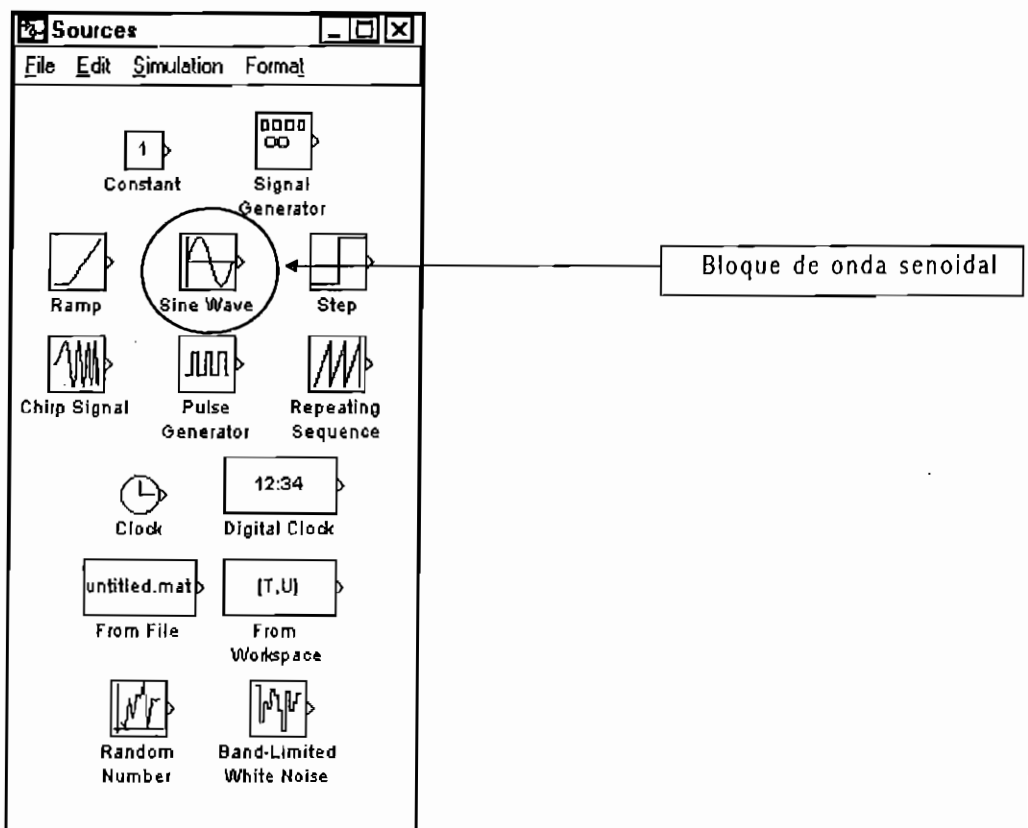


Figura 1.9 Librería *Sources* o fuentes del *SUMLINK*

Se añaden bloques al modelo nuevo copiando el bloque requerido e incorporándolo de esa forma al modelo, para este ejemplo se copia el bloque del generador de onda senoidal, llevándolo directamente con el ratón, manteniendo presionado el botón derecho, de esta manera se tiene:

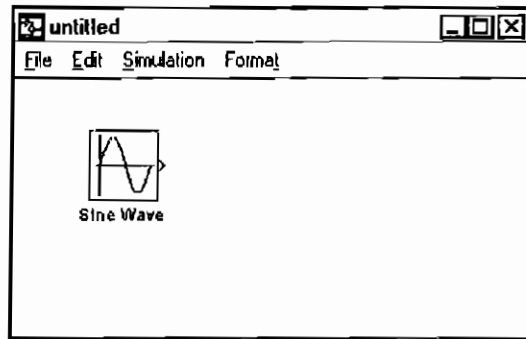


Figura 1.10 Modelo a simular denominado por defecto en un inicio *untitled* (sin título).

Una vez copiado el bloque, se puede volverlo a copiar en la misma ventana, además es posible definir el tamaño del bloque y cambiar su nombre, de esta manera se copian todos los bloques requeridos para el ejemplo, como se puede apreciar en la figura 1.11:

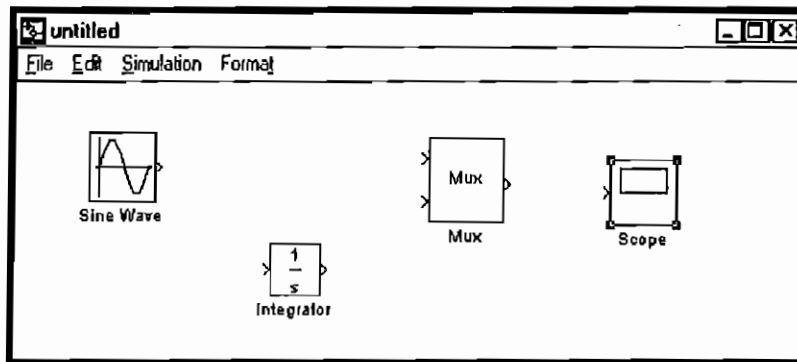


Figura 1.11 Modelo a simular con los bloques copiados y sin conexiones.

En cada bloque se tiene un tipo de señalización, cuando se tiene el símbolo ">" y a la izquierda, el bloque entonces es un puerto de entrada

de una señal; de lo contrario es una salida del bloque. Entonces se tienen puertos de entrada y salida como se observan en la figura 1.12:

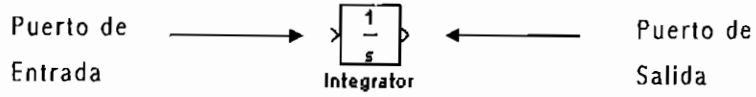


Figura 1.12 Puertos de entrada y salida de un bloque

Como se puede observar en la figura 1.11 el bloque *mux* debe tener 2 entradas, con dos "clicks"<sup>1</sup> de ratón en el bloque se muestra un recuadro de dialogo (Figura 1.13), entonces se coloca el valor de 2 en el número de puertos de entrada.

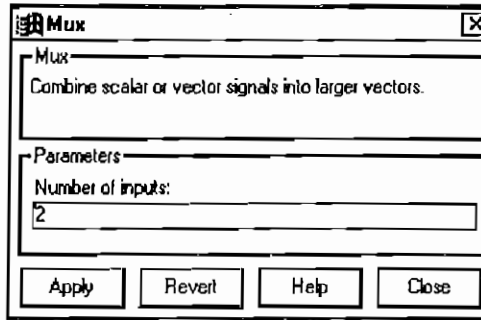


Figura 1.13 Recuadro de dialogo del bloque *Mux* con sus propiedades

Manteniendo presionado el botón derecho del ratón se puede conectar puntos desde un puerto de salida a un puerto de entrada; para realizar conexiones desde una línea, se presiona adicionalmente la tecla *control*. Siguiendo este procedimiento se realizan las conexiones del modelo propuesto como se muestra en la figura 1.7.

Antes de la simulación, abriendo el bloque *Scope*, se debe inicializar los parámetros seleccionando del menú *Simulation* la opción

---

<sup>1</sup> Suele denominarse "click" a la acción de presionar un botón del ratón.



*Parameters* (figura 1.14), aquí se indican los valores de tiempo de simulación, tipo de simulación y características, si se desea que los datos se envíen al campo de datos del *MATLAB* y otras opciones más. Posteriormente a esto se inicia la simulación seleccionando del menú la palabra *Start*(figura 1.15).

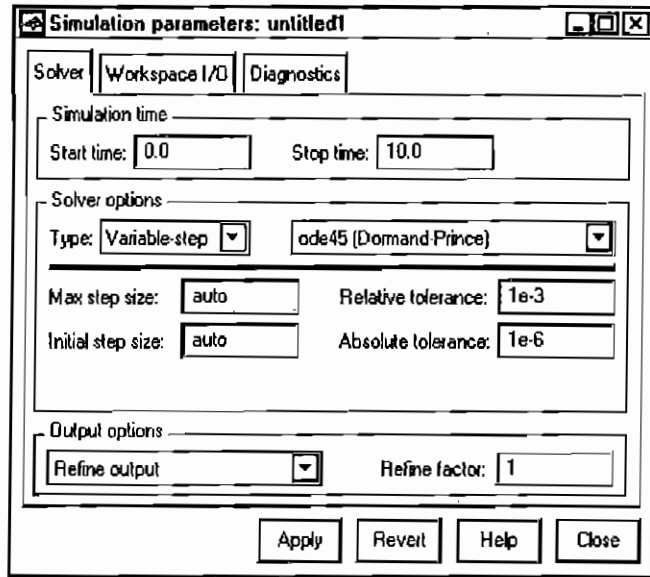


Figura 1.14 Menú de los Parámetros de la Simulación.

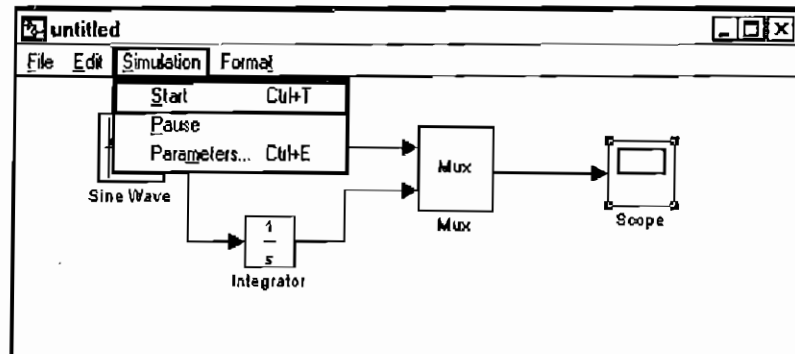


Figura 1.15 Menú de selección para iniciar la simulación, detener temporalmente (*pause*) o definir parámetros.

Los resultados se presentan en la figura 1.16 después de hacer dos "*clicks*" en el bloque *Scope*:

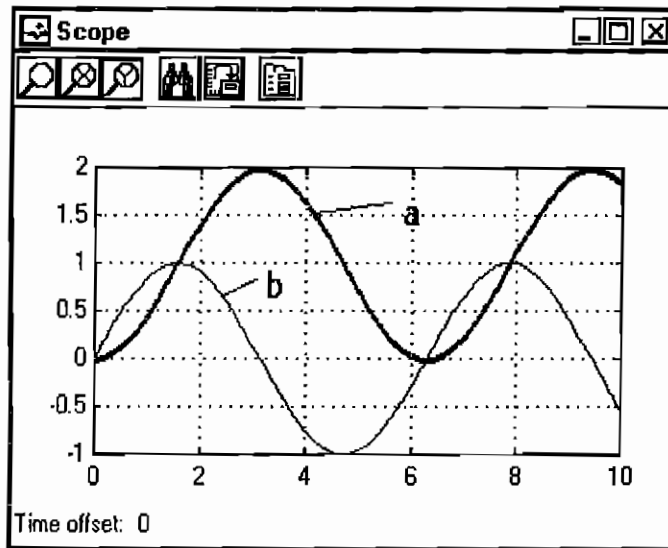





Figura 1.16 Resultado gráfico de la simulación del ejemplo, a es la integral de la señal senoidal y b es la señal senoidal


De esta manera se ha indicado brevemente como generar un modelo sencillo. Es necesario explicar las funciones de esta pantalla o bloque *Scope*, ya que sirve para analizar los resultados, lo cual es muy importante para la medición de resultados.

 Esta opción selecciona una región del gráfico, la cual será ampliada hasta el tamaño de la ventana *Scope*.

 Esta ayuda realiza una ampliación del gráfico, pero solo en el eje horizontal.

 Este botón realiza una ampliación del gráfico en el eje vertical.

 Define los límites de los ejes con el valor máximo y mínimo de las señales a representarse gráficamente.

 Una vez visualizada la imagen requerida se define los valores de escalas actuales por defecto para las siguientes simulaciones.



Permite ajustar los máximos y mínimos de las escalas de acuerdo a valores escogidos por el usuario de manera manual. Permite al usuario definir la cantidad de puntos a presentarse en el bloque de simulación y si el bloque *scope* puede funcionar como bloque flotante. Cuando se toma esta opción se tiene la plantilla de datos representada en la figura 1.17.

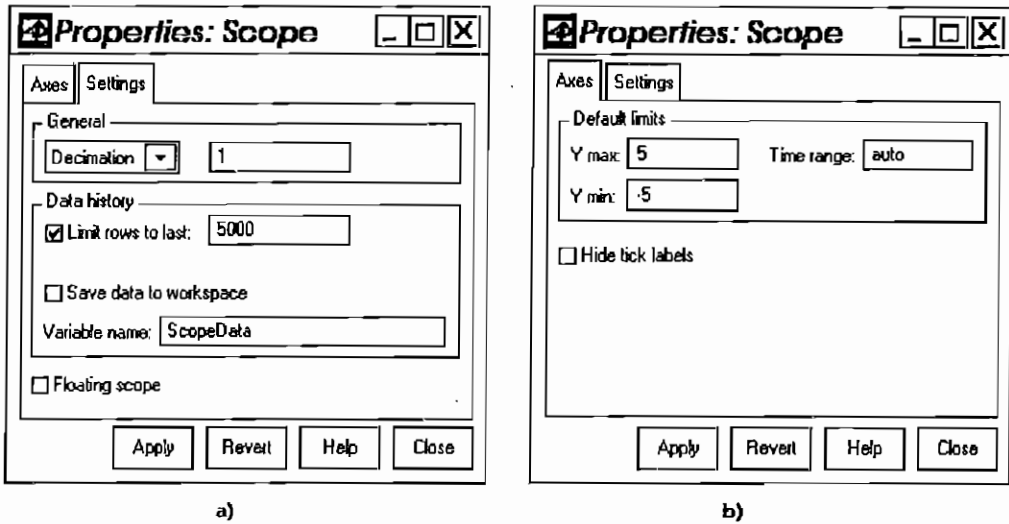


Figura 1.17 Plantilla de datos del bloque *Scope*: a) Opción *Setting* b) Opción *Axes*

En la figura 1.17b se presenta la opción *Axes*, en la cual se tiene campos para ingresar el valor máximo (*Y max:*) y el mínimo (*Y min:*) para la variable dependiente. El valor ingresado en *Time range* puede ser un número o dejar la opción *auto*; cuando se tiene esta opción el tiempo es aquel ingresado en la parte *Simulation: Parameters*.

La página *Settings* (figura 1.17a) permite controlar el número de puntos mostrados y guardar los datos (no el gráfico) a la memoria general del *MATLAB* o denominada también *workspace*.

La sección *General* de la página *Settings* presenta dos campos. El uno es un menú desplegable con dos opciones y el otro es una ventana que permite recibir datos numéricos. Las dos opciones del menú son:

*Decimation* y *Sample time*. Si se elige la opción *Decimation* y se ingresa en la ventana de datos el valor de 1 (valor por defecto), cada punto que ingresa al bloque *Scope* es dibujado; si se ingresa el valor de 2, cada dos puntos son dibujados y así sucesivamente. Si se elige la opción *Sample time*, se debe ingresar en la ventana de datos el espacio absoluto entre puntos a dibujarse.

El bloque *Scope* almacena los datos que representan gráficamente un vector cuyo límite está dado por el valor ingresado en la sección *Data History*, con la opción *Limit row to last*. Estos datos almacenados son los que más tarde ayudan a la manipulación del gráfico, como son ampliaciones o búsquedas de coordenadas de un punto dibujado. Luego en esta misma sección se tiene la posibilidad de grabar en el *workspace* los datos de este bloque, para utilizarse dentro del *MATLAB*.

Finalmente en esta página *Settings* se tiene la opción de que el bloque *scope* funcione como un osciloscopio flotante o fijo en la elección *Floating scope*. Cuando se elige esta opción se debe indicar, la línea de interconexión entre los bloques del modelo que se desea ver la señal; y, en caso de no tomar esta opción se necesita conectar una línea desde la parte de interés a la entrada del bloque *scope*.

## **Subsistemas**

Los subsistemas son modelos sencillos agrupados con el propósito de simplificar la visión conceptual de un modelo a simularse, ésta es una manera de generar nuevos bloques de simulación a partir de otros ya existentes. La forma de agrupación de los bloques para formar un subsistema debe tener en cuenta una relación funcional entre los bloques seleccionados, es decir que en conjunto cumplen con una determinada

tarea. Por ejemplo en la figura 1.18 se indica como conformar un grupo de bloques que sirvan para evaluar la ecuación  $y = mx + b$ .

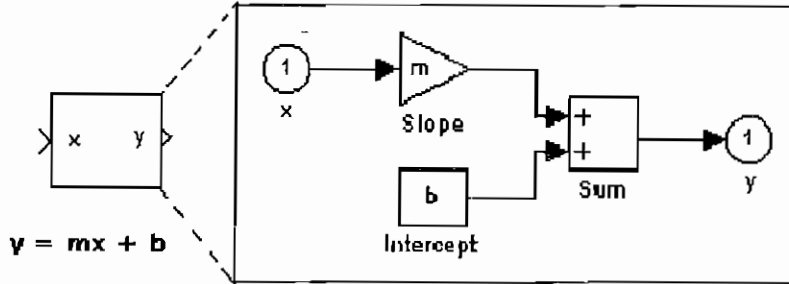


Figura 1.18 Ejemplo de Subsistema que calcula los resultados de la ecuación  $y=mx+b$

Después de desarrollar un modelo de simulación como se vió anteriormente (es decir buscando en las librerías respectivas los bloques y realizando las conexiones necesarias) en este modelo se selecciona con ayuda del ratón los bloques del recuadro (lado derecho figura 1.18) y se ha tomado la opción *Create Subsystem* del *menú Edit*, creándose el bloque que se observa a la izquierda de la figura 1.18. El subsistema tiene como entrada la señal  $x$  y como salida la señal  $y$  de la ecuación  $y=mx+b$ . Este bloque puede cambiar su presentación gráfica y se le puede añadir una plantilla de datos con la técnica denominada *Masking* cuyo proceso se encuentra detallado en la pág. 119 de la referencia [5].

Un modelo de simulación puede contener varios subsistemas, lo que permite al usuario simplificar el sistema que está simulando y concentrarse en cambiar los parámetros de interés para el estudio que esté realizando.

### **1.3 LIMITACIONES.**

Existen varias limitaciones, ya que lo que en verdad se está haciendo es la simulación de señales discretas y no realmente continuas, por lo cual se debe tener mucho cuidado con los tiempos de muestreo de las señales en cuanto a obtener sus gráficas.

Otra limitación la presenta la versión para estudiantes que se está empleando, la cual es para cálculos no superiores a una matriz de 16.300 datos y un almacenamiento en los bloques de simulación no mayor a 50, estos límites en la versión profesional no se encuentran.

#### **1.4 BIBLIOGRAFÍA.**

1. Math Works INC. "Using MATLAB", 1996.
2. Math Works INC. "Matlab accelerating inovation and development", 1997.
3. Math Works INC. "SIMULINK® Simulación de Sistemas Dinámicos", 1997.
4. Math Works INC. "Using\_SIMULINK®", 1996.
5. James B. Dabney y Thomas L. Harman. "The Student Edition of SIMULINK® User's Guide", Prentice Hall, 1997.
6. Math Works INC. "Instalation for PC and Macintosh 1996.

## **CAPÍTULO 2. INTRODUCCIÓN AL SISTEMA DE SIMULACIÓN.**

El principal objetivo de esta tesis es conseguir una visión didáctica de los códigos de línea y los moduladores de transmisión digital. Para conseguir este objetivo se necesita generar nuevos bloques de simulación, no solo que permitan la codificación y modulación, sino que ayuden a complementar la labor didáctica de los modelos desarrollados. Es por esta razón que se han generado librerías que permitirán obtener datos digitales, analizar la Densidad Espectral de Potencia, simular un canal con ruido blanco y representar el diagrama del ojo de una señal.

### **2.1 LIBRERÍAS DEL SISTEMA DE SIMULACIÓN.**

En el anexo D se encuentra la descripción detallada de los nombres y los tipos de archivos desarrollados, así como también la manera de hacer la instalación del sistema.

El sistema está diseñado para que se puedan crear nuevos modelos para simulación a partir de librerías ya creadas, las librerías se presentan en forma gráfica escribiendo en la línea de comandos del *MATLAB* la palabra correspondiente. Se han creado catorce librerías y son: fuentes, análisis, codecs, decods, askmodular, askdemodular, fskmodular, fskdemodular, pskmodular, pskdemodular, pskdsmodular, pskdsdemodular, qammodular y qamdemodular. Las dos primeras de propósito general, que se van estudiar en este capítulo, las dos siguientes para los códigos de Línea y las últimas para la modulación digital.



Adicionalmente se desarrolló un ambiente gráfico que le permite al usuario encontrar los bloques de simulación desarrollados en esta tesis y ejemplos que le serán útiles para la mejor comprensión de su funcionamiento. Este ambiente gráfico es una ventana interactiva que presenta cuatro opciones, la primera activa una venta para la presentación de las librerías, la siguiente opción es otra ventana que permite la elección de modelos de ejemplos de codificación lineal y modulación digital, la tercera opción es una ventana de ayuda básica y la última opción cierra la ventana interactiva. Esta ventana interactiva se presenta al escribir en la línea de comandos la palabra "comdig", como se muestra en la figura 2.1 y las opciones se seleccionan con botones como los de cualquier aplicación de *windows*.

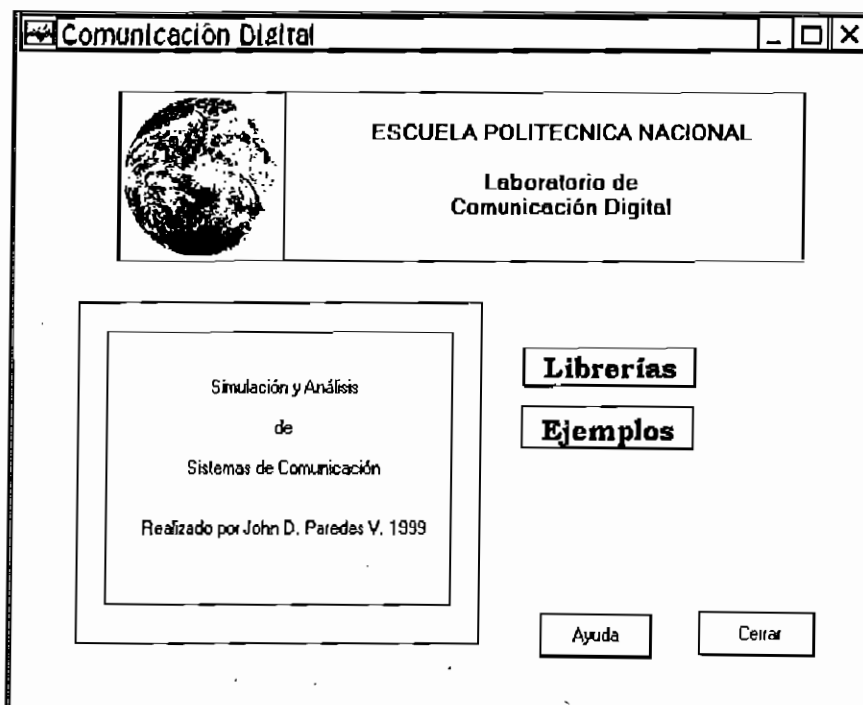


Figura 2.1 Ventana General del comando 'comdig'

Una vez que se tiene la ventana de la figura 2.1 se pueden elegir las opciones de librerías o ejemplos. Al seleccionar el botón librerías se tiene la la ventana de la figura 2.2, luego con ayuda del ratón se puede disponer de la librería deseada. Con la opción de Nuevo modelo se activa

un modelo con el nombre de *untitled*, que quiere decir sin título, el cual se encuentra vacío y con los valores por defecto de los parámetros de simulación.

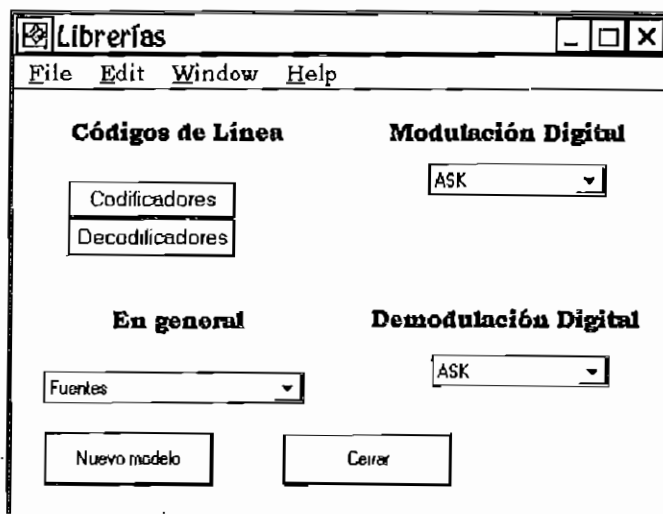


Figura 2.2 Librerías Desarrolladas en esta Tesis

Entonces para la simulación de un nuevo modelo se toma la opción de nuevo modelo y los bloques necesarios que se encuentran en las librerías de Modulación Digital, Demodulación Digital y las librerías que se encuentran bajo el título de "En general", donde se tienen las fuentes digitales, filtros y otras utilidades bajo la librería análisis.

Se diseñaron varios ejemplos de modulación y el usuario puede disponer de ellos bajo la opción ejemplos de la ventana principal de la figura 2.1. Cuando se toma la selección ejemplos aparece la ventana desarrollada de la figura 2.3 y el usuario puede elegir el tipo de modulación bajo el menú Modulación Digital, el tipo de ejemplo disponible aparecerá en el menú denominado Ejemplos en la ventana; y, por último una vez seleccionado el ejemplo de este menú el usuario puede llamar a dicho ejemplo a través del botón Ejecutar.

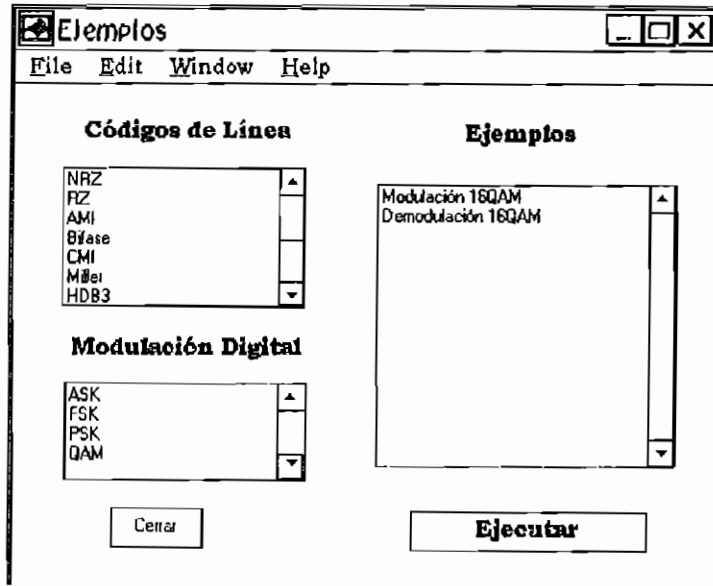


Figura 2.3 Ejemplos Desarrollados en esta Tesis

Estos son los pasos generales que el usuario puede utilizar para no escribir en la línea de comandos el nombre de cada librería o el nombre de cada modelo de ejemplo para realizar consultas de su propio interés. Los ejemplos desarrollados permiten al usuario realizar cualquier tipo de cambio, por lo que se recomienda tener un respaldo de los mismos.

### 2.1.1 Librería Fuentes.

La librería fuentes está conformada por tres tipos de fuentes de datos digitales, una fuente denominada de reloj y dos bloques de retardo de señal; tal como se puede apreciar en la figura 2.4.

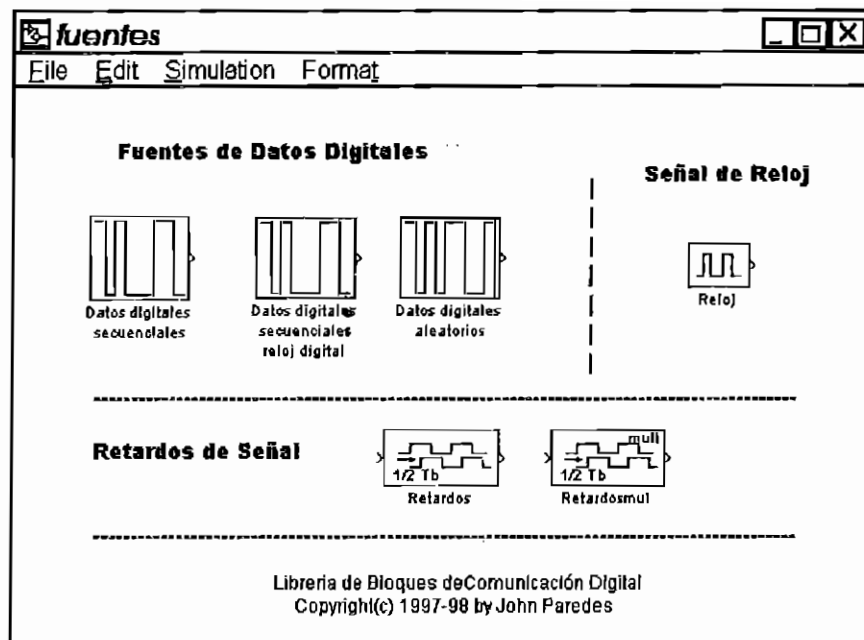


Figura 2.4 Librería Fuentes.

### 2.1.1.1 Simulación de Fuentes de Datos Digitales.

Estos bloques de simulación, están desarrollados a partir de una señal de tiempo, un bloque *S-function*. Se utilizan dos tipos de señales de tiempo, pero el archivo "gendatos.m" del bloque *S-function* es el mismo para los tres tipos de fuente de datos.

En el anexo A se da una descripción de las condiciones que debe cumplir un archivo "\*.m" para ser utilizado en el bloque *S-function*. La programación en sí se encuentra en este archivo "gendatos.m"; y, la diferencia de los bloques de simulación está en la manera que los datos ingresan a este bloque *S-function*. Los datos que se necesitan en el archivo "gendatos" son tiempo de bit, amplitud de salida y los datos binarios unos y ceros en forma de vector como por ejemplo [101010001]. En el archivo gendatos cada tiempo de muestreo se define en cada tiempo de bit, es en ese instante de tiempo donde se empieza secuencialmente a enviar como resultado cada elemento del vector de datos binarios y si se terminaran los elementos volvería a enviarlos nuevamente.

En la figura 2.5 se encuentran los bloques de simulación de las fuentes de datos digitales secuenciales con su respectiva plantilla de datos. En el mismo gráfico (partes d y e) se encuentran como están diseñados esos dos bloques. Se desarrolló el bloque de secuencias digitales con una señal de entrada que proviene del bloque *clock*, la cual permite tener el valor de la variable tiempo de simulación y para el caso del bloque Datos digitales secuenciales de reloj digital en lugar del bloque *clock* se tiene el bloque *Digital clock*, el cual proporciona lo mismo que el anterior pero con intervalos de tiempo bien definidos, en este caso 10 veces el tiempo de bit deseado.

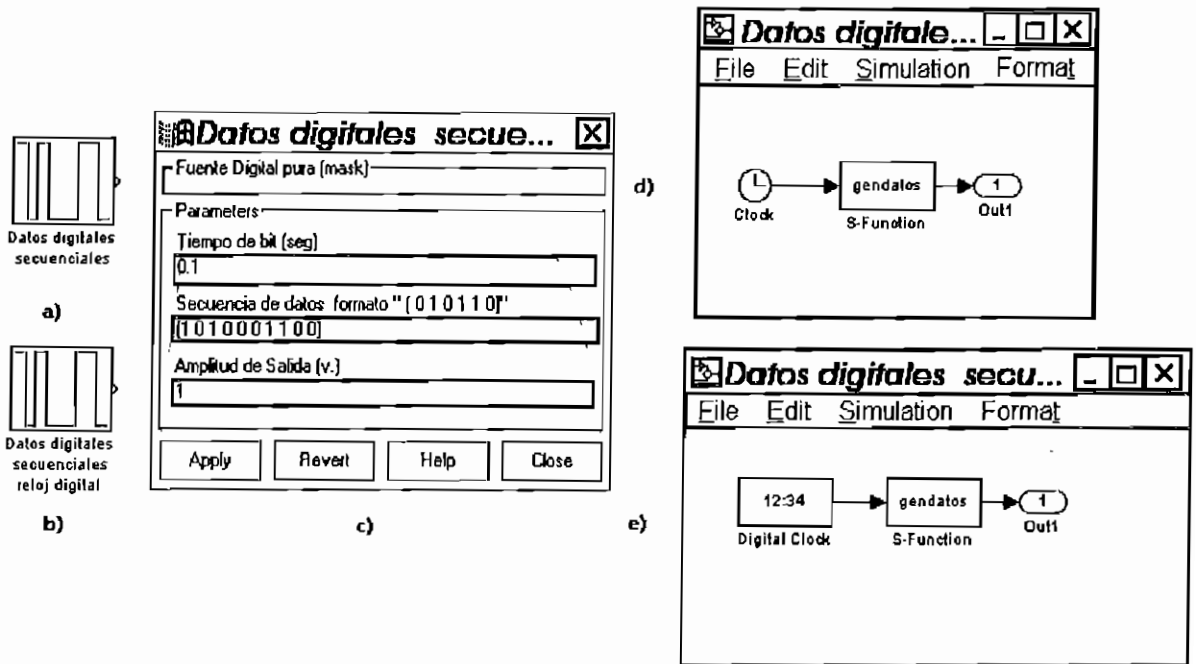


Figura 2.5 Fuentes de Datos Secuenciales:

El comportamiento de estas dos fuentes es muy similar, la diferencia se da en que el que utiliza el bloque *clock* después de dos o tres segundos de simulación sufre unos pequeños corrimientos de fase, mientras que el otro bloque siempre va a responder cada tiempo ya

definido. Las dos fuentes de datos se utilizan tanto para Codificación como para Modulación Digital.

El bloque denominado Datos digitales aleatorios es importante para el estudio de Densidad Espectral de Potencia, ya que genera datos binarios (unos y ceros) equiprobables, pero con una distribución del tipo gaussiana. Se diseñó dicho bloque a partir del bloque Datos secuenciales con reloj digital, con la única diferencia de que la variable de datos binarios es generada con ayuda de la función del *MATLAB* denominada *rand*, el algoritmo que permite esta generación es el siguiente:

```
r = rand(1,nsize)
at = ones(size(r))
index = (r<0.5);
at(index) = zeros( length( r(index) ), 1 );
```

donde *at* es el vector de datos (unos y ceros aleatorios), *nsize* es el número de datos aleatorios deseados. Este método crea un vector de unos de *nsize* elementos luego aquellos sitios de este vector en los que *r* es menor que 0.5 se llenan de ceros; al ser *r* números aleatorios entre 0.0 y 1.0 con distribución gaussiana hace que los ceros y unos también estén en esta relación.

#### **2.1.1.2 Simulación de Retardos de Señal.**

Este bloque se diseñó para conseguir que los datos generados con cualquiera de las fuentes de datos digitales o cualquier señal digital pueda retardarse un número finito de intervalos de tiempo. Por afinidad con la duración de un bit, se le solicita al usuario un valor que se denomina tiempo de bit, el cual se utilizará como referencia y además el

usuario debe indicar el número de semitiempos de bit que la señal se retrasará. Este bloque almacena datos hasta un límite de cuarenta cada semiperiodo de bit hasta que sea el tiempo de retransmitirlos, almacena los siguientes y se continua transmitiendo con el retardo respectivo.

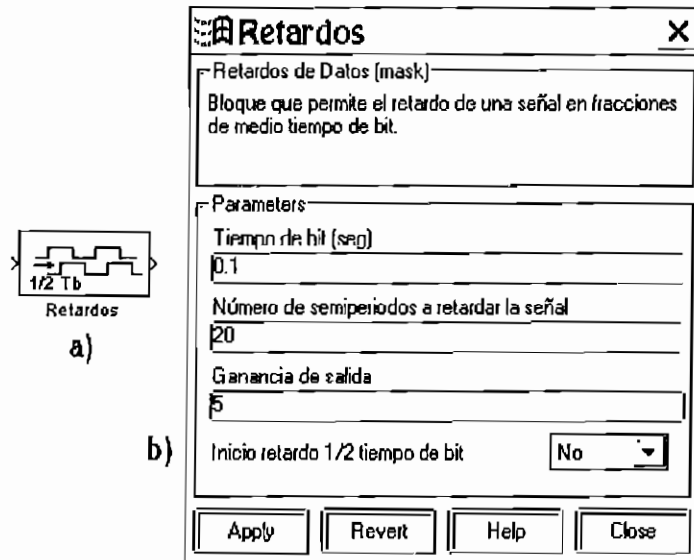


Figura 2.6 Bloque de Simulación de Retardos de señal digital: a) Bloque simulador y b) Plantilla de datos.

En la figura 2.6 se representa el bloque desarrollado y su respectiva plantilla de datos. El usuario puede también modificar un factor de ampliación de la señal con la opción Ganancia de salida. El indicador denominado "inicio de retardo 1/2 tiempo de bit" de la figura 2.6b, indica si el usuario desea que se empiece la lectura de datos en el primer semiperiodo de bits o en el segundo. Se habla de tiempo de lectura de datos ya que se diseñó este bloque a partir del bloque *S-function* con el archivo de programación "retardos.m", el cual tiene un comportamiento de bloque digital, es decir a la entrada del bloque se toma el valor del dato cada cierto tiempo, se procesa la información y se calcula la salida respectiva del bloque hacia el modelo de simulación<sup>1</sup>.

<sup>1</sup> Más información de bloques digitales en referencias [4] y [7].

El bloque denominado Retardosmul se representa en la figura 2.7. Sus funciones son las mismas que las diseñadas para el bloque anterior y su programación se encuentra en el archivo "retardosmul.m". Este bloque se desarrolló con el fin de que se identifiquen señales con valores discretos de +1,+3,0,-1 y -3 cuando se toma la opción multinivel en la plantilla de datos que se representa en la figura 2.7b, al desactivar esta opción el bloque es igual al bloque retardos.

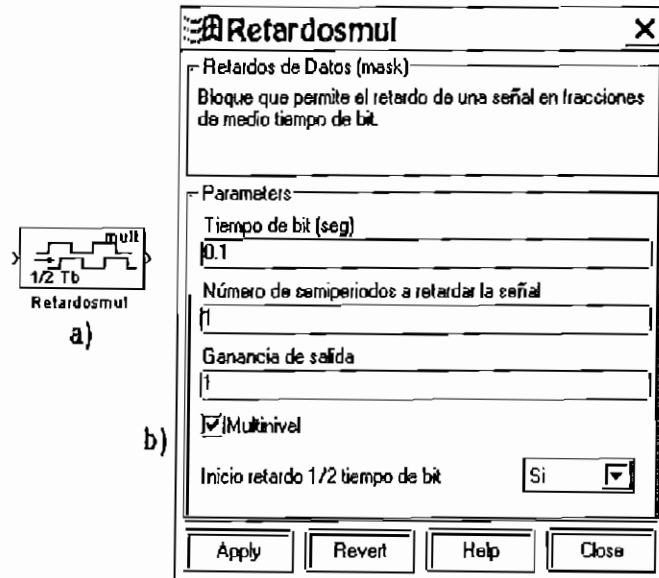


Figura 2.7 Bloque de Simulación de Retardos que puede identificar hasta tres niveles de señal: a) Bloque simulador y b) Plantilla de datos.

### 2.1.1.3 Simulación de Señal de Reloj.

Este bloque se desarrolló con el propósito de tener disponible una señal periódica cuadrada con una frecuencia definida. En la figura 2.8 se representa el bloque de simulación, su contenido o diseño y la plantilla de datos donde el usuario puede modificar el tiempo de bit (o también denominado frecuencia) y la magnitud de salida.



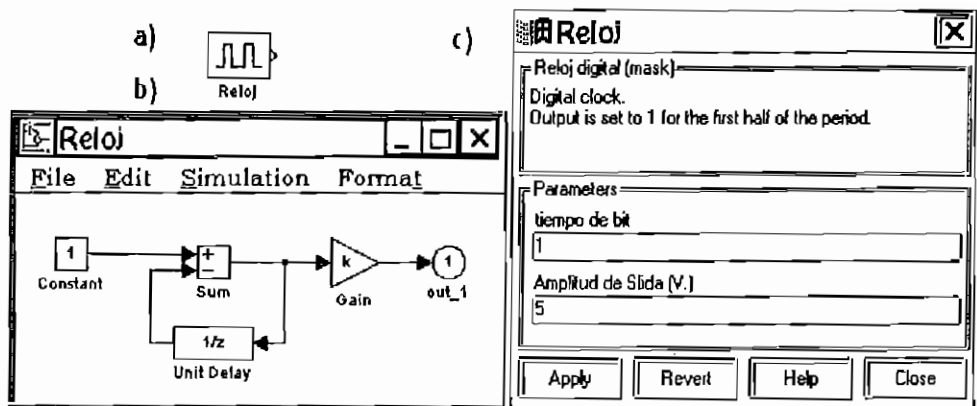


Figura 2.8 Bloque Reloj: a) Bloque de simulación de señal de reloj, b) Contenido del bloque reloj y c) Plantilla de datos del bloque reloj.

### 2.1.2 Librería Análisis.

En esta librería se encuentran bloques desarrollados para realizar estimaciones de la Densidad Espectral de Potencia, la simulación de un canal con ruido blanco, el diagrama del ojo; y, los restantes son de utilidad general en el proceso de simulación. En la figura 2.9 se tiene el contenido de esta librería.

Los bloques que se tratarán en esta sección son los osciladores controlados por voltaje, el bloque monoestable y el repetidor regenerativo. En las secciones 2.2, 2.3 y 2.4 se tratará el funcionamiento de los bloques restantes.

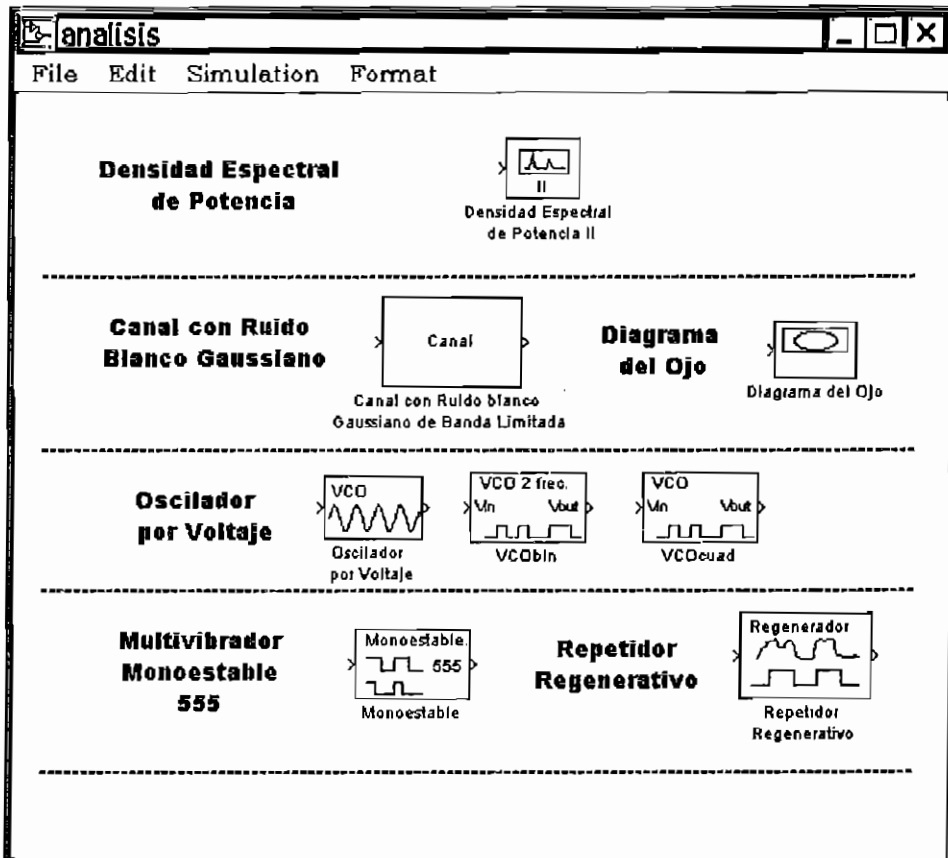


Figura 2.9 Librería Análisis

### 2.1.2.1 Simulación de Osciladores Controlados por Voltaje.

Para la simulación de los osciladores generados por voltajes o también denominados con las siglas VCO (*Voltage Contoled Oscilator*), se diseñaron a partir de dos esquemas, el primero un VCO senoidal de fase continua y el segundo basado en un generador de onda triangular.

En la figura 2.10 se muestra el bloque diseñado para simular un VCO senoidal de fase continua. El usuario puede modificar la frecuencia de muestreo, la frecuencia portadora y el valor de la desviación máxima de frecuencia de oscilación. La frecuencia de muestreo indica al bloque simulador cada cuanto tiempo se debe calcular la salida del VCO en base a los datos y el tiempo de simulación respectivos. La frecuencia portadora

es la frecuencia de oscilación libre es decir la frecuencia de salida cuando el voltaje en la entrada del bloque es cero. La desviación máxima de frecuencia indica la frecuencia máxima y mínima que alcanza el VCO para ciertos valores también máximos y mínimos respectivos de señal de entrada al bloque.



Figura 2.10 Bloque VCO de onda senoidal de fase continua.

Este bloque se desarrolló con la ayuda del bloque *S-function* y su respectiva programación se diseñó en el archivo "ospv.m". La programación cumple con la ecuación (2.1) :

$$s(t) = A_c \cos[2\pi \cdot f_c \cdot t + 2\pi \cdot \Delta f \cdot t \cdot p_k + \gamma_k] \quad (2.1)$$

donde, si  $f_1$  es la frecuencia mínima del VCO y  $f_2$  es la frecuencia máxima entonces  $f_c = \frac{f_1 + f_2}{2}$ ,  $\Delta f$  se define como  $\frac{f_2 - f_1}{2}$ ,  $p_k$  es una función binaria con estados posibles  $\pm 1$  que representan datos binarios de entrada ( $k = 1, 2, \dots$ ) y  $\gamma_k = \gamma[(k-1)T_b]$ , exceso de fase al inicio del  $k$ -ésimo intervalo de bit necesario para mantener una señal continua, donde  $T_b$  es el tiempo de duración del dato binario. Este bloque se empleará en modulación FSK y los detalles respectivos a la ecuación (2.1) se encuentran en la sección 4.2.2.1.

El VCO de la figura 2.11 se diseñó de tal forma que cuando se tiene a la entrada un nivel de cero se tenga una señal cuadrada a cierta frecuencia y cuando la entrada sea uno, la señal cuadrada de salida tendrá otra frecuencia. Para el usuario estas dos frecuencias se pueden modificar en la plantilla de datos respectiva.

El principio de funcionamiento de este VCO es el mismo que el de un generador de onda triangular a partir de un comparador y un integrador como el VCO del anexo B, puede estudiarse también en la referencia [9]. El generador de onda cuadrada se diseñó a partir de un bloque lógico, un integrador y un comparador con histéresis, tal como se ve en la figura 2.11c. La señal de entrada mediante el circuito lógico proporciona la adecuada pendiente adecuada del integrador, el cual envía su señal triangular al comparador, el mismo que entrega un valor en uno o en cero dependiendo de la señal de entrada. El circuito lógico toma en cuenta el valor de entrada al VCO y el de salida del comparador, cuando la pendiente del integrador es positiva el comparador entregará una señal de uno y cero en el caso contrario. Al tenerse en el bloque diseñado un lazo realimentado puede presentarse al inicio de la simulación incertidumbre en los valores y puede ocasionar que esta simulación no se pueda ejecutar, por esta razón se añadió el bloque *IC* de la librería *Conectios* que permite definir condiciones iniciales.

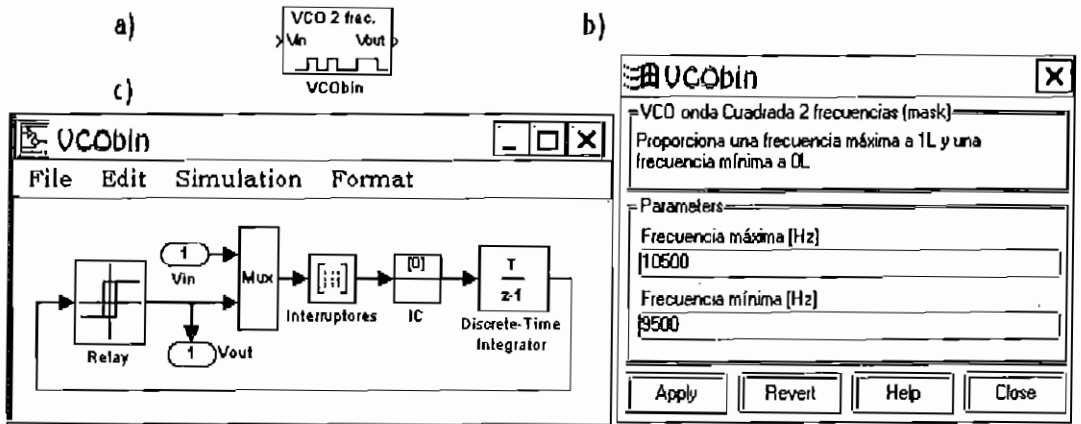


Figura 2.11 Bloque de Simulación del VCO de onda cuadrada: a) Bloque de Simulación, b) Plantilla de datos y c) Contenido del Bloque de simulación.

El bloque desarrollado de la figura 2.12 presenta una onda cuadrada a partir de una señal de entrada y su funcionamiento es muy similar al de la figura 2.11. Este bloque se diseñó para que de como respuesta una frecuencia máxima a una entrada de uno y una frecuencia mínima cuando el valor de entrada sea cero, pero a diferencia del VCO anterior si acepta valores intermedios de señal de entrada que obtendrá su respectiva respuesta con la frecuencia proporcional a este nivel de entrada; es decir que la constante proporcional de fase  $K_o = f_{max} - f_{min}$ . Además el usuario en la opción "Nivel de señal a frecuencia máxima" puede mejorar la resolución de respuesta ampliando el nivel de señal para la frecuencia máxima.

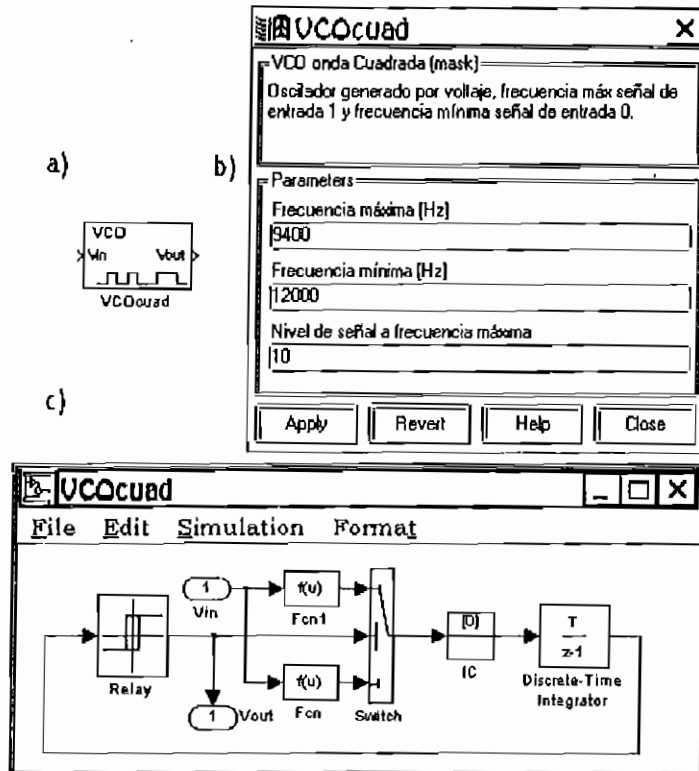


Figura 2.12 Bloque de Simulación del VCO de onda cuadrada: a) Bloque de Simulación, b) Plantilla de datos y c) Contenido del Bloque de simulación.

### 2.1.2.2 Simulación de un circuito monoestable.

Este bloque se desarrolló con la necesidad de simular un circuito monoestable para la detección de señales digitales. El bloque se representa en la figura 2.13, y el usuario puede definir el tiempo en que la salida permanece en alto después de haber detectado un cambio de cero a uno en la entrada. Se diseñó de manera similar a un circuito temporizador 555, es decir un integrador y un *flip-flop S-R* de la librería *flip-flops*. El funcionamiento es idéntico a un 555 con un integrador que se carga muy rápidamente es decir casi linealmente<sup>1</sup>.

<sup>1</sup> Puede verse la referencia [9].



Figura 2.13 Bloque de Simulación de un circuito Monoestable

### 2.1.2.3 Simulación de un repetidor regenerativo.

Para la recuperación de una señal binaria se tomó un método descrito en la sección 7.2 de la referencia [2], es decir un esquema que consiste en colocar en serie con los datos recibidos un filtro pasabajos, un circuito de muestreo y retención (*Sampling and Hold*) y un comparador de nivel tal como se muestra en la figura 2.14.

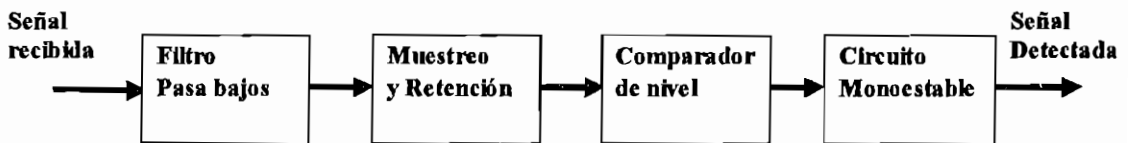


Figura 2.14 Esquema de un repetidor regenerativo de una señal binaria.

En la figura 2.15 se muestra el bloque diseñado con su respectiva plantilla de datos y el contenido del mismo. Para el filtro pasabajos se utilizó un filtro Butterworth de orden 4 y una frecuencia de corte igual al inverso del tiempo de bit. El muestreo y retención se realiza con un *flip-flop* tipo D activado por flanco positivo de una señal de sincronización. La sincronización se realiza mediante un VCO (bloque "VCOcuad"), el cual empieza a generar su señal biestable cuando hay una transición de cero a

uno de la señal del filtro pasabajos. Por último se tiene un comparador de señal y un multivibrador monoestable del tipo 555.

La señal de entrada ingresa al bloque, es amplificada, luego filtrada, posteriormente es muestreada y luego con un comparador de nivel se obtiene la señal binaria regenerada. Finalmente con el circuito monoestable se puede controlar cuando el dato es realmente uno o cero con los tiempos de bit adecuados.

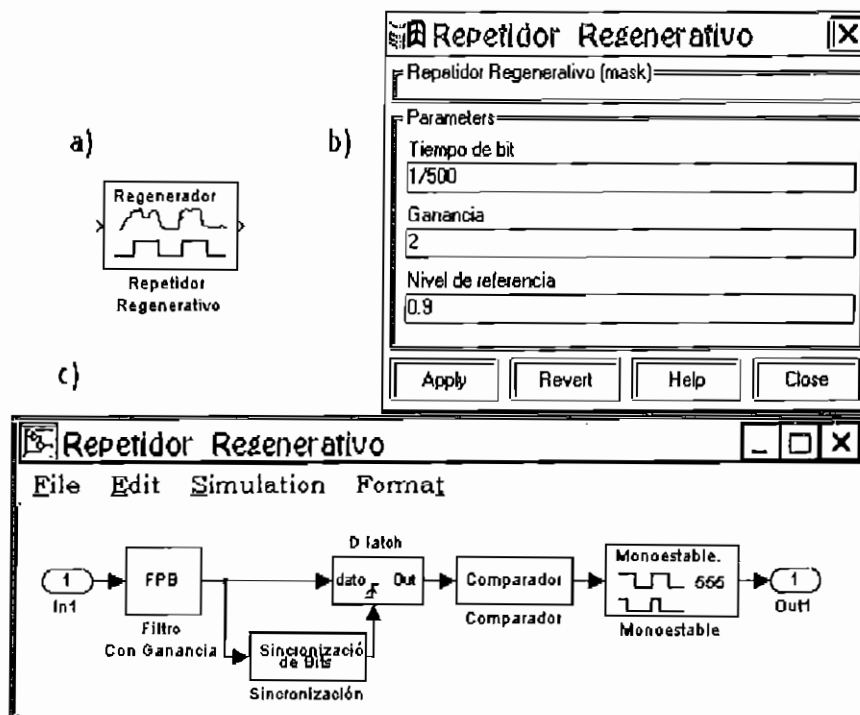


Figura 2.15 Bloque de simulación de un repetidor regenerativo de una señal binaria: a) Bloque de simulación, b) Plantilla de datos y c) Contenido del bloque de simulación.

## 2.2 DENSIDAD ESPECTRAL DE POTENCIA.

Para analizar las características de Densidad Espectral de Potencia de una Señal se dispone de un bloque en el *SIMULINK* en la librería de *extras*, utilizando la transformada rápida de Fourier del *MATLAB*.



Para una mejor comprensión del funcionamiento y con el objeto de que se tenga de este bloque un adecuado criterio al ingresar los valores solicitados en dicho bloque de simulación, se realizará una breve descripción de lo que significa la Transformada discreta de Fourier (*Discret Fourier Transform*, DFT) y la transformada rápida de Fourier (*Fast Fourier Transform*, FFT). El *MATLAB* en el bloque "Densidad espectral de Potencia" (*Power Spectral Density*, PSD), utiliza la FFT para el cálculo numérico. .

### 2.2.1 Transformada Discreta de Fourier.

Con el conveniente uso de la computadora y el desarrollo de circuitos integrados para el procesamiento digital de señales, se puede estimar con buena aproximación las características de frecuencia de una señal a partir de la transformada discreta de Fourier. El análisis a continuación es tomado de las referencias [2] y [1], pero se puede consultar también la referencia [3].

Definición: La Transformada Discreta de Fourier (DFT) de una señal discreta  $x(k)$  se define como<sup>1</sup>:

$$X(n) = \sum_{k=0}^{N-1} x(k) \cdot e^{-j\left(\frac{2\pi}{N}\right)nk} \quad (2.2a).$$

donde  $n=0,1,2,\dots,N-1()$ ; y, la correspondiente transformada *inversa* (*Inverse Discret Fourier Transform*, IDFT) se define así:

---

<sup>1</sup>Esta definición puede variar según otros autores, el sumatorio también puede ir de 1 a N.

$$x(k) = \frac{1}{N} \sum_{n=0}^{n=N-1} X(n) \cdot e^{j \left( \frac{2\pi}{N} \right) nk} \quad (2.2b).$$

N es el número de valores calculados de IDFT y DFT.

La definición del *MATLAB* para la DFT es la siguiente:

$$X(k+1) = \sum_{n=0}^{n=N-1} x(n+1) \cdot e^{-j \left( \frac{2\pi}{N} \right) kn} \quad (2.3),$$

los subíndices  $n+1$  y  $k+1$  se deben a que los índices de las matrices en el programa empiezan desde 1.

La Comparación entre DFT y la transformada continua (en ingles *Continuous Fourier Transform*, CFT) se entiende a través de tres conceptos fundamentales truncamiento (en ingles *windowing*), muestreo y la generación de señales periódicas como consecuencia del muestreo; esto se ilustra en la figura 2.16 donde se tiene las señales en el dominio de la frecuencia a la izquierda y en el dominio del tiempo a la derecha.

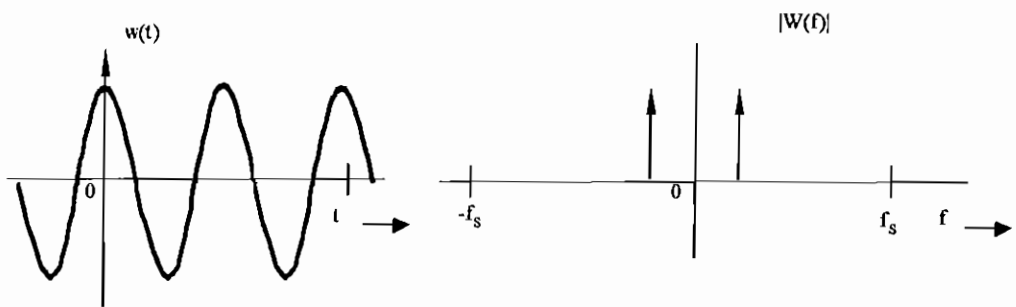


Figura 2.16a) señal coseno y su diagrama espectral

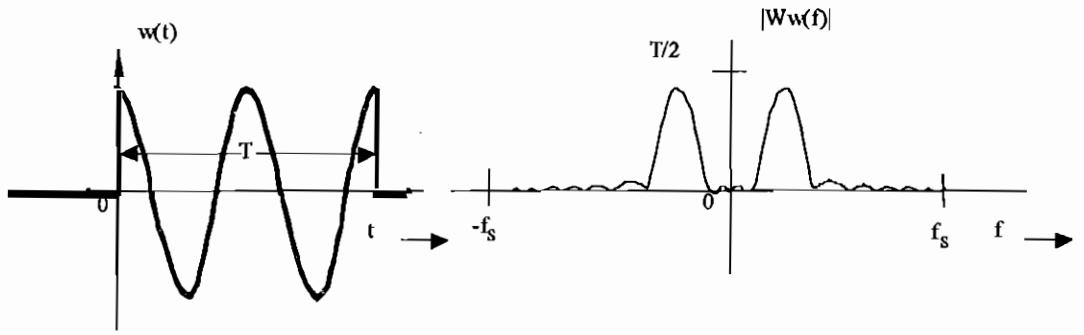


Figura 2.16 b) señal coseno truncada y su diagrama espectral

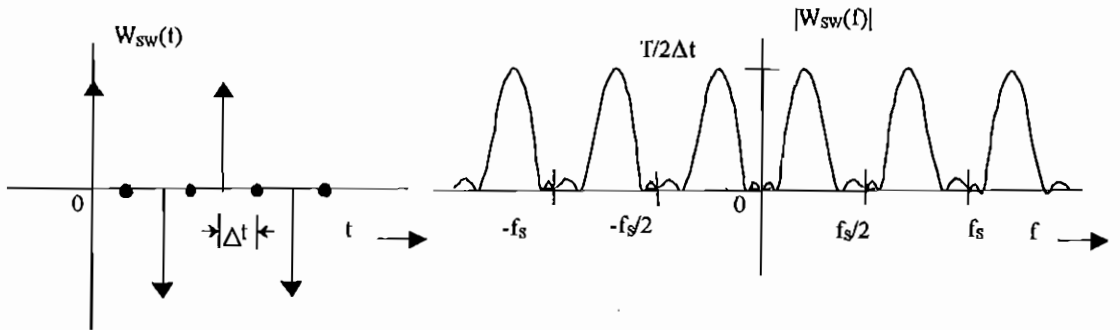


Figura 2.16 c) señal muestreada y truncada en el tiempo y su diagrama espectral ( $f_s = 1/\Delta t$ )

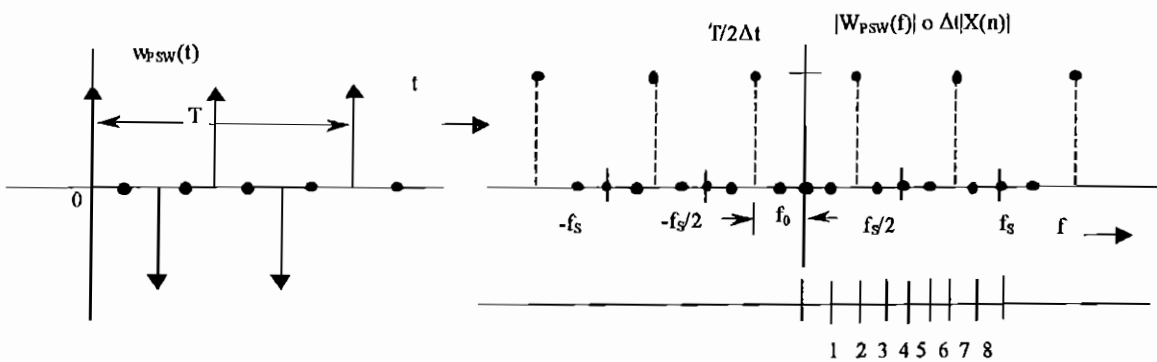


Figura 2.16 d) señal muestreada periódicamente, truncada en el tiempo y su diagrama espectral ( $f_0 = 1/T$ )

Figura 2.16 Comparación entre DFT y CFT (Transformada continua de Fourier)

Partimos del análisis de una señal continua coseno la cual se desea evaluar a través de la DFT, la señal en el dominio del tiempo es primero truncada en el intervalo  $(0, T)$  ya que se requiere calcular solo un número

de muestras finitas, la función truncada (o multiplicada por una función denominada ventana) denotada con el subíndice  $w$  se define matemáticamente así:

$$w_w = \begin{cases} w(t), & 0 \leq t \leq T \\ 0, & \text{en otro caso} \end{cases} = w(t) \Pi\left(\frac{t - (T/2)}{T}\right) \quad (2.4),$$

donde  $\Pi(t/a)$  es la función tal que su valor es uno en  $-a/2 < t < a/2$  y cero en otro caso, la transformada de Fourier de la señal truncada es la siguiente:

$$W_w(f) = \int_{-\infty}^{\infty} w_w(t) \cdot e^{-j2\pi ft} dt = \int_0^T w(t) \cdot e^{-j2\pi ft} dt \quad (2.5),$$

utilizando la definición aproximada de la integral para representar la CFT se tiene, los siguientes valores:

$$t = k \Delta t, f = n/T, \text{ y } \Delta t = 1/N \text{ entonces}$$

$$W_w(f) \Big|_{f=n/T} \approx \sum_{k=0}^{N-1} w(k\Delta t) \cdot e^{-j(2\pi/N)nk} \Delta t \quad (2.6)$$

comparando con (2.2a) la relación entre CFT y DFT se obtiene sí:

$$W_w(f) \Big|_{f=n/T} \approx \Delta t \cdot X(n) \quad (2.7)$$

donde  $f = n/T$  y  $\Delta t = T/N$ . El valor de la muestra usada para el cálculo de la DFT es  $x(k) = w(k \Delta t)$  como se representa en la figura 2.16c. Al utilizarse el término  $e^{-j\left(\frac{2\pi}{N}\right)nk}$  en la ecuación (2.6), las muestras tomadas son periódicas en  $n$ , es decir los mismos valores se repetirán para  $n = N, N + 1, \dots$  tal como fueron obtenidos en  $n = 0, 1, \dots$

entonces  $X(n)$  es periódico (aunque solo los primeros valores son utilizados por los programas de cálculo de la DFT ya que los otros se repiten). Otra manera de ver que la DFT (y la IDFT) son periódicas es reconocer que se utiliza muestras discretas, la transformada discreta es un ejemplo de modulación de pulso y consecuentemente el diagrama espectral debe ser periódico para la frecuencia de muestreo,  $f_s = 1/\Delta t = N/T$  (como se ilustra en 2.3.1.c y d). Además si se necesita el diagrama espectral para frecuencias negativas el computador entrega valores de  $X(n)$  para valores de  $n$  de  $0, 1, \dots, N - 1$ , la ecuación (2.7) debe ser modificada para obtener valores del espectro sobre el rango fundamental de  $-f_s/2 < f < f_s/2$  así para valores positivos de frecuencia se utiliza:

$$W_w(f)_{f=n/T} \approx \Delta t \cdot X(n), \quad 0 \leq n < \frac{N}{2} \quad (2.8)$$

y para frecuencias negativas:

$$W_w(f)_{f=(n-N)/T} \approx \Delta t \cdot X(n), \quad \frac{N}{2} \leq n < N \quad (2.9)$$

En conclusión aparecen frecuencias extrañas primeramente al realizar el truncamiento de una señal continua, luego se debe muestrear esta señal truncada y el espectro resultante vuelve a presentar frecuencias extrañas que además son periódicas en todo el espectro. Entoces cuatro consideraciones fundamentales se deben tomar en cuenta cuando se evalúa la CFT a través de la DFT:

1. El truncamiento en el intervalo  $(0, T)$  que se utiliza al calcular un número finito de muestras.
2.  $N$  valores de muestras son utilizadas, las cuales están equidistantes a  $t = k \Delta t$  (donde  $k = 0, 1, \dots, N-1$  y  $\Delta t = T/N$ ).
3. La DFT entrega un espectro de  $N$  frecuencias ( $f = n/T$ ,  $n = 0, 1, \dots, N-1$ ) sobre el intervalo  $(0, f_s)$ , donde  $f_s = 1/\Delta t = N/T$ .
4. La DFT y la IDFT son periódicas con período  $f_s$  y  $T$  respectivamente.

La figura 2.16 indica que si no se tiene cuidado con los valores obtenidos de DFT, se pueden tener errores significativos al aproximarlos a la CFT, estos errores son básicamente: filtración de frecuencias extrañas, efecto de alias (aliasing) y resolución de frecuencia inadecuada.

El primer efecto es causado por el truncamiento en el dominio del tiempo. En el dominio de la frecuencia esto corresponde a la convolución entre el espectro de la función no truncada con el espectro (transformada de Fourier) de la función ventana, esto expande las componentes de frecuencia de  $w(t)$ , como se indica en la figura 2.16b y causa que se "filtren" frecuencias extrañas adyacentes, esta "filtración" puede generar errores al compararse con el espectro de CFT, este efecto puede disminuir incrementado el ancho de la ventana  $T$  o con el mismo resultado aumentando el número de puntos  $N$  que se usan para la DFT. Se puede también utilizar otro tipo de ventana la cual pueda reducir el ancho de los lóbulos espectrales adyacentes al lóbulo principal de interés, las componentes de períodos grandes en  $w(t)$  causan más "filtración" y pueden ser eliminados antes de evaluar la DFT para reducir la "filtración" de frecuencias. En particular en el bloque de simulación que realiza el cálculo de la DFT antes de la densidad espectral de potencia utiliza la ventana de Hanning, la cual tiene una transformada de Fourier con lóbulos laterales bajos, esta ventana también es denominada de coseno elevado y se define matemáticamente a continuación en la ecuación (2.10).

$$\left. \begin{array}{l} \frac{1}{2} \cdot \left( 1 + \cos\left(\frac{2\pi t}{T}\right) \right) \quad |t| < T/2 \\ 0 \quad \quad \quad |t| > T/2 \end{array} \right\} \quad (2.10),$$

la cual se muestra en la figura 2.17, la transformada de Fourier de la ventana coseno elevado tiene lóbulos laterales más bajos que la ventana rectangular a expensas de un lóbulo principal más ancho y presenta alguna atenuación; presente en distinta medida estos compromisos son típicos de las diferentes ventanas de uso común.

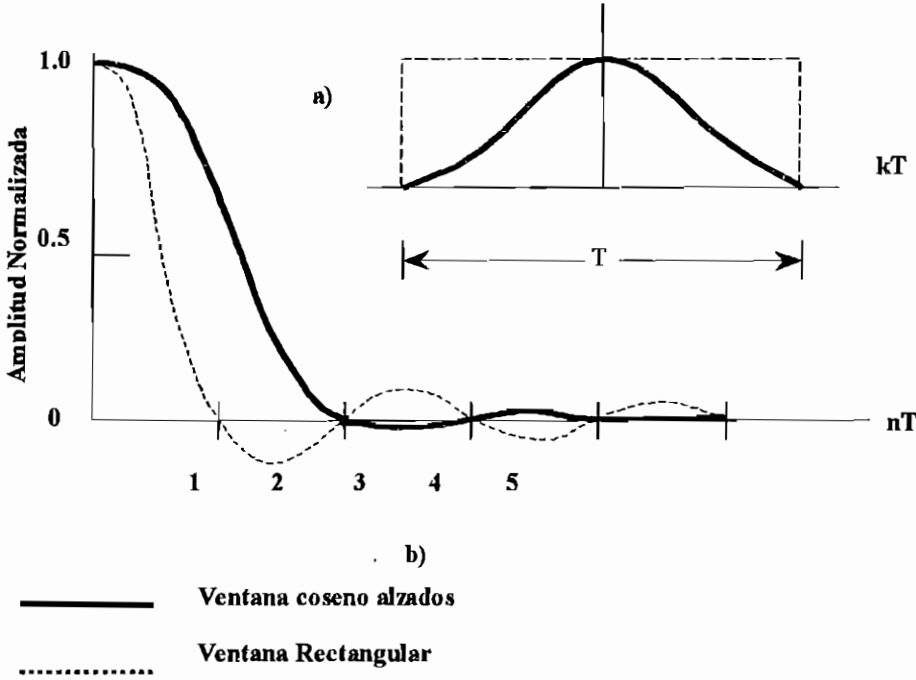


Figura 2.17 a) Funciones ventana rectangular y coseno elevado, y b) sus transformadas de Fourier.

Del teorema del muestreo<sup>1</sup>, es conocido que el diagrama espectral de una señal muestreada se constituye de replicas del espectro de la señal original (sin hacer muestreo) en valores de frecuencias múltiplos de la frecuencia de muestreo. Si  $f_s < 2B$ , donde  $B$  es la componente significativa de mayor valor de frecuencia de la señal sin muestreo, se presenta un error de alias, este error se disminuye al utilizar una frecuencia de muestreo de mayor valor o utilizando un filtro pasa - bajos previo al muestreo. También debe tomarse en cuenta que la componente

<sup>1</sup> El teorema del muestreo se encuentra demostrado por vario autores entre ellos la referencia [1], págs 126-130, referencia [2] págs 96-101.

de frecuencia más alta puede ser evaluada con  $N$  puntos de DFT si  $f = f_s/2 = N/(2T)$ .

El último tipo de error, es la inadecuada resolución de frecuencia, la cual ocurre debido a que  $N$  puntos de DFT no pueden obtener como resultado las componentes espectrales que se encuentren dentro del intervalo de frecuencias  $\Delta f = 1/T = f_s/N$ , si se aumenta la tasa de muestreo  $\Delta f$ , puede ser más pequeño, pero esto no incrementaría el número de puntos de la DFT. Si el límite de los datos en tiempo es  $T_0$  segundos donde  $T_0 \leq T$ ,  $T$  puede extenderse añadiéndose ceros adicionales a los puntos de muestreo, estos ceros añadidos se denominan "ceros aumentados" <sup>1</sup> y esto reducirá  $\Delta f$  lo que producirá una mejor resolución de frecuencia.

El siguiente ejemplo<sup>2</sup> ilustra como el *MATLAB* realiza el cálculo de la DFT de una ventana rectangular:

El ejemplo propone obtener la forma del espectro de una señal de pulso rectangular, para esto se define el número de muestras a tomarse con la variable  $M$ ,  $T$  es el valor hasta donde se va a muestrear la señal y  $tend$  es el ancho del pulso rectangular. Se calcula la *fft* a partir de los valores del pulso rectangular tomados cada instante de muestreo, el rango de frecuencias es  $f$  y es igual a el número total de muestras dividido para  $T$ . Los parámetros  $M$ ,  $tend$  y  $T$  se han seleccionado para que el cómputo de la magnitud, fase espectral dadas en la ecuación (2.5), el lector si lo desea puede redefinir los valores de  $M$ ,  $tend$  y  $T$  para observar los tres tipos de errores descritos anteriormente los que se incrementan

---

<sup>1</sup> En algunos textos se conoce como zero-padding como en [2]

<sup>2</sup> Tomado de [2] pág. 108 o del ejemplo 2-178 de la quinta edición disponible disco flexible.



si los parámetros no son seleccionados adecuadamente; T tiene que ver con el truncamiento de la señal, M con la resolución de frecuencia y los dos con el muestreo de la señal. Entonces el lenguaje codificado del ejemplo es el siguiente:

```
% Cálculo de la FFT para una función paso troncada
% Se define tend como la duración de la función paso.
M = 8;
N = 2^M;
n = 0:1:N-1;
tend = 1;
T = 10;
dt = T/N;
t = n*dt;
w = zeros(length(t),1);
for (i = 1:length(w))
    if (t(i) <= tend)
        w(i) = 1;
    end;
end;
% Cálculo de la FFT

W = dt*fft(w);
f = n/T;
subplot(211);
plot(t,w);
xlabel('t');
ylabel('w(t)');
title('Señal del pulso en el tiempo');
subplot(212);
plot(f(1:N/2),abs(W(1:N/2)));
xlabel('f');
title('Magnitud del espectro hasta fs/2');
subplot(111);
```

y los resultados se encuentran en la figura 2.18:

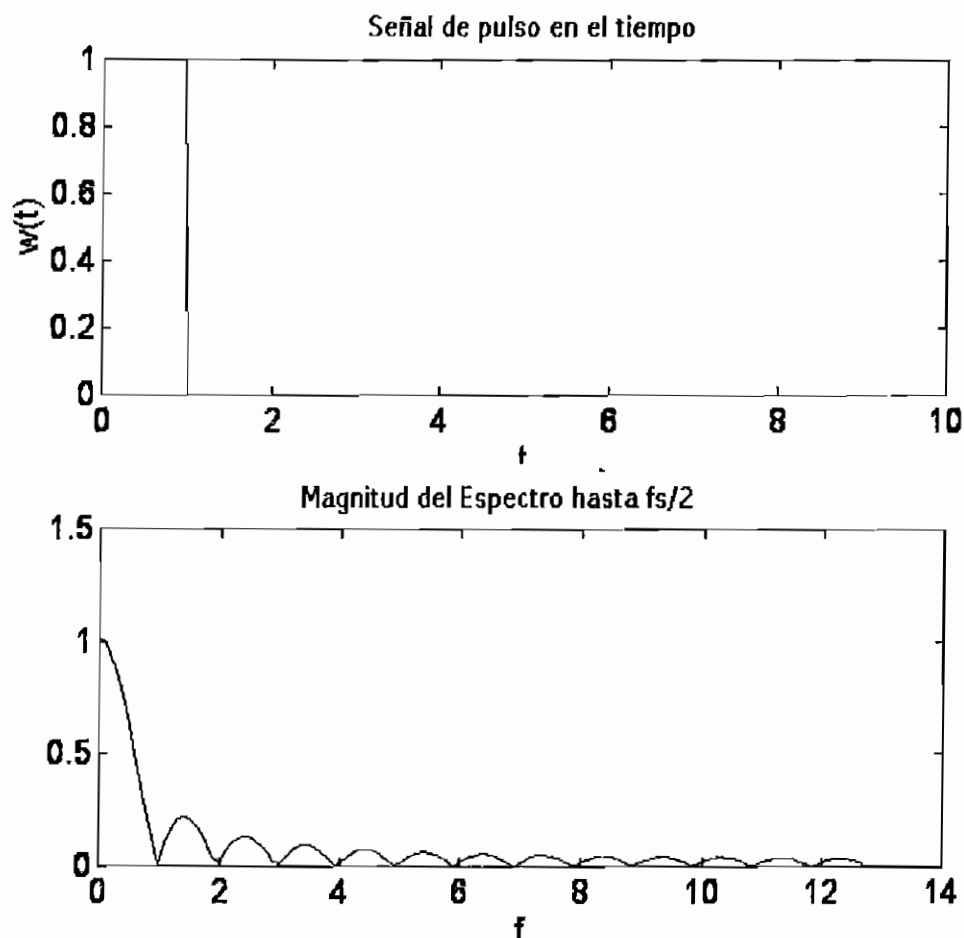


Figura 2.18 gráficos de la ventana rectangular y su diagrama espectral calculado por el MATLAB.

### 2.2.2 Transformada Rápida de Fourier.

El cálculo de la DFT requiere de  $N^2$  multiplicaciones (esto es,  $0 \leq k < N$ ,  $0 \leq n < N$ ), y el tiempo de cálculo se vuelve excesivo cuando  $N$  se hace muy grande. La clave de los métodos de cálculo más eficientes, es aprovechar la simetría de los exponentes complejos antes de realizar las multiplicaciones.

La Transformada rápida de Fourier (Fast Fourier Transform, FFT) no es más que un algoritmo para obtener la DFT con un número más reducido de operaciones. Se tratará brevemente el método utilizado por *MATLAB* que es el radix-2 o más conocido como el método de Cooley y

Tukey<sup>1</sup>. Este algoritmo calcula N componentes de frecuencia discreta a partir de N muestras en el tiempo discretas donde  $N = 2^r$ , con r cualquier entero. Esta restricción de una potencia de dos no es seria en la práctica, así si  $2^r$  es mayor que el número de muestras; se puede completar con ceros.

Reemplazando en la ecuación (2.2 a)

$W^\ell = e^{-j2\pi\ell/N}$   $\ell = 0, 1, 2, \dots$  (2.11) y  $f(k)$  por  $x(k)$  se tiene entonces:

$$F_D(n) = X(n) = \sum_{k=0}^{N-1} f(k)W^{nk} \quad (2.12)$$

La ecuación (2.11) describe un fasor de magnitud unitaria y ángulo de fase  $\theta_\ell = -2\pi\ell/N$ . Como ejemplo sea  $r = 3$  en donde los valores correspondientes a (2.11) se muestran en la figura 2.19. De esta figura, se puede ver que  $W^\ell$  y  $W^{-\ell}$  están localizados en forma simétrica alrededor del eje real, y se cumplen las siguientes propiedades en la ecuación 2.13:

$$\begin{aligned} W^N &= W^0 = 1 \\ W^{N/2} &= -1 = -W^0, \\ W^{N-\ell} &= [W^\ell]^* \end{aligned} \quad (2.13)$$

---

<sup>1</sup> J.W. Cooley y J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series", Mathematics of Computation 19, abril de 1965, págs. 297-301

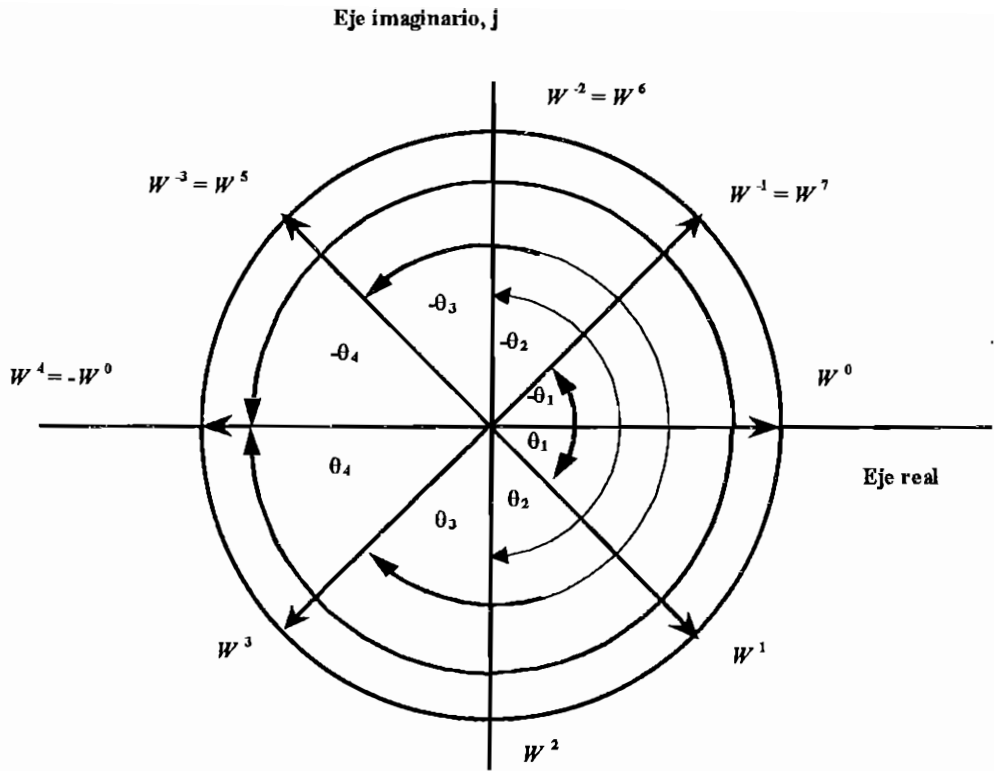


Figura 2.19 Potencias de la función exponencial  $W$  para  $N = 8$ .

Con el fin de sacar el mayor provecho del uso de  $N = 2^r$ , se expresan  $n$ ,  $k$  como números binarios. Supóngase que  $r = 2$  (esto es,  $N = 4$ ) y

$$k = (k_1, k_0) = \{00, 01, 10, 11\},$$

$$n = (n_1, n_0) = \{00, 01, 10, 11\}, \quad (2.14a)$$

donde  $n_0, n_1, k_0, k_1$  sólo pueden tomar valores de 0 y 1. Un método compacto de escribir el valor numérico de  $k$  y  $n$  es

$$k = 2k_1 + k_0,$$

$$n = 2n_1 + n_0 \quad (2.14b)$$

Usando las ecuaciones (2.14a) y (2.14b) en la ecuación (2.12), se obtiene:

$$F_D(n_1, n_0) = \sum_{k_0=0}^1 \sum_{k_1=0}^1 f(k_1, k_0) \cdot W^{(2n_1+n_0)(2k_1+k_0)} \quad (2.15),$$

donde ahora es necesario un doble sumatorio para enumerar la representación binaria completa de k.

Considérese el termino exponencial de (2.15)

$$W^{(2n_1+n_0)(2k_1+k_0)} = W^{2n_0k_1} W^{(2n_1+n_0)k_0} \quad (2.16)$$

Debido a que  $W^{(4n_1k_1)} = 1$  para todos los enteros  $n_1, k_1$ . Este último paso es crucial para la eficiencia de la FFT porque ahora la ecuación (2.15) se puede escribir como<sup>1</sup>

$$F_D(n_1, n_0) = \sum_{k_0=0}^1 \left[ \sum_{k_1=0}^1 f(k_1, k_0) \cdot W^{2n_0k_1} \right] \cdot W^{(2n_1+n_0)k_0} \quad (2.17),$$

Ahora se puede mostrar el algoritmo reescribiendo el sumatorio entre los corchetes como

$$f_1(n_0, k_0) = \sum_{k_1=0}^1 f(k_1, k_0) \cdot W^{2n_0k_1} \quad (2.18)$$

y el sumatorio externo como

---

<sup>1</sup> Para una interpretación matricial de esto véanse los capítulos 10 y 11 de Brigham, E.O., The Fast Fourier Transform, Englewood Cliffs, NJ., Prentice Hall, 1974

$$f_2(n_0, n_1) = \sum_{k_0=0}^1 f_1(n_0, k_0) \cdot W^{(2n_1+n_0)k_0} \quad (2.19)$$

De las ecuaciones (2.17) y (2.19), se obtiene

$$F_D(n_1, n_0) = f_2(n_0, n_1) \quad (2.20)$$

Este último paso se incluye porque el mencionado algoritmo mezcla el orden de salida. Una variante es mezclar el orden de entrada para que la salida esté en el orden correcto.

Las ecuaciones (2.18) y (2.20) son las relaciones recursivas que constituyen el algoritmo de Cooley y Tukey para  $N = 4$ . En la figura 2.20 se muestra una gráfica del flujo de señales de estas relaciones. Los algoritmos para potencias de dos mayores son similares, con una suma en la ecuación (2.15) por cada potencia de dos considerada.

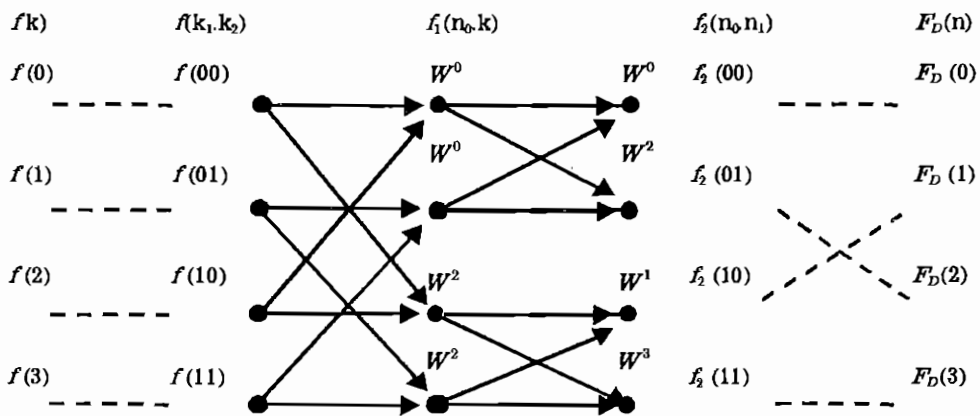


Figura 2.20 Gráfica del Flujo de señales para  $N = 4$  del algoritmo de Cooley y Tukey.

Se necesita alrededor de  $N^2$  operaciones complejas de suma y multiplicación en una evaluación directa de la DFT mientras que el

algoritmo de la FFT requiere sólo alrededor de  $N \log_2 N$  operaciones. Los ahorros netos se vuelven apreciables para  $N$  grande. Por ejemplo, el tiempo de cálculo para una DFT de 1024 puntos por evaluación directa es casi 100 veces superior al que necesita usando el algoritmo de la FFT. Sin embargo, el algoritmo requiere una cantidad considerable de memoria y esto puede limitar su aplicación en situaciones donde se tiene disponible una cantidad limitada de memoria.

Por último se resumirán algunos puntos útiles que se deben tomar en cuenta al momento de utilizar este algoritmo<sup>1</sup>:

1. Se elige un número de muestras  $N$  tal que  $N = 2^r$ , con  $r$  entero. Este número puede incluir ceros aumentados
2. Para  $N$  muestras en el tiempo existen  $N$  frecuencias discretas.
3. Como el resultado de la extensión periódica de la DFT, los puntos 0 y  $N$  son idénticos en ambos dominios.
4. Se considera que las componentes de frecuencia positiva están en el rango de  $(0, N/2)$ ; las componentes de frecuencia negativa se encuentran en el intervalo  $(N/2, N)$ .
5. Para funciones de valor real, las componentes de frecuencia positiva son conjugadas complejas de las componentes de frecuencia negativa. Los puntos  $n = 0, N/2$  son iguales y tienen valor real.
6. La componente más alta de frecuencia (esto es,  $n = N/2$ ) es  $(2T)^{-1}$  Hz; se puede conseguir un margen de frecuencia más alto reduciendo el intervalo de tiempo entre muestras.

---

<sup>1</sup> Algunos de los puntos son los que se señalaron en la DFT

7. El intervalo de tiempo entre componentes de frecuencia es  $(1/NT)$ ; este puede disminuir añadiendo ceros aumentados a la secuencia de muestras para mejorar la resolución de frecuencia.
8. La relación exacta de los valores de la FFT depende de las constantes de multiplicación particulares asignadas en el algoritmo; un procedimiento común es dividir entre N de tal forma que los valores calculados sean  $1/N$  veces los de la DFT.

### 2.2.3 Cálculo numérico de la densidad Espectral de Potencia.

La Densidad espectral de Potencia (PSD) de una señal puede ser evaluada por varios métodos, tanto analógicos como digitales, siendo en cualquier caso tan solo una aproximación al valor exacto de la PSD porque la medición se realiza en un intervalo de observación lo cual no corresponde a la definición, la cual de acuerdo a la definición exige sea en un intervalo infinito<sup>1</sup>.

Para un procesos aleatorio, se define la densidad espectral de potencia de la siguiente manera<sup>2</sup>:

$$S(f) = \lim_{T \rightarrow \infty} \left( E \left\{ \frac{|F_T(f)|^2}{T} \right\} \right) \quad (2.21).$$

---

<sup>1</sup> Para nuestro Interés se dará la definición más general de PSD, como es el caso de variables aleatorias.

<sup>2</sup> Esta definición se puede encontrar en [1] pág. 506 y [2] pág. 507 otra referencia útil es Bendat J.S. y Pierlson A.G. "Random Data: Análisis and Measurement Procedures, 2ª. Ed. NY. Wiley-Interscience, 1986 "



Para un proceso aleatorio se define el valor promedio o medio  $E\{X\}$  de la siguiente manera:

$$E\{X\} = \int_{-\infty}^{\infty} xp(x)dx = \lim_{T \rightarrow \infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t)dt \quad (2.22)$$

La ecuación (2.21) es la señal densidad espectral de potencia de la señal  $f(t)$  cuya transformada de Fourier es  $F_T(f)$  en  $(-T/2, T/2)$ . No es práctico llevar al límite cuando  $T \rightarrow \infty$ , por lo que se tendrán consecuencias al examinar un intervalo de tiempo finito, pero existe un valor aproximado de la densidad espectral de potencia derivada de esta definición; así para una función  $f(t)$  sobre un intervalo de observación de  $T$  unidades se tiene la ecuación (2.23)

$$S_T(f) = \frac{|F_T(f)|^2}{T} \quad (2.23)$$

Para evaluar numéricamente la PSD, se utiliza la transformada discreta de Fourier en el cálculo aproximado de  $F_T(f)$ .

Como los valores aproximados dados por la ecuación (2.23) son con respecto a variables aleatorias, se espera al menos que en un promedio de lecturas tienda a ser igual al valor de PSD<sup>1</sup>; basándose en este promedio estadístico y en el hecho de que se tiene el efecto de una ventana, se presenta una inconsistencia del resultado del cálculo discreto con el valor real de PSD.

---

<sup>1</sup> Se pretende dar una explicación cualitativa para mejor comprensión, para ampliar la demostración de lo expuesto se tiene en la referencia [1] págs. 515-517 y [2] págs 519-520.

En la práctica las elecciones adecuadas para el cálculo de la densidad espectral se puede hacer más sencillamente conociendo alguna información sobre las características de la densidad espectral de potencia y deberían hacerse varias gráficas hasta obtenerse las mejores combinaciones de truncamiento, promediación y longitudes de registro de datos.

#### 2.2.4 Bloque y Algoritmo de simulación para el análisis de la Densidad Espectral de Potencia de una señal.

Para el cálculo numérico de la densidad espectral de potencia se dispone de un bloque de simulación desarrollado por Andrew Grace en 1991 para MathWorks INC. y revisado por Wes Wang y Charlie Ko en 1993 para su adaptación a las versiones de *MATLAB 5.0* y *SIMULINK 2.0.*, su presentación se muestra en la figura 2.21:

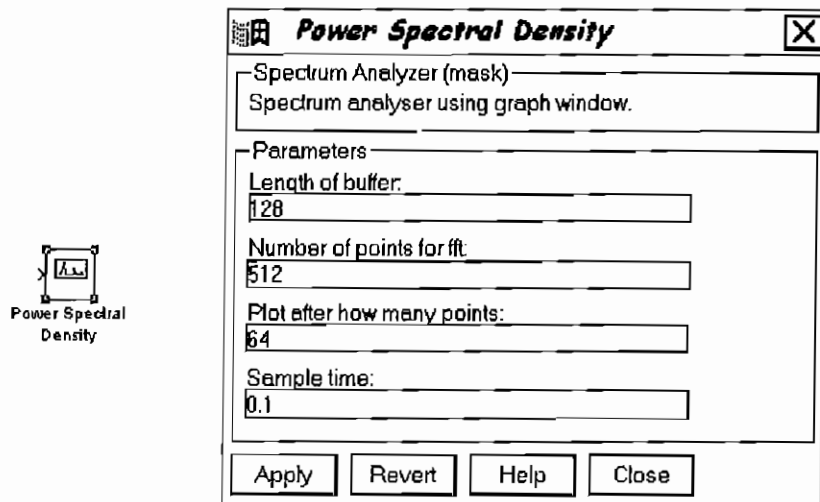


Figura 2.21 presentación del bloque PSD y sus requerimientos

Los datos que el usuario debe ingresar son: número de muestras a tomar de la señal que ingresa al bloque, número de datos que se utilizarán para el cálculo de la FFT, el número de puntos después de los cuales se graficará la FFT calculada. Este bloque utiliza el bloque de

simulación *S-function*<sup>1</sup>, cuyo programa se encuentra bajo el nombre de "sfunpsd.m", la cual emplea la FFT con un tipo de ventana de Hanning. Para observar como se realiza detalladamente el proceso se recomienda revisar el archivo antes mencionado. Cuando la simulación se ejecuta se presenta una ventana con los gráficos en el dominio del tiempo y con la Densidad Espectral de Potencia respectivas. Por fines didácticos se modificó la función "sfunpsd.m" en el archivo "sfunpsd3.m", el bloque se representa resultante se muestra en la figura 2.22.

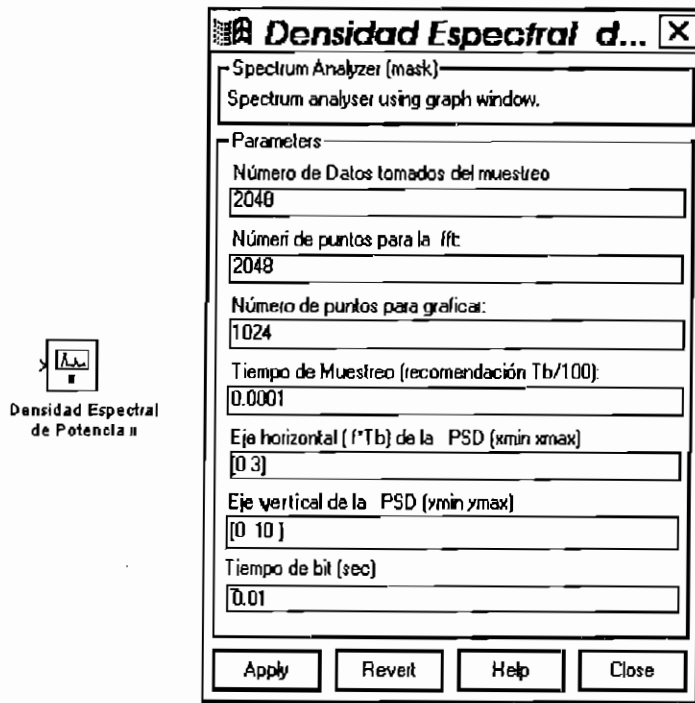


Figura 2.22 Bloque de Simulación y Plantilla de datos del bloque de Densidad Espectral de Potencia

Las modificaciones hechas al bloque son tal que el usuario puede manipular los ejes del gráfico de la PSD que se obtiene como resultado.

<sup>1</sup> Para mayor información de cómo funciona la *S-function* revisar el Capítulo 8 de la referencia [4] o el archivo sfuntempl.m que está bajo el directorio blocks del SIMULINK.

La respuesta dada por este bloque se da después de haber almacenado los suficientes datos, se sugiere que "número de datos del muestreo" y el "Número de puntos para la FFT" sean cercanos en un inicio, para evitar relleno de ceros en exceso.

En las secciones 3.3.2 y 4.7.2 se encuentran ejemplos de utilización de este bloque.

## **2.3 SIMULACIÓN DE UN CANAL DE TRANSMISIÓN CONTAMINADO CON RUIDO GAUSSIANO BLANCO.**

Un canal ideal de comunicación es aquel que transmite una señal sin ningún tipo de distorsión, desde luego por las características físicas de los medios de transmisión esto es imposible de conseguir, siempre habrá distorsión. Es de mucha utilidad la simulación de canales de transmisión que presenten características similares a las reales para ver como es afectada la información.

### **2.3.1 Transmisión sin Distorsión.**

Asumiendo que el canal de transmisión es un sistema lineal invariante en el tiempo se deben cumplir ciertos requisitos para la transmisión sin distorsión. La transmisión sin distorsión implica que a la salida del sistema se pueda tener una señal  $y(t)$ , que sea proporcional a la señal de entrada  $(x(t))$  con un cierto retardo, de la siguiente manera.

$$y(t) = A \cdot x(t - Td) \quad (2.24)$$

Donde  $A$  es la ganancia y  $T_d$  es el retraso, tomando la transformada de Fourier a ambos lados de la ecuación (2.24) se tiene:

$$Y(f) = A \cdot X(f) \cdot e^{-j2\pi f T_d} \quad (2.25)$$

Para una transmisión sin distorsión se requiere una función de transferencia del canal igual a:

$$H(f) = \frac{Y(f)}{X(f)} = A \cdot e^{-j2\pi f T_d} \quad (2.26)$$

Por lo tanto, para tener una transmisión sin distorsión la respuesta global del sistema debe tener una magnitud constante y su desplazamiento de fase debe ser lineal en frecuencia. No es suficiente que el sistema atenúe (o amplifique) por igual todas las componentes de frecuencia. Estas deben llegar con el mismo retardo para que se sumen correctamente, lo que demanda que el desplazamiento de fase sea proporcional a la frecuencia.

Matemáticamente estas dos condiciones se suelen resumir con las siguientes ecuaciones:

1. Magnitud de la respuesta de frecuencia constante

$$|H(f)| = \text{constant} = A \quad (2.27).$$

2. Desplazamiento de fase lineal en función de la frecuencia

$$\theta(f) = \angle H(f) = -2\pi f T_d \quad (2.28).$$

En la práctica una señal se puede distorsionar al pasar por ciertas partes del sistema; se puede introducir redes de corrección de amplitud o fase (igualación) en cualquier parte del sistema para corregir esta distorsión. La característica global del sistema es lo que determina su comportamiento.

### 2.3.2 Características del Filtro de un canal de transmisión.

Un sistema lineal invariante en el tiempo<sup>1</sup> (representado en la figura 2.23) actúa como filtro de las diferentes frecuencias aplicadas al sistema. Algunas componentes de frecuencia pueden amplificarse otras atenuarse y otras más permanecer inalteradas. Cada componente se puede desplazar en frecuencia, al pasar por el sistema.

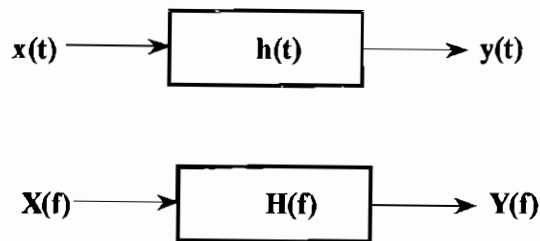


Figura 2.23 Representación de un sistema en el tiempo y en la frecuencia

Por la definición de transmisión sin distorsión, al incumplir con las condiciones (2.27) y (2.28) se tiene un canal no ideal que presenta distorsión, de allí que se puede comparar este canal no ideal con el comportamiento de un filtro pasabajos, cuya atenuación y frecuencia de corte muy baja distorsionan la señal transmitida a través de este filtro.

Para el tratamiento de señales de banda limitada se puede interpretar la segunda condición para transmisión sin distorsión como<sup>2</sup>:

$$-\frac{1}{2\pi} \frac{d\theta(f)}{df} = Tg \quad (2.29)$$

---

<sup>1</sup> Esta sección y la anterior son tomadas de la referencia [1] págs. 114 -115, 247-248 y [2] págs 92-94

<sup>2</sup> Según la referencia [2] pág. 247.

donde  $T_g$  es una constante denominada retardo de grupo, esta condición es menos estricta que en la ecuación (2.28), ya que si se cumple con (2.29) no necesariamente se cumple con (2.28), porque la integral de (2.29) es igual a la expresión (2.30):

$$\theta(f) = -2\pi f T_g + \theta_0 \quad (2.30)$$

donde  $\theta_0$  es una constante de fase, si  $\theta_0$  es diferente de cero la condición (2.28) no se cumple; es decir que para señales de banda limitada es suficiente que se cumpla con (2.29) para tener una transmisión sin distorsión. Por esta razón más tarde se ve que dando una constante de fase distinta de cero se incumple con (2.29) y ya se tiene una señal con distorsión. Al simular un canal con distorsión no es necesario que se presente un retardo  $T_d$  o  $T_g$  es suficiente con una constante de fase para tener distorsiones. Por estas razones un filtro pasabajos utilizado para modelar el canal puede o no tener retardo de fase como  $T_d$  o  $T_g$ , basta con una constante de fase distinta de cero.

En la práctica para simulación de canales de transmisión se utilizan los filtros pasabajos<sup>1</sup> e incluso pasabandas. Existen muchos otros modelos de los canales de transmisión dependiendo de la aplicación que se desee estudiar.

Se ha tomado como modelo de simulación de un canal de transmisión un filtro pasabajos Chebyshev tipo I<sup>2</sup>, principalmente por su

---

<sup>1</sup> Se encuentran varios métodos de simulación de canales, aquí como ejemplo un estudio para transmisión digital en un sistema móvil celular con TDMA (IS-54) donde se utiliza un filtro Chebyshev tipo I pasabajos para la simulación de un canal en la referencia [5].

<sup>2</sup> Se tomo esta decisión con ayuda de la referencia [6] y la nota anterior.

respuesta de frecuencia bastante aproximada a un filtro pasabajos real ya que presenta un rizado al inicio. Las características del filtro Chebyshev se encuentran en la referencia [3].

### 2.3.3 Ruido Blanco Gaussiano de banda limitada.

La simulación completa del canal de transmisión incluye ruido blanco de banda limitada por lo que resulta importante conocer algunas definiciones<sup>1</sup>.

Un proceso aleatorio  $x(t)$  se dice representar ruido blanco si la PSD es constante para todas las frecuencias así:  $S(f) = N_0/2$  donde  $N_0$  es una constante positiva.

La función de autocorrelación para un proceso de ruido blanco se obtiene a través de la transformada inversa de Fourier de la ecuación anterior

$$R(\tau) = \frac{N_0}{2} \delta(\tau) \quad (2.31)$$

Por ejemplo un proceso térmico donde  $N_0 = kT$  ( $T$ , temperatura  $k$ , constante de Boltzmann).

Esta definición de ruido blanco no puede usarse para describir ningún proceso físico<sup>2</sup> porque implica una cantidad infinita de potencia, debido a que las mediciones se restringen a anchos de banda finitos, lo que en realidad interesa es el ruido blanco de banda limitada. En otras

---

<sup>1</sup> Tomado de [2] págs 518,533-534 y [1] sección 4.7

<sup>2</sup> Tomado de [1] pág. 189.



palabras si una señal de ruido tiene una densidad espectral de potencia plana, que se extiende más allá del ancho de banda de un sistema dado, el sistema percibe el ruido como si fuera efectivamente blanco.

Para ruido blanco de banda limitada, la potencia del ruido es independiente de la elección de la frecuencia de operación, por esta razón se eligió el ruido blanco de banda limitada para la simulación de una canal con ruido aditivo.

#### 2.3.4 Bloque de Simulación que permite el análisis del efecto de un canal con ruido Blanco Gaussiano.

El modelo del canal comprende un filtro pasabajos Chebyshev tipo I, de un orden alto (de orden 8)<sup>1</sup>, se le añade ruido blanco de banda limitada con el fin de tener una señal de salida distorsionada por el medio de transmisión y el ruido. El ruido tiene características de distribución normal o Gaussiana.

El bloque desarrollado para la simulación del canal se encuentra en el gráfico 2.24. Los datos a ingresar son; frecuencia de corte del canal, la potencia del ruido, tiempo de bit y atenuación deseada.

---

<sup>1</sup> Para una revisión completa del diseño de filtros digitales se recomienda la referencia [3] donde encuentra ampliamente desarrollado este tema.

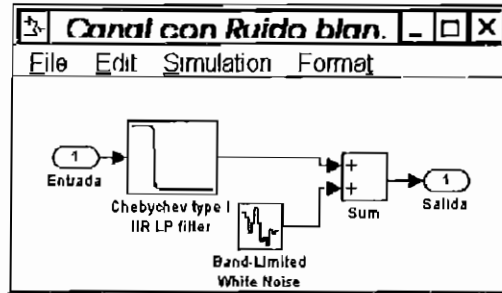


Figura 2.24 Bloque de simulación de un canal contaminado con ruido blanco

Se tiene un ejemplo de simulación de un código NRZ polar afectado por el ruido del canal de en la siguiente sección conjuntamente con el estudio del diagrama ocular.

## 2.4 DIAGRAMA DEL OJO.

El diagrama del ojo es un sistema práctico de analizar la interferencia intersímbolos (ISI<sup>1</sup>) en una señal recibida.

Este diagrama se observa con un osciloscopio, poniendo la señal en el eje vertical, y en el eje horizontal un barrido de frecuencia  $1/T_b$ , donde  $T_b$  es el tiempo de bit recibido.

En la figura 2.25 a se representa una secuencia de pulsos binarios sin interferencia entre símbolos y el diagrama del ojo que se vería en la pantalla del osciloscopio, después de una larga serie de pulsos.

<sup>1</sup> Se conoce como Interferencia Intersímbolo (ISI) la superposición parcial de un bit con otro adyacente en el tiempo.

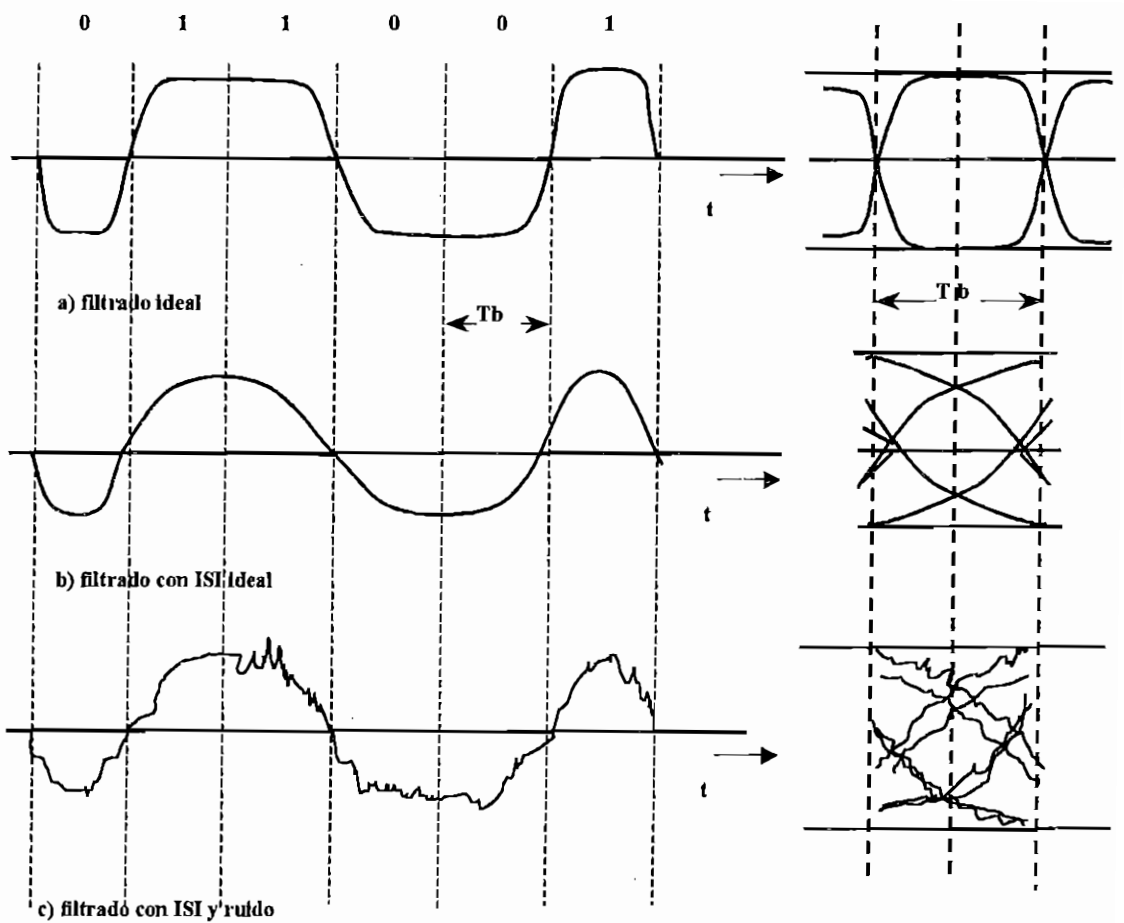


Figura 2.25 Señal binaria con su respectivo Diagrama del Ojo.

Si hubiera interferencia entre símbolos, los trazos en el diagrama del ojo ya no se superponen, como puede apreciarse en la figura 2.25b; sin embargo, si consideramos el efecto del ruido como en un caso real la señal se vería como en la figura 2.25c.

Del diagrama del ojo se puede extraer mucha información tanto de las características del sistema transmisor, como del mejor método de detección.

En la figura 2.26 se ha representado un modelo simplificado del diagrama del ojo donde con letras se ha señalado los puntos más significativos:

- A. Margen contra errores en el muestreo.
- B. Margen de defensa contra el ruido.
- C. Distorsión en los cruces por cero.
- D. Sensibilidad al error en el instante de muestreo.
- E. Distorsión en el instante del muestreo.
- F. Instante de Muestreo óptimo.

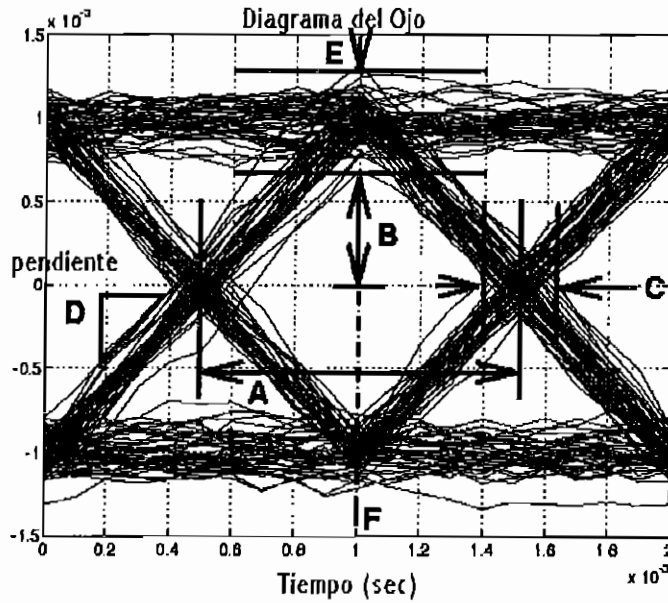


Figura 2.26 Interpretación del Diagrama del ojo.

#### 2.4.1 Simulación para la obtención del diagrama del Ojo.

El proceso de simulación implica tomar los datos binarios y graficarlos en un mismo intervalo de tiempo sin borrar los trazos del símbolo anterior con el actual. Para cumplir con este objetivo se desarrolló el bloque denominado Diagrama del ojo a partir del bloque *S-function* y el archivo creado con el nombre de "sfunxyjp". Los datos de entrada a la programación de este bloque son dos: los datos propiamente y una señal diente de sierra cuya pendiente es igual a dos tiempos de bit. El usuario puede modificar los ejes y el valor del tiempo de bit como se muestra en la figura 2.27b.

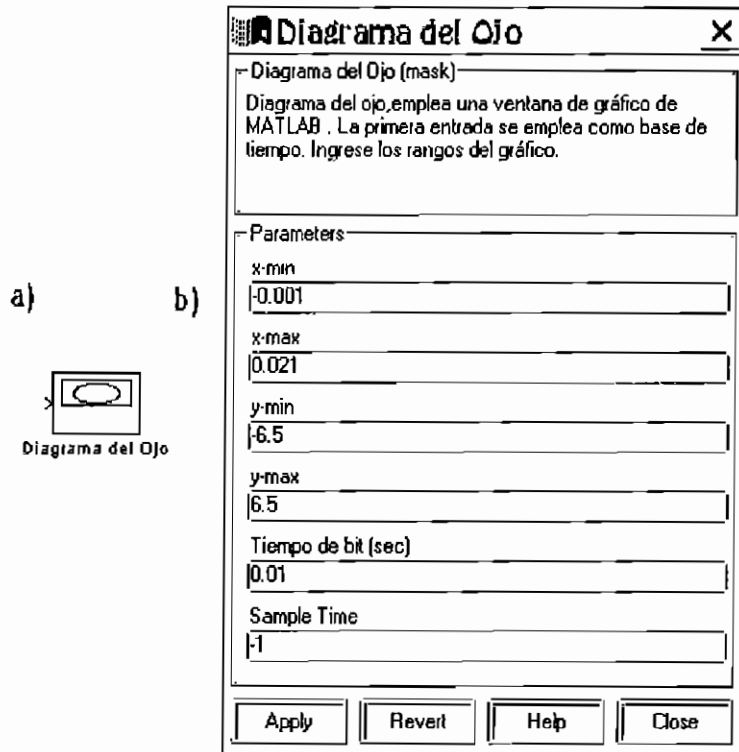


Figura 2.27 Bloque de Simulación del Diagrama del ojo y su plantilla de datos.

#### 2.4.2 Ejemplo de Simulación del diagrama ocular y de un canal ruidoso.

Este ejemplo se desarrolla en dos partes: en la primera parte una señal binaria bipolar será contaminada con ruido y luego al modelo se añadirá el bloque diseñado para la obtención del diagrama del ojo.

No se entrará en detalles acerca de los parámetros generales de simulación, simplemente se desea mostrar el resultado proporcionado por los bloques de simulación del canal de transmisión y el bloque del diagrama del ojo.

En la figura 2.29a, se tienen los bloques necesarios para obtener la señal binaria bipolar, el bloque de simulación del canal y un conjunto de bloques que permiten la visualización de los resultados. Los bloques necesarios para la señal binaria bipolar son el bloque de "Datos digitales secuenciales aleatorios" de la librería "fuentes" y el bloque "NRZ polar" de la librería "codecs"(del cual se tratará en la sección 3.2.2), en estos bloques se define el número de datos aleatorios deseados y también el tiempo de duración de cada símbolo con su magnitud de salida. En el bloque canal se especifica la frecuencia de corte, la potencia de ruido, la atenuación y el tiempo de muestreo de este bloque. Finalmente el bloque *Mux* de la librería *connections* ,el bloque *constant* de la librería *source* y el bloque *sum* de la librería *linear* se utilizan para ver los resultados en el bloque *scope* de la librería *sinks*.

Se eligió un tiempo de símbolo de 0.01, una frecuencia de corte del filtro de 80 Hz y una potencia muy baja de ruido con atenuación de uno; y, los resultados pueden observarse en la figura 2.29b. La señal superior corresponde a los datos digitales y la inferior corresponde a la señal afectada por efecto del canal.

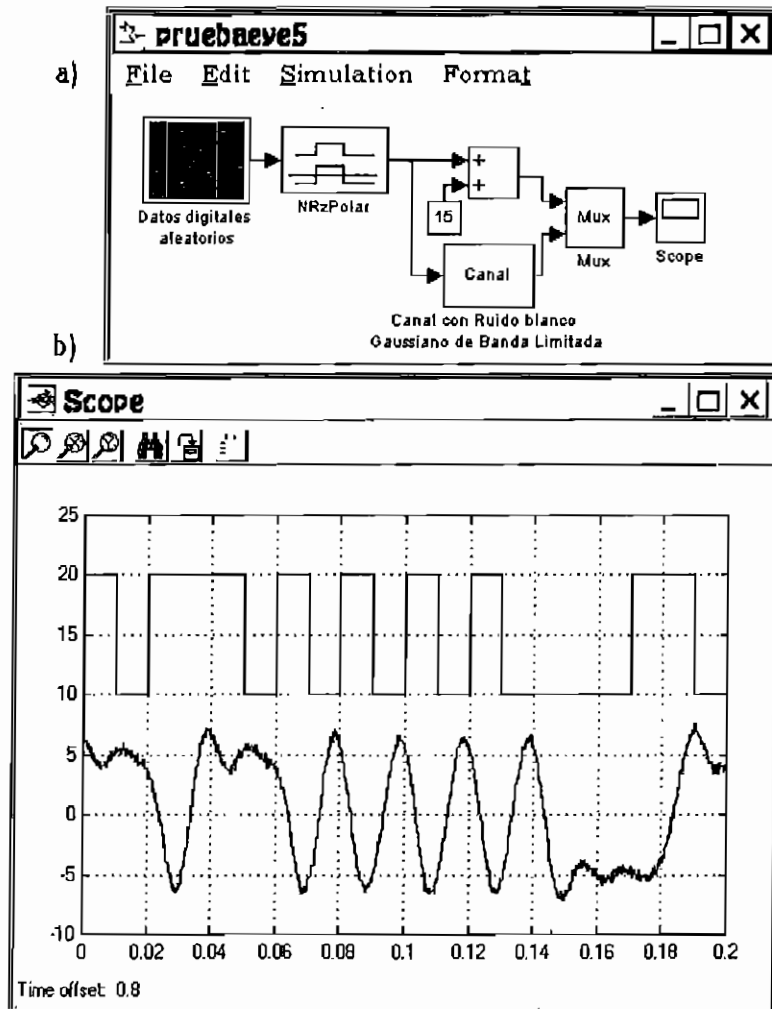


Figura 2.29 Ejemplo de Simulación de canal con ruido: a) Modelo a simular y b) Resultado de la simulación

El segundo ejemplo es mucho más simple, se dejan únicamente los bloques que entregan la señal binaria bipolar y el bloque del canal de transmisión añadiendo únicamente el bloque "Diagrama del Ojo", especificando en éste último el tiempo de duración del símbolo, los márgenes de los ejes que se desean tener. El modelo construido se muestra en la figura 2.30a y el resultado se encuentra en la parte b del mismo gráfico. La calidad del diagrama del ojo, se diseñó para que dependa del tiempo de muestreo que el usuario puede modificar, si en lugar de un valor positivo se pone -1 se tendrá la máxima eficiencia del

bloque dependiendo del tiempo de muestreo propio del bloque de donde provenga la señal.

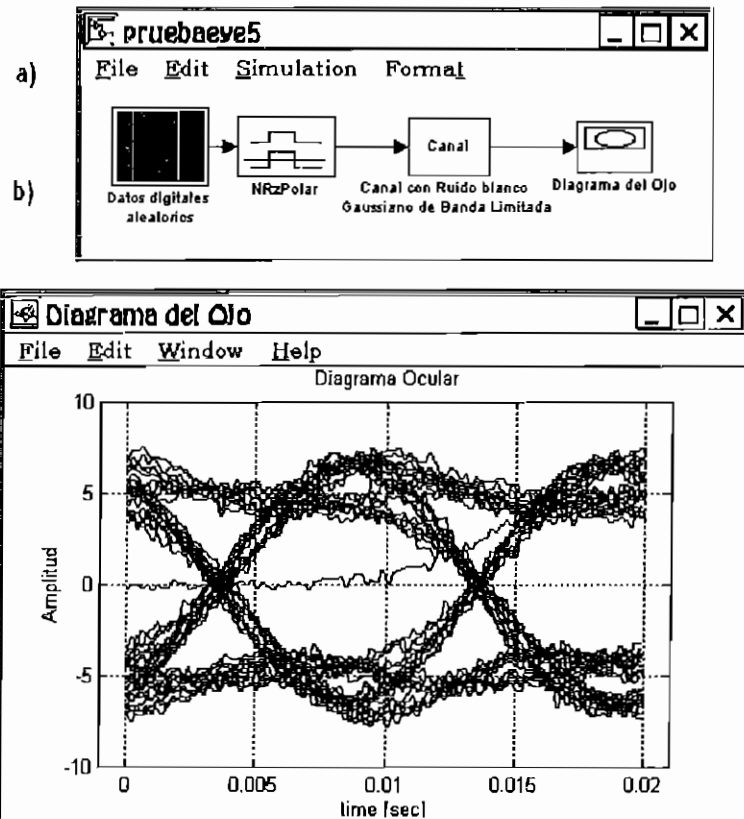


Figura 2.30 Ejemplo de Simulación del Bloque Diagrama del Ojo: a) Modelo a simular b) Resultado de la simulación

## 2.5 REFERENCIAS.

1. Strembler F.G, Introducción a los Sistemas de Comunicación, Addison Wesley, México 1993
2. Leon W. Couch II, Digital and Analog Communication Systems, four edition, Macmillan Publishing Company New York 1993.
3. Oppenheim, AV. And R.W. Schafer. Discrete Time Signal Processing. Englood Cliffs, NJ: Prentice Hall, 1989.
4. Math Works INC. "Using SIMULINK®", 1996.



5. Matolak David y Wilson Stephen, "Detection for a Statically Known, Time - Varing Dispersive Channel", IEEE Transactions on Comunications Vol 44 No.12 December 1996.
6. O.M. Zeytinoglu & N.W Ma, "Communication Systems II ELE 045", Communication Toolbox and Manual, Departament of Electrical and Computer Engineering , Ryerson Polytechnic University, Canada,1994.
7. James B. Dabney y Thomas L. Harman. "The Student Edition of SIMULINK® User's Guide", Prentice Hall, 1997
8. Coughlin R. y Driscoll F. Amplificadores Operacionales y Circuitos Integrados Lineales, Ed. Prentice Hall, 1993.

## CAPÍTULO 3. SIMULACIÓN DE CÓDIGOS DE LÍNEA.

### 3.1 CÓDIGOS DE LÍNEA.

En transmisión digital se emplean varios métodos para adaptar la señal de información a un determinado medio físico (canal de transmisión), con el objetivo de lograr una transmisión eficiente, uno de estos métodos constituyen los denominados Códigos de Línea para transmisión en banda base<sup>1</sup>.

Según la UIT-T<sup>2</sup>, un código de línea es un "Código elegido de modo que convenga a las características de un canal y que define la equivalencia entre un conjunto de dígitos presentados para su transmisión y la correspondiente secuencia de elementos de señal transmitidos por ese canal"<sup>3</sup>. Es necesario indicar que en transmisión digital, un dígito puede estar representado por un elemento de señal, caracterizado por su naturaleza dinámica, su estado discreto y su posición discreta en el tiempo; por ejemplo, por un impulso de amplitud y duración especificadas.

Los códigos de Línea deben cumplir con las siguientes condiciones:

- a) Ningún código de línea debe imponer alguna restricción al mensaje a transmitirse, esta propiedad se denomina "Transparencia".
- b) La decodificación del mensaje debe ser única.

---

<sup>1</sup> Cuando la información digital se representa con pulsos digitales, se dice que la transmisión se realiza en Banda Base.

<sup>2</sup> UIT-T es el sector de Normalización de las Telecomunicaciones de la UIT (Unión Internacional de las Telecomunicaciones)

<sup>3</sup> Recomendación UIT-T G.701, revisada por la Comisión de Estudio XV (1988-1993) del UIT-T, fue aprobada por la CMNT (Helsinki, 1-12 de marzo de 1993).

- c) Cada símbolo debe contribuir para la transmisión del mensaje, esto se traduce en una adecuada eficiencia de codificación.
- d) Un código de línea debe procurar una baja tasa de error.

Además de estas condiciones, en los códigos de línea se procura encontrar las siguientes propiedades:

- a) El ancho de banda de transmisión debe ser lo más pequeño posible.
- b) Con un determinado ancho de Banda y su correspondiente potencia de transmisión, el código debe tener la suficiente inmunidad contra el ruido y la interferencia Intersímbolo (ISI).
- c) Todo canal físico puede causar errores en la transmisión, en la recepción de datos sería una ventaja que el código de línea pueda detectar que existieron errores con respecto a la señal de origen.
- d) Densidad Espectral de Potencia favorable. La presencia de transformadores y capacitores en la mayoría de los acoplamientos de los sistemas de transmisión, por lo general implica que se deba disminuir las componentes que se ubican cerca de la frecuencia cero.
- e) Para una adecuada recuperación de la información en la recepción es necesaria una buena sincronización; un buen código de línea debe permitir una fácil recuperación de sincronismo.

En el momento de elegir un código de línea se deben tomar en cuenta todas estas propiedades, por lo que es muy importante el análisis espectral de los códigos de Línea; esta es la razón por la que se ha creado esta herramienta didáctica, con el fin de aplicar las reglas de codificación e ilustrar la forma de la Densidad Espectral de Potencia de los códigos más utilizados.

### 3.2 ALGORITMOS DE CODIFICACIÓN Y BLOQUES DE SIMULACIÓN.

En cuanto a la simulación de los codificadores y decodificadores de Línea, se han creado dos librerías denominadas "codecs" y "decods", donde se encuentran los bloques de simulación de los codificadores y los decodificadores respectivamente. Es oportuno para una buena utilización de estas librerías, tener presente para cada código de Línea, sus respectivas reglas de codificación y Densidad Espectral de Potencia. Las ecuaciones y gráficas de las Densidades Espectrales de Potencia, se obtuvieron de las referencias [1], [2], [4] y [5]. Adicionalmente para cada código de línea, se presentará detalladamente las características que deben tomarse en cuenta para la utilización de los bloques de simulación y una descripción general del proceso, tanto para la codificación como para la decodificación.<sup>1</sup>

Por cada código de Línea se realizó un ejemplo con el propósito de ayudar al usuario a familiarizarse con la herramienta desarrollada y también con el fin de tener a mano una sección de información, donde conste una breve descripción del código estudiado.

Las librerías antes mencionadas se presentan al usuario al momento de invocar los comandos "codecs" y "decods". Los bloques de simulación desarrollados están listos para su utilización, tal como se muestra en la figura 3.1 y 3.2.

---

<sup>1</sup> Para mayor detalle acerca de los bloques de simulación, se recomienda revisar cada bloque bajo la opción del Menú *Edit*, *Look Under Mask* o su respectivo archivo *.m* si se trata de un bloque con *S-Function*.

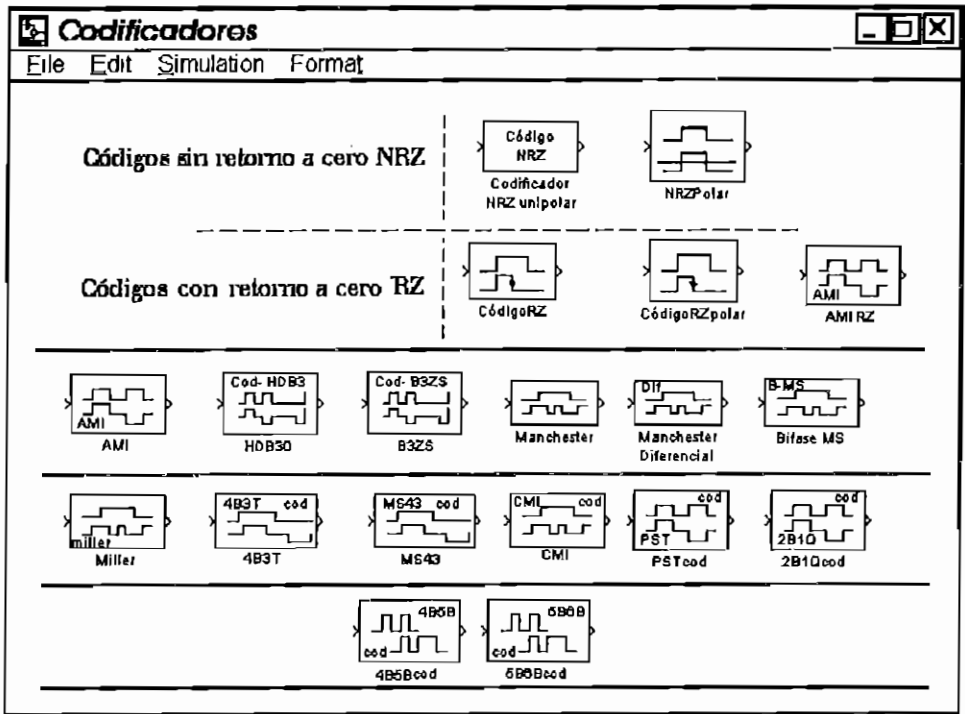


Figura 3.1 Librería codecs.

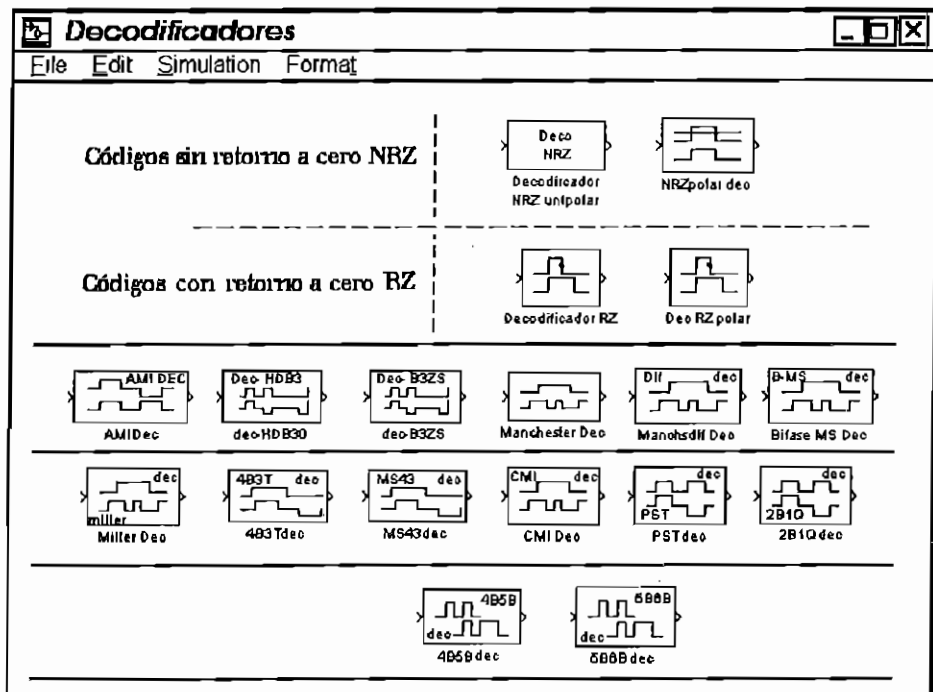


Figura 3.2 Librería decods.

Cada uno de estos bloques ejecuta las reglas de codificación correspondiente a cada código de línea. Para simular un código se necesita de fuentes de datos digitales, las cuales fueron descritas en el capítulo 2. Además para visualizar los resultados de la simulación se necesita utilizar el bloque *scope* de la librería *sinks*. En la sección 3.3 se tiene un ejemplo completo del uso de los codificadores y decodificadores conjuntamente con el análisis espectral y los efectos de un canal de transmisión.

Otro acceso a esta librería es mediante la instrucción "comdig" en la línea de comandos, la cual permite obtener un ambiente gráfico para revisar las librerías y ejemplos de una manera más sencilla para el usuario.

Para describir los bloques de simulación, no es necesario profundizar en el desarrollo de las funciones de Densidad Espectral de Potencia de estos códigos, debido a que existe suficiente material de consulta referente a este tema.<sup>1</sup> Los resultados del análisis de la Densidad Espectral de Potencia se enuncian como una función de la frecuencia en Hz con la notación  $S(f)$ ; donde se incluyen valores como la amplitud de la señal ( $A$ ), tiempo de bit ( $T$ ), función de impulso unitario ( $\delta(f)$ ) y el valor de probabilidad de los unos y ceros, denotados como  $p$  y  $q$  respectivamente, debido a que se tratan de señales de información, que tienen como datos el uno y el cero.

Antes de analizar los procesos de codificación y decodificación, conviene recordar que cada dato proviene de un bloque denominado fuente de datos digitales (con un nivel de 1 para el uno lógico y 0 el cero

---

<sup>1</sup> Tesis: "Diseño y Construcción de un Codec Didáctico Para transmisión en Banda Base ", de Nelson Avila, EPN. 1994

lógico). Estos datos son ingresados a cada bloque de simulación con sus respectivas características de amplitud y duración de tiempo (denominado también tiempo de bit).

### 3.2.1 Código NRZ Unipolar (*Non Return to Zero*) .

Es el más sencillo de los códigos de Línea, a cada símbolo correspondiente a un cero lógico ( $0_L$ ) y uno lógico ( $1_L$ ) se le asigna los niveles de 0 y A respectivamente, a esta asignación se denomina lógica positiva y lógica negativa se tiene cuando se asignan, de la misma manera, los niveles 0 y -A. Un ejemplo de codificación NRZ con lógica positiva se muestra en la figura 3.3, donde se representa una señal de datos y su correspondiente codificación.

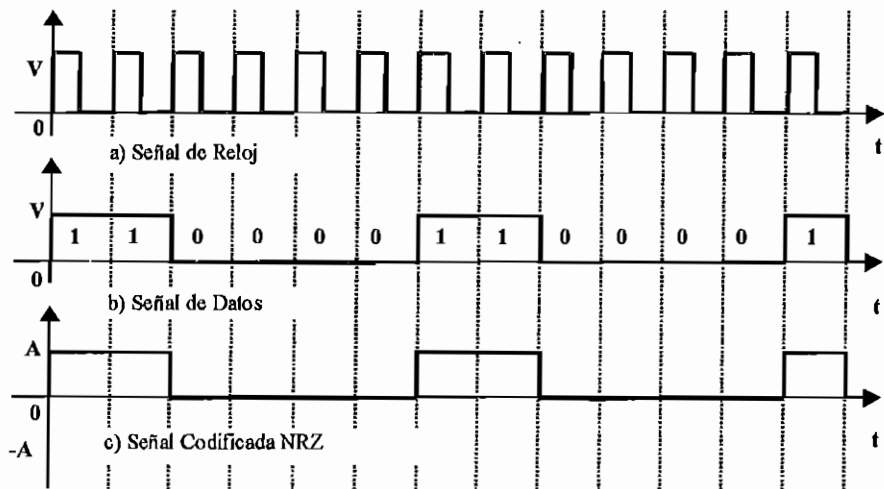


Figura 3.3 Ejemplo de codificación NRZ

Para el análisis espectral de la señal NRZ, se considera la probabilidad con la que ocurren los unos y ceros, así:

$p$  = probabilidad de tener un  $0_L$ ,

$q$  = probabilidad de tener un  $1_L$ , evidentemente  $p + q = 1$

El cálculo de la Densidad espectral de potencia ( $S(f)$ ) requiere de un análisis específico<sup>1</sup>. Para la señal NRZ el resultado se muestra en la ecuación 3.1:

$$S(f) = A^2 T p q \left( \frac{\text{sen}(\pi f T b)}{\pi f T b} \right)^2 + A^2 p^2 \left( \frac{\text{sen}(\pi f T b)}{\pi f T b} \right)^2 \sum_i \delta \left( f - \frac{i}{T b} \right) \quad (3.1)$$

donde :

A es la amplitud del pulso.

f es la frecuencia en general.

Tb es el tiempo de bit de cada dato y no tiene relación con f.

$\delta(f)$  es la función impulso o Delta de Dirac.

La representación de la densidad espectral de potencia de la ecuación (3.1) se encuentra en la figura 3.4.

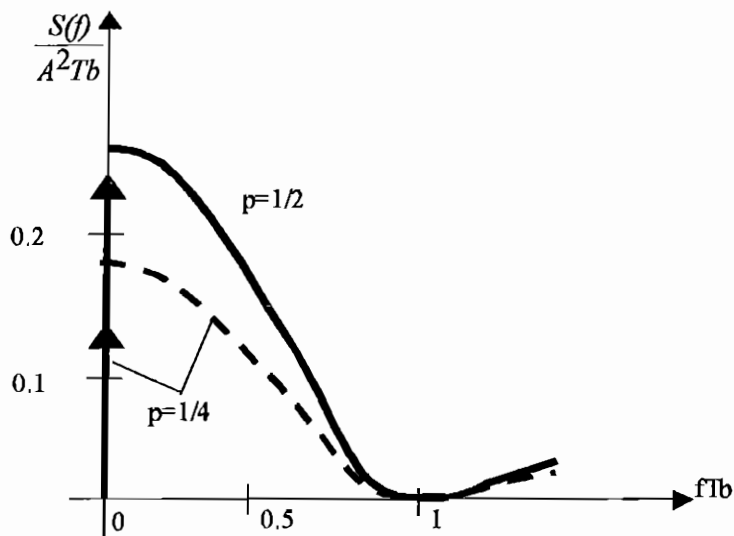


Figura 3.4 Densidad espectral de potencia para el código NRZ unipolar

<sup>1</sup> Ver las referencias [1],[2] y [6].



## Simulación del Código NRZ unipolar.

Una vez conocida la regla de codificación del código NRZ, se puede iniciar el proceso de simulación, para lo cual se diseñan dos bloques, uno para la codificación y otro para la decodificación, además cada bloque tiene sus respectivas plantillas de ingreso de datos como se representa en la figura 3.5.

La señal de entrada ingresa al bloque de simulación, se procesa y la respuesta es enviada al sistema de simulación por el puerto de salida. En las plantillas de datos se ingresan valores que permiten al usuario definir tanto el tiempo de duración del bit como la lógica de la codificación.

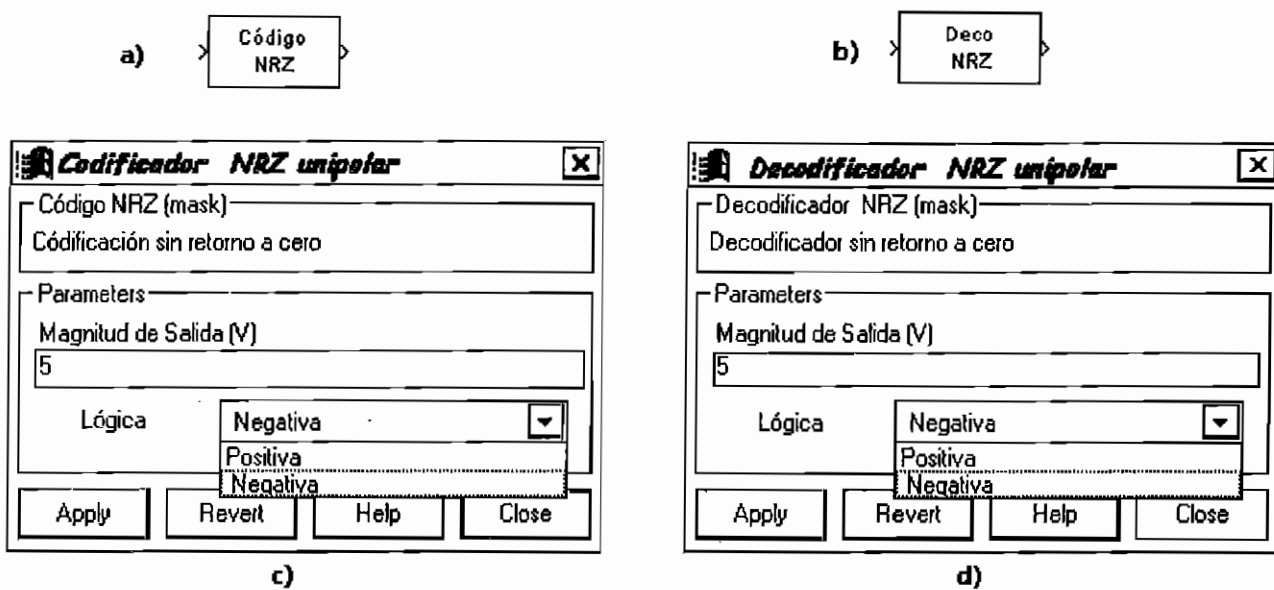


Figura 3.5 a) Bloque de Simulación del Codificador, b) Bloque de Simulación del Decodificador, c) Plantillas de Datos del Codificador y d) Plantillas de Datos del Decodificador

### a) Codificación.

El dato proveniente de la fuente digital, es ingresado al bloque codificador y es amplificado con el valor de la variable "Magnitud de Salida" de la plantilla de datos (ver figura 3.5 c), para cumplir con este objetivo se utiliza un bloque controlador de ganancia (librería "linear" del

*SIMULINK*). Además se incluye la opción de escoger si el uno lógico tiene un valor positivo o negativo de la magnitud de salida; el usuario puede modificar este parámetro mediante el menú "Lógica" (ver figura 3.5 c). Mediante una técnica que permite crear plantillas de datos<sup>1</sup> se ha modificado el bloque controlador de ganancia, esta técnica se empleará en todos los bloques desarrollados en esta tesis.

## **b) Decodificación.**

La decodificación es muy similar a la codificación, ya que también utiliza el bloque de control de ganancia, pero con el factor de amplitud correspondiente. Es decir que si se amplifica el dato con la magnitud de  $+A$  (ó  $-A$ ), en la decodificación debe multiplicarse esta señal por el factor  $1/A$  (ó  $-1/A$ ).

Es pertinente indicar que para la simulación de la decodificación de este código, tanto la magnitud como el tipo de lógica, deben ser los mismos para los bloques de codificación y decodificación.

### **3.2.2 Código NRZ polar (*Non Return to Zero*).**

Este código de línea es similar al NRZ unipolar, al símbolo uno lógico se le asigna el nivel de  $+A$  y para el cero lógico  $-A$ . En la figura 3.6 se encuentra un ejemplo de codificación.

---

<sup>1</sup> Esta técnica se denomina *Masking* y se encuentra descrita en la referencia [3].

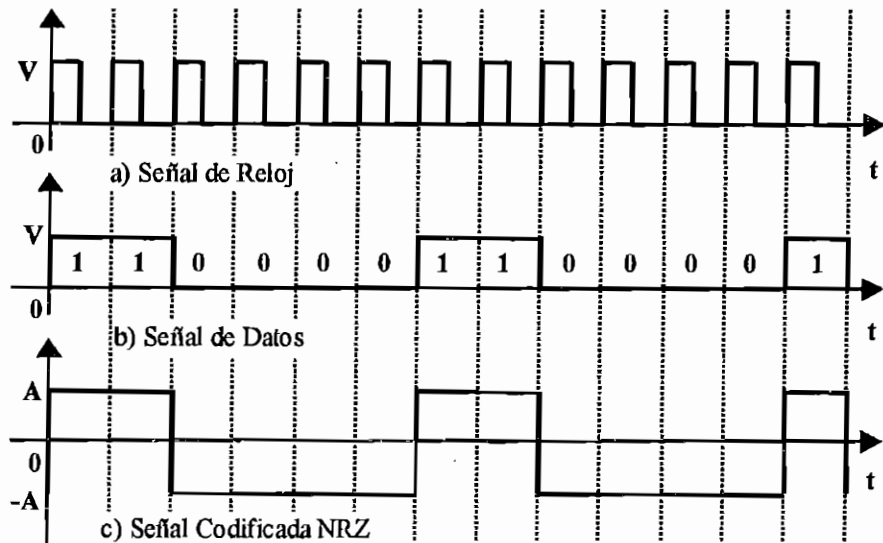


Figura 3.6 Ejemplo de Codificación NRZ polar (uno +A y cero -A)

Este código se tiene una densidad espectral de potencia, dada por la ecuación (3.2) y representada en la figura 3.7.

$$S(f) = 4A^2 T p q \left( \frac{\text{sen}(\pi f T b)}{\pi f T b} \right)^2 + A^2 (2p - 1)^2 \left( \frac{\text{sen}(\pi f T b)}{\pi f T b} \right)^2 \sum_i \delta \left( f - \frac{i}{T b} \right) \quad (3.2)$$

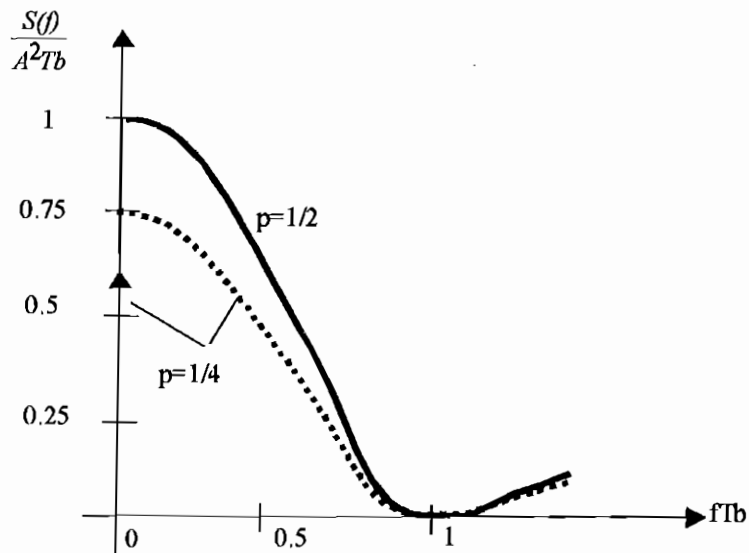


Fig. 3.7 Densidad espectral de potencia para el código NRZ polar

## **Simulación NRZ polar.**

Varios bloques conforman el codificador y el decodificador NRZ polar mediante la agrupación de subsistemas de la misma forma que se describió al final de la sección 1.2.9.

### **a) Codificación.**

Para la codificación el usuario define un valor de amplitud de la señal de salida mediante la opción "Magnitud de señal de salida" (figura 3.8 b). La señal proviene de la fuente digital con niveles uno y cero e ingresa al bloque codificador. La señal de datos es multiplicada por un factor, a este resultado se le resta un nivel para conseguir una señal polar como se indica en la figura 3.7 c.

El subsistema diseñado del bloque del codificador se muestra en la figura 3.8b y puede modificarse utilizando la opción del *Menú Edit, Look Under Mask*. En esta figura se muestran los bloques que realizan el proceso indicado en el inicio, se tiene un amplificador de ganancia  $2 \cdot k_1$  (donde  $k_1$  es el valor de la amplitud deseada de la codificación), y un sumador que mediante el bloque *sum* de la librería *linear* suma esta señal con la señal proveniente de la fuente constante de valor  $k_1$ . Adicionalmente en el bloque de simulación (figura 3.8 c) el usuario tiene la opción de elegir utilizar lógica positiva o lógica negativa para representar sus datos mediante la variable "logica".

## b) Decodificación.

La decodificación consiste en comparar la señal codificada con un nivel cero, si la señal es mayor que cero se interpretará como uno y si es menor cero será un cero.

En la figura 3.9 b se muestra el proceso de decodificación donde la señal codificada ingresa al subsistema y luego se amplifica con la constante de ganancia  $k_1$  que es 1 si la lógica es positiva y -1 si es negativa; luego se utiliza el bloque *relational operator* (librería *nonlinear* del *SIMULINK*), en el cual se compara con cero el valor de la señal codificada multiplicada por el factor  $k_1$ . El bloque *relational operator* tiene una salida de 1 si la señal de entrada es mayor que la constante 0 y entrega un valor de cero en el caso opuesto. El valor de la constante  $k_1$  hace que este diseño pueda decodificar tanto lógica positiva como negativa. El bloque de decodificación incluye la opción de cambiar la magnitud de salida decodificada (constante  $k_2$  en la figura 3.9 b). Es necesario que el codificador y el decodificador tengan la misma "Lógica" en la simulación.

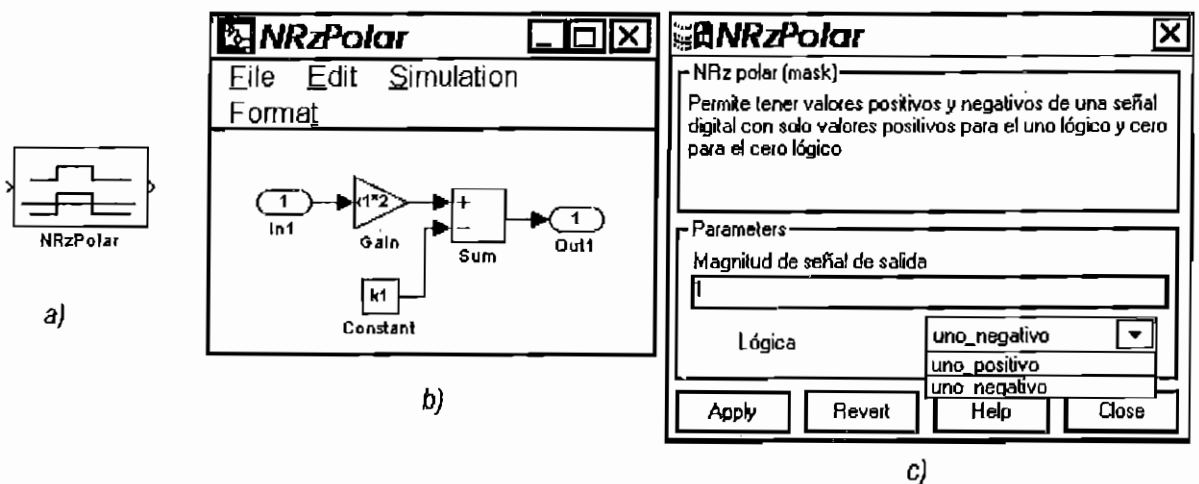


Figura 3.8 Codificador NRZ polar: a) bloque de simulación, b) subsistema contenido y c) Plantilla de datos

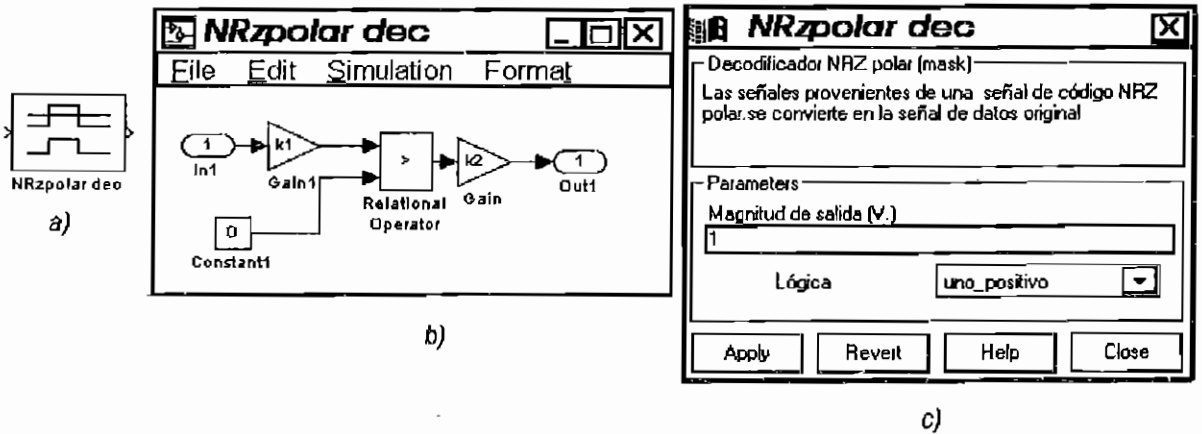


Figura 3.9 Decodificador NRZ polar: a) bloque de simulación, b) subsistema contenido y c) Plantilla de datos

### 3.2.3 Código RZ unipolar (*Return to Zero*).

Este código es denominado con retorno a cero, porque cuando se tiene uno lógico, durante un cierto porcentaje del tiempo de bit regresa nuevamente a nivel 0 la señal. En la figura 3.10 se presenta un ejemplo de codificación RZ. La relación del porcentaje de tiempo de retorno a cero se denota aquí como  $r = a \cdot 100 / T_b$ , donde  $a$  es el tiempo antes de retornar a cero y  $T_b$  es el tiempo de bit.

La referencia [2], proporciona la Densidad Espectral de Potencia  $S(f)$  de este código y está representada en la siguiente ecuación:

$$S(f) = \frac{A^2 \cdot T}{4} \left[ \frac{\sin\left(\frac{\pi f T_b}{2}\right)}{\frac{\pi f T_b}{2}} \right]^2 \left[ p \cdot q + \frac{p^2}{T_b} \sum_{i=-\infty}^{\infty} \delta\left(f - \frac{i}{T_b}\right) \right] \quad (3.3)$$

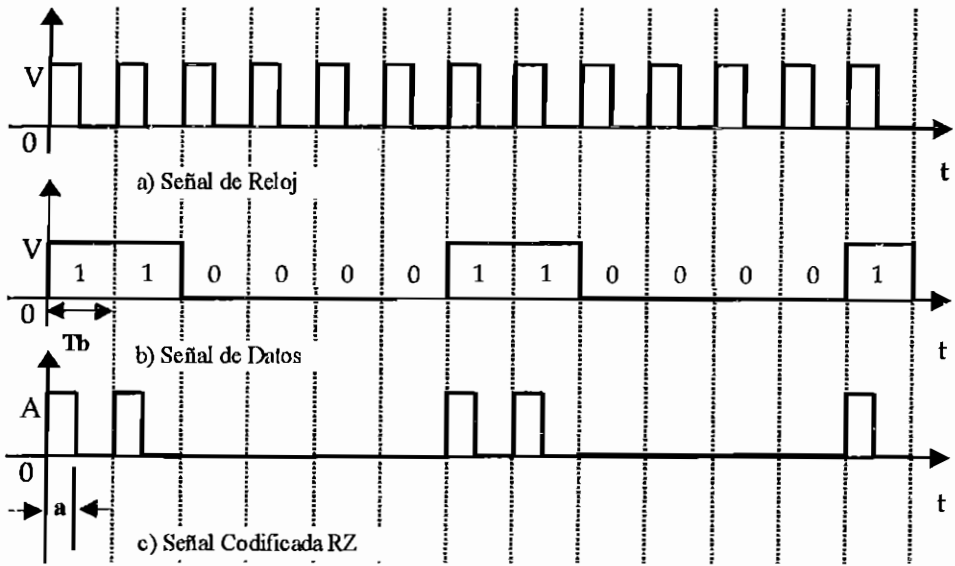


Figura 3.10 Ejemplo de Codificación RZ unipolar

En el gráfico 3.11, se tiene la representación de la ecuación (3.3), puede observarse que el primer anulamiento de la Densidad Espectral de Potencia ocurre a dos tiempos de bit y no a uno como el caso del código NRZ. Además se tienen pulsos discretos de área o peso 0.25 y 0.1 a las frecuencias 0 y  $1/T_b$  respectivamente.

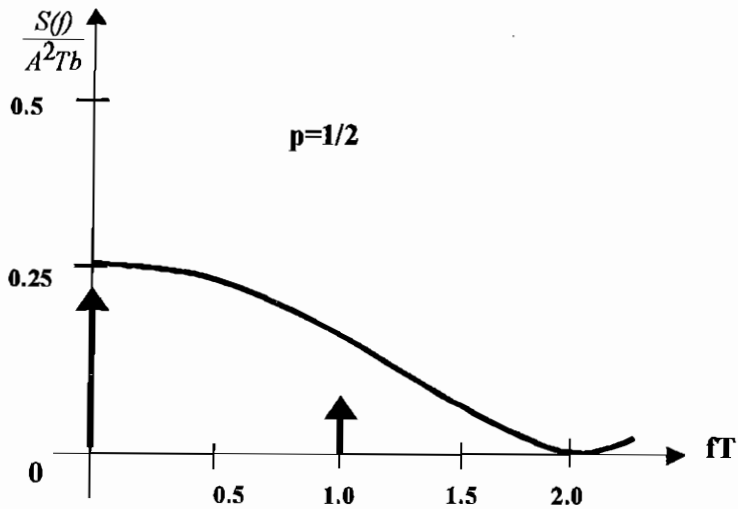


Fig. 3.11 Densidad espectral de potencia para el código RZ unipolar

## Simulación del Código RZ unipolar.

En esta simulación se utiliza la fuente de datos con reloj digital de la sección 2.1.1.1, de otra manera la fuente con el reloj normal produce un ligero desfase haciendo que se pierdan algunos datos.

### a) Codificación.

Para obtener la señal RZ se controla, mediante los datos ingresados, la señal de un generador de onda cuadrada de ancho de pulso variable; si el dato es uno pasa la señal de ancho de pulso variable y si el dato es cero pasa un nivel cero de señal. El ancho del pulso esta dado por la razón de retorno a cero.

Entonces para la codificación se tiene un generador de pulsos (librería *source* del *SIMULINK*) en el que se controla tanto el ancho de los pulsos como su periodicidad y además se dispone de un bloque interruptor (librería *nonlinear* del *SIMULINK*) que se utiliza en el bloque codificador desarrollado para conmutar entre la señal del generador y una constante de valor cero, dependiendo de la señal proveniente de los datos.

Como se ve en la figura 3.12 b, el subsistema desarrollado para la codificación RZ tiene como entrada la señal de datos, la cual controla el bloque del *switch*<sup>1</sup>, conmutando entre la señal del *pulse generator* (generador de pulsos) y la constante 0. El retorno a cero se da en virtud de que se puede controlar el ancho deseado del pulso en el bloque generador de pulsos, con el valor del porcentaje de la "razón de retorno a

---

<sup>1</sup> La conmutación del bloque switch es así: señal de control mayor que cero pasa el nivel superior, señal de control menor o igual que cero pasa la señal inferior.



zero"; además se puede controlar la amplitud del pulso y el período. En la parte c del gráfico 3.12 se tiene la plantilla de datos. El dato más importante de esta plantilla es la "razón de retorno a cero", la cual debe ingresarse como porcentaje del tiempo de bit. No se recomienda un valor menor que el 6 %, ya que se necesita de una mayor definición de puntos para tener un resultado gráfico adecuado.

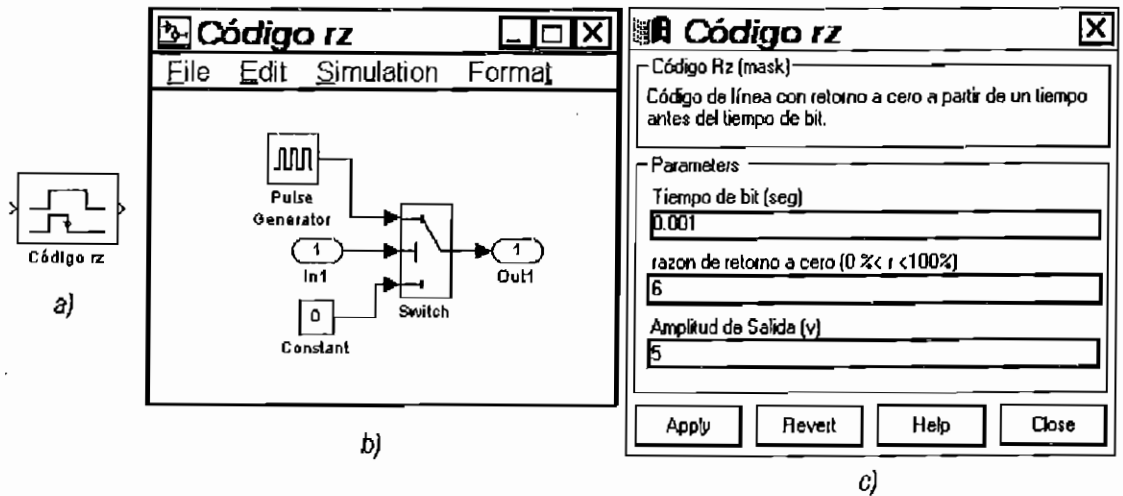


Figura 3.12 Codificación RZ unipolar: a) Bloque de simulación b) Contenido c) Plantilla de datos

## b) Decodificación.

Se realiza la recuperación de datos mediante un oscilador controlado por voltaje que sirve como señal de reloj a un bloque que se diseñó para que cumpla la función de un flip-flop tipo D<sup>1</sup>. En la figura 3.13 b se tiene el subsistema de decodificación y en la parte c se tiene la plantilla de datos en la cual es necesario especificar la magnitud de entrada para el control de la magnitud de salida.

<sup>1</sup> Para más informes sobre flip-flops Ver Tocci R., Sistemas Digitales, ed Prentice Hall 1993.

La señal a decodificar ingresa al flip-flop y a un bloque de toma de decisiones, este bloque entrega un valor de uno si la señal cumple con un mínimo, luego este resultado va a un comparador y se tiene el voltaje necesario para iniciar la oscilación del VCO a la frecuencia del tiempo de bit, hasta que esto no ocurra la oscilación es de una frecuencia muy pequeña (el inverso de 10 veces el tiempo de bit). Esta señal de reloj controla el flip-flop tipo D, que responde al flanco positivo de su respectiva señal de reloj.

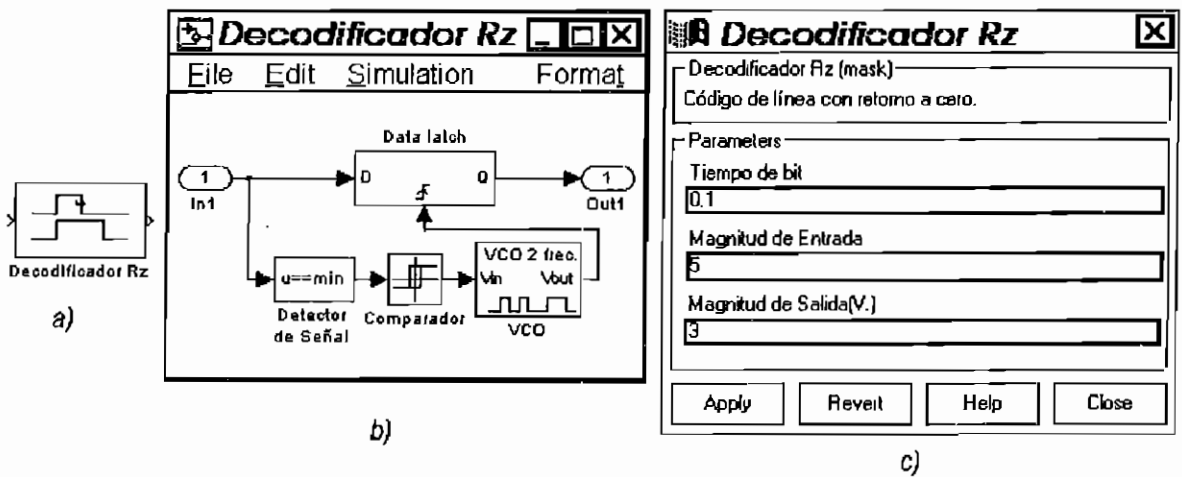


Figura 3.13 Decodificación RZ: a) bloque de simulación b) Subsistema c) Plantilla de datos

### 3.2.4 Código RZ polar (Return to Zero).

Básicamente es similar al código RZ unipolar, con la diferencia que para el 0, se tiene un nivel  $-A$  por el mismo porcentaje de tiempo hasta retornar a cero, tal como se muestra en la figura 3.14. Ahí se puede observar los valores del tiempo de bit  $T_b$  y el tiempo de retorno a cero denotado como "a".

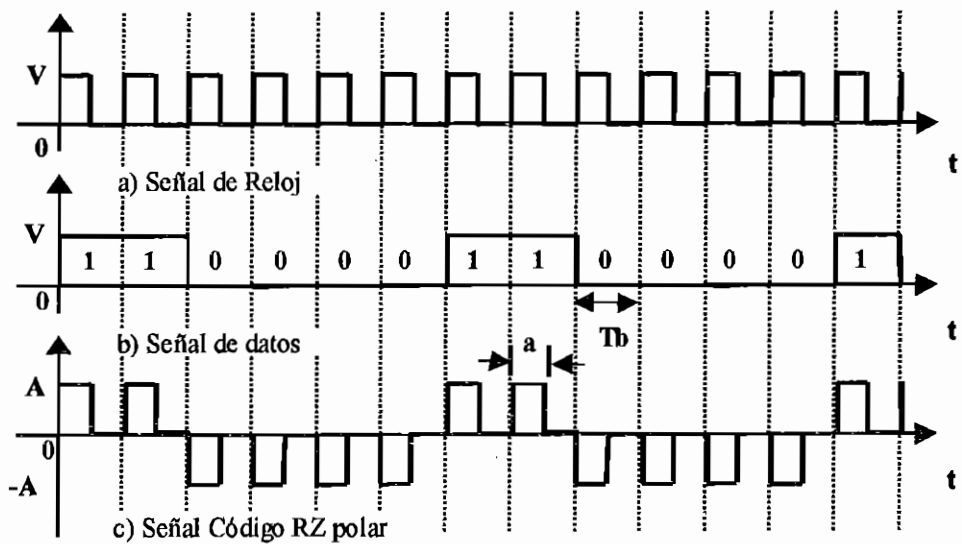


Figura 3.14 Ejemplo de Codificación RZ polar

La Densidad Espectral de Potencia de este código se encuentra descrita en la ecuación (3.4) y representada en el gráfico 3.15, estos datos se tomaron de la referencia [6].

$$S(f) = A^2 T_b p q \left( \frac{\text{sen}(\pi f T_b / 2)}{\pi f T_b / 2} \right)^2 + \frac{A^2}{4 T_b} (2p - 1)^2 \left( \frac{\text{sen}(\pi f T_b)}{\pi f T_b} \right)^2 \sum_i \delta \left( f - \frac{i}{T_b} \right) \quad (3.4)$$

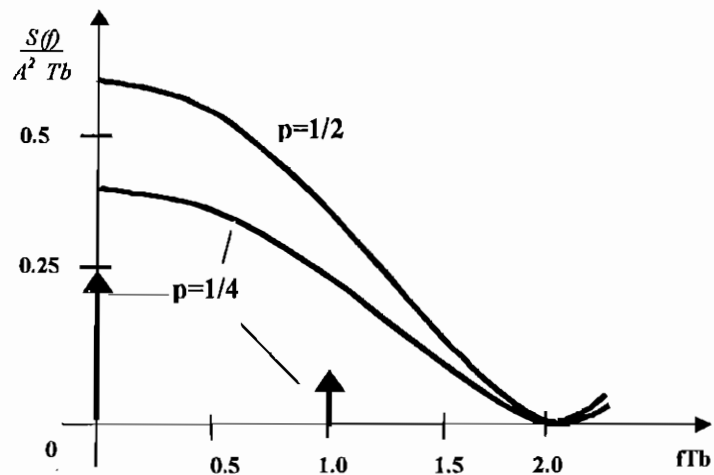


Fig. 3.15 Densidad espectral de potencia para el código RZ polar

## **Simulación RZ polar.**

Este código presenta un esquema de simulación similar al caso unipolar, pero con la diferencia que se toma en cuenta el nivel correspondiente para el dato cero.

### **a) Codificación.**

El bloque codificador está compuesto prácticamente de dos codificadores unipolares, ya que el cero lógico debe tener el mismo proceso de codificación que el uno lógico. Existen entonces dos partes principales del bloque de codificación, una que entrega los pulsos positivos y otra los pulsos negativos.

Las partes del bloque de simulación que generan los pulsos positivos y negativos se diseñaron de tal forma que cuando el uno actúe el otro entregue un nivel cero. Cuando se tiene un uno lógico se permite el paso de la señal de pulsos positivos y nivel cero en el sector de pulsos negativos; al tratarse de un cero lógico se tendrá un nivel cero en el sector de pulsos positivos y pulsos negativos en el otro sector. Finalmente se suman los dos resultados y esta es la señal codificada.

En la figura 3.16 b se puede observar la composición del bloque de simulación desarrollado, en la parte superior se tiene lo que corresponde al paso de pulsos positivos y en la inferior el sector que controla el paso de pulsos negativos. Para finalmente sumar las dos señales resultantes obteniéndose la señal deseada.

El bloque se diseñó de la siguiente forma: los pulsos positivos y negativos están dados por la misma clase de bloque denominado *Pulse generator* de la librería *Source*, el control del paso de la señal está dado por la señal de datos que ingresa a dos bloques que sirven como interruptores, los cuales se denominan *switch* en la librería *nonlinear*; y, por último se tiene el bloque *Sum* de la librería *linear*, que sirve para sumar dos señales. De la misma forma que en la codificación RZ unipolar, la razón de retorno a cero está dado por la facilidad que presenta el bloque *Pulse generator* de modificar el ancho del pulso.

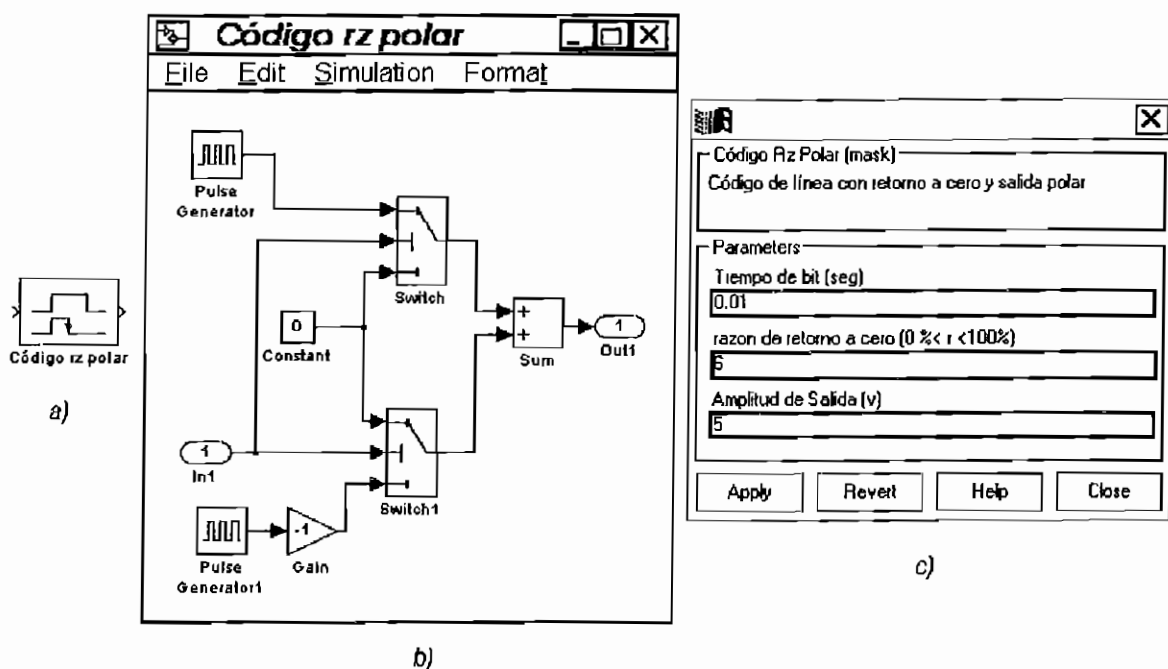


Figura 3.16 Codificación RZ polar: a) Bloque de simulación c) Subsistema contenido

### b) Decodificación.

Es la misma decodificación que para el caso unipolar, como puede verse en la figura 3.17, con la diferencia que la señal codificada es recortada su nivel negativo y procesada de manera similar que en la decodificación RZ unipolar. Para recortar el nivel negativo se emplea el bloque *Relational Operator* de la librería *nonlinear*, el cual compara la señal de entrada con la señal del bloque *constant* que se le dió el valor de cero. Si la señal es mayor que la constante se tiene como salida un

nivel de uno y cero en el caso contrario. Luego de quitar a la señal el nivel negativo se tiene una señal RZ unipolar y la decodificación se convierte en la decodificación RZ unipolar.

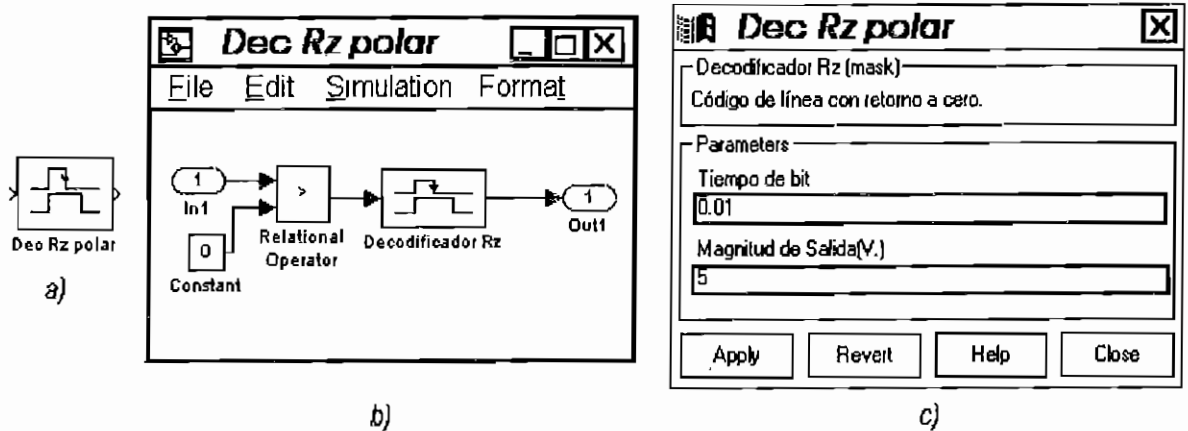


Figura 3.17 Decodificación RZ polar: a) Bloque de simulación b) Subsistema contenido c) Plantilla de datos

### 3.2.5 Código AMI (Alternate Mark Inversion).

Es un código en el cual se emplean pulsos de polaridad alternada para codificar los unos binarios, en tanto que los ceros se codifican mediante la ausencia de pulsos (figura 3.18). La Densidad espectral de Potencia se encuentra determinada por la ecuación (3.5) y está representada en la figura 3.19, según la referencia [6] con el nombre de código NRZ bipolar.

$$S(f) = A^2 T_b \left[ \frac{\text{sen}^2(\pi f T_b)}{\pi f T_b} \right]^2 \frac{4pq}{1 + (2p-1)^2 + 2(2p-1)\cos(2\pi f T_b)} \quad (3.5)$$

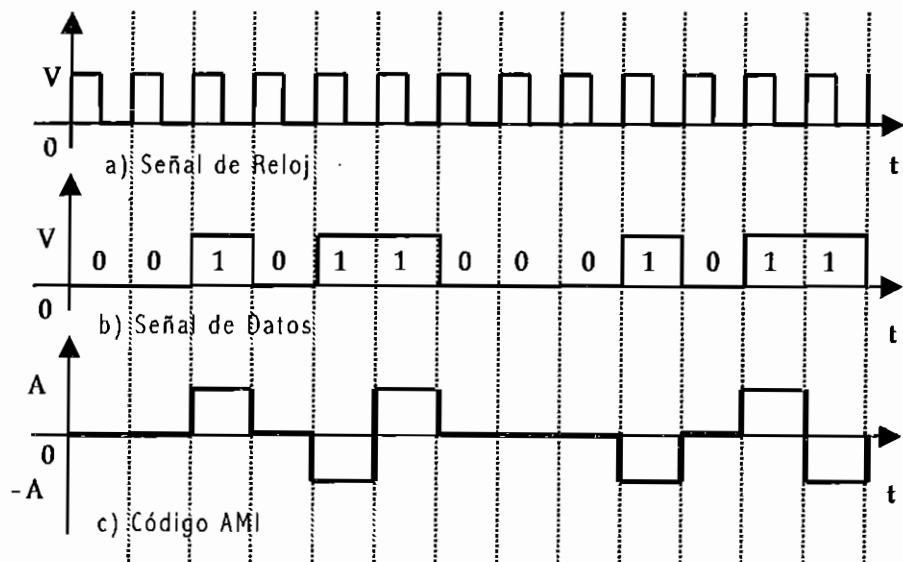


Figura 3.18 Codificación AMI NRZ

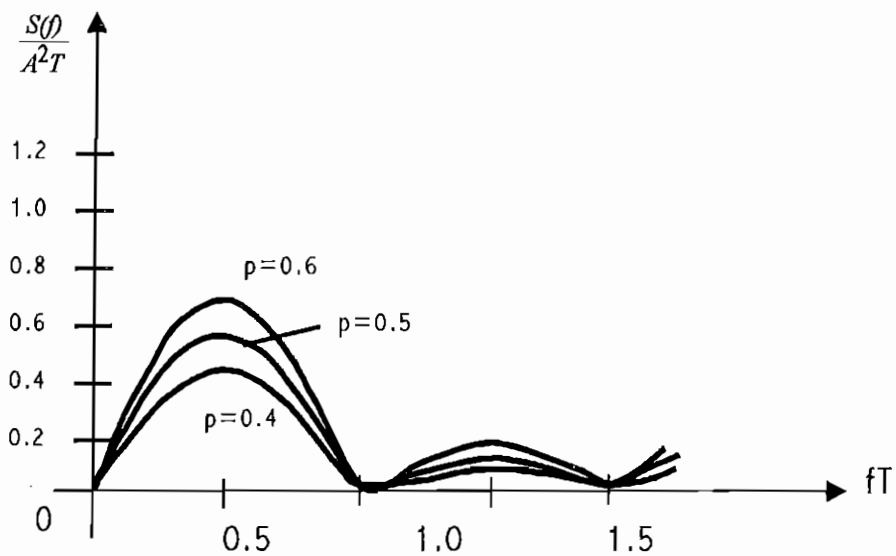


Fig. 3.19 Densidad espectral de potencia para el código AMI

Este código también es empleado en la versión con retorno a cero en telefonía, por ejemplo en el sistema americano de telefonía digital<sup>1</sup>.

<sup>1</sup> Ver referencia [5].

## **Simulación código AMI NRZ y AMI RZ.**

Este código requiere como dato adicional de la polaridad del uno lógico precedente, es decir necesita memoria durante la codificación. Este tipo de códigos con memoria se ha desarrollado a través del bloque *S-function* y necesita un archivo de programación (archivo .m). En el anexo A se presenta una guía acerca de la manera como se emplea la *S-function*, en esta Tesis.

La parte de la programación que permite la codificación con la *S-function*, se encuentra relacionada con el respectivo archivo de extensión m por lo que se recomienda la lectura del anexo A. En adelante solo se describirá en general el algoritmo del archivo.m y algún detalle adicional de los datos requeridos en el programa.

### **a) Codificación.**

El bloque codificador toma la señal de entrada, la identifica y procede a determinar (mediante análisis de los datos previamente ingresados) si se recibió un uno lógico o cero lógico y que tipo de polaridad que tenía el último uno. Si el dato es cero siempre se tiene un nivel cero como salida y si el dato es uno se le asigna un pulso de polaridad contraria a la del uno precedente, se actualizan cambios y se envía como resultado el nivel de señal del uno lógico.

Este proceso se diseña a través del bloque *S-function*, el cual para mejor presentación es empleado como subsistema para poder dar al usuario una manera fácil de ingresar los datos, por medio de una plantilla de datos diseñada para ser compatible con la original del bloque en mención. La plantilla de datos original del bloque *S-function* se muestra



en la figura 3.20 b; en esta plantilla de datos se encuentra el nombre del archivo de programación ("*S-function name*") "amicod" y los parámetros necesarios para que este archivo realice el proceso de codificación, estos parámetros son: sa, uma y ampp.

En la figura 3.20 c se tiene la plantilla de datos desarrollada para este bloque codificador, cuyos datos se relacionan con los datos de la plantilla de la figura 3.20 b de la siguiente manera: "Marca última" con la variable uma, "Amplitud de Salida" con ampp y sa con el valor "Tiempo de duración del bit" ( $sa=T_b$ ).

En "amicod"<sup>1</sup> se definió un tiempo de muestreo igual a  $T_b/2$ , es decir la mitad del tiempo de bit para la lectura de datos. Además se determinó que si el valor ingresado al bloque codificador es cero se mantiene un nivel cero en la salida; pero si es uno, dependiendo del tipo de marca anterior (-1 o +1), se asigna el valor de +A ó -A.

La señal de salida tiene un retardo de medio tiempo de bit con respecto a la señal de datos, debido a que la lectura de los datos ocurre después de haber transcurrido el tiempo de muestreo.

---

<sup>1</sup> En el anexo A, la estructura de este archivo se explica en el ejemplo A.1.

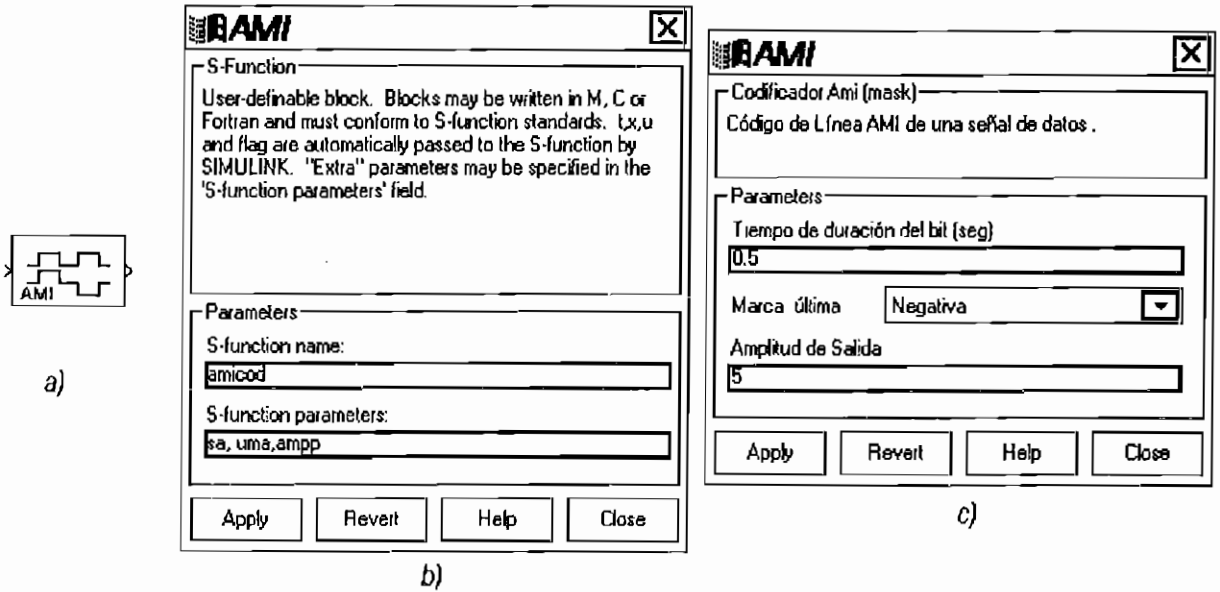


Figura 3.20 Codificación AMI (sin retorno a cero): a) Bloque de Simulación b) Subsistema c) Plantilla de datos

## b) Decodificación.

La recuperación de los datos se consigue tomando el valor absoluto de la señal codificada. El bloque *Abs* de la librería *nonlinear* entrega el valor absoluto de la señal de entrada y es utilizado para la decodificación AMI tal como se muestra en la figura 3.21 b. Adicionalmente se utiliza el bloque *Gain* de la librería *linear* para que el usuario tenga la opción de determinar el valor de la salida ("Magnitud de Salida" figura 3.21 c), entonces se solicita también el nivel de la señal de entrada ("Magnitud de Entrada") con el propósito de que la ganancia *k* sea igual a la magnitud de salida dividida para la magnitud de entrada y de esta manera tener el nivel de salida deseado por el usuario.

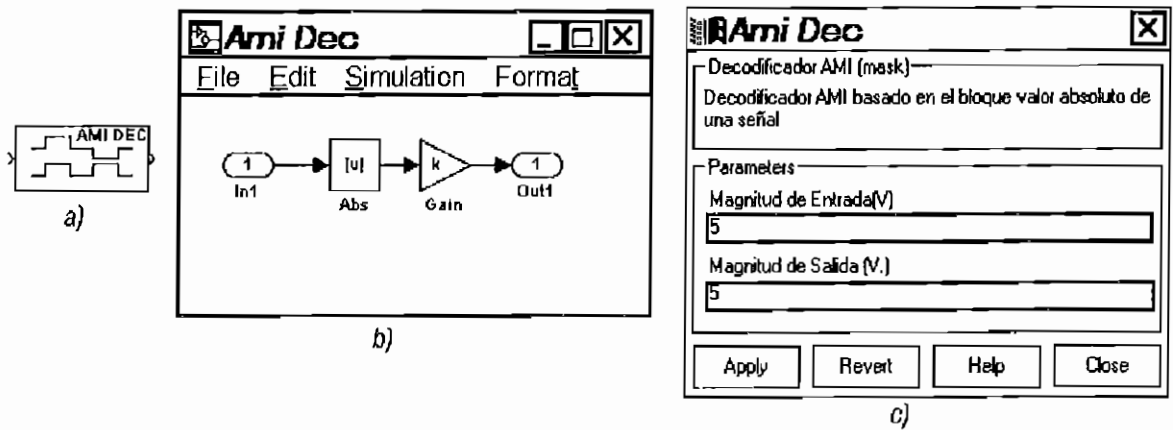


Figura 3.21 Decodificación AMI: a) Bloque de Simulación b) Subsistema c) Plantilla de datos

### c) Codificación y Decodificación AMI RZ.

Para realizar esta codificación se utiliza: el bloque desarrollado para el código AMI, el bloque del código RZ polar y el bloque de retardo de una señal digital desarrollado en la sección 2.1.1.2. La decodificación se consigue simplemente añadiendo después de la codificación el bloque decodificador AMI y luego el decodificador RZ polar.

Se puede apreciar en la figura 3.22b como se diseñó el codificador. El bloque AMI empleado es exactamente el mismo desarrollado antes, solamente que sin la plantilla de datos. El bloque de retardo de señal está entre los otros dos bloques con el fin de tener a la señal AMI retardada un tiempo de bit antes de entrar al bloque RZ polar. El bloque codificador RZ polar tiene una pequeña modificación, se añadió un bloque llamado *Enable*<sup>1</sup> en su interior, lo que produce el pedido de una entrada adicional

<sup>1</sup> este bloque se encuentra en la librería *connections* y se recomienda ver la referencia [5] para una descripción detallada de su modo de empleo.

al bloque RZ, esta señal de entrada proviene de un bloque de denominado paso (que se lo encuentra en la librería *source* con el nombre de "step").

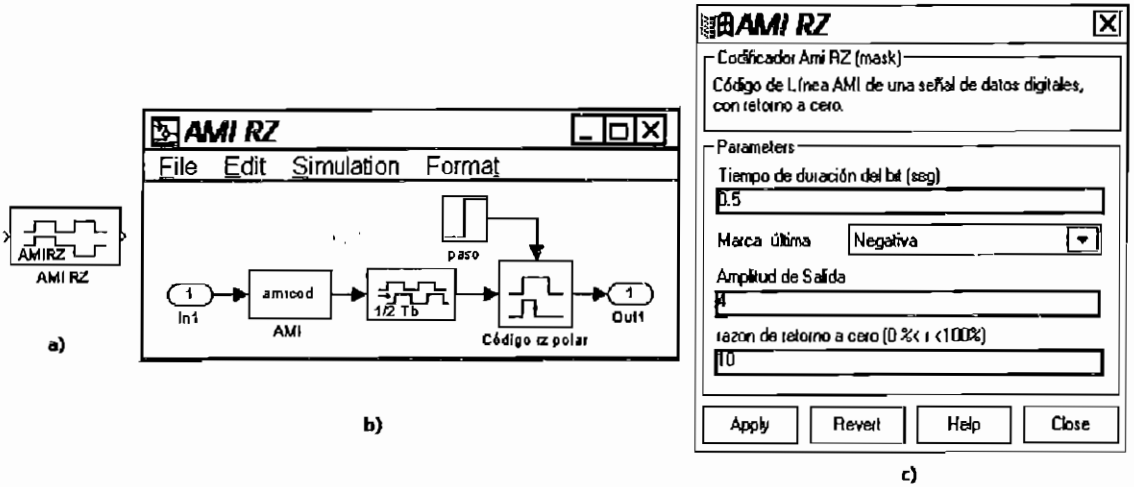


Figura 3.22 Codificación AMI RZ: a) Bloque de Simulación b) Subsistema c) Plantilla de datos

El bloque denominado paso entrega una señal del tipo escalón, es decir de valor cero por un tiempo definido y luego un nivel unitario durante el resto del tiempo de simulación. Se utilizó el bloque paso de tal forma que su nivel unitario empiece luego de un tiempo de bit.

La señal AMI tiene por la codificación un retardo de medio tiempo de bit y después del bloque retardos tiene un retardo total de un tiempo de bit. El propósito de retrasar la señal AMI es el de sincronizarla con los pulsos generados al interior del bloque RZ polar. Entonces la función del bloque *Enable* es la de habilitar el bloque RZ polar al mismo tiempo que empieza el código AMI retardado, es decir a un tiempo de bit.

### 3.2.6 Código HDB3(*High Density Bipolar 3*).

En general el código HDB<sub>n</sub> no admite un número mayor a n ceros consecutivos para una señal AMI. El código HDB<sub>n</sub> más utilizado es el HDB<sub>3</sub>, el cual no tolera el paso de más de tres ceros consecutivos, reemplazando el cuarto cero por un pulso de violación, los unos se codifican con la regla AMI. Los tres ceros consecutivos son reemplazados obedeciendo las siguientes reglas descritas en la tabla 3.1.

Polaridad del pulso precedente	Número de unos desde la última sustitución	
	Impar	par
-	0 0 0 -	+ 0 0 +
+	0 0 0 +	- 0 0 -

Tabla 3.1 Reglas para la sustitución HDB3

Cuando se tienen los cuatro ceros consecutivos, para proceder a la sustitución es necesario saber la polaridad del pulso precedente y el número de unos, ocurridos a partir de la última sustitución, es par o impar. Si el número de unos ocurridos desde la última sustitución es impar, se produce un pulso de violación de la regla AMI<sup>1</sup> en lugar del cuarto cero; si el número de unos es par, además del pulso de violación (ubicado siempre en lugar del cuarto cero), se tiene un pulso de relleno<sup>2</sup> en lugar del primer cero. A continuación se tiene un ejemplo de la codificación HDB<sub>3</sub> en la figura 3.23.

<sup>1</sup> El pulso de violación quiere decir, que el pulso debe tener la misma polaridad del pulso precedente.

<sup>2</sup> Pulso de relleno quiere decir, que el pulso tiene la polaridad opuesta a la polaridad del pulso precedente.

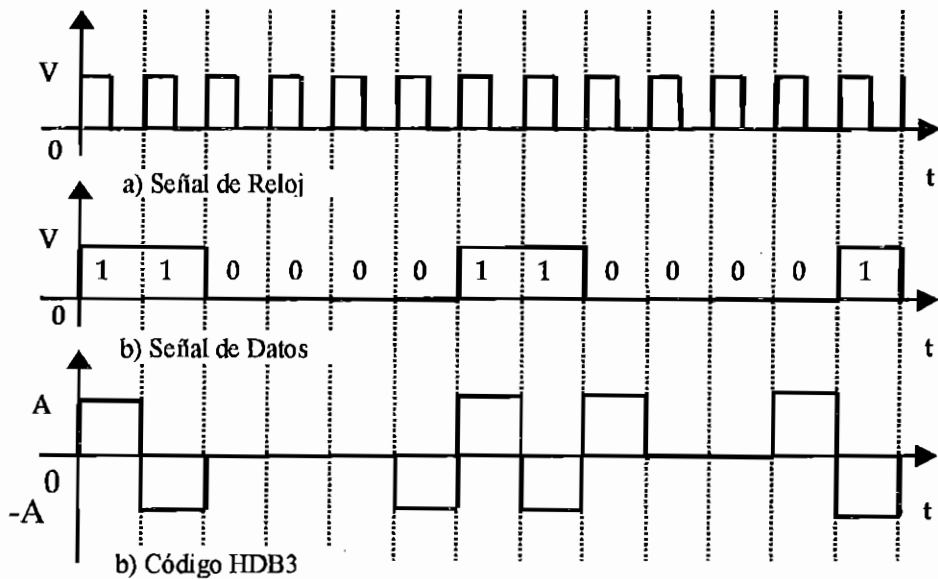


Figura 3.23 Ejemplo de Codificación HDB<sub>3</sub>

Para el código HDB<sub>3</sub> la expresión de su densidad espectral de potencia (ver figura 3.24), según la referencia [6], es más compleja que las anteriores y para una entrada equiprobable de ceros y unos se expresa en los siguientes términos:

$$S(f) = \frac{40 - 32 \cos \Phi}{465T(1025 - 64 \cos 5\Phi)} \left[ \frac{7258.5 - 1929 \cos \Phi - 1424 \cos 2\Phi - 160 \cos 3\Phi + 32 \cos 4\Phi - 131288.5 - 41399 \cos \Phi - 86112 \cos 2\Phi}{85 - 44 \cos \Phi - 24 \cos 2\Phi - 16 \cos 3\Phi} \right] [X(f)]^2 \quad (3.6)$$

donde  $\Phi = 2\pi fTb$  y  $X(f)$  es la transformada de Fourier del un pulso unitario.

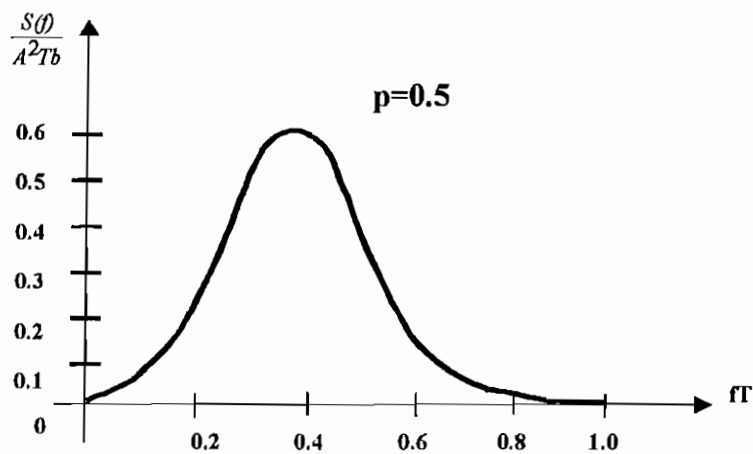


Fig. 3.24 Densidad espectral de potencia para el código HDB<sub>3</sub>

En la figura 3.24 se tiene una representación de la densidad espectral de potencia, para un valor equiprobable de ceros y unos.

### **Simulación del código HDB3.**

Esta simulación presenta un retraso mayor debido a que se necesita conocer el cuarto bit consecutivo y se debe considerar el retardo por la lectura de los datos. Por lo tanto el bloque codificador y decodificador están diseñados para entregar un valor de cero hasta que los datos requeridos sean debidamente procesados.

#### **a) Codificación.**

El programa (cuyo algoritmo esta representado en la figura 3.25) se denomina hdb3cod.m. El usuario puede modificar cuatro parámetros principales: el "tiempo de duración del bit", "Amplitud de la salida", tipo de polaridad precedente o "Marca última" y la posibilidad de elegir si es par o impar el número de unos después de la supuesta sustitución precedente (supuesta, ya que en el tiempo  $t$  igual a cero segundos, no puede hablarse de que existió o no una sustitución).

La lectura de los datos se realiza cada medio tiempo de bit, por lo tanto se tiene que el dato válido se encuentra pasando un tiempo de bit después de la última lectura. Dentro de este tiempo de lectura de datos válidos se realiza la programación respectiva de la codificación.

Dentro del archivo hdb3cod.m se determinó cinco elementos del vector variables de estado ( $x$ ) con las siguientes funciones:

$x(7)$ .- elemento cuyo valor se transmite como salida del bloque de codificación.

x(20).- Primer elemento codificado, que actualiza el valor de x(7).

x(21).- Segundo elemento codificado.

x(22).- Tercer elemento codificado.

x(23).- Cuarto elemento codificado.

Cada tiempo válido de lectura se actualiza el valor de x(7) con el contenido de x(20), posteriormente el último dato codificado se almacena en x(23) y los valores anteriores son guardados en sus inmediatos superiores perdiéndose el primer valor de x(20) (que ya fue transmitido). No es necesario esperar que ingrese el cuarto bit para la transmisión debido a que, se tienen con valor cero desde la variable x(20) a la x(23)<sup>1</sup>. Se denominó al conjunto x(20), x(21), x(22), x(23) vector de transmisión.

Si el bit recibido es uno se cambia la polaridad precedente, se incrementa el contador de unos, se define como cero al contador de los ceros, se escribe en x(23) +1 o -1 según la polaridad precedente y se continúa con la transmisión de los datos correspondientes. Si el dato es cero lógico, al no tener cuatro ceros consecutivos se mantiene ese valor en el vector de datos, pero si el cuarto cero ha llegado se realiza lo siguiente: se define como cero el contador de ceros, según si es par o no el número de unos antes de la última sustitución, se reemplaza el vector x (en sus elementos 20,21,22,23) con los valores correspondientes a los de la tabla 3.1.

Los parámetros requeridos para la codificación son ingresados mediante una plantilla de datos que se encuentra en la figura 3.26.

---

<sup>1</sup> En la parte de inicialización (hdbrcod.m), se debe modificar x0 (ver anexo A) en sus elementos 20 al 23, para tener valores de niveles deseados, antes de los correspondientes a la codificación, mientras tanto se transmite 0.



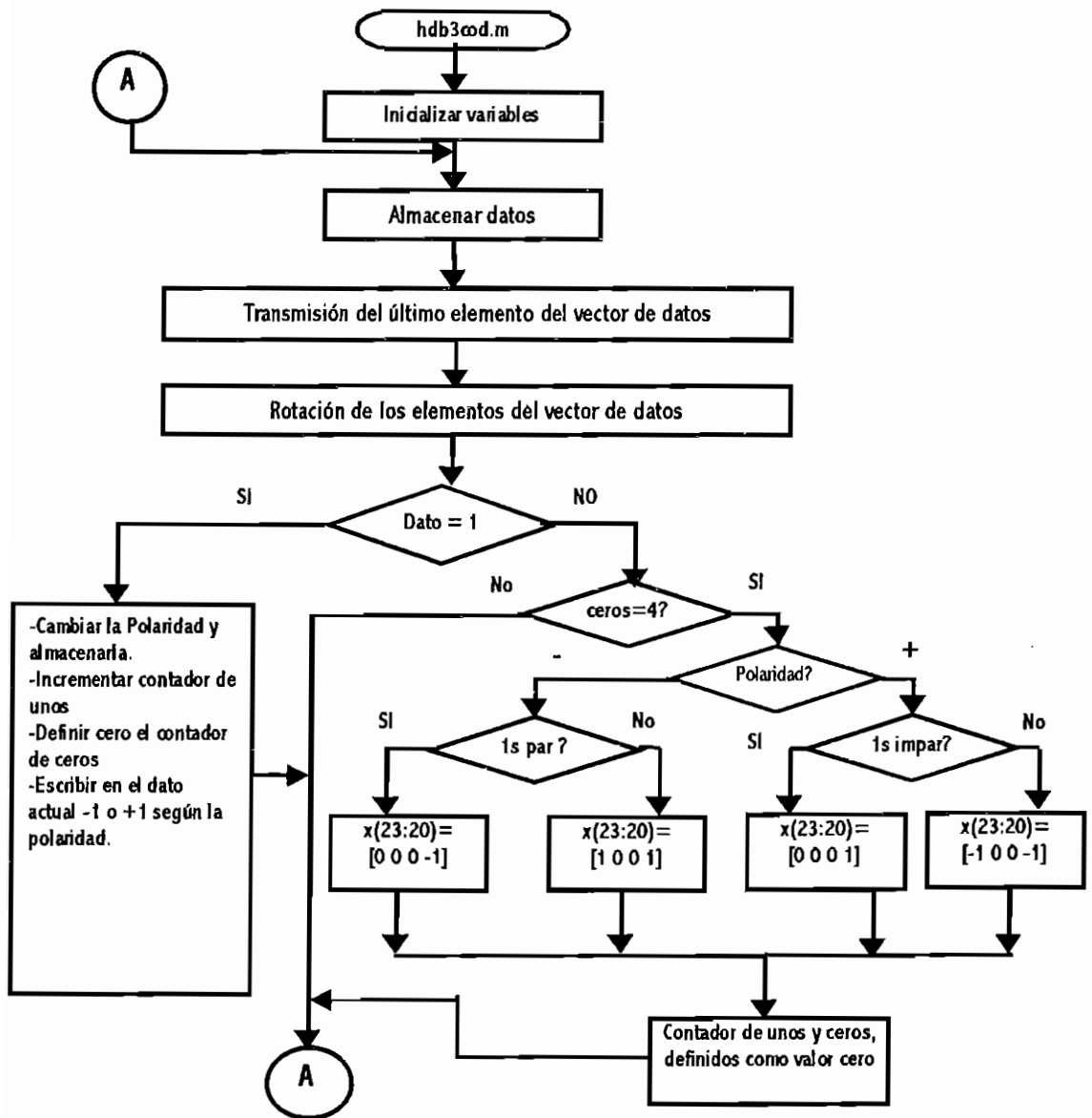


Figura 3.25 Codificación hdb3cod.m

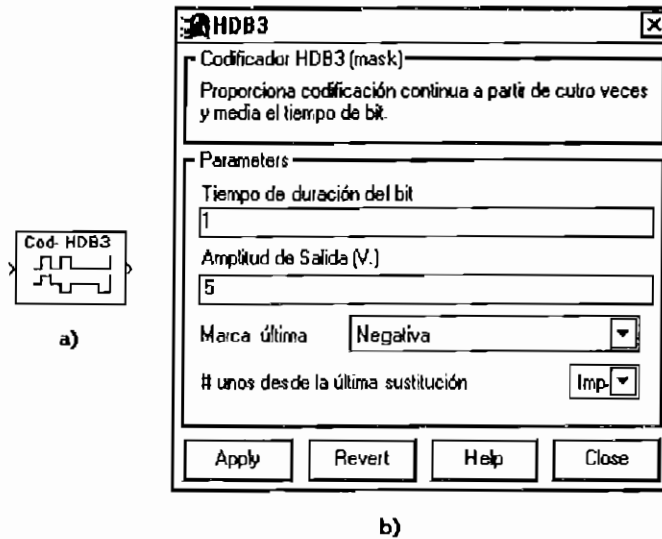


Figura 3.26 Codificador HDB3: a) Bloque de Simulación b) Plantilla de datos

## b) Decodificación.

Es importante indicar que la lectura de los datos recibidos del bloque de codificación, debe hacerse en el intervalo de tiempo en el cual se tiene un dato de la codificación real, en este caso después de  $4.5 T_b$ . Esto es válido para todos los códigos que tienen memoria ó que utilicen el bloque *S-function* tomando en cuenta el retardo respectivo.

Al recibir el elemento válido, si éste es cero se escribe un  $0_L$  en el último elemento del vector de transmisión ( $x(23)$ ), de la misma manera que en la codificación. Por otra parte, si el elemento de la señal recibido es un pulso (positivo o negativo), se compara con la polaridad del último pulso recibido, si existe diferencia, se ha cumplido con la regla AMI de alternabilidad en la polaridad y se escribe un  $1_L$  en el vector de transmisión. Si las polaridades de los pulsos son iguales, se ha producido una violación a la regla AMI y por tanto se reemplazan todos los valores del vector de transmisión por ceros.

Como se observa en la figura 3.27 b, es necesario especificar el "tiempo de duración del bit", "Amplitud de salida" y la "Marca última". Se debe tomar en cuenta que los valores de tiempo de bit y del tipo de marca deben ser los mismos que los del bloque codificador.

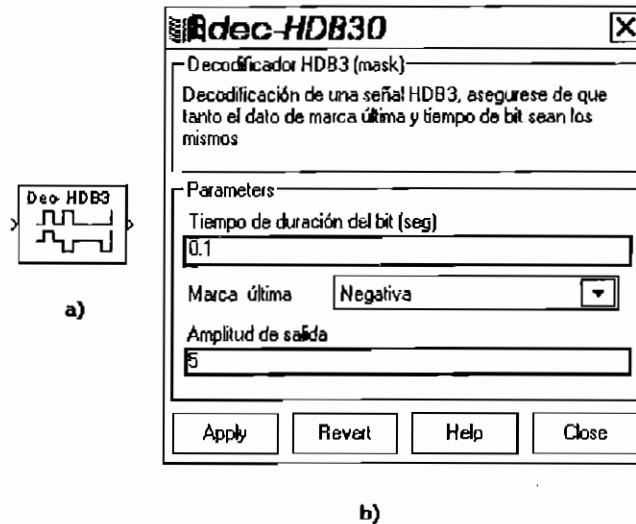


Figura 3.27 Decodificador HDB3: a) Bloque de Simulación b) Plantilla de datos

### 3.2.7 Código B3ZS (*Binary with 3 Zeros Substitution*).

El código B3ZS es conocido como el código bipolar con sustitución de tres ceros, es una versión modificada del código AMI. Los uno lógicos se codifican con pulsos positivos y negativos alternadamente; los ceros con un nivel cero. La diferencia con el código AMI se presenta cuando se tienen tres ceros seguidos.

Cada bloque de tres ceros consecutivos se sustituye por BOV o OOV, donde B representa un pulso conforme a la regla AMI (pulso de relleno) y V representa un pulso que viola la regla bipolar (pulso de violación). Se elige entre BOV y OOV de tal manera que el número de pulsos B

comprendidos entre dos pulsos de violación consecutivos sea impar<sup>1</sup>. Lo dicho anteriormente se puede resumir en la tabla 3.2.

Polaridad del pulso precedente	Número de unos desde la última sustitución	
	Impar 0 0 V	par B 0 V
-	0 0 -	+ 0 +
+	0 0 +	- 0 -

Tabla 3.2 Reglas para la sustitución B3ZS

A continuación se tiene un ejemplo de codificación B3ZS, donde el caso par supone que el número inicial de 1<sub>L</sub> es par, en tanto que el impar supone lo contrario.

Entrada Binaria	101 000 11 000 000 001 000 1
Caso par	+0- +0+ -+ -0- +0+ 00- 00- +
Caso impar	+0- 00- +- +0+ -0- 00+ 00+ +

### Simulación del código B3ZS.

La simulación es muy similar al código HDB3 con la diferencia dada por el número de ceros, según se muestra en la tabla 3.2.

<sup>1</sup> Definición que se encuentra en el Anexo A.1 a la Recomendación UIT-T G.703, preparada por la Comisión de Estudio XVIII y aprobada por el procedimiento de la Resolución N.º 2 el 5 de abril de 1991.

### **a) Codificación.**

Cuando el bit ingresado es  $1_L$  es suficiente escribir en el vector de transmisión el código correspondiente a un pulso de polaridad opuesta al anterior transmitido. Cuando se ha recibido un  $0_L$  será necesario verificar si es el tercer cero recibido, ya que éste es el caso donde se aplica la regla de codificación; caso contrario, se escribirá en el vector de transmisión el valor cero respectivo. Al recibirse el tercer cero se tiene que tomar en cuenta si el número de unos antes de la última sustitución es par o impar y la polaridad del pulso anterior para reemplazar el vector de transmisión con el equivalente dado por la tabla 3.2.

### **b) Decodificación.**

La decodificación se realiza con el siguiente algoritmo: cuando el tercer dato (o bit válido, es decir después de  $3.5 T_b$ ) que ingresa es cero, se escribe en el vector de transmisión. Si este último dato es un pulso (positivo o negativo) se verifica que cumpla con la regla AMI, para identificar como  $1_L$  este dato; de no ser el caso, se escriben tres ceros en el vector de transmisión.

Las plantillas de datos, tienen las mismas características que las del código HDB3 por lo cual, no es necesario añadir nada al respecto.

### **3.2.8 Código Bifase L o Manchester.**

Para este tipo de código se dan transiciones entre niveles de voltaje (+A y -A) a medio tiempo de bit ( $T_b/2$ ). Para  $1_L$  se da una polaridad prefijada y para el  $0_L$  lo opuesto.

Existe además una variación del código Manchester y se denomina Manchester Diferencial, un bit  $1_L$  se representa por la ausencia de transición al inicio del tiempo de bit y el  $0_L$  por la presencia de transición al inicio del intervalo, en ambos casos, existe una transición en la mitad del intervalo del tiempo de bit. Con un ejemplo en la figura 3.28 se muestra tanto la codificación Manchester como la Manchester Diferencial; donde se tiene como dato de entrada: 1100001100111. Para los dos códigos en el ejemplo, se tiene que las transiciones de medio tiempo de bit son: para el  $1_L$  de  $+A$  a  $-A$ , mientras que para el  $0_L$  de  $-A$  a  $+A$ ;

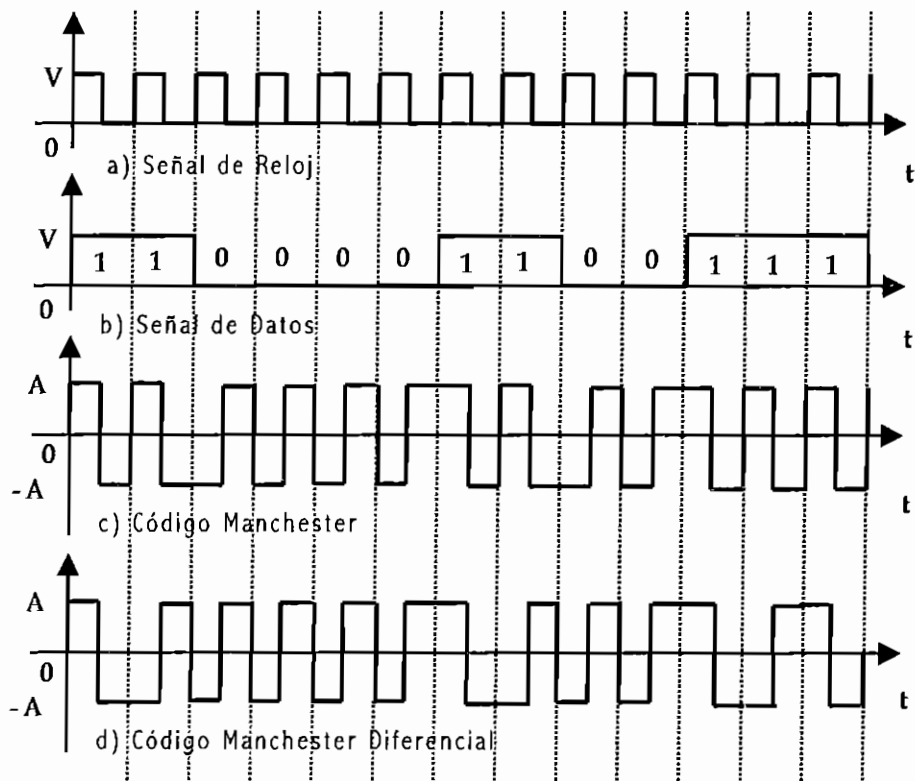


Figura 3.28 Ejemplo de Codificación Manchester

La densidad espectral del código Manchester (representada en la figura 3.29) está dada por la ecuación (3.7), tomada de la referencia [2].

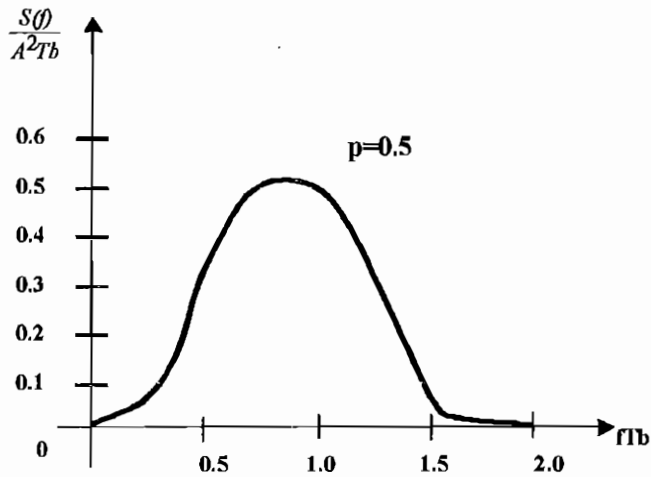


Fig. 3.29 Densidad espectral de potencia para el código Manchester

La ecuación de la densidad espectral  $S(f)$ , es la siguiente:

$$S(f) = \frac{A^2T}{4} \left[ \frac{\text{sen}^2(\pi fTb/2)}{\pi fTb/2} \right]^2 \frac{4pq}{1 + (2p-1)^2 + 2(2p-1)\cos(2\pi fTb)} \quad (3.7)$$

### Simulación del código Manchester

La simulación también emplea el bloque S-function y los archivos desarrollados manchs.m y manchsdec.m.; además se tiene la codificación y decodificación diferencial en los archivos manchsdif.m y manchsdifdec.m. Al igual que en los códigos anteriores la lectura de datos válidos se diseñó para que se realice cada medio tiempo de bit.

#### a) Codificación.

Para el código Manchester, se requieren los datos de: "tiempo de bit", "Amplitud de salida" y "Tipo de polaridad del 1<sub>L</sub> (es decir si desea que la transición del 1<sub>L</sub> vaya de +A a -A, la polaridad es "positiva" lo que

se representa con una variable igual a uno y -1 para la polaridad negativa).

La señal de entrada es muestreada<sup>1</sup> cada medio tiempo de bit, por el bloque codificador y es manipulada de la siguiente manera: si el dato ingresado es  $1_L$ , se le asigna el valor del "tipo de polaridad de  $1_L$ " y si es  $0_L$  el de la polaridad contraria. Al siguiente medio tiempo de bit solamente se cambia la polaridad de la variable de salida. La plantilla de datos se encuentra en la figura 3.30 b).

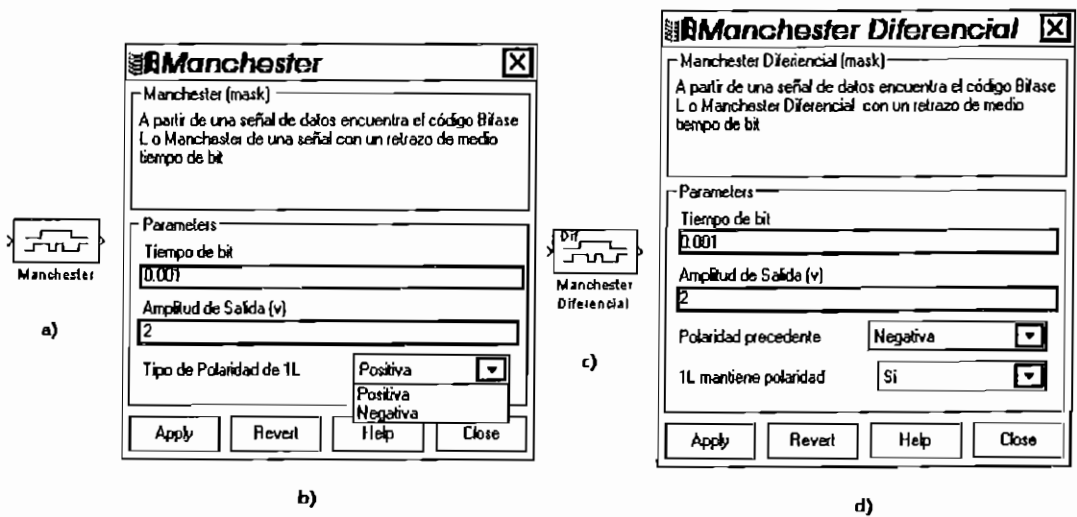


Figura 3.30 Codificación Manchester: a) Bloque de Simulación b) Plantilla de datos Codificación Manchester Diferencial: c) Bloque de Simulación d) Plantilla de datos

Como lo indica la figura 3.30 d, para la codificación diferencial se hace necesario un nuevo parámetro, que permite indicar si el  $1_L$  mantiene la polaridad del pulso anterior o no. Los datos son también leídos cada medio tiempo de bit, al ser ingresados se realiza la operación lógica XOR con el valor del parámetro antes mencionado, si el resultado es  $0_L$ , se cambia el valor de la polaridad precedente y se transmite esta polaridad actualizada; siempre se cambia la polaridad actualizada al siguiente

<sup>1</sup> Al emplear el término muestrear se refiere a que el bloque de simulación es discreto, ver referencia [3].



medio tiempo de bit. Para mayor claridad este algoritmo se representa en un diagrama de flujo, el cual se encuentra en la figura 3.31.

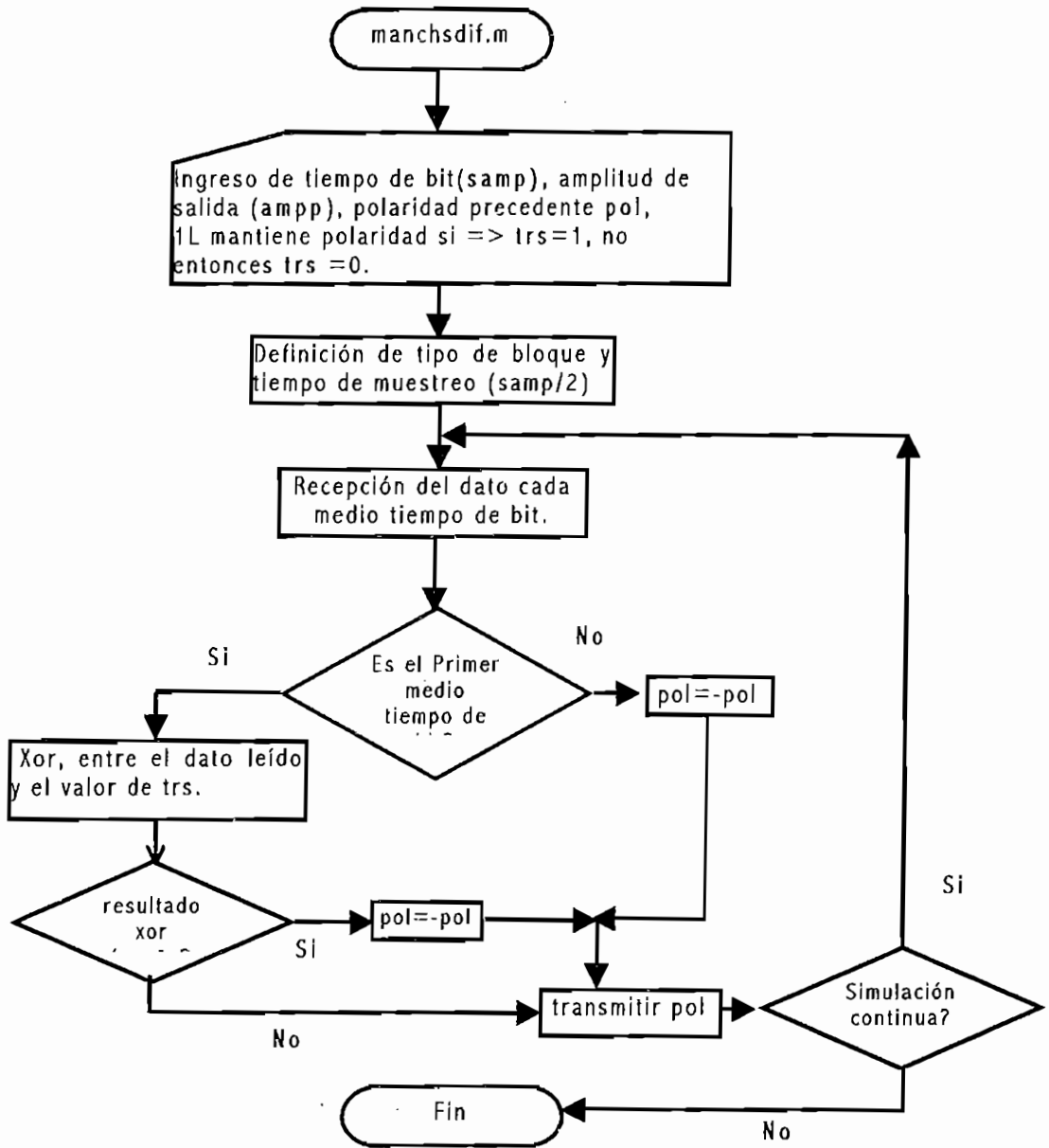


Figura 3.31 Codificación Manchester Diferencial

## **b) Decodificación.**

La decodificación Manchester requiere los mismos parámetros que para la codificación. En el primer medio tiempo de bit de lectura de datos, si el dato es +A se tiene  $1_L$  y si es -A se interpreta como  $0_L$ . En caso de tener la polaridad negativa se complementa la salida. Los unos y ceros resultantes se mantienen para el siguiente medio tiempo de bit.

En el caso del código diferencial, se compara el signo del dato actual con el anterior, si son iguales se trata de  $1_L$  y  $0_L$  si son diferentes. Al final se almacena el signo contrario al que tiene el dato actual.

### **3.2.9 Código Bifase - M (*Biphase- Mark*) y Bifase - S (*Biphase - Space*).**

Estos son códigos de dos niveles, uno positivo (+A) y otro negativo (-A). Una transición aparece siempre al principio del intervalo. Se denomina de tipo M si el símbolo  $1_L$  produce otra transición medio periodo después, en tanto que el símbolo  $0_L$  no produce transición. Si se intercambia lo correspondiente al  $0_L$  con lo del  $1_L$  se tiene el código bifase denominado tipo S. En la figura 3.32 se muestran con un ejemplo estas formas de codificación.

### **Simulación del código Bifase M y S.**

Se diseñó un mismo bloque de codificación para los dos tipos de codificación, el usuario tiene la opción de elegir entre tipo M y S; de igual modo se tiene un solo bloque de decodificación.

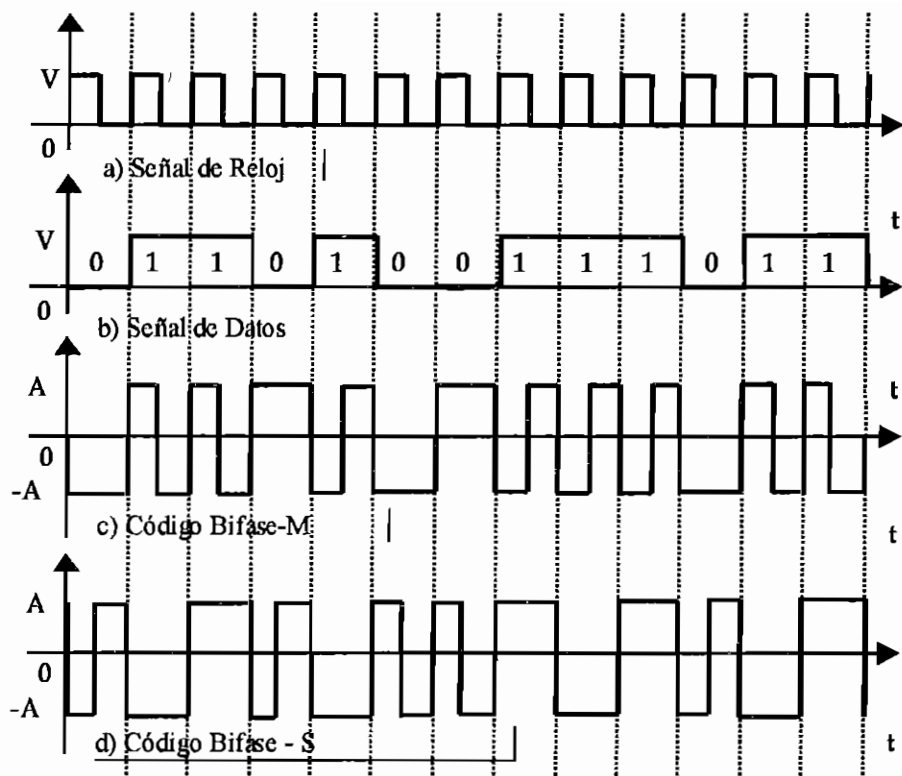


Figura 3.32 Ejemplo de Codificación Bifase M y S

### a) Codificación.

El usuario puede elegir el tipo de código con la opción "Tipo de Bifase M o S", según lo indica la figura 3.33 b. Además el usuario puede modificar los siguientes parámetros: tiempo de bit (samp), magnitud de salida (ampp), polaridad precedente (pol). Estos datos son necesarios para el archivo bifasemscod.m de la codificación.

Se toman los datos cada medio tiempo de bit. En el primer medio tiempo de bit se escribe en la variable de salida (x(4)) el valor de la polaridad contraria a la precedente y en el segundo medio tiempo de bit se efectúa la operación lógica XOR entre el dato y el tipo de código (1 si es tipo M o 0 si es tipo S); si este resultado es igual a uno, se cambia de polaridad el valor de salida.

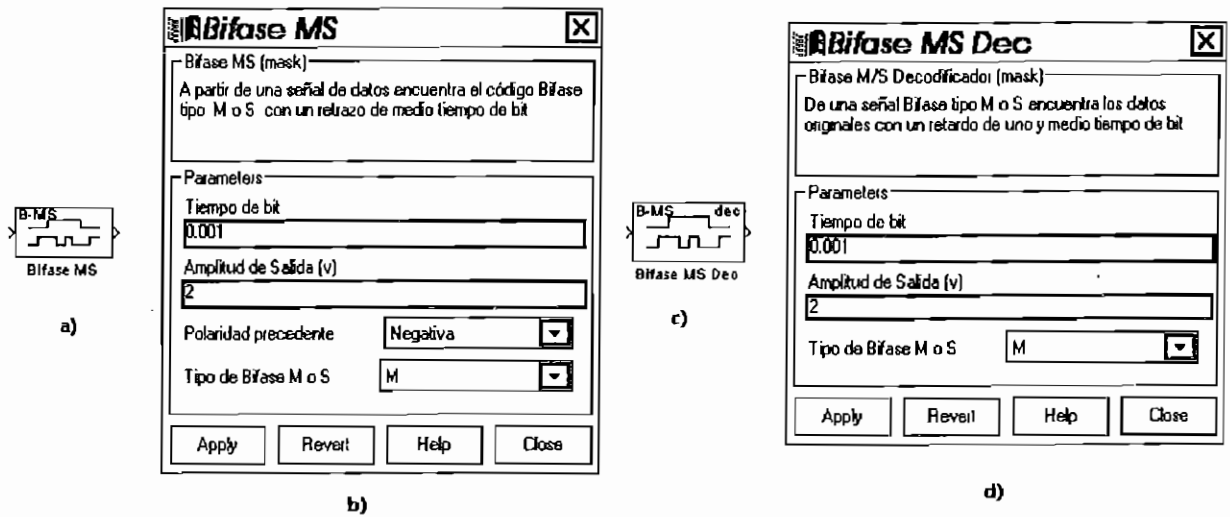


Figura 3.33 Código Bifase tipo M y S: a) Bloque de simulación codificador, b) Plantilla de datos del codificador, c) Bloque de simulación decodificador, d) Plantilla de datos del decodificador.

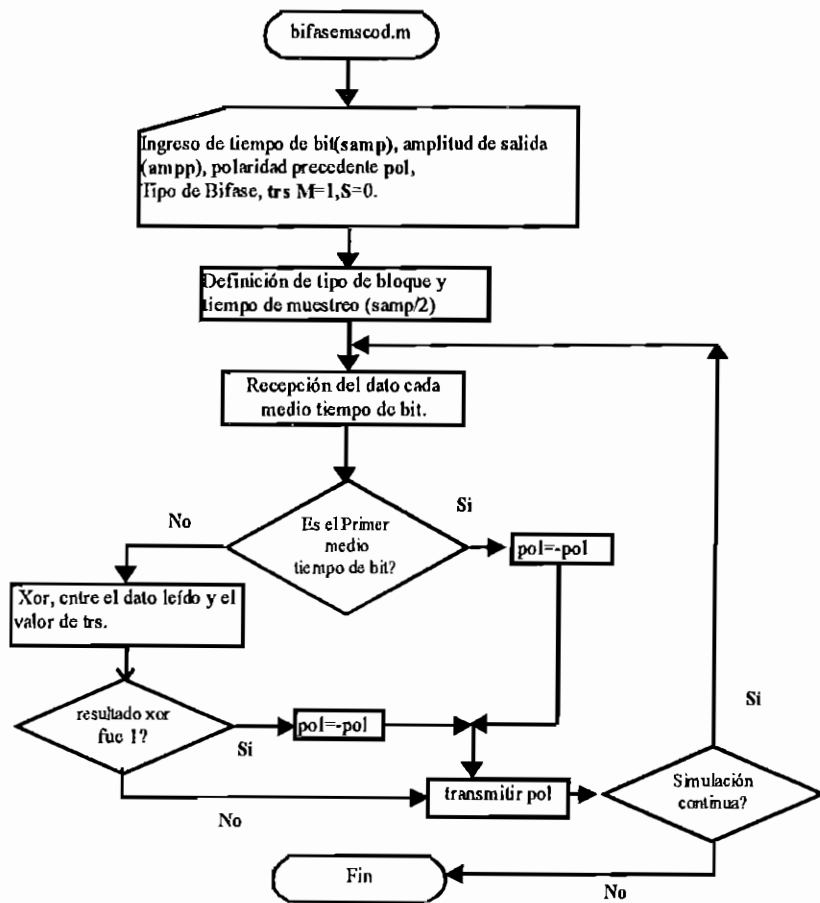


Figura 3.34 Codificación Bifase tipo M y S

## b) Decodificación.

La detección de los datos es idéntica a la efectuada en la decodificación Manchester. En el caso del código Bifase-M se compara el signo del dato actual con el anterior, si son iguales es 0<sub>L</sub> y 1<sub>L</sub> si son diferentes, almacenando al final el signo contrario al que tiene el dato actual. Para el código Bifase-S se complementa la salida del proceso anterior. Según la figura 3.32 d, los parámetros solicitados son los mismos que para el bloque codificador.

### 3.2.10 Código Miller o Código de modulación por retardo.

Este código presenta una transición entre dos niveles de la misma amplitud y polaridad contraria (+A y -A). El símbolo uno lógico produce una transición en el punto medio del período; mientras que el cero lógico no produce ninguna transición, a no ser que vaya seguido por otro cero lógico en cuyo caso se produce una transición entre los dos ceros al final del primer período. Un ejemplo de codificación se tiene en la figura 3.35.

El cálculo de la densidad espectral de potencia es bastante complejo y según la referencia [6] la ecuación que la representa es la (3.8):

$$S(f) = \frac{1}{\theta^2 T b (17 + 8 \cos \theta)} \left[ 23 - 2 \cos \theta - 22 \cos 2\theta - 12 \cos 3\theta - 12 \cos 3\theta + \right. \\ \left. + 5 \cos 4\theta + 12 \cos 5\theta + 2 \cos 6\theta - 8 \cos 7\theta + 2 \cos 8\theta \right] \quad (3.8)$$

donde  $\theta = \pi f T b$

La representación gráfica de la densidad espectral se encuentra en la figura 3.36.

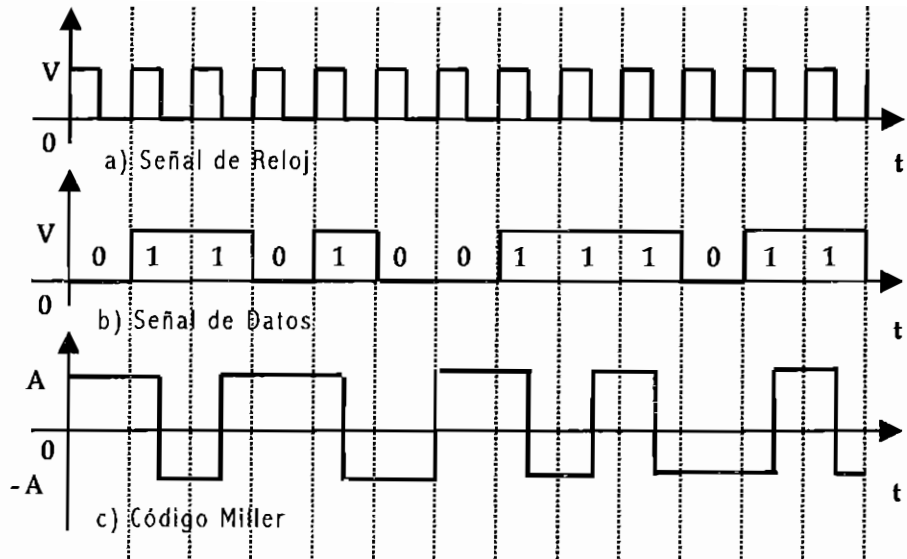


Figura 3.35 Codificación Miller

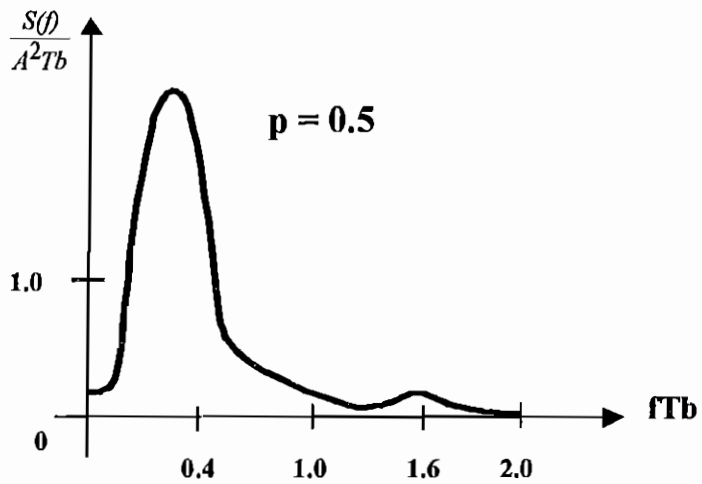


Fig. 3.36 Densidad espectral de potencia para el código

### Simulación del código Miller.

Para la simulación del código Miller se diseñaron los archivos millercod.m y millerdec.m tanto para la codificación como para la decodificación respectivamente.

#### a) Codificación.

La transmisión de datos, tiene las mismas características y mismo retardo que los códigos Bifase. Se toma en el primer medio tiempo de bit

el dato ingresado al bloque de codificación, si el dato es uno se mantiene la polaridad del vector de transmisión durante este medio tiempo de bit y se cambia de polaridad el siguiente medio tiempo de bit. Si el bit de datos es cero se cambia de polaridad de la variable de transmisión, solamente en el caso de que el dato anterior haya sido cero.

Los parámetros requeridos (figura 3.37 b) son: tiempo de bit (samp), amplitud de salida (ampp), polaridad bit precedente y dato anterior.

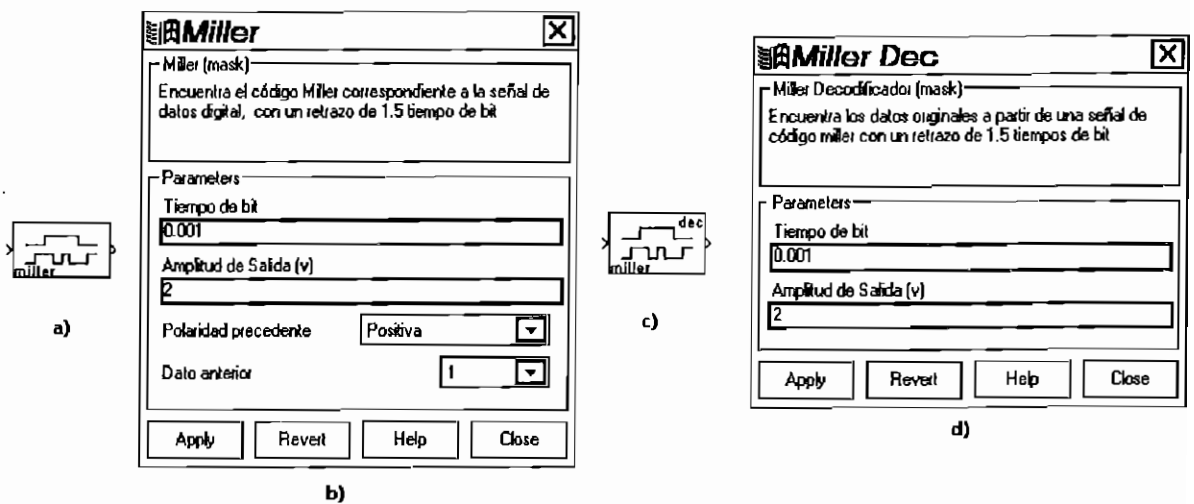


Figura 3.37 Código Miller: a) Bloque de Simulación codificador b) Plantilla de datos codificación c) Bloque decodificador d) Plantilla de datos, decodificación.

## b) Decodificación.

Como lo indica la figura 3.37 d, solamente se solicita al usuario dos parámetros: el tiempo de bit y la amplitud de salida. El proceso es muy similar al de los códigos bifase, con los mismos retardos y tipos de detección de los datos. En el segundo medio tiempo de bit (de lectura del dato válido, es decir después de  $1.5 T_b$ ), se compara con el dato recibido en el primer medio tiempo de bit; si son iguales el vector de salida tiene el valor 0 y 1 en caso contrario.

### 3.2.11 Código 4B3T(4 Binary into 3 Ternary).

Este código produce una sustitución de grupos de 4 dígitos binarios por grupos de tres dígitos ternarios. Según el procedimiento propuesto por Jessop-Waters (tabla 3.3) se realiza la elección de los códigos ternarios, el objetivo será mantener la "disparidad" de componente positiva y negativa en cero, con ello se logrará que la componente continua sea mínima. Para lograr este objetivo se debe considerar la disparidad acumulada que se tiene al momento de realizar la codificación de un grupo de 4 bits. Si la disparidad es positiva se elegirá el modo negativo para la codificación, en tanto que si la disparidad es negativa, se usará el modo positivo.

Se necesita en cada momento de la codificación del valor de la polaridad acumulada, la cual debe estar dentro de los valores de +3 y -3, no se permite tener una polaridad acumulada de valor 0, y en el caso de que se presente, se codificará con la polaridad contraria a la última realizada; con el fin de conseguir la siguiente polaridad acumulada diferente del valor 0. Todo esto se resume en el siguiente diagrama de estados (figura 3.38).

---

<sup>1</sup> Disparidad quiere decir polaridad acumulada.



Entrada Binaria (4B)	Palabra Ternaria (3T)		Disparidad de la palabra código
	Modo Positivo	Modo Negativo	
0000	0 - +	0 - +	0
0001	- + 0	- + 0	0
0010	- 0 +	- 0 +	0
0011	+ - +	- + -	+1, -1
0100	0 + +	0 - -	+2, -2
0101	0 + 0	0 - 0	+1, -1
0110	0 0 +	0 0 -	+1, -1
0111	- + +	+ - -	+1, -1
1000	0 + -	0 + -	0
1001	+ - 0	+ - 0	0
1010	+ 0 -	+ 0 -	0
1011	+ 0 0	- 0 0	+1, -1
1100	+ 0 +	- 0 -	+2, -2
1101	+ + 0	- - 0	+2, -2
1110	+ + -	- - +	+1, -1
1111	+ + +	- - -	+3, -3

Tabla 3.3 Codificación 4B-3T

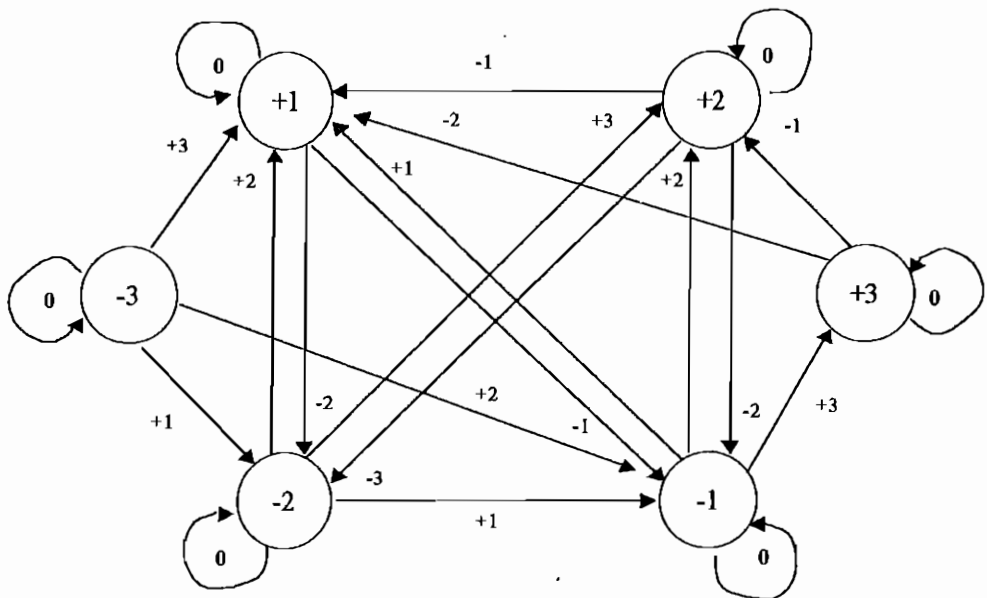


Figura 3.38 Diagrama de Transición de estados del código 4B-3T

A continuación se expone un ejemplo de codificación en 4B-3T, en el cual se supone que la disparidad inicial acumulada es +2.

Entrada Binaria	0000	1000	0110	0111	1011	0101	1111	0000	0000	0000	1101
Salida Ternaria	0 - +	0 + -	0 0 -	+ - -	+ 0 0	0 - 0	+ + +	0 - +	0 - +	0 - +	- - 0
Disparidad Actual	+2	+2	+1	-1	+1	-1	+3	+3	+3	+3	+1

## Simulación del código 4B3T.

Mediante los archivos qb3tcod.m y qb3tdec.m cada bloque de simulación realiza la codificación y decodificación respectivamente. El codificador realiza la lectura de datos cada  $T_b/3$  segundos<sup>1</sup>, ya que se diseñó la codificación de tal manera que, la salida ternaria tenga la misma duración que los cuatro bits de entrada. En cuanto a la decodificación también se determinó un tiempo de toma de datos de  $T_b/3$ .

### a) Codificación.

Para la codificación el dato tomado y considerado válido se lo obtiene en el primer tercio de bit, es decir se vuelve a tomar como dato válido luego pasados los dos tercios restantes. En la figura 3.39 se muestra como está conformado el denominado vector de datos ingresados, en el lado izquierdo se representa las variables en donde se almacena la información y en la parte derecha el contenido de dichas variables y como se actualizan.

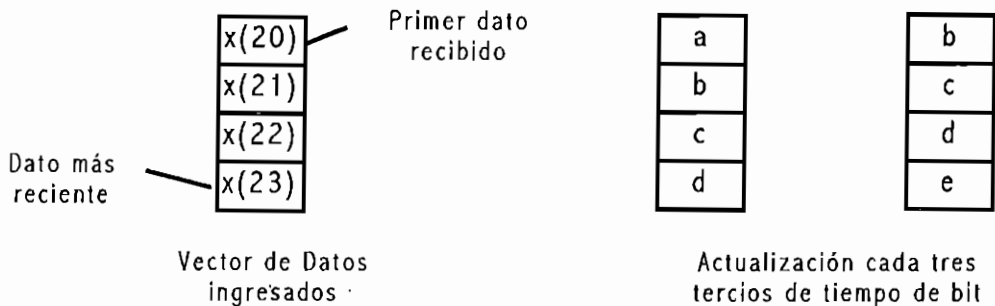


Figura 3.39 Vector de Recepción

<sup>1</sup> Al tratarse de bloques con muestreo, siempre debe tomarse en cuenta que el bloque entra en actividad después de transcurridos  $t_s$  segundos (véase Nota 1 Anexo A).

Se determinó que hasta recibir el cuarto bit el bloque codificador transmite cero, cuando este cuarto bit ingresa se buscan los tres bits correspondientes según la tabla 3.3, para el modo positivo y se almacena también la polaridad respectiva. Posteriormente se hace el cálculo adecuado de la polaridad acumulada y se determina el tipo de salida según la polaridad acumulada inicial, es decir si la polaridad acumulada fue antes positiva o cero, los bits correspondientes antes encontrados son la señal de respuesta a la codificación; si la polaridad acumulada es negativa se multiplica por -1 a la anterior salida.

Después del respectivo proceso de codificación, la transmisión de los resultados respectivos también se efectúa por medio de un vector denominado de transmisión, pero que está formado por los elementos  $x(5)$ ,  $x(6)$  y  $x(7)$  del vector de estados  $x$ . En realidad la salida del bloque de simulación es el elemento  $x(2)$  del vector de estados (definido su valor inicial en cero), pero cuando se recibe el cuarto bit de datos se cambia el contenido de  $x(2)$  por los contenidos de  $x(5)$ ,  $x(6)$  y  $x(7)$ . Las actualizaciones del valor a transmitirse ( $x(2)$ ) se lo hace secuencialmente y varias veces según dure el tiempo de simulación, un ejemplo de la actualización utilizando los índices sería :5,6,7,5,6,7,5,6,7...,etc. Es oportuno indicar que el tiempo de transmisión (o duración de la actualización) es de cuatro tercios de bit, es decir que cuando se terminó el tiempo de transmisión de la actualización del valor de  $x(7)$  en  $x(2)$ , ha llegado ya el cuarto bit y por lo tanto se tienen los nuevos valores para  $x(5)$ ,  $x(6)$  y  $x(7)$ . el gráfico 3.40 ilustra de mejor forma esta última parte.

---

<sup>1</sup> Ver anexo A, sobre el vector de variables de estado.

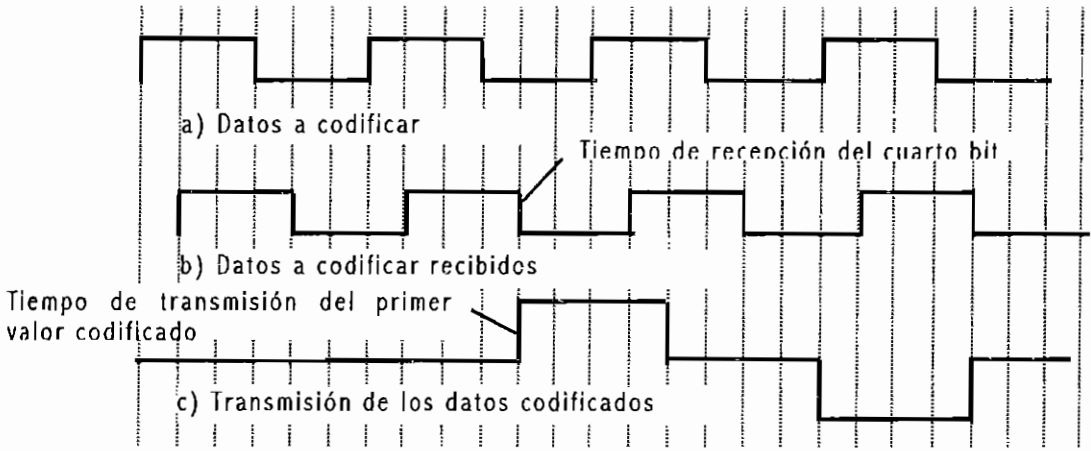


Figura 3.40 Transmisión de la codificación 4B-3T

En la figura 3.40 se puede ver unas líneas verticales que están separadas a un tercio del tiempo de bit, en la parte a) se tiene los datos a codificar, luego se representa la lectura de datos en el bloque codificador y finalmente la señal de salida del bloque de simulación. Claramente se observa el retardo de la señal de salida con los datos a codificar.

Para la codificación (ver figura 3.42) se necesita conocer, el tiempo de bit, la amplitud de salida de la señal y la polaridad acumulada, estos datos se solicitan al usuario en la plantilla de datos de la figura 3.41 b.

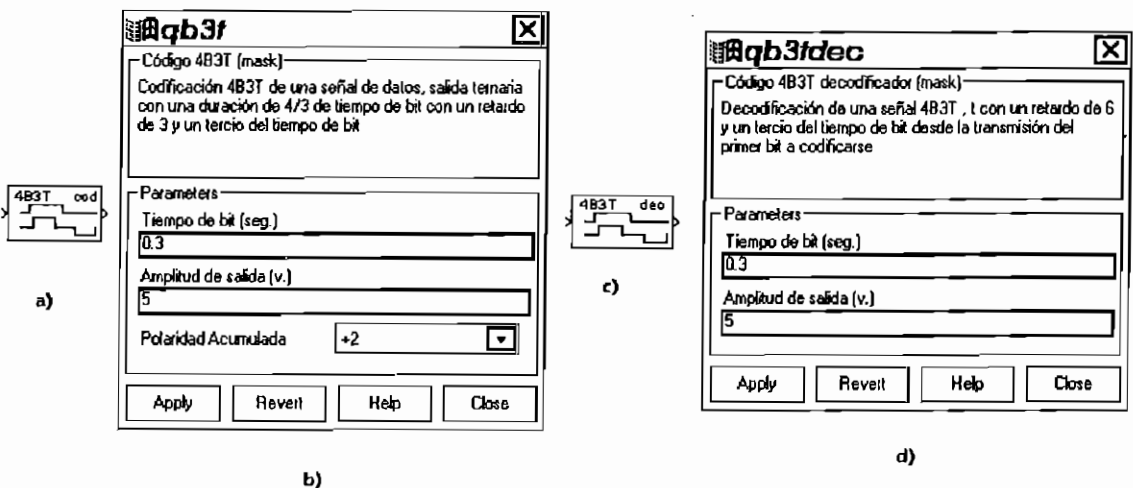


Figura 3.41 Código 4B-3T: a) Bloque de Simulación codificador b) Plantilla de datos codificación c) Bloque decodificador d) Plantilla de datos, decodificación.

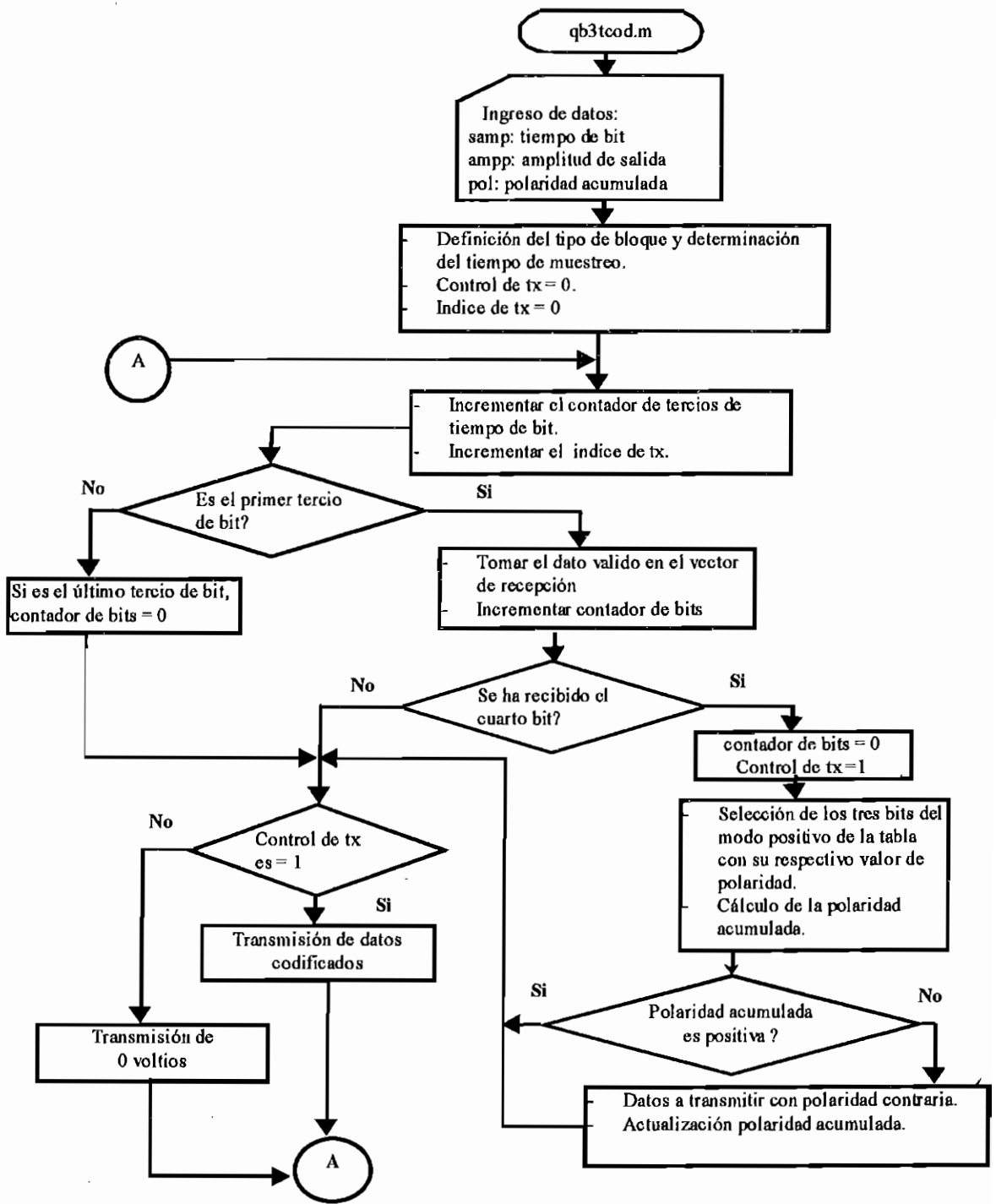


Figura 3.41 Diagrama de Flujo de la Codificación 4B-3T

## **b) Decodificación.**

La transmisión y recepción son análogas a las del proceso de codificación con ciertas diferencias. Para la recepción de datos debe tomarse en cuenta, que el primer tercio de bit está ubicado después del tiempo de la transmisión del primer bit codificado, señalado en la figura 3.40. También es necesario indicar que el próximo dato válido (a decodificarse) está a  $4/3$  de  $T_b$ . En la transmisión se tiene una duración igual a la que tiene el dato decodificado, evidentemente de un tiempo de bit (vale decir tres tercios del tiempo de bit).

Se requieren del usuario como datos adicionales, la magnitud de salida decodificada y el tiempo de bit (figura 3.41 d). Cuando se ha recibido el tercer dato válido a decodificarse, se determina su polaridad, si ésta es negativa se multiplica por  $-1$  los datos ingresados, teniendo de esta manera una sola equivalencia entre datos recibidos ternarios y sus correspondientes de cuatro bits, mediante una tabla de equivalencias.

### **3.2.12 Código CMI (*Coded Mark Inversion*).**

En el código CMI o codificación por inversión de marca. El  $0_L$  es codificado con un cambio de polaridad de negativo a positivo a la mitad del tiempo de bit; y, el símbolo  $1_L$  es codificado asignándole los niveles positivos y negativos alternadamente cada uno durante un período de bit. En la figura 3.43 se da un ejemplo de la codificación CMI.

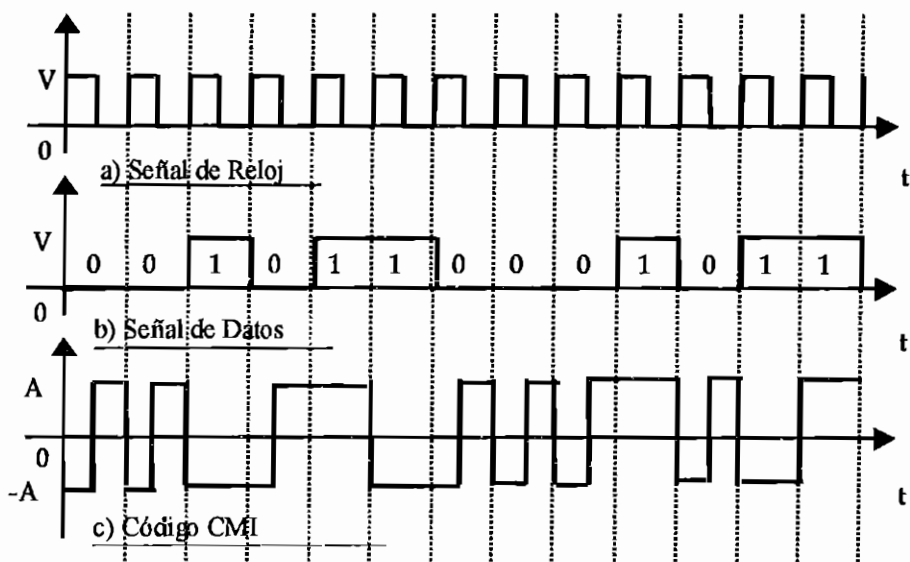


Figura 3.43 Codificación CMI

### Simulación CMI.

La simulación es similar a la de los códigos bifase, por lo tanto la manera de recepción y transmisión es la misma que los códigos bifase. En los archivos cmicod.m y cmidec.m se encuentran la codificación y decodificación respectivas.

#### a) Codificación.

Según la figura 3.44 b, se necesitan los siguientes parámetros para la codificación: "tiempo de bit", "Amplitud de salida" y "Polaridad precedente", que se refiere a la polaridad del bit anterior (para la codificación del  $1_L$ ). El tiempo de muestreo es de medio tiempo de bit, los datos válidos se recogen en el primer tiempo de bit; si el dato es  $0_L$  se transmite un pulso de polaridad negativa y si el dato ingresado es  $1_L$  se invierte la polaridad del  $1_L$  precedente y se transmite la misma durante este medio tiempo de bit. En el segundo medio tiempo de bit se cambia la polaridad de la transmisión, únicamente si se recibió en el tiempo de muestreo anterior un  $0_L$ .

Al tratarse de un bloque con muestreo se tiene un retardo (de la señal codificada respecto a la señal de datos) de medio tiempo de bit y en cuanto a la transmisión se la hace a través del elemento  $x(2)$  del vector de estados.

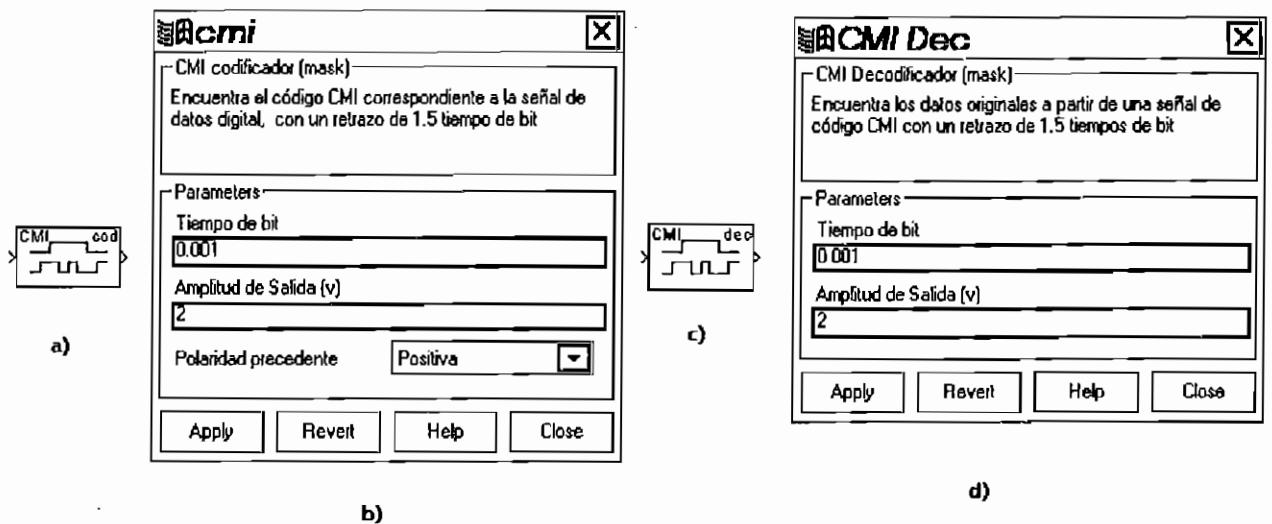


Figura 3.44 Codificación CMI: a) Bloque de Simulación codificador b) Plantilla de datos codificación

## b) Decodificación.

Los parámetros requeridos (ver figura 3.44 d) para la decodificación son dos: "tiempo de bit"(el mismo que el bloque codificador) y la "Amplitud de salida" deseada.

Al retardo de la codificación debe añadirse el retardo de lectura de un tiempo de bit, ya que la decodificación consiste en saber si hubo o no cambio de polaridad a medio tiempo de bit. De esta manera la lectura de datos se inicia luego de dos tiempos de muestreo (tiempo de muestreo es igual a la mitad del tiempo de bit) y la decodificación se inicia un tiempo de muestreo más tarde (ya que necesita conocer el valor de la entrada a la siguiente mitad del tiempo de bit).



Una vez que se inicia la lectura de datos, en el primer tiempo de muestreo se toma el valor de la señal codificada y se almacena, en el siguiente tiempo de muestreo se compara el dato ingresado recientemente con el anterior, si son iguales el bit decodificado es 1<sub>L</sub> y en caso contrario 0<sub>L</sub>.

**3.2.13 Código PST (*Pair Selected Ternary*).**

Se conoce este código también con el nombre de parejas ternarias seleccionadas. La codificación PST procesa pares de datos binarios de entrada para producir secuencias de caracteres de dos dígitos ternarios, los que serán transmitidos. El formato más usual de codificación se encuentra en la Tabla 3.4, para realizar la codificación se selecciona inicialmente uno de los modos (positivo o negativo) hasta que se transmita un pulso simple (pulso acompañado de un nivel cero), en este punto el codificador cambia el modo y selecciona códigos apropiados hasta que igualmente se transmita un pulso simple (de polaridad opuesta).

<b>Entrada Binaria</b>	<b>Modo Positivo</b>	<b>Modo Negativo</b>
<b>00</b>	- +	- +
<b>01</b>	<b>0</b> +	<b>0</b> -
<b>10</b>	+ <b>0</b>	- <b>0</b>
<b>11</b>	+ -	+ -

Tabla 3.4 Regla de codificación para el código PST.

Un ejemplo se presenta en la figura 3.45. donde se tiene un grupo de bits y se muestra las dos posibilidades de codificación.

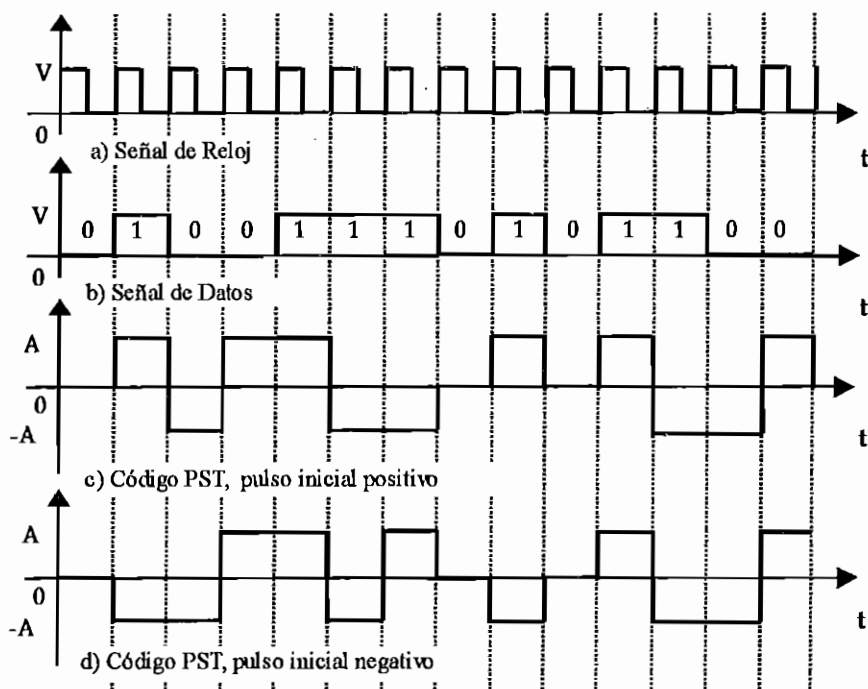


Figura 3.45 Ejemplo de Codificación PST.

### Simulación del código PST.

Una desventaja de la codificación PST es que el flujo de bits debe ser arreglado en pares. Por lo tanto la decodificación debe reconocer los pares de elementos de señal para que pueda asignar los bits correctamente, lo que proporciona un mayor retardo. El retardo en la codificación es de tres tiempos de muestreo, es decir de uno y medio tiempo de bit, mientras que en la decodificación el retardo es de siete tiempos de muestreo o tres y medio tiempos de bit.

Para la simulación se generan los archivos pscod.m y pstdec.m. Dos elementos del vector x se emplean para la lectura de datos y otros

dos para almacenar los valores a transmitirse. La transmisión se la hace actualizando un elemento diferente del vector  $x$  en el intervalo de tiempo adecuado con el valor correspondiente. Este proceso es el mismo tanto para la codificación como para la decodificación y en estos dos casos los bloques transmiten un nivel cero en ausencia de datos, es decir al inicio de la codificación y decodificación (por retardo del mismo proceso).

### **a) Codificación.**

Se reciben los dos primeros datos para la codificación y luego se toma el equivalente respectivo (según la tabla 3.4) para transmitir. De igual manera que en codificaciones anteriores se emplean elementos del vector  $x$  para almacenar los datos ingresados al bloque codificador, en este caso las variables:  $x(4)$ (primer dato) y  $x(5)$ (segundo dato). Desde luego la lectura de datos se lleva a cabo a la mitad del tiempo del bit y con un intervalo de un tiempo de bit entre lecturas. Una vez que se reciben dos datos se almacena el par equivalente (según la tabla 3.4) en dos variables:  $x(8)$  (primer dato codificado) y  $x(9)$  (segundo dato codificado). Luego se transmite el valor de  $x(8)$ , para que en el siguiente tiempo de lectura de datos se transmita lo contenido en  $x(9)$ . El usuario debe indicar (ver figura 3.46 b): el "Tiempo de duración del bit", la "Amplitud de Salida" y el "Modo anterior de codificación". El modo anterior de codificación si es positivo tiene el valor de 1 y -1 si no lo es, la tabla de equivalencias antes mencionada está diseñada para que se elija lo correspondiente al modo de transmisión y luego se cambie el modo de transmisión para el próximo par de datos.

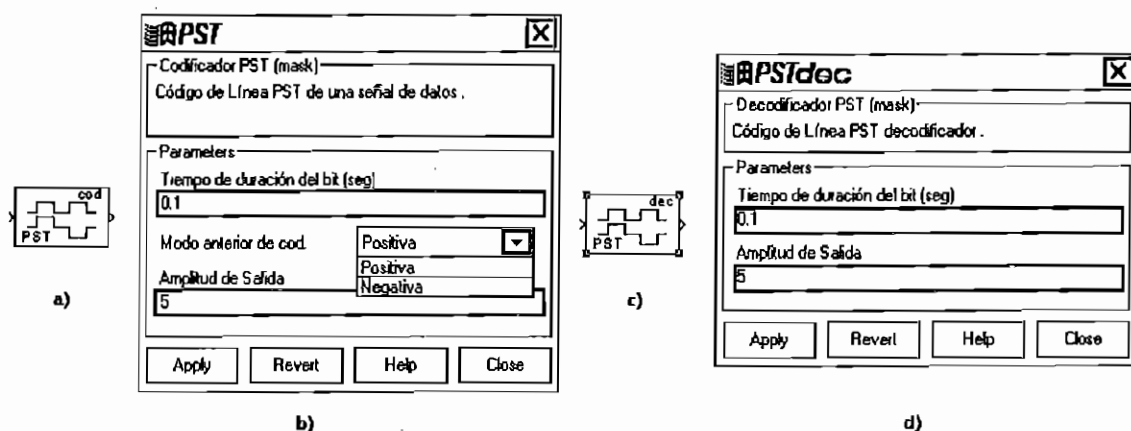


Figura 3.46 Codificación PST: a) Bloque de Simulación codificador b) Plantilla de datos codificación c) Bloque decodificador d) Plantilla de datos decodificación

## b) Decodificación.

Hasta tener datos válidos el bloque decodificador transmite el nivel cero. El proceso en sí es el mismo que la codificación, la tabla de equivalencias se diseñó de tal forma que los datos de modo positivo y negativo tienen su equivalencia adecuada. Se requiere que el usuario indique el "Tiempo de duración del bit" y la "Amplitud de Salida" según la figura 3.46d.

### 3.2.14 Código 2B1Q (*Two Binary One Quaternary*).

Este código se emplea para enviar más información por menor ancho de banda en redes ISDN<sup>1</sup> de banda estrecha (*Integrated Service Digital Network*) o Red Digital de Servicios Integrados, debido a que se trata de un código de línea multinivel.

Este código es diferencial, es decir depende del nivel anterior para la codificación. El código 2B1Q tiene cuatro niveles: +1,+3,-1 y -3; a

<sup>1</sup> Ver pág., 410 referencia [2].

cada grupo de dos bits de dato le corresponde un nivel de estos dependiendo del nivel precedente, de acuerdo con la tabla 3.5. En esta tabla se tiene un tipo de equivalencias cuando el nivel precedente fue positivo y otro tipo si fue negativo.

Nivel 2B1Q Previo	Entrada Binaria	Nivel 2B1Q Actual
+1 o +3	00	+1
	01	+3
	10	-1
	11	-3
-1 o -3	00	-1
	01	-3
	10	+1
	11	+3

Tabla 3.5 Equivalencia para el código 2B1Q

A continuación se representa en la figura 3.47 un ejemplo de datos binarios y su respectiva codificación con un estado inicial negativo.

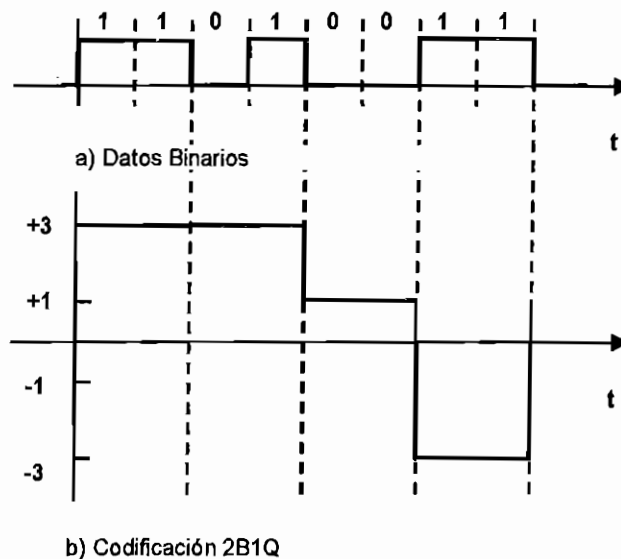


Figura 3.47 Ejemplo de Codificación 2B1Q.

Existe otro tipo de codificación 2B1Q no diferencial pero la simulación se desarrolló para la codificación diferencial únicamente.

## **Simulación del código 2B1Q.**

La simulación se diseñó en los archivos `db1qcod.m` y `db1qdec.m` para ser utilizados con el bloque *S-function*. Debido a que el código necesita conocer los bits de información en pares, la codificación tiene un retardo de uno y medio tiempo de bit y para la decodificación el retardo es de dos tiempos de bit con respecto al dato ingresado al codificador.

Las redes ISDN de banda estrecha por lo general utilizan par trenzado o *twisted pair*, y este código tiene la ventaja de que si se invierten las conexiones de los dos cables de cobre la decodificación es la misma, lo que puede comprobarse en la simulación.

### **a) Codificación.**

Se almacenan los datos ingresados en dos variables ( $x(4)$  primer dato y  $x(5)$  segundo dato), la lectura de los datos válidos se realiza a medio tiempo de bit, cada tiempo de bit. Dentro del intervalo de tiempo de lectura del dato de interés se lleva a cabo la codificación. Se almacena el dato válido en  $x(4)$  y se desplaza su contenido anterior a  $x(5)$ , luego se transmite el valor contenido en  $x(8)$ ; que en un inicio tiene el valor de cero y posteriormente el valor codificado. Con un contador se determina si es el segundo dato recibido, si es este caso se almacena en  $x(8)$  el valor correspondiente según la tabla 3.5. En la figura 3.47 b. se solicitan los valores de "Tiempo de duración del bit" el cual se almacena en la variable `samp` y la "Polaridad previa" que se almacena en `um` y en  $x(6)$ , esta última toma el valor de 1 si es positiva y -1 si es negativa; si el valor de  $x(6)$  es menor que cero se multiplica  $x(8)$  por -1, luego se actualiza el valor de  $x(6)$  con el contenido de  $x(8)$ . Finalmente se transmite el valor de  $x(8)$ .

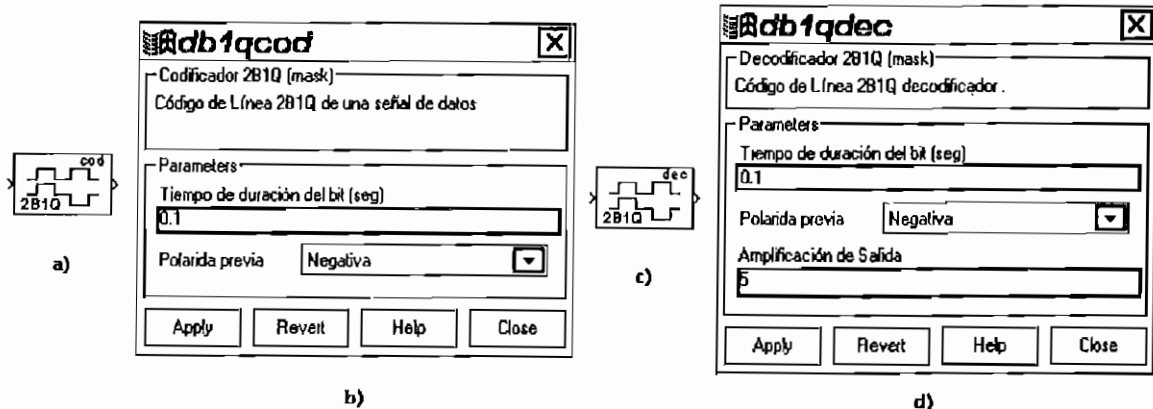


Figura 3.48 Codificación 2B1Q: a) Bloque de Simulación codificador b) Plantilla de datos codificación c) Bloque decodificador d) Plantilla de datos de la codificación.

## b) Decodificación.

La lectura de los datos en el bloque decodificador se la hace cada medio tiempo de bit y se denomina tiempo de muestreo del bloque decodificador. Cada dos tiempos de muestreo (con un retardo inicial de cuatro tiempos de muestreo) se tiene el proceso de decodificación. Se recibe el dato almacenándolo en una variable  $x(4)$  y se transmite el valor de  $x(9)$ , que en un inicio es cero, luego será el último valor decodificado. Cuando se recibe el segundo dato válido es tiempo de almacenar los equivalentes binarios en  $x(8)$  y  $x(9)$  en ese orden; evidentemente esto conforme a la polaridad precedente, la cual posteriormente se actualiza con el valor de  $x(4)$  y en este mismo intervalo de tiempo se transmite  $x(8)$ . El tiempo de muestreo es la mitad del valor  $\text{samp}$  que es el dato pedido al usuario como "Tiempo de duración del bit", según la figura 3.48 d, de la misma manera se solicitan los valores de "Polaridad previa" (variable  $um$ ) y "Amplificación de Salida" en la variable  $ampp$ .

### 3.1.15 Código mBnB.

Es una nueva clase de códigos de línea<sup>1</sup>, que ha sido desarrollado para codificar  $m$  bits de datos en grupos de  $n$  bits binarios, donde  $m < n$ . Debido a que se pueden elegir  $2^n$  valores codificados en cada grupo, existe la posibilidad de utilizar esa información como medios de control de la transmisión. Un ejemplo de esto es el código 4B5B ( $m=4$  y  $n=5$ ), empleado en redes de fibra óptica, en el cual se toman de las 32 palabras de 5 bits, 16 para la representación de datos y las restantes para control de transmisión.

El código de línea 4B5B tiene 16 símbolos para representar 16 datos binarios (0 a F), 8 símbolos de control (Q, H, I, J, K, T, R, S) y 8 símbolos de violación (V). La codificación de los símbolos está diseñada de tal manera que en condiciones normales nunca se tenga cuatro ceros consecutivos, es necesario que se mantenga una buena sincronía en la transmisión. Los símbolos de violación indican que el receptor tiene cuatro ceros consecutivos. La tabla 3.6 presenta la distribución de los símbolos.

Se tiene otro ejemplo de esta clase de códigos de línea, como el 5B6B, que tiene únicamente símbolos para transmisión de datos y no presenta símbolos que cumplan otra función. Este código se representa en la tabla 3.7, donde se puede observar que los símbolos de cuatro pulsos se alternan con los de dos pulsos para mantener el nivel de DC.

---

<sup>1</sup> Tomado de la referencia [5], págs 394-397.



**Campo de Datos**

Datos	Símbolo
-------	---------

0 (binario 0000)	11110
1 (binario 0001)	01001
2 (binario 0011)	10100
3 (binario 0101)	10101
4 (binario 0100)	01010
5 (binario 0101)	01011
6 (binario 0110)	01110
7 (binario 0111)	01111
8 (binario 1000)	10010
9 (binario 1001)	10011
A (binario 1010)	10110
B (binario 1011)	10111
C (binario 1100)	11010
D (binario 1101)	11011
E (binario 1110)	11100
F (binario 1111)	11101

**Símbolos de Control**

Q 00000  
H 00100  
I 11111  
J 11000  
K 10001  
T 01101  
R 00111  
S 11001

Nota: (algunos símbolos V pueden tomarse como H)

**Símbolos de Violación**

V. H 00001  
V. H 00010  
V 00011  
V 00101  
V 00110  
V. H 01000  
V 01100  
V. H 10000

Tabla 3.6 Tabla de símbolos del código de Línea 4B5B

Datos	Símbolo	Datos	Símbolo
00000	011101/100010	10000	111010/000101
00001	101110/010001	10001	100011
00010	010111/101000	10010	100101
00011	000111	10011	100110
00100	101011/010100	10100	101001
00101	001011	10101	101010
00110	001101	10110	101100
00111	001110	10111	011011/100100
01000	110101/001010	11000	110001
01001	010011	11001	110010
01010	010101	11010	110100
01011	010110	11011	101101/010010
01100	011001	11100	111000
01101	011010	11101	110011/001100
01110	011100	11110	101101/010010
01111	110110/001001	11111	100111/011000

Tabla 3.7 Tabla de símbolos del código de Línea 5B6B

## **Simulación del código mBnB.**

La simulación de los códigos 4B5B y 5B6B se la realiza a través de la S-function con los archivos para la codificación qb5bcod.m y cb6bcod.m, mientras que para la decodificación con los archivos qb5bdec.m y cb6bdec.m respectivamente.

En la codificación se tiene más número de bits de salida que los datos ingresados a cada bloque, de tal manera que se diseñaron los bloques de codificación para que el tiempo de duración total de los  $n$  bits sean el mismo que el de los  $m$  bits, es decir el tiempo de cada  $n$  bit es igual a  $n$  veces el tiempo de duración de un bit dividido para  $m$ .

Tanto los bloques codificadores como los decodificadores (desarrollados) tienen un tiempo de muestreo igual al tiempo de bit dividido para  $n$ . La salida de los bloques de simulación mientras no se han recibido los datos apropiados a ser codificados o decodificados, será siempre cero y solo puede alterarse esta condición cambiando el valor de la variable (elemento del vector  $x$ , es decir  $x(2)$ ) denominada "dato a transmitirse".

Como en esencia se diseñaron de la misma forma los procesos de codificación y decodificación para los códigos 4B5B y 5B6B se presenta a continuación la descripción general de estos procesos. Los datos requeridos para la simulación son "tiempo de bit" y "Amplitud de salida", según las plantillas de datos de las figuras 3.48 y 3.49.

### **a) Codificación.**

Se tienen tres partes en la codificación, una es la interpretación de la señal de entrada para tomarse como uno o cero, la segunda parte es la comparación de los datos ingresados con la tabla de equivalencias y la última parte es la transmisión de los datos a la variable de "dato a transmitirse".

Si la señal de entrada es mayor o igual que uno se le asigna el valor de uno y si es menor se le asigna el valor de cero. Luego se hace un control de lectura de datos, el cual toma el dato válido a dos tiempos de muestreo en el caso 4B5B y a 3 tiempos de muestreo para el código 5B6B. Dentro de este tiempo de dato válido se lleva un conteo de datos válidos ingresados, cuando ya se tienen los  $m$  bits se compara los valores con los de la tabla de equivalencia respectiva (tabla 3.6 o 3.7) y se almacena la equivalencia pertinente.

La transmisión de las equivalencias de la tabla se la hace a través de la variable antes mencionada. El valor de la variable a transmitirse es actualizado si ya se ingresaron los  $m$  bits y si está dentro del tiempo de duración del bit de salida.

### **b) Decodificación.**

El proceso de decodificación es prácticamente el mismo que para la codificación, con la diferencia que se debe controlar el número de datos válidos ingresados para realizar la comparación con la tabla de equivalencias respectiva y el tiempo de duración de los datos transmitidos.

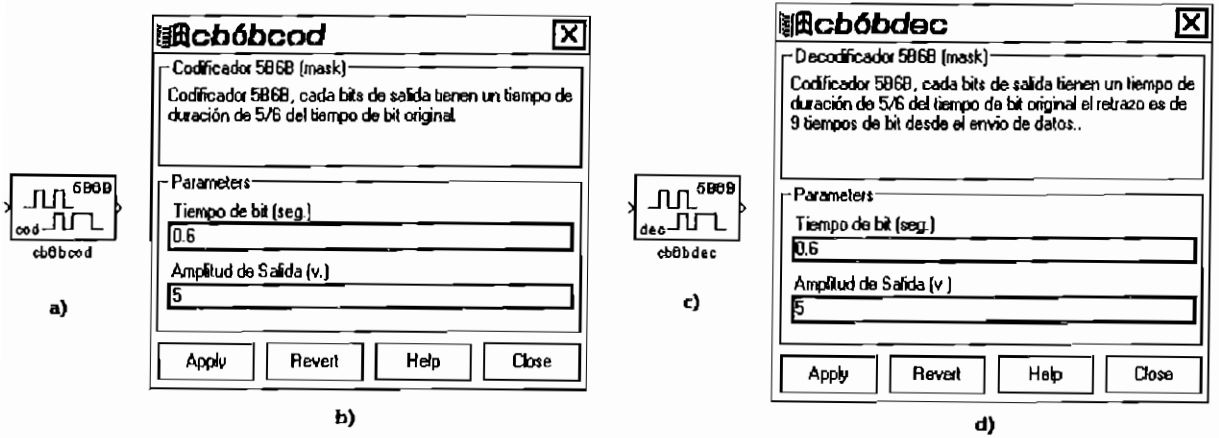


Figura 3.50 Codificación 5B6B: a) Bloque de Simulación codificador b) Plantilla de datos codificación c)Bloque decodificador d) Plantilla de datos de la codificación.

### 3.1.16 Código MS43(Monitoring State 4 3).

El código MS43, propuesto por Franaszek<sup>1</sup>, tiene un concepto similar al 4B-3T descrito anteriormente. Este código produce una sustitución de grupos de 4 dígitos binarios por grupos de tres dígitos ternarios. En la figura 3.51 se presenta el diagrama de transición de estados y en la tabla 3.8 la correspondiente codificación.

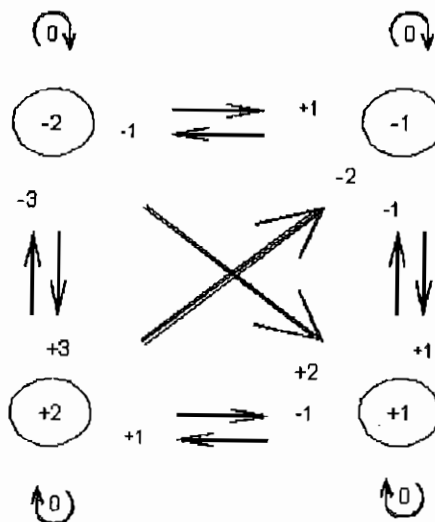


Figura 3.51 Diagrama de transición de estados del código

<sup>1</sup> P.A. Franaszek. "Sequence - State Coding for Digital Transmission", B.S.T.J., V-47, este es el documento fuente según la referencia [6].

Entrada Binaria	Palabra Ternaria				Disparidad acumulada
	Modo 1	Modo 2	Modo 3	Modo 4	
0000	+++	- + -	- + -	- + -	+3,-1,-1,-1
0001	++0	00-	00-	00-	+2,-1,-1,-1
0010	+0+	0-0	0-0	0-0	+2,-1,-1,-1
0011	0-+	0-+	0-+	0-+	0
0100	0++	-00	-00	-00	+2,-1,-1,-1
0101	-0+	-0+	-0+	-0+	0
0110	-+0	-+0	-+0	-+0	0
0111	-++	-++	--+	--+	+1,+1,-1,-1
1000	+ - +	+ - +	+ - +	- - -	+1,+1,+1,-3
1001	00+	00+	00+	- - 0	+1,+1,+1,-2
1010	0+0	0+0	0+0	- 0 -	+1,+1,+1,-2
1011	0+-	0+-	0+-	0+-	0
1100	+00	+00	+00	0 - -	+1,+1,+1,-2
1101	+0-	+0-	+0-	+0-	0
1110	+ - 0	+ - 0	+ - 0	+ - 0	0
1111	++-	++-	+ - -	+ - -	+1,+1,-1,-1

Tabla 3.8 Regla de codificación MS43

Para realizar la codificación se parte del hecho de que existen cuatro estados posibles de polaridad acumulada (+2,+1,-1,-2), por lo tanto para cada uno de estos estados se establece una tabla de codificación ternaria que se puede tener para cada entrada de cuatro dígitos binarios; con esto, se logra simplificar la codificación que consistirá en elegir la palabra ternaria correcta en uno de los cuatro modos establecidos, tal como se indica a continuación.

Disparidad acumulada	Palabra Ternaria
-2	Modo 1
-1	Modo 2
+1	Modo 3
+2	Modo 4

Para la decodificación puede simplificarse la tabla 3.8 en la tabla 3.9, con esto si la señal de entrada corresponde al modo negativo o positivo de un bloque de 4 bits, se toman estos 4 bits para ser transmitidos.

A continuación se plantea un ejemplo de la codificación MS43 con una disparidad inicial de +2:

Entrada Binaria	0000	1000	0110	0111	1011	0101	1111	0000	0000	0000	1101
Salida Ternaria	-+-	+--+	-+0	--+	0+-	-0+	+--	--+	+++	--+	+0-
Disparidad Actual	+1	+2	+2	+1	+1	+1	-1	-2	+2	+2	+1

Entrada Binaria	Palabras Ternarias	
	Modo Positivo	Modo Negativo
0000	+++	-+-
0001	++0	00-
0010	+0+	0-0
0011	0-+	0-+
0100	0++	-00
0101	-0+	-0+
0110	-+0	-+0
0111	-++	--+
1000	+ - +	- - -
1001	00+	- - 0
1010	0+0	- 0 -
1011	0+-	0+-
1100	+00	0--
1101	+0-	+0-
1110	+ - 0	+ - 0
1111	++-	+ - -

Tabla 3.9 Palabras binarias y ternarias MS43 reducidas

## Simulación del código MS43

Se diseñó un bloque de simulación para la codificación y otro para la decodificación. Se empleó en la simulación el bloque *S-function*, por lo tanto la programación tanto de la codificación como de la decodificación se encuentran en los archivos `ms43cod.m` y `ms43dec` respectivamente.

Como se puede observar en la figura 3.52 los bloques de simulación permite al usuario señalar el "Tiempo de bit" y modificar la "Amplitud de salida". Adicionalmente el usuario debe seleccionar una polaridad acumulada para la codificación.

Se presenta un retraso en la señal codificada y en la decodificada con respecto a los datos iniciales de idéntica forma que en el código 4B3T. Además el proceso de la manipulación de las señales en la codificación y decodificación del código MS43 también es muy similar que el código 4B3T, únicamente difieren en las tablas de las reglas de codificación y decodificación.

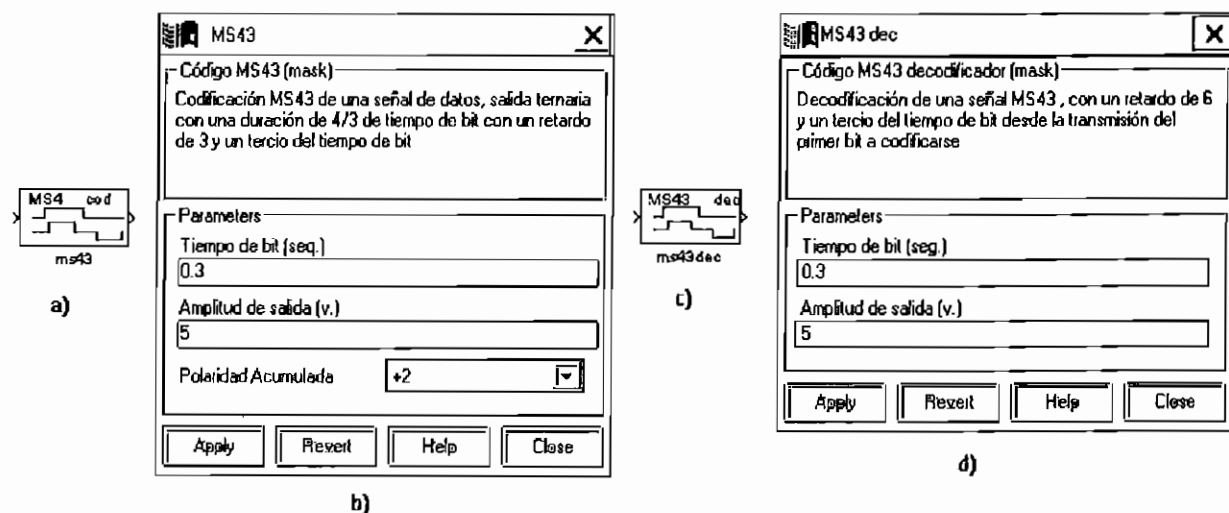


Figura 3.52 Codificación MS43: a) Bloque de Simulación codificador b) Plantilla de datos de la codificación c) Bloque decodificador d) Plantilla de datos de la decodificación.

### 3.3 DENSIDAD ESPECTRAL DE POTENCIA DE LOS CÓDIGOS DE LÍNEA.

La Densidad Espectral es una manera de medir la energía de una señal al ser transmitida, además permite observar las características de frecuencia de la señal para elegir el medio de transmisión. Con estos propósitos se modificó el bloque "*Power spectral density*" de la librería de "*sinks/ extras*", como se indica en la sección 2.2.4; este bloque modificado se denomina "Densidad Espectral de Potencia" y tiene las características gráficas para el estudio de los códigos de línea.

A continuación se tiene un ejemplo de simulación del código de línea Manchester, con su respectivo resultado de la Densidad Espectral de Potencia.

#### 3.3.1 Ejemplo de Simulación de los códigos de Línea.

Para el ejemplo de aplicación de se ha tomado el código Manchester, en este ejemplo se presenta tanto la codificación como la decodificación. El modelo de aplicación se denominó manchesterdec y utilizando este nombre en la línea de comandos puede ejecutarse y modificarse o tomando la opción ejemplos del programa "comdig" bajo la selección Bifase.

Este modelo se construyó principalmente a partir de las librerías *codecs*, *decods* y *fuentes*<sup>1</sup> de una manera similar al modelo descrito en la

---

<sup>1</sup> La librería *fuentes* no es la misma que *source*, sino la descrita en la sección 2.1.1.1.



sección 1.2.9. Primeramente se abre un nuevo modelo, llamado por defecto *Untitled*, luego se coloca el bloque *mux* de la librería *connections*, más tarde se colocan 3 bloques *Fcn* de la librería *nonlinear*, se interconectan estos bloques con el bloque *mux* como se indica en la parte derecha de la figura 3.53. Al seleccionar con ayuda del ratón estos cuatro bloques (ya que en un inicio solo están los tres bloques *Fcn* y el bloque *mux*) y después de tomar la opción del menú *Edit* que corresponde a crear subsistemas, se genera de esta manera el nuevo bloque que se denomina *muxm* (se da este nombre). Luego se da la apariencia de tamaño y color como se ve en la parte izquierda de la figura 3.53. De la librería fuentes (figura 2.1) se toma la fuente "Datos digitales secuenciales". Se toma luego el bloque manchester de la librería codecs (figura 3.1) y manchester Dec de la librería decods (figura 3.2). Por último se toma el bloque *scope* de la librería *sinks*.

Una vez colocados los bloques en el modelo, se hacen las conexiones pertinentes y el modelo se ve como en la figura 3.53. El contenido del bloque *muxm* se encuentra representado en la parte derecha de la figura 3.53. Adicionalmente se añade un bloque de información y como se puede observar también se puede agregar texto, como el título del modelo.

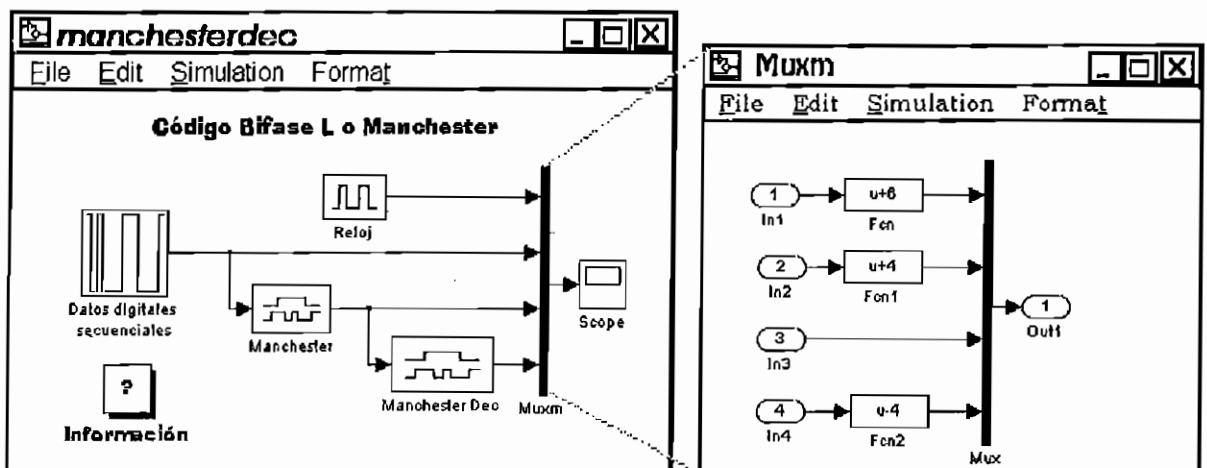


Figura 3.53 Modelo de Codificación y Decodificación del código Manchester

Una vez terminado el aspecto gráfico, es necesario llenar los requerimientos de cada bloque de simulación, se hace doble "click" en cada bloque y aparece una plantilla de datos que el usuario debe completar. A continuación se presentan los datos a llenarse para cada bloque.

#### Datos digitales secuenciales.

En este bloque se ingresa los valores de 0.1 en el "Tiempo de bit", [1 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1] en "Secuencia de datos" y uno en "Amplitud de Salida".

#### Manchester y Manchester Dec

Se tomaron las cantidades de 0.1 para el "Tiempo de bit", 2 en la "Amplitud Salida" y se escogió la opción positiva en el "Tipo de polaridad del 1".

#### Bloques *Fcn*

Estos tres últimos bloques solo tienen que ver con la visualización de los resultados en el bloque Scope, sus contenidos son  $u+6$ ,  $u+4$  y  $u-4$  que solo indican que a la señal de entrada ( $u$ ) se le suma un cierto nivel.

#### Bloque Información

El bloque de información es un bloque denominado *Subsystem* de la librería *connections* y con la técnica de *Masking* se da el signo "?" y luego se da el nombre de "Información". Al hacer un doble *click* en este bloque se tiene una nueva ventana en la cual se ha añadido texto y gráficos con el propósito de ilustrar las características de los códigos.

Antes de la simulación se debe definir los parámetros de la simulación en una plantilla de datos, que se tiene en el menú principal del modelo bajo la opción *Simulation* ; es decir la elección *parameters*. La plantilla de datos se representa en la figura 3.54. Esta plantilla de datos tiene tres partes, la parte *Workspace I/O* tiene que ver con variables de entrada y salida de la simulación por parte de la ventana de comandos, luego lo que corresponde a la parte *Diagnostic*, que presenta un tipo de diagnóstico de la simulación y por último la parte de *Solver* está relacionada con el tiempo de simulación, el tipo de simulación y las características de las salidas de los bloques en la simulación. En la referencia [3] se proporciona una amplia explicación sobre los requerimientos de los parámetros de simulación.

En la figura 3.54 se encuentra la representación de los parámetros más comunes en el uso de la simulación y que el usuario va a estar en contacto permanentemente, como es el caso de la sección *Solver* donde se tienen tres sectores, el primero es el sector de *Simulation Time*, el segundo *Solver options* y el tercero denominado *Output options*.

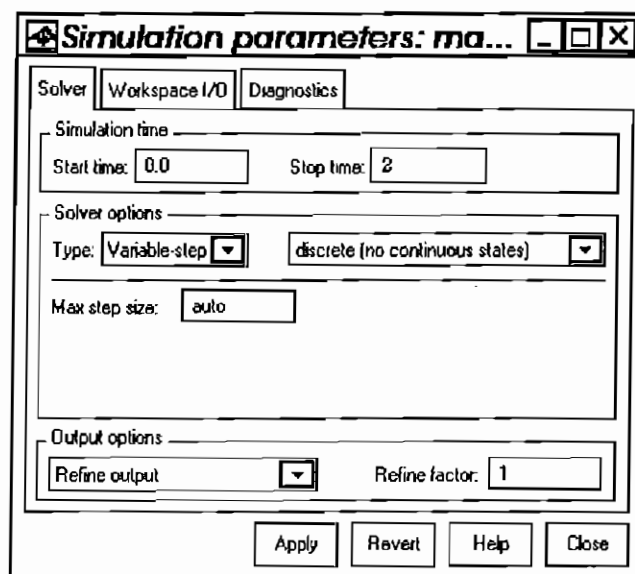


Figura 3.54 Plantilla de datos de los Parámetros de la Simulación

En el sector *Type* existe dos posibilidades *Variable-step* y *fixed-step*, que corresponden a dos grupos de métodos de solución de ecuaciones, tanto diferenciales como en diferencias o las dos al mismo tiempo. Como se explicó en el anexo A, el *SIMULINK* trabaja esencialmente con solución de ecuaciones diferenciales y en diferencias. Cuando se elige la opción *Variable-step* existe campos que permiten modificar valores de los intervalos máximos de integración, el valor de la integración inicial, y las tolerancias absoluta y relativa. Al tomar la opción *fixed-step* solo existe un solo campo que permite ingresar el valor del intervalo de pasos de simulación. En la pagina 61 de la referencia [3] se presenta una tabla de uso de cada uno de los tipos de solución de ecuaciones.

En esta simulación se utiliza la opción *Variable-step* junto con la opción *Discret (no continuous states)*, debido a que en la simulación no se presentan bloques que tengan estados continuos. Además se deja la opción *auto* en *Max step size*, para que el *MATLAB* de su primera respuesta y se puede cambiar el valor del tamaño del intervalo entre puntos de simulación, según las necesidades de dar más puntos a la simulación o no.

Finalmente en la definición de parámetros se tiene la sección *Output options*, la cual solo trabaja con la opción *Variable-step* y controla el espaciamiento entre los puntos de salida de los bloques de simulación. Esta sección contiene tres posibles elecciones; *Refine output*, *Produce additional output*, y *Produce specified output only*. Cuando se elige *Refine output* se tiene que dar un valor en el campo *Refine factor* el cual permite agregar de 1 a más puntos entre los puntos de solución de

la salida de un bloque que se está simulando, con el propósito de tener más información gráfica. En el caso del ejemplo se ha elegido un factor de 1 simplemente, pero cuando se trata de simulaciones con señales seno y coseno de alta frecuencia se recomienda un factor de 2 a 4, aunque esto repercute en el uso de memoria del programa. La opción *Produce additional output*, indica al *SIMULINK* que se calculen valores de la salida a ciertos valores de tiempo adicionalmente a los ya calculados, con un vector en el campo *Output times*. Esto se hace con el propósito de comparar resultados. La opción *Produce specified output only*, indica al *SIMULINK* que se calculen valores de la salida únicamente a ciertos valores definidos por el usuario, para esto se debe definir un vector en el campo *Output times*.

De la misma manera como en el ejemplo de la sección 1.2.9, se señalaron las escalas del bloque *scope*. Obteniéndose el siguiente resultado en la figura 3.55. Este resultado puede copiarse o ser impreso.

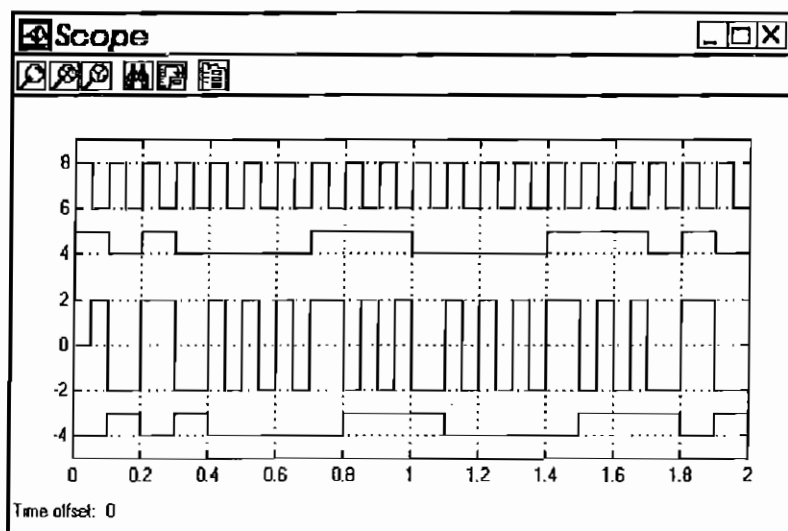


Figura 3.55 Resultado de la Simulación del código manchester.

<sup>1</sup> Para más información de los estados continuos y discretos ver anexo A.

En la simulación, estas señales representadas en esta figura 3.55 cada una tiene un color, según el orden con que ingresan al bloque *mux*; es decir la primera señal tiene el color amarillo, magenta para la segunda, turquesa para la tercera y rojo para la cuarta. Por lo tanto la primera señal (de arriba hacia abajo) corresponde a la señal de reloj, la segunda a los datos binarios, la tercera a la codificación manchester y la cuarta a la decodificación.

### 3.3.2 Ejemplo de la obtención de la Densidad Espectral de Potencia de un Código de Línea.

De la misma manera como se desarrolló el ejemplo anterior se genera un modelo llamado *manchesterpsd* (figura 3.56). En el modelo se colocan de la librería fuentes el bloque Datos digitales aleatorios, de la librería codecs el bloque manchester y de la librería análisis el bloque Densidad Espectral de Potencia.

Los datos de los bloques de fuente de datos, de la codificación y parámetros de simulación se definen del mismo modo que en el ejemplo anterior. Con un tiempo de bit de 0.01segundos, 300 datos en la fuente aleatoria, amplitud de 1 en la fuente de datos, 2 unidades de amplitud de la señal manchester y tipo de polaridad 1<sub>L</sub> positiva, se procede a ingresar los datos necesarios para la obtención de la Densidad Espectral de Potencia, los que se muestran en la figura 3.57.

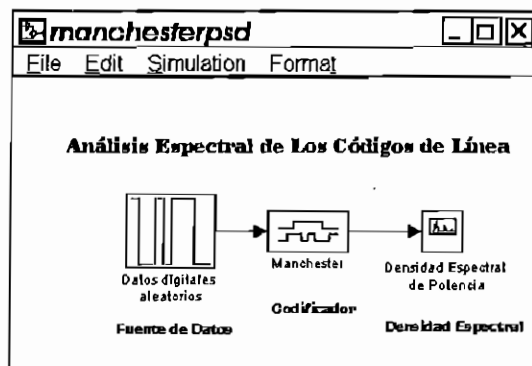


Figura 3.56 Modelo para la obtención de la Densidad Espectral de Potencia del código Manchester

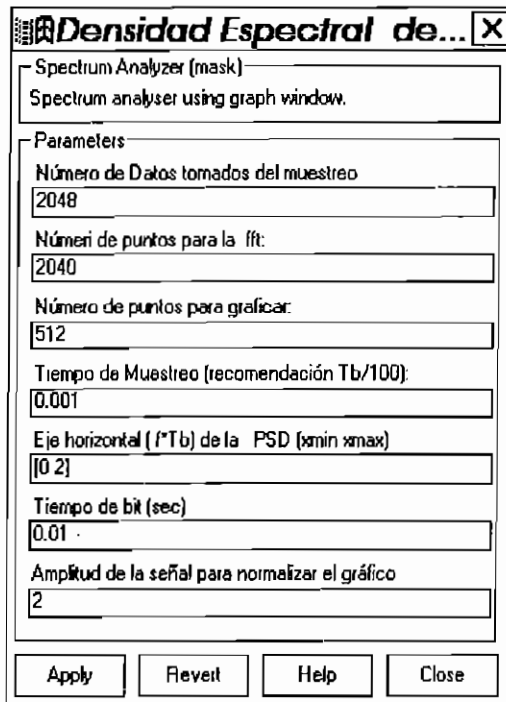


Figura 3.57 Plantilla de datos del Bloque de Densidad Espectral de Potencia.

Una vez que se ha cumplido con estos requisitos se tiene al iniciar la simulación una ventana con las respuestas respectivas, de acuerdo a los puntos analizados, así en la figura 3.58 se tiene el resultado de esta simulación.

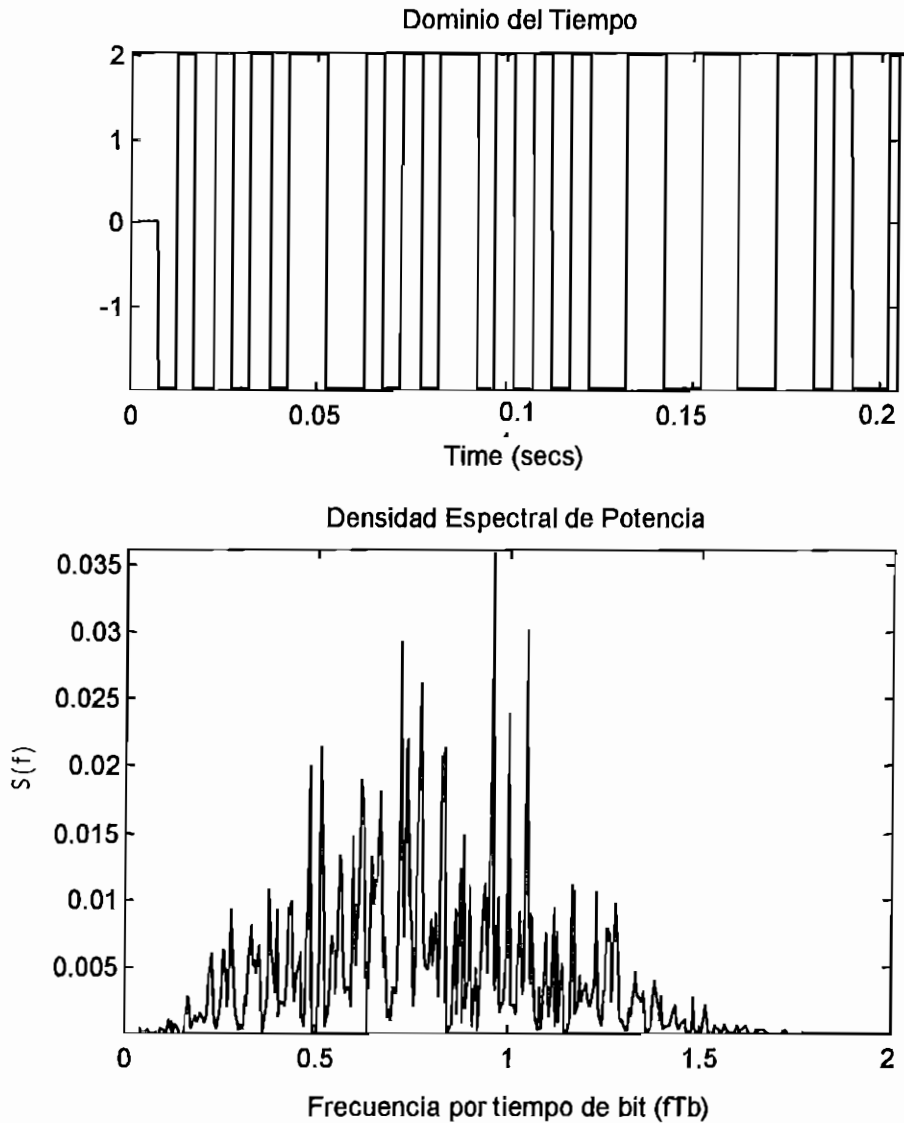


Figura 3.58 Resultados del Bloque Densidad Espectral de Potencia

En la versión para estudiantes del *MATLAB* y *SIMULINK* es necesario no sobrepasar un límite de un número de elementos de una matriz y un número de 50 bloques en un modelo. Es necesario que en la opción *Workspace I/O* no se guarden datos de tiempo y señal de salida para que



no se sobrepase el límite de elementos de un vector. De esta forma se puede simular el modelo más tiempo.

### **3.4 EJEMPLO DE SIMULACIÓN DE UN CANAL DE TRANSMISIÓN CON RUIDO GAUSSIANO BLANCO EN UNA TRANSMISIÓN EN BANDA BASE.**

Este tipo de simulación se hizo con el propósito de ver el efecto de ruido blanco en la transmisión en banda base. Se puede colocar el bloque de simulación del canal después de cualquier codificador y observar los resultados. El usuario puede variar los valores de frecuencia de paso del canal y la potencia de ruido.

Además es muy importante el observar el diagrama del ojo de la señal afectada por el ruido, realizar mediciones y comparaciones.

En el gráfico 3.59 se indica como se llenaron los valores pedidos para esta simulación. Los valores para el canal ruidoso que el usuario puede modificar son la potencia de ruido y la frecuencia máxima de transmisión del canal, el tiempo de bit debe ser igual al de la señal de origen y este dato es utilizado tanto para el tiempo de muestreo del filtro como del ruido que conforman el bloque de simulación del canal. La misma recomendación sobre el tiempo de bit se aplica al diagrama del ojo. El usuario puede también en el diagrama del ojo modificar escalas con propósitos de medición.

En la figura 3.60 se tiene el modelo desarrollado y el respectivo resultado.



Figura 3.59 Plantillas de datos del bloque Canal de ruido y Diagrama del ojo respectivamente

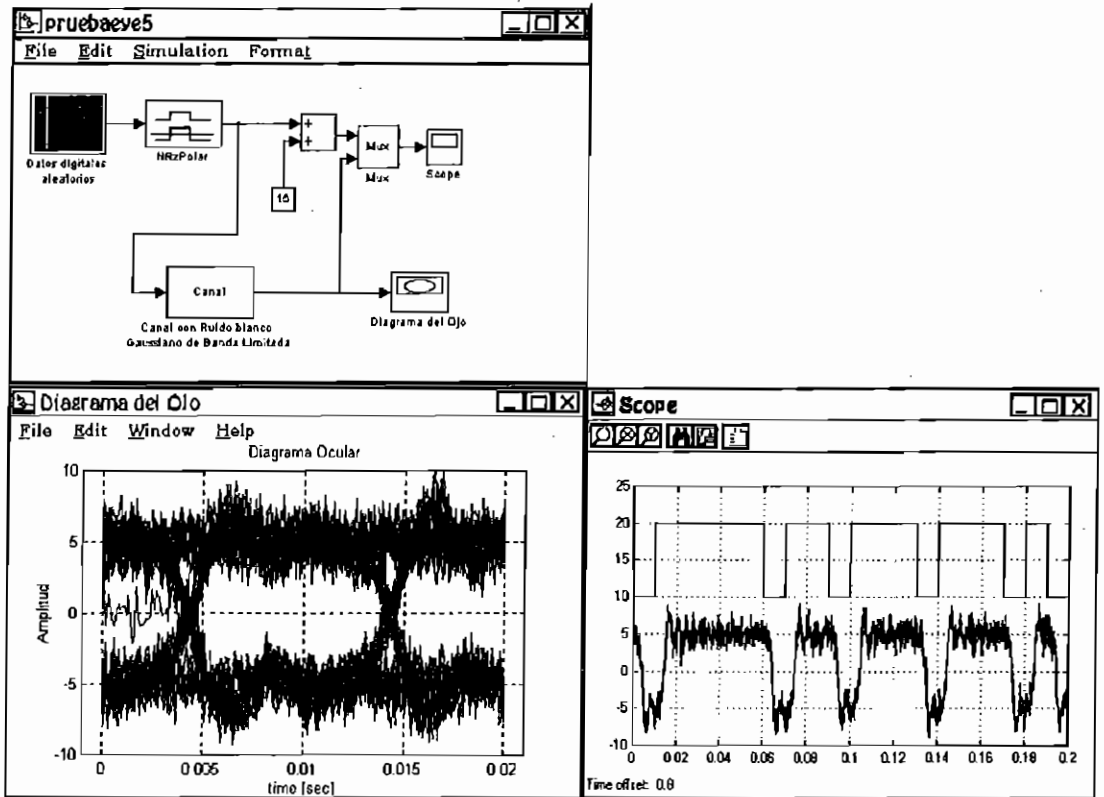


Figura 3.60 Ejemplo de Simulación de Canal con ruido blanco y diagrama del ojo

### 3.5 REFERENCIAS.

1. Strembler F.G, Introducción a los Sistemas de Comunicación, Addison Wesley, México 1993
2. Leon W. Couch II, Digital and Analog Communication Systems, four edition, Macmillan Publishing Company New York 1993.
3. James B. Dabney y Thomas L. Harman. "The Student Edition of SIMULINK® User's Guide", Prentice Hall, 1997.
4. Nelson Avila, "Diseño y Construcción de un Codec Didáctico", Tesis de Grado previa a la Obtención del Título de Ingeniero en Electrónica y Telecomunicaciones, EPN. 1994.
5. Bellamy J., Digital Telephony, John Wiley & Sons, Nuew York, 1991.
6. Vidaller J. y Otros, Transmision de Datos, E.T.S. Ingenieros Telecomunicaciones Universidad Politécnica de Madrid, Madrd 2ª ed., 1979.

## **CAPÍTULO 4. MODULACIÓN DIGITAL Y SIMULACIÓN.**

En el capítulo anterior se trató la transmisión de datos en Banda Base, pero la transmisión se ve restringida en cuanto al ancho de banda limitado, el ruido a bajas frecuencias afecta de manera importante, la transmisión por múltiples canales tiene mucha interferencia por estar en el mismo rango de frecuencias y es en estos casos que la modulación digital aventaja a la transmisión en banda base.

Un sistema de comunicación digital toma las señales en Banda Base y las modula mediante una señal portadora utilizando métodos tales como AM(modulación de amplitud), PM(modulación de fase) y FM(modulación de frecuencia) con el fin de adaptar la señal que lleva la información a los diferentes medios de transmisión (par trenzado, cable coaxial, fibra óptica, espacio libre, etc.). Al ser la señal modulante una señal binaria, los diferentes métodos de modulación adquieren otro tipo de denominación como conmutación de amplitud (ASK, Amplitud Shift Keying), conmutación de fase (PSK Phase Shift Keying) y conmutación de frecuencia (FSK, Frequency Shift Keying).

En este capítulo se tratan brevemente estos métodos de modulación y se describen los bloques desarrollados para la simulación. Todos los bloques diseñados para cada tipo de modulación se encuentran disponibles a través de la opción Librerías del menú principal que se presenta con el comando "comdig", lo que se tratará con más detalle en la sección 4.6.

## 4.1 MODULACIÓN ASK.

En la modulación de amplitud con modulante digital, la amplitud de una señal portadora de alta frecuencia se conmuta entre dos o más valores. Se trata de una modulación lineal que puede generarse según la ecuación 4.1. El proceso general de una modulación lineal se muestra en el gráfico 4.1, en donde se representa la señal modulante  $g(t)$  multiplicada por un índice de modulación  $m$ , sumada con un nivel unitario y luego multiplicada por un señal senoidal de amplitud uno y alta frecuencia  $f_c$  (portadora).

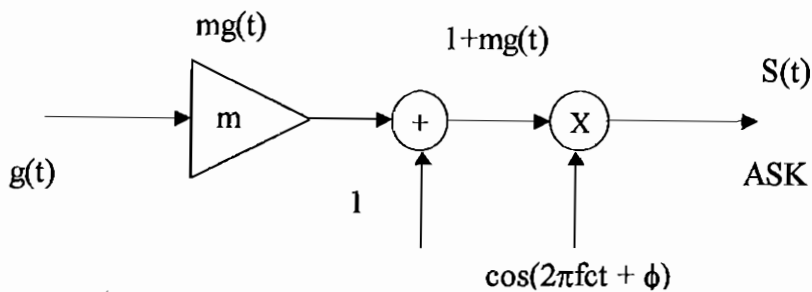


Figura 4.1 Modulación Lineal en general

$$S_{AM}(t) = [1 + m \cdot g(t)] \cos(2 \cdot \pi \cdot f_c \cdot t + \phi) \quad (4.1)$$

El índice de modulación  $m$  puede variar entre cero y uno, para evitar distorsión de envolvente ya que valores negativos de  $m$  solo provocarían un cambio de fase,  $f_c$  es la frecuencia de la portadora y  $\phi$  es el ángulo de fase. La señal modulante  $g(t)$  puede ser del tipo binario (puede tomar dos niveles), así por ejemplo si toma valores de 0 y 1 con el índice de modulación unitario se tienen los siguientes resultados en las ecuaciones 4.2 y 4.3.

$$\text{Si } g(t)=1 \Rightarrow S_{ASK}(t) = 2 \cdot \cos(2 \cdot \pi \cdot f_c \cdot t + \phi) \quad (4.2).$$

$$\text{Si } g(t)=0 \Rightarrow S_{ASK}(t)=\cos(2 \cdot \pi \cdot f_c \cdot t+\phi) \quad (4.3).$$

En la figura 4.2 se representa una señal modulada con la secuencia de datos 1001, con una duración de bit de  $T_b$ .

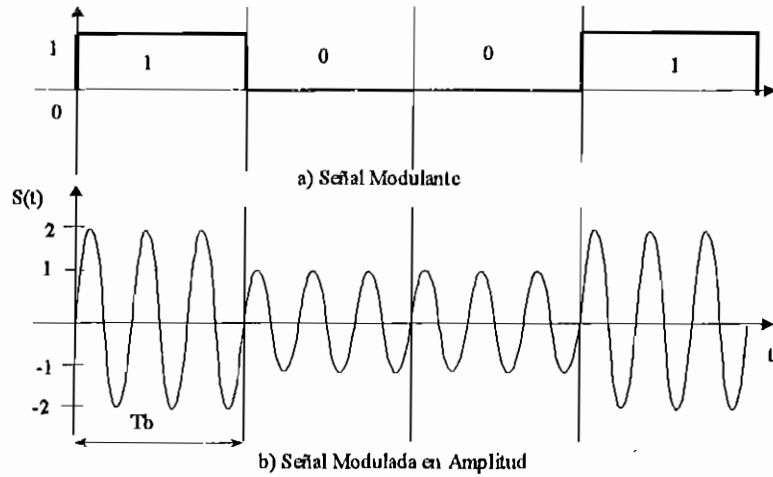


Figura 4.2 Ejemplo de Señal AM con Modulante Binaria

Si la señal  $g(t)$  tiene componente continua se tiene modulación de doble banda lateral con o sin portadora<sup>1</sup>. La técnica más empleada para generar señales ASK es la de modulación por doble banda lateral con portadora suprimida o también llamada modulación balanceada, cuya señal modulante es una señal binaria. Si la señal binaria modulante es de tipo unipolar se tiene la Modulación OOK (*On-Off Keying*) y si es del tipo polar se denomina Modulación PRK (*Phase-Reversal Keying*)<sup>2</sup> o BPSK (*Binary Phase Shift Keying*).

<sup>1</sup> Ver referencia [6]

<sup>2</sup> Ver referencia [1]

#### 4.1.1 Modulación On-Off Keying (OOK).

La señal modulada OOK está matemáticamente representada por la ecuación 4.4, donde  $g(t)$  es una señal unipolar de datos como se muestra en la figura 4.3a,  $A_c$  es la amplitud de la señal portadora que por lo general es de valor unitario,  $f_c$  es la frecuencia de la portadora y  $\phi$  es la fase inicial de la portadora.

$$s(t) = A_c \cdot g(t) \cdot \cos(2 \cdot \pi \cdot f_c \cdot t + \phi) \quad (4.4)$$

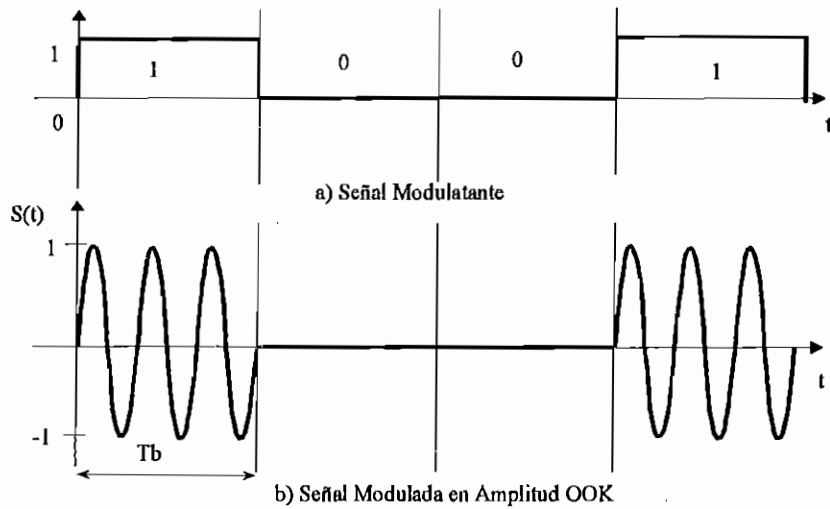


Figura 4.3 Ejemplo de una Señal modulada OOK

En general para una señal modulada en amplitud, la densidad espectral de potencia viene dada por la ecuación 4.5<sup>1</sup>.

$$S(f) = \frac{1}{4} \cdot [S_{eg}(f - f_c) + S_{eg}(-f - f_c)] \quad (4.5)$$

donde  $S_{eg}(f)$  es la densidad espectral de potencia de la función  $eg(t)$ , la cual en la referencia [2] se denomina envolvente compleja y se denotada  $g(t)$  que no tiene el mismo significado que la señal modulante de este

<sup>1</sup> Ver referencia [2], se sustituyo  $S_{eg}$  por  $S_g$  por facilidad de nomenclatura

capítulo, ya que es función de la señal modulante y es diferente para cada tipo de modulación. Para el caso de modulación de doble banda lateral con portadora suprimida la señal  $eg(t)$  es igual a  $g(t)$ .

Por tanto la densidad espectral de potencia para una señal OOK con una probabilidad igual de unos y ceros viene dada para frecuencias positivas por la ecuación 4.6; la cual es resultado de reemplazar la ecuación 3.1 en la ecuación 4.5 ya que la señal modulante es una señal unipolar sin retorno a cero.

$$S_{OOK}(f) = \frac{1}{8} \left[ \delta(f - f_c) + Tb \cdot \left( \frac{\text{sen}(\pi \cdot (f - f_c) \cdot Tb)}{\pi \cdot (f - f_c) \cdot Tb} \right)^2 \right] \quad (4.6)$$

En la figura 4.4 se tiene la representación de esta ecuación para el caso de frecuencias positivas.

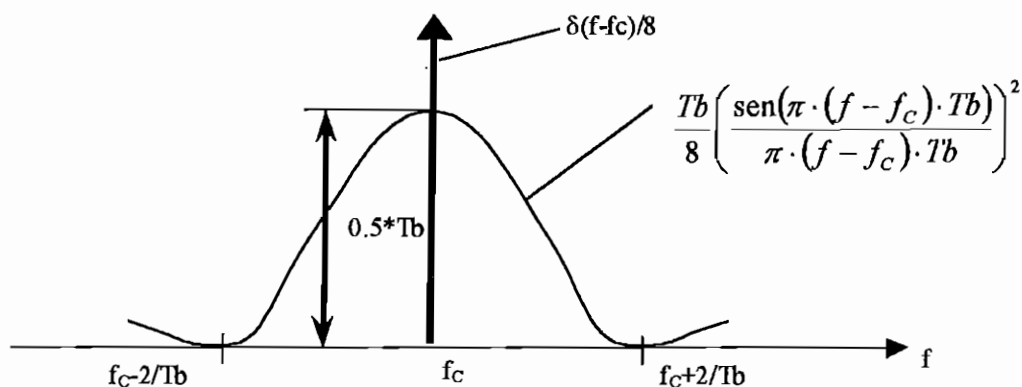


Figura 4.4 Densidad espectral de potencia para la señal OOK (Frecuencias positivas)

#### 4.1.1.1 Simulación de la modulación OOK.

Se diseñó un bloque de simulación de una señal con modulación lineal y luego se desarrolló un bloque que simula únicamente la modulación



OOK. En los dos bloques el usuario debe ingresar el valor de la frecuencia portadora y en el caso de la modulación lineal el usuario puede modificar el índice de modulación.

El ángulo de defasaje  $\phi$  para la simulación se ha tomado por defecto como cero, pero en el bloque diseñado el usuario puede cambiar este valor, no directamente desde la plantilla de datos sino a través del bloque que genera la función senoidal.

### Bloque de Modulación Lineal

Este bloque se diseñó con el fin de cumplir con la ecuación 4.1, es decir ingresar una señal de datos (polar o unipolar) que se multiplica por un índice de modulación ( $m$ ), luego se suma la unidad y se multiplica este resultado por el  $\cos(2\pi fct + \phi)$ . El subsistema desarrollado (representado en la figura 4.5) consiste de un bloque *Gain* de la librería *linear* con ganancia igual a  $m$  (índice de modulación), luego se tiene un bloque *sum* también de la librería *linear* para sumar el resultado del bloque *Gain* con el de un bloque *constant* de la librería *source*. A continuación se tiene un bloque *clock* de la librería *source* que da la variable de tiempo de modulación cuyo resultado se ingresa a un bloque *Fcn* de la librería *No linear* donde se calcula el  $\cos(2\pi fct + \phi)$  ( $t$  proviene del bloque *clock*) y finalmente se tiene un bloque *product* de la librería *no linear* para multiplicar el resultado de la suma con la salida del bloque *Fcn*.

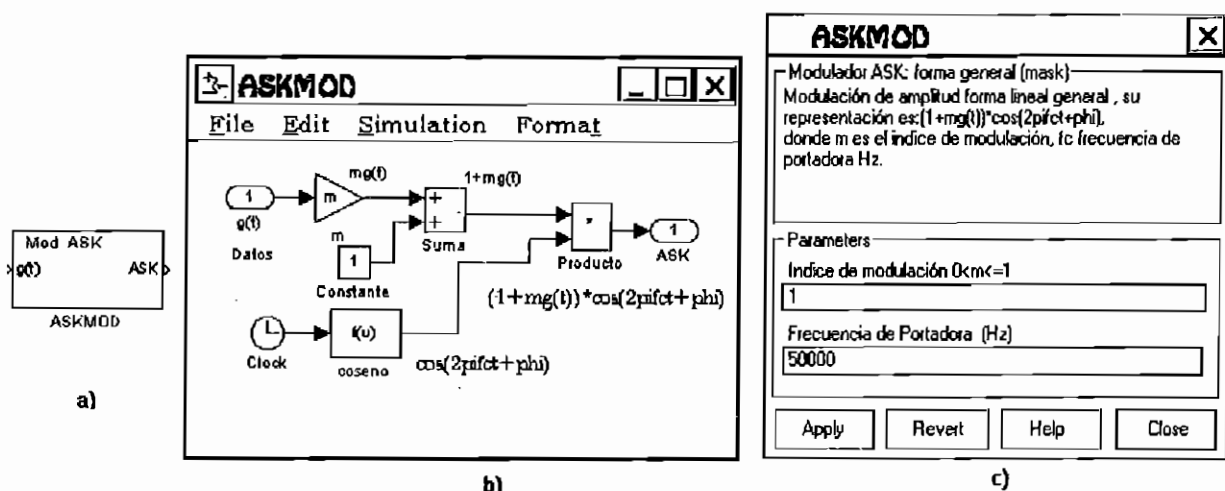


Figura 4.5 Modulación Lineal a) Bloque de Modulación, b) Subsistema contenido y c) Plantilla de datos

Para el caso de ASK se puede tomar como fuente de datos una señal NRZ unipolar proveniente de los bloques de datos secuenciales de la librería fuentes descrita en el capítulo 2, o también se puede tener como señal modulante al resultado del bloque de codificación NRZ polar (sección 3.1.2).

### Bloque de Modulación OOK

Se desarrolló un bloque particular de la señal OOK, este bloque recibe la señal proveniente del bloque fuente y se ejecuta la ecuación 4.4 de una manera más directa.

Como se indica en la figura 4.6 la señal de datos junto con la señal proveniente del bloque *clock* ingresan al bloque *Fcn*, con ayuda del bloque *mux*. Una vez que estas dos señales ingresan en el bloque *Fcn* se procede a definir el contenido de este bloque; la señal de tiempo se llama dentro del bloque *Fcn* como *u[1]* y la señal de datos toma el nombre de *u[2]*, luego la operación interna del bloque *Fcn* se define como

$u[2] \cdot \cos(2 \cdot \pi \cdot f_c \cdot u[1])$ , que cumple con la ecuación 4.4. El parámetro  $f_c$  es la frecuencia en Hz de la señal portadora, el usuario debe especificar este parámetro solamente en la plantilla de datos de la figura 4.6 c).

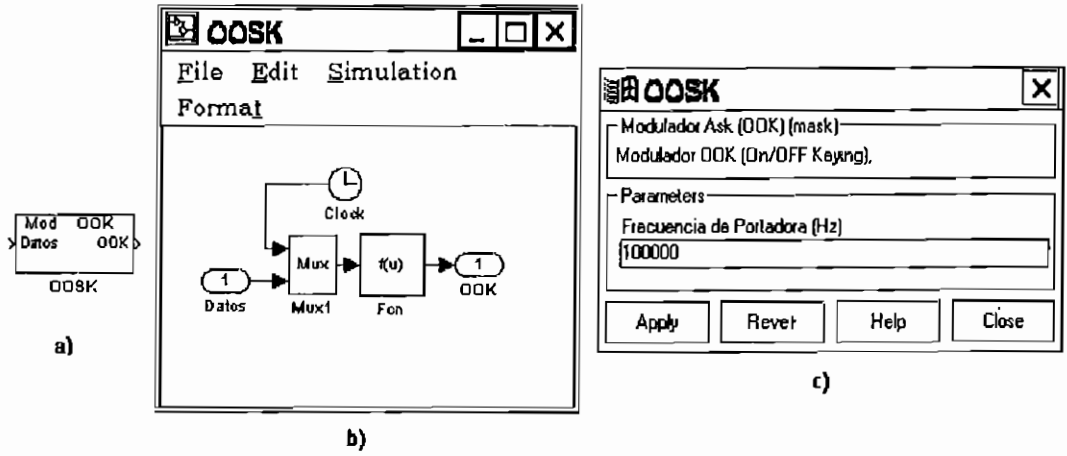


Figura 4.6 Modulación OOK a) Bloque de Modulación, b) Subsistema contenido y c) Plantilla de datos

#### 4.1.2 Modulación PRK (Phase Reversal Keying) o BPSK (Binary Phase Shift Keying).

La señal PRK está representada por la misma ecuación 4.4, donde  $g(t)$  es una señal binaria polar de valores 1 y -1 y  $A_c$  es la amplitud de la señal portadora que se considera unitaria, entonces el resultado puede presentarse así:

$$\begin{aligned}
 \text{si } g(t) = 1 &\Rightarrow s_{\text{PRK}}(t) = \cos(2 \cdot \pi \cdot f_c \cdot t) \\
 \text{si } g(t) = -1 &\Rightarrow s_{\text{PRK}}(t) = \cos(2 \cdot \pi \cdot f_c \cdot t + \pi)
 \end{aligned}$$

Se puede observar que no hay cambio de amplitud sino un cambio de fase entre 0 y  $\pi$ , por lo que se considera esta señal como BPSK o señal 2-PSK. En la figura 4.6 se representa cómo se puede conseguir esta modulación y en la figura 4.7 se tiene un ejemplo.

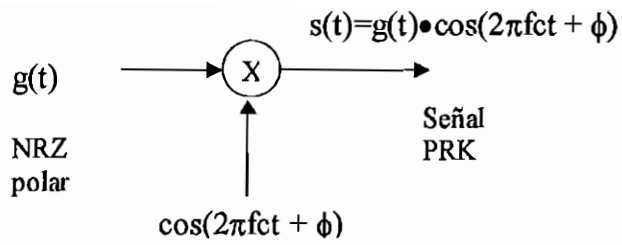


Figura 4.6 Modulación PRK

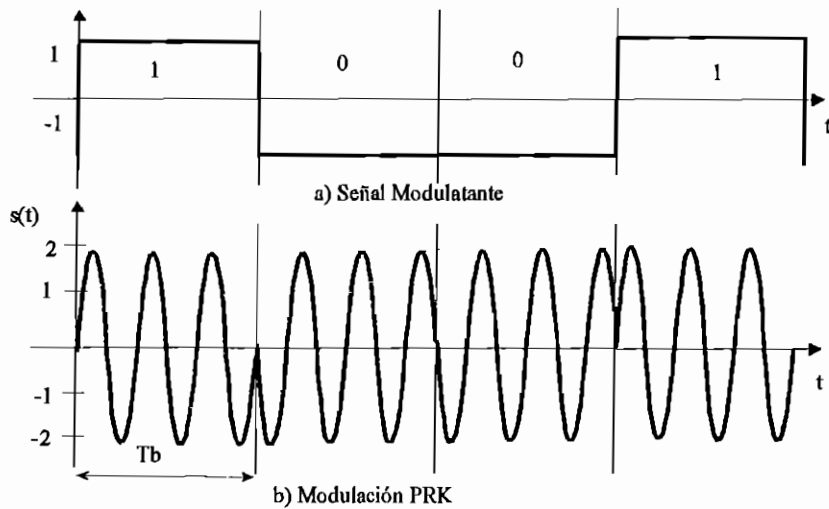


Figura 4.7 Ejemplo de una Señal modulada PRK

Como PRK es en realidad de un caso de modulación de amplitud, la densidad espectral de potencia tiene un tratamiento igual al del caso OOK, es decir que se trata de una señal de doble banda lateral con portadora suprimida, así para frecuencias positivas y datos equiprobables se tiene la ecuación 4. 7.

$$S_{OOK}(f) = \frac{1}{4} \left[ Tb \cdot \left( \frac{\text{sen}(\pi \cdot (f - fc) \cdot Tb)}{\pi \cdot (f - fc) \cdot Tb} \right)^2 \right] \quad (4.7)$$

Esta ecuación es el resultado de la densidad espectral de potencia de una señal NRZ polar trasladada a la frecuencia  $f_c$ , debido a que es una modulación lineal. A continuación se representa esta ecuación en la figura 4.8.

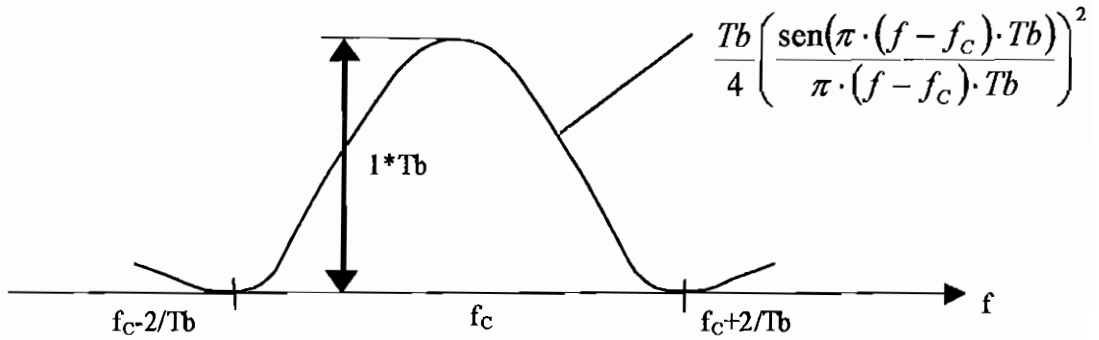


Figura 4.8 Densidad espectral de potencia para la señal PRK (Frecuencias positivas)

#### 4.1.2.1 Simulación PRK

La modulación PRK, se la obtiene del mismo bloque desarrollado para la simulación de la señal OOK, con la única diferencia que entre la fuente de datos y el bloque modulador se coloca el bloque de codificación NRZ polar, con amplitud unitaria.

#### 4.1.3 Demoduladores ASK.

Para realizar la detección de una señal de amplitud modulada existen dos métodos, la detección coherente y la no coherente.

#### 4.1.3.1 Demodulación ASK no coherente.

Una señal modulada en doble banda lateral con portadora, puede demodularse como en el caso analógico por recuperación de la envolvente mediante una rectificación y un filtrado.

Este tipo de demodulación resulta más sensible al ruido, aumentando la probabilidad de error, por lo que no es recomendable excepto cuando se dispone de una relación señal a ruido muy elevada.

En la figura 4.9 se representa la demodulación no coherente, aquí se puede observar como la señal ingresa al rectificador primero y luego al detector de envolvente, que no es más que un filtro pasabajos. Este método se utiliza en ASK con índice de modulación de hasta 100 %.

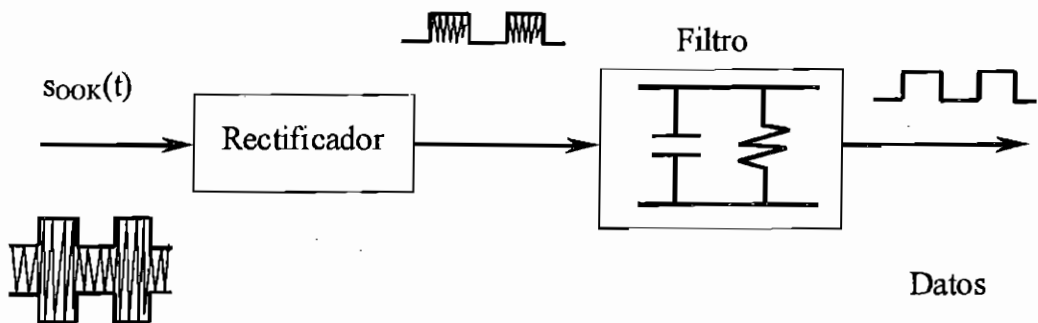


Figura 4.9 Demodulación OOk no coherente

En el capítulo cuarto de la referencia [2], se recomienda que la frecuencia de corte de la red RC, que denominaremos  $f_0$  esté entre dos valores tales que :  $B \ll f_0 \ll f_c$ , donde B es el ancho de banda base de la señal modulante y  $f_c$  es la frecuencia de la portadora.

#### 4.1.3.2 Simulación Demodulación no coherente.

Se diseñó un bloque que permite la detección de envolvente, aquí el usuario debe ingresar la frecuencia de portadora y especificar si la modulación contiene o no portadora.

La señal modulada ingresa al bloque demodulador, luego se toma el valor absoluto y el resultado ingresa a un filtro pasabajos. Si la señal contiene portadora se resta un nivel de 0.5 y después se filtra; el nivel de 0.5 se debe a que la portadora se supone de amplitud unitaria y un índice de modulación unitario, sino debe modificarse esta constante hasta obtener la señal demodulada deseada.

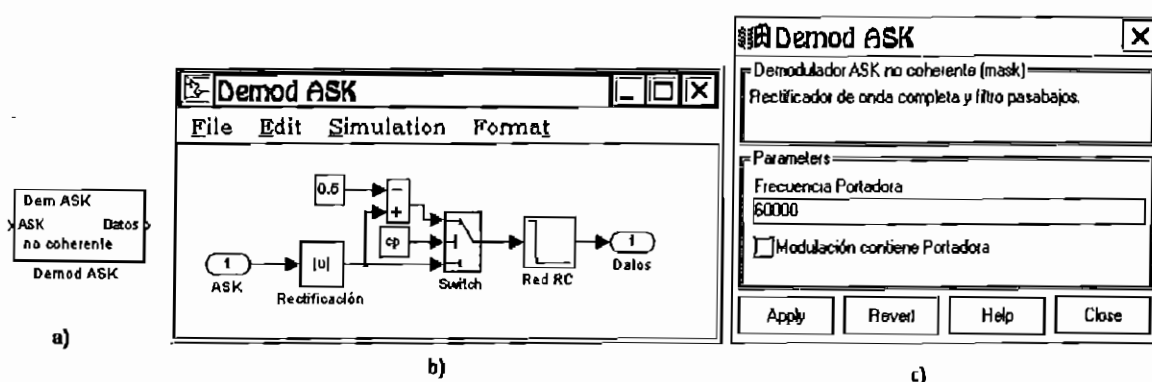


Figura 4.10 Demodulación no coherente a) Bloque de Modulación, b) Subsistema contenido y c) Plantilla de datos.

En la figura 4.10 b, se puede observar como se realiza el proceso de rectificación con el bloque *abs* de la librería *no linear*. Si se utiliza la opción "Modulación contiene portador" el valor en el bloque *cp* es de uno y cero en caso contrario. El valor de *cp* controla el bloque *switch*, cuando vale uno realiza la resta de la señal rectificada con la constante 0.5 y en caso contrario la señal va directamente hacia el filtro.

El filtro si se quiere simular lo más aproximado a una red RC se puede emplear un filtro *Butterworth* de orden uno, o un filtro *chebychev* de orden bajo. Los dos tipos de filtros son más selectivos si son de mayor orden. Estos filtros utilizan funciones del *Signal Processing Toolbox* (*butter* y *cheby1*) que son desarrollados por *Mathworks* y su modo de empleo se encuentra en la referencia [7]. Los bloques de simulación de filtros pueden obtenerse en la opción de librerías del programa de presentación *comdig.m*.

#### 4.1.3.3 Demodulación ASK coherente.

La demodulación coherente presenta más dificultad debido a que se necesita recuperar portadora. La señal modulada se multiplica con la portadora recuperada y luego se aplica este resultado a un filtro pasabajos, tal como se representa en la figura 4.11.

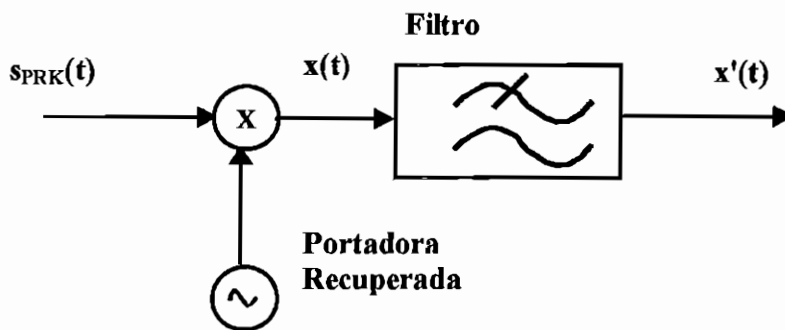


Figura 4.11 Demodulación PRK coherente

Las señales moduladas PRK o BPSK, necesariamente requieren de una detección coherente, en cambio las señales ASK-OOK, pueden emplear las dos formas de demodulación.



#### 4.1.3.4 Simulación de la Demodulación ASK coherente.

En la simulación desarrollada no se realiza la recuperación de portadora, la portadora se genera conociendo la frecuencia a la que se va a transmitir, en las referencias [1] y [2] se encuentran varios métodos de recuperación de portadora, para los usuarios interesados en el tema. Una vez generada la portadora se multiplica por la señal de entrada y luego se utiliza un filtro pasabajos; si la señal de entrada contiene portadora se resta un nivel antes de ingresar al filtro, caso contrario va directamente al filtro.

Este diseño se puede observar en la figura 4.12b, aquí se encuentra el bloque *Fcn* para la generación de la señal portadora  $\cos(2\pi fct)$ , nuevamente la señal del bloque *clock* da el tiempo de simulación para el bloque *Fcn*. Luego se introdujo un bloque *switch* que permite restar o no un valor de 0.5 si la señal viene con portadora o sin portadora; el parámetro *cp* controla el bloque *switch* y el usuario puede modificarlo con la opción "Modulación contiene portadora" de la plantilla de datos (figura 4.12c). A continuación se tiene el mismo filtro pasabajos que la demodulación no coherente.

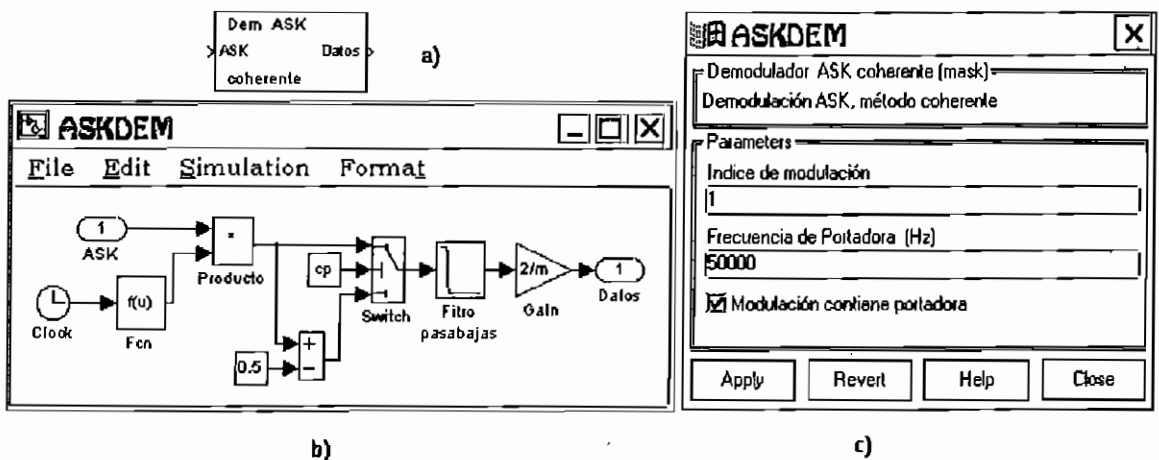


Figura 4.12 Demodulación coherente a) Bloque de Demodulación, b) Subsistema contenido y c) Plantilla de datos.

## 4.2 MODULACIÓN FSK (*FREQUENCY SHIFT KEYING*).

FSK es una técnica de modulación en frecuencia cuya señal modulante es digital, en este caso la modulación no se puede analizar como un proceso lineal. Existen dos maneras de generar una señal modulada FSK, la primera se denomina modulación con discontinuidad de fase y la segunda con continuidad de fase o FSK coherente.

### 4.2.1 Modulación FSK con discontinuidad de fase.

Este tipo de modulación genera una señal discontinua, esta señal puede representarse con la ecuación 4.8; y, en la figura 4.14 se encuentra un ejemplo de la modulación FSK con discontinuidad de fase. Este modulador consiste en disponer de varios osciladores a las frecuencias deseadas (típicamente dos) e ir conectando uno u otros de acuerdo a la señal de datos binarios, tal como se muestra en la figura 4.13.

$$s(t) = \begin{cases} A_C \cos(2\pi f_1 + \theta_1) & \text{cuando el dato es } 1_L \\ A_C \cos(2\pi f_2 + \theta_2) & \text{cuando el dato es } 0_L \end{cases} \quad (4.8)$$

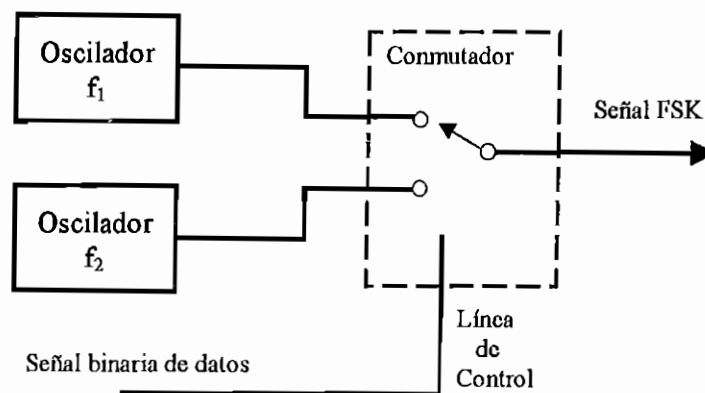


Figura 4.13 Modulador FSK con Discontinuidad de Fase

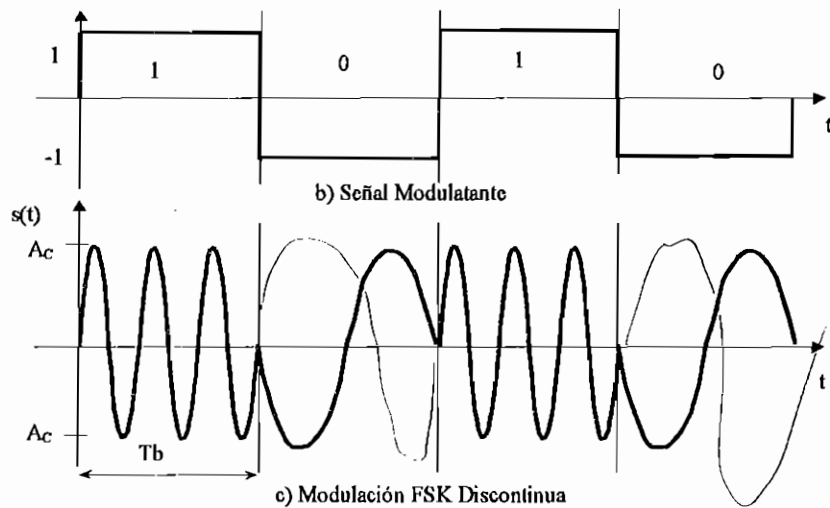


Figura 4.14 Ejemplo de una señal FSK con discontinuidad de fase.

#### 4.2.1.1 Simulación FSK con discontinuidad de fase.

Para la modulación FSK, se desarrolló un bloque que permite el cambio de la señal de un oscilador de frecuencia  $f_1$  a uno de frecuencia  $f_2$  mediante un interruptor controlado por la fuente de datos digitales.

El contenido del bloque de simulación se puede observar en la figura 4.15b, se genera dos señales senoidales con las frecuencias determinadas por el usuario (figura 4.16c), mediante dos bloques *Fcn* de la librería *nolinear* que tienen como entrada la señal del bloque *clock* de la librería *source*. Con el propósito de controlar la frecuencia de una salida senoidal, se ha colocado el bloque *switch* de la librería *connections*; la señal de entrada al bloque *switch* es la señal de datos binarios que pueden ser los de la fuente de datos digitales de la librería fuentes.

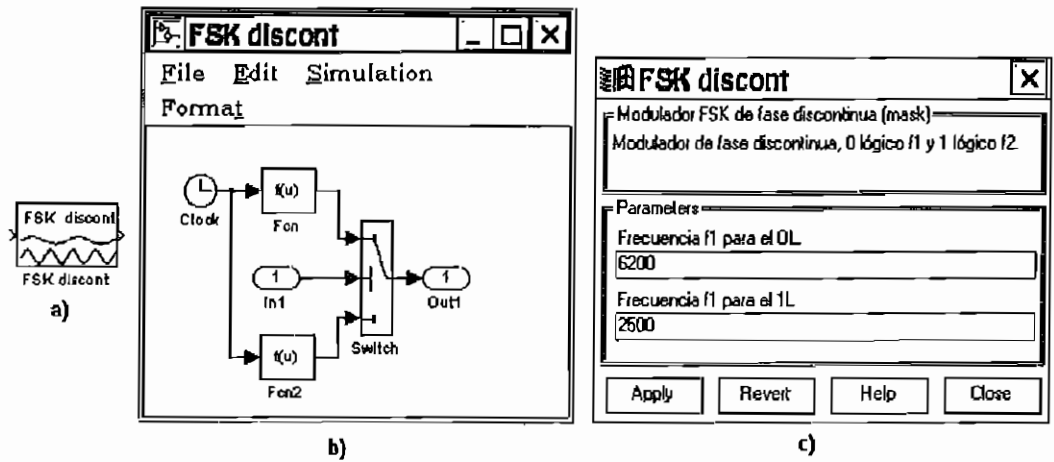


Figura 4.15 Modulador FSK de fase discontinua: a) Bloque de Simulación b) Subsistema de modulación, c) Plantilla de datos.

#### 4.2.2 Modulación FSK con fase continua.

Según la referencia [1], la señal FSK de fase continua se define por la ecuación 4.9.

$$s(t) = A_c \cos[2\pi \cdot f_c \cdot t + \gamma(t)] \quad (4.9).$$

Donde la fase con respecto a la señal portadora,  $\gamma(t)$ , es una función continua del tiempo. La ecuación 4.10 se obtiene de reescribir la expresión anterior en función de la constante de desviación de frecuencia pico  $\Delta f$  y todo esto válido para un intervalo del tiempo de bit de  $0 \leq t < T_b$ .

$$s(t) = A_c \cos[2\pi \cdot f_c \cdot t \pm 2\pi \cdot \Delta f \cdot t + \gamma(0)] \quad (4.10).$$

Si para los símbolos 0 y 1, se utilizan las frecuencias  $f_1$  y  $f_2$  entonces  $f_c$  es igual a  $\frac{f_1 + f_2}{2}$  y  $\Delta f$  se define como:  $\frac{f_2 - f_1}{2}$ . En 4.10 la fase inicial,  $\gamma(0)$ , depende de los valores precedentes de la señal y se elige su valor de tal forma que se evite cualquier discontinuidad; para el inicio se suele

tomar como cero. Se puede reescribir la ecuación 4.10 en 4.11, para extenderla más allá de un intervalo de bit

$$s(t) = A_C \cos[2\pi \cdot f_C \cdot t + 2\pi \cdot \Delta f \cdot t \cdot p_k + \gamma_k] \quad (4.11).$$

donde:

$p_k$  = función binaria con estados posibles  $\pm 1$ , que representan los datos binarios de entrada;  $k=1,2,\dots$ ,

$\gamma_k = \gamma[(k-1)T_b]$ , exceso de fase al inicio del  $k$ -ésimo intervalo de bit, necesario para mantener una señal continua.

Esta señal modulada es generada mediante un modulador de FM, tal como un oscilador controlado por voltaje (VCO), que se puede representar como en la figura 4.16; y, un ejemplo de la señal resultante se representa en la figura 4.17.

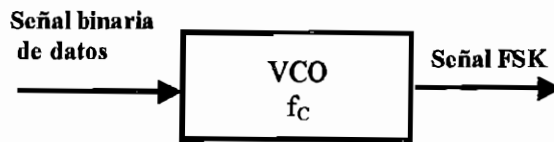


Figura 4.16 Generación de FSK de fase continua.

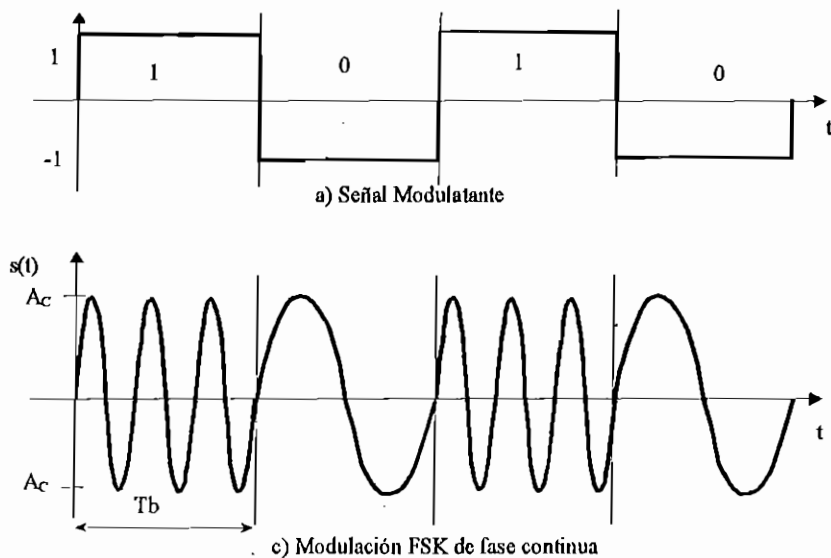


Figura 4.17 Ejemplo de Señal FSK de Fase continua

#### 4.2.2.1 Simulación FSK con fase continua.

Se desarrolló la modulación con fase continua a partir de la ecuación 4.11 y, se empleó el bloque *s-function*, ya que la ecuación 4.11 requiere de memoria, es decir de tener presente la fase anterior para calcular la próxima fase y de esta manera mantener la continuidad de fase.

La programación de este oscilador controlado por voltaje (VCO) se encuentra en el archivo creado con el nombre de ospv.m y consiste en aplicar la ecuación 4.11 en el tiempo de muestreo necesario del bloque *s-function*. El bloque está disponible en la librería de modulación FSK mediante la interface comdig o en la librería desarrollada y que se denominó análisis. La apariencia del bloque VCO se encuentra en la figura 4.18a; y, en la figura 4.18c se encuentran los datos que el usuario debe ingresar, es decir el tiempo de muestreo, la frecuencia de portadora y la constante de desviación de frecuencia.

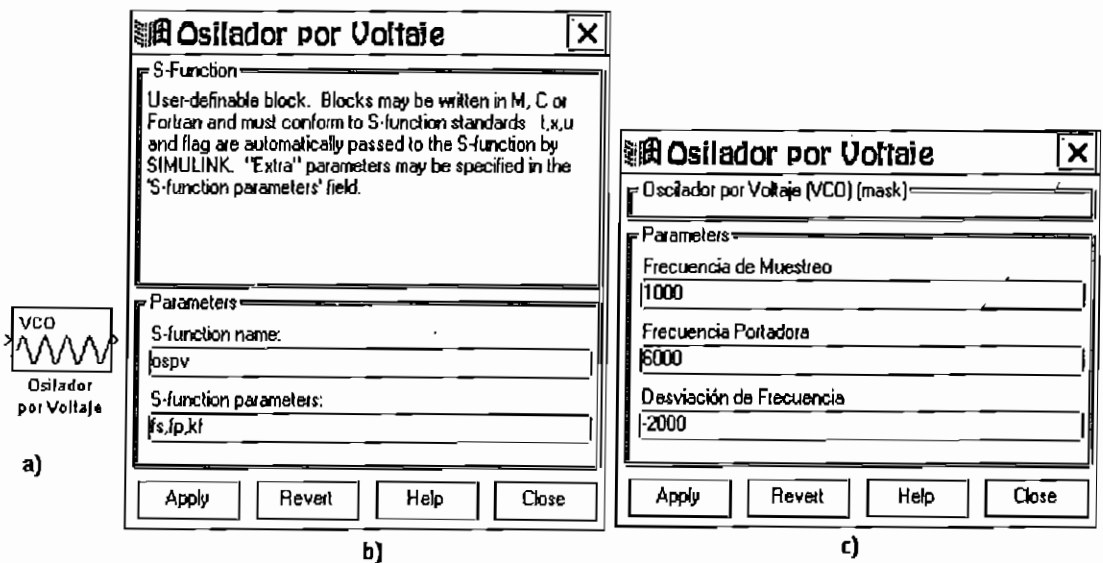


Figura 4.18 Modulador FSK de fase continua: a) Bloque de Simulación, b) Plantilla del bloque S-function y c) Plantilla de datos para el usuario.

### 4.2.3 Demodulación FSK mediante Detector de cruces por cero.

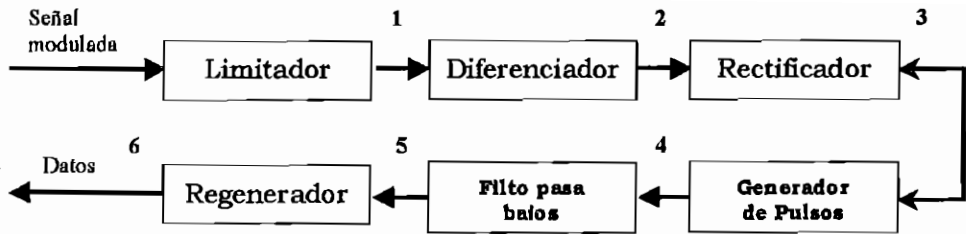
Este es uno de los métodos<sup>1</sup> para detectar señales FSK y, se basa en generar pulsos de anchura fija y detectar el nivel de componente continua de los mismo. El esquema de bloques y las formas de onda correspondientes pueden verse en la figura 4.19.

El generador de pulsos suele ser un monoestable. El filtro pasabajos debe tener una pendiente suficientemente elevada como para eliminar las componentes de alta frecuencia y su frecuencia de corte debe ser lo bastante alta para que en los instantes de cruces por cero de la señal de datos regenerada no se vea muy distorsionada.

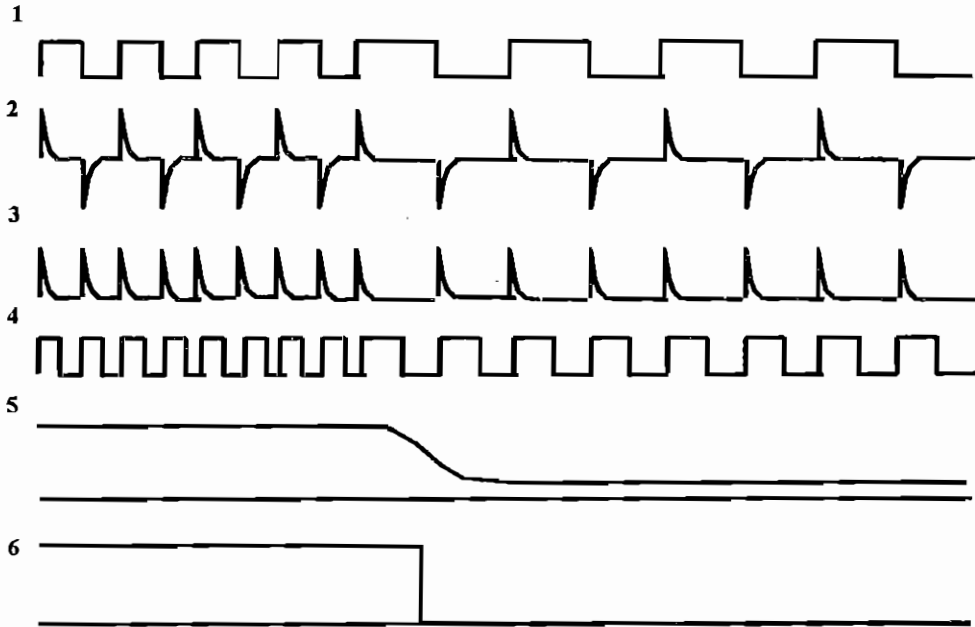
Esta realización presenta las ventajas de una gran simplicidad, sin embargo, cuando las dos frecuencias características de la señal modulada FSK son muy próximas resulta difícil discriminar entre los dos niveles a la salida del filtro pasabajos.

---

<sup>1</sup> Tomado de referencia [6]



a) Detector de Cruces por cero



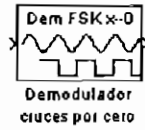
b) Formas de Onda

Figura 4.19 Demodulador FSK mediante detección de cruces por cero

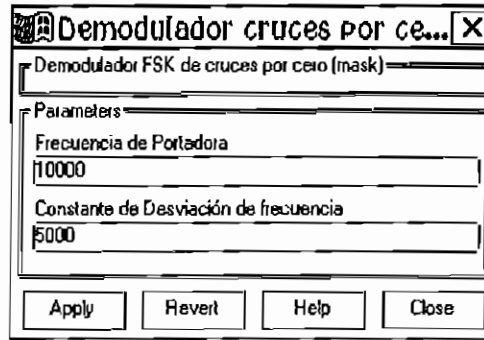
#### 4.2.3.1 Simulación del demodulador FSK mediante detección de cruces por cero.

Básicamente se sigue paso a paso lo que propone el diagrama de bloques, el bloque desarrollado permite al usuario que pueda verificar cada parte del proceso de demodulación. Se diseñó el bloque demodulador con la posibilidad de modificar únicamente los datos de frecuencia de portadora y la constante de desviación de frecuencia.





a)



b)

Figura 4.20 Demodulador FSK por detección de cruces por cero: a)Bloque de Simulación

El bloque de simulación está compuesto por varios subsistemas (o subbloques) correspondientes a las funciones que desempeñan los bloques de la figura 4.19a. Los contenidos de estos subsistemas se describen a continuación.

Como limitador se utiliza el bloque *Relay* de la librería *no-linear*, que funciona como un comparador de nivel, si se tiene una entrada senoidal proporcionará una señal cuadrada de la misma frecuencia; y, se eligió como salidas niveles  $\pm 1$ .

Para el bloque diferenciador se diseñó un subsistema compuesto por un filtro pasa altos (red derivadora<sup>1</sup>) y un amplificador con saturación de nivel máximo.

Con el bloque *Abs* de la librería *no-linear* se desarrolló un rectificador de señal, similar a un rectificador de onda completa.

<sup>1</sup> Capítulo 14 referencia [8].

Para conseguir el efecto de un generador de pulsos se utilizó un bloque *Relay* junto con el bloque *Memory*, los cuales generan una señal de pulsos de ancho fijo desde la señal proveniente del bloque rectificador.

El filtro pasabajos es un *Butterworth* con una frecuencia de corte cercana a la frecuencia máxima de la señal modulada, este valor de frecuencia de corte se obtuvo experimentalmente.

Finalmente el bloque regenerador de señal está compuesto de un bloque *gain* (amplificador) y un bloque *Relay* como comparador.

#### 4.2.4 Demodulación FSK mediante PLL (*Phase Locked Loop*).

La estructura de un lazo cerrado en fase (PLL), figura 4.21 es otro método para detección de una señal modulada en frecuencia.

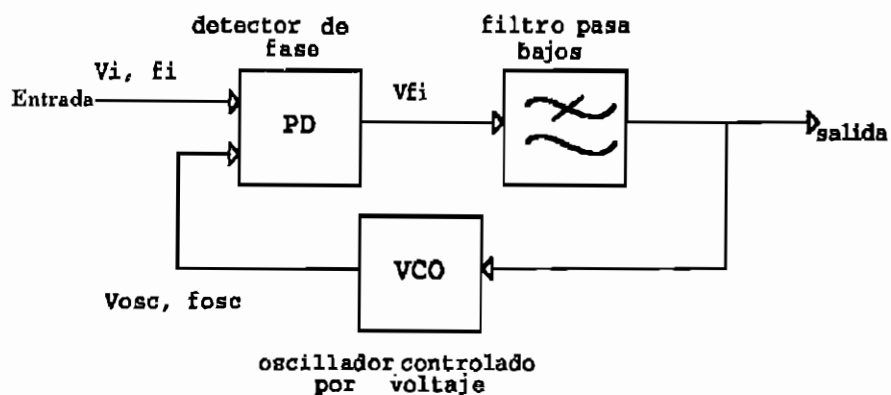


Figura 4.21 Esquema general del PLL.

La señal ingresa al detector de fase, el cual compara la fase de la señal suministrada por el oscilador controlado por tensión (VCO) con la de la señal de entrada generando una señal proporcional a la diferencia. Esta señal se filtra para eliminar las componentes de alta frecuencia y se utiliza para manejar el VCO. Al introducir la señal modulada, el VCO deja de oscilar

en régimen libre y se produce un fenómeno de captura hasta que las dos señales consiguen sincronizarse en frecuencia y en fase. A partir de entonces, cualquier incremento de frecuencia en la señal de salida o en la de entrada producirá un incremento proporcional en la señal de salida del detector de fase, para que este incremento se produzca también en el VCO y se mantenga el enganche.

Para una señal modulada en FSK a dos frecuencias, por ejemplo, la salida del filtro pasabajos tendrá dos niveles correspondientes a los niveles 0 y 1 de la señal de datos.

El ancho de banda del filtro pasabajos debe ser mayor o al menos igual al de la señal modulante. La defensa contra el ruido será tanto mayor, cuanto mayor sea la pendiente y menor el ancho de banda del filtro. Sin embargo no conviene elevar demasiado el orden del filtro ya que el lazo tiende a ser inestable.

Tanto el VCO como el detector de fase pueden ser digitales y el demodulador puede realizarse de forma integrada resultando un montaje compacto, de pequeñas dimensiones.

#### 4.2.4.1 Simulación del demodulador FSK mediante PLL.

La simulación del demodulador consta de tres partes, un limitador, un PLL y un regenerador de señal.

El limitador se diseñó a partir del bloque *Relay*, que se puede utilizar como un comparador de nivel. La señal senoidal ingresa al limitador y es muy similar a una señal cuadrada de la misma frecuencia. El regenerador de señal tiene las mismas características que el limitador de

señal, es decir es un comparador al cual se le da un nivel de referencia adecuado. Para la simulación del *PLL* se utilizó un esquema basado en el circuito integrado (CMOS)<sup>1</sup>4046 y cuyo estudio se encuentra en el anexo C. Según este esquema el detector de fase es una compuerta lógica XOR, el filtro pasabajos es del tipo *Butterworth* y el VCO es un generador de onda cuadrada a partir de una onda triangular<sup>2</sup>.

El bloque de simulación se representa en la figura 4.22 y los bloques que lo conforman se representa en la figura 4.23. En esta última figura se puede observar que el *PLL* es prácticamente el mismo descrito antes y está conformado por VCO1 (*VCO*), FS1 (Filtro pasabajos) y el Detector de fase 1. Adicionalmente se introdujo una señal de referencia cercana a la frecuencia central del VCO1 que proviene del VCO2, mediante un detector de fase (Detector de fase 2) y un filtro pasabajos (FS2).

La configuración fina del PLL empleada en la simulación se fundamenta en el modelo tomado de la referencia [9] pero utilizando un detector de fase en lugar de un detector de frecuencia. El funcionamiento es prácticamente el mismo que para un PLL como el descrito anteriormente, con la ventaja que la señal introducida cercana a la frecuencia central disminuye las oscilaciones bruscas de VCO1 (denominado también ruido de *VCO*) y permite extender la calidad de detección del PLL a un ancho de banda mayor, todo sin mejorar la calidad de VCO1.

---

<sup>1</sup> Circuito integrado basado en el transistor de efecto de campo de metal óxido semiconductor (MOSFET).

<sup>2</sup> Ver anexo B y Generadores de Señal de la referencia [9].

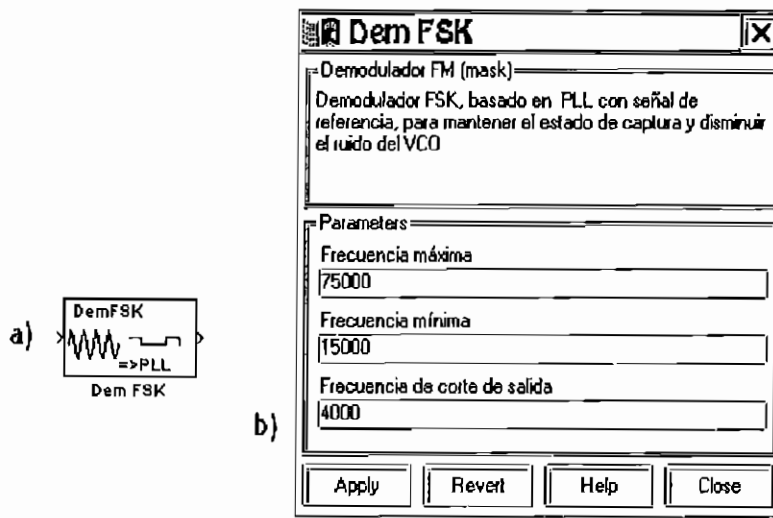


Figura 4.22 Demodulador FSK: a) Bloque de Simulación  
b) Plantilla de datos

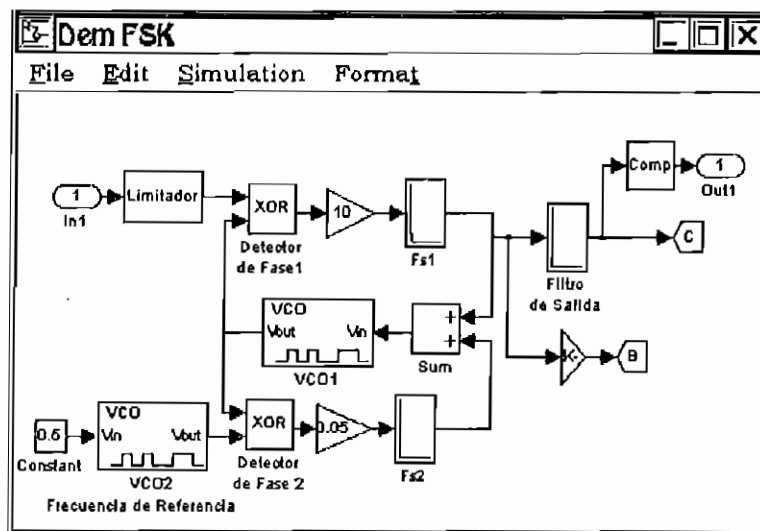


Figura 4.23 Contenido del Demodulador FSK de 4.22 a.

#### 4.2.5 Densidad Espectral de Potencia de la señal FSK.

El ancho de banda de transmisión de una señal FSK depende de la separación de frecuencias utilizada. La obtención de una fórmula para la densidad espectral de potencia para una señal FSK con una entrada aleatoria es bastante complicada. No obstante, las siguientes son algunas tendencias en los resultados<sup>1</sup>. Para valores bajos de  $2\Delta fT_b$ , la densidad espectral de potencia tiene un solo pico centrado en la frecuencia

<sup>1</sup> Estas tendencias se tomaron de la referencia [1], una expresión para la PSD de la FSK se la puede tener en la referencia [2].

portadora o central  $f_c = \frac{f_1 + f_2}{2}$  y disminuye suavemente hacia ambos lados del pico. Conforme aumenta  $2\Delta f T_b$ , el pico central de la densidad espectral de potencia de la FSK tiende hacia dos grupos espectrales identificables por separado, centrados en  $f_c \pm \Delta f$ . Para valores aún mayores de  $2\Delta f T_b$ , la densidad espectral de potencia de la FSK tiende hacia dos grupos espectrales identificables por separado, centrados en  $f_c \pm \Delta f$ . La densidad espectral de potencia es continua para señales binarias aleatorias a excepción de valores de  $2\Delta f T_b = m$ , donde  $m$  es un valor entero que genera además pulsos discretos. Para la elección particular de  $2\Delta f T_b = 1$ , el 50% de la potencia total en la señal FSK se halla en componentes de línea a las dos frecuencias transmitidas, por lo que esta opción suele evitarse para prevenir posibles interferencias entre canales. Los sistemas proyectados para receptores menos costosos no coherentes utilizan  $2\Delta f T_b > 1$ , mientras que los sistemas que utilizan FSK que se proyectan en principio para detección coherente usan a menudo valores en el intervalo de  $0.5 < 2\Delta f T_b < 1$  para obtener cierta ventaja contra el ruido.

En la referencia [2] se encuentra un archivo .m que puede calcular la densidad espectral de potencia según técnicas de estadística (Proakis, "Digital Communications", ed. New York McGraw Hill, 1983) y se encuentra representada en la ecuación 4.12a.

$$S(f) = \frac{Ac^2 T_b}{2} \cdot (A_1^2(f)[1 + B_{11}(f)] + A_2^2(f)[1 + B_{22}(f)] + 2B_{12}(f)A_1(f)A_2(f)) \quad (4.12a)$$

donde:

$$A_n(f) = sa(\pi \cdot T_b \cdot \{f - \Delta f(2n-3)\}) = \frac{\text{sen}(\pi \cdot T_b \cdot \{f - \Delta f(2n-3)\})}{\pi \cdot T_b \cdot \{f - \Delta f(2n-3)\}} \quad (4.12b)$$

y

$$B_{nm}(f) = \frac{\cos[2\pi f T_b - 2\pi \Delta f T_b (n + m - 3)] - \cos(2\pi f T_b) \cos[2\pi \Delta f T_b (n + m - 3)]}{1 + \cos^2(2\pi f T_b) - 2 \cos(2\pi \Delta f T_b) \cos(2\pi f T_b)} \quad (4.12c)$$

$\Delta f$  es la máxima desviación de frecuencia,  $R = 1/T_b$  es la tasa de bits,  $T_b$  es el tiempo de bits y el índice digital de modulación es  $h = 2\Delta f/R$ .

A continuación se proporciona el listado original de comandos tomados del listado de programas que vienen con la referencia [2], el cual permite al *matlab* representar la densidad espectral de potencia de una señal FSK.

```
% Setting the value of the modulation index
h = 0.7;

Ac = 1;
R = 1;
Tb = 1/R;
DF = h*R/2;

df = 1.5*R/50;
f = 0:df:1.5*R;

xf = 2*pi*f*Tb;
xF = 2*pi*DF*Tb;

% Generando Equation 4-12b para An(f) for n = 1,2;
A1 = sa(pi*Tb*(f-DF*(2*1-3)));
A2 = sa(pi*Tb*(f-DF*(2*2-3)));

% Generando Blij(f) for ij=11,12,22 (Ver ecuación 4-12a)
xc11 = xF*(1+1-3);
xc12 = xF*(1+2-3);
xc22 = xF*(2+2-3);
B11 = zeros(length(xf),1);
B12 = zeros(length(xf),1);
B22 = zeros(length(xf),1);
for (i = 1:length(xf))
    B11(i) = (cos(xf(i)-xc11) - cos(xF)*cos(xc11));
    B11(i) = B11(i)/(1+cos(xF)^2-2*cos(xF)*cos(xf(i)));
    B12(i) = (cos(xf(i)-xc12) - cos(xF)*cos(xc12));
    B12(i) = B12(i)/(1+cos(xF)^2-2*cos(xF)*cos(xf(i)));
    B22(i) = (cos(xf(i)-xc22) - cos(xF)*cos(xc22));
    B22(i) = B22(i)/(1+cos(xF)^2-2*cos(xF)*cos(xf(i)));
end;

% Combinando terminos para producir ecuación 4.12a
P1 = (A1.*A1).*(1 + B11);
P2 = (A2.*A2).*(1 + B22);
```

```

P12 = (2*B12).*(A1.*A2);
P = 0.5*Ac^2*Tb*(P1+P2+P12);

```

```

plot(f,P);
xlabel('f');
ylabel('P(f)');
grid;

```

El resultado se muestra en la figura 4.24; y, se puede variar el valor de  $h$ , para observar la influencia del índice de modulación sobre la densidad espectral de potencia.

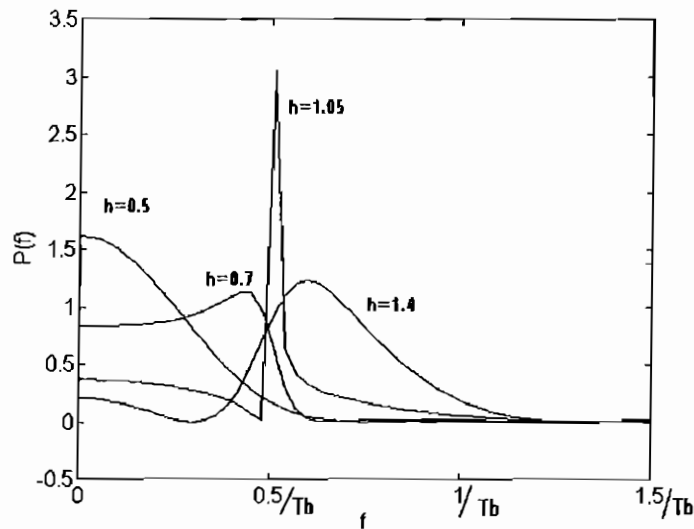


Figura 4.24 Densidad espectral de potencia de la señal envolvente de la modulación FSK

#### 4.2.6 Modulación MSK (*Minimun Shift Keying*).

La modulación MSK es una modulación FSK de fase continua con un mínimo índice de modulación ( $h=0.5$ ), El índice de modulación( $h$ ) al cual se refiere la definición es igual a  $2\Delta f T_b$ , donde  $\Delta f$  es la desviación máxima de frecuencia y  $T_b$  es el tiempo de duración de un bit.

Se mostrará por qué  $h=0.5$  es el mínimo índice para obtener la ortogonalidad en la señal FSK de fase continua. Para un uno lógico dentro del intervalo de tiempo  $0 < t < T_b$ , la señal FSK será  $s_1(t) = A_c \cos(2\pi f_1 t + \phi_1)$  y para un cero lógico se tendrá una señal FSK igual a



$s_2(t) = A_c \cos(2\pi f_2 t + \phi_2)$ , donde  $\phi_1 = \phi_2$ , para mantener continuidad de fase, entonces la condición de ortogonalidad para estas dos señales se encuentra en que la integral del producto de  $s_1$  y  $s_2$  sea igual a cero, como se muestra en la expresión 4.13 :

$$\int_0^{Tb} s_1(t) s_2(t) dt = \int_0^{Tb} A_c^2 \cos(2\pi f_1 t + \phi_1) \cdot \cos(2\pi f_2 t + \phi_2) dt = 0 \quad (4.13).$$

trabajando en esta ecuación se obtiene 4.14:

$$\begin{aligned} & \frac{A_c^2}{2} \left[ \frac{\text{sen}[(2\pi f_1 - 2\pi f_2)Tb + (\phi_1 - \phi_2)] - \text{sen}(\phi_1 - \phi_2)}{(2\pi f_1 - 2\pi f_2)} \right] + \\ & + \frac{A_c^2}{2} \left[ \frac{\text{sen}[(2\pi f_1 + 2\pi f_2)Tb + (\phi_1 + \phi_2)] - \text{sen}(\phi_1 + \phi_2)}{(2\pi f_1 + 2\pi f_2)} \right] = 0 \end{aligned} \quad (4.14).$$

el segundo término de esta expresión es de valor despreciable debido a que se puede suponer  $(2\pi f_1 + 2\pi f_2)$  bastante grande o tomando en cuenta que la frecuencia central  $f_c$  es mayor que  $1/Tb$ . En la primera parte de la ecuación 4.14 se tiene que  $(2\pi f_1 - 2\pi f_2)Tb = 2\pi(2\Delta f)Tb$ , además  $h = 2\Delta f Tb$ , entonces se cumplirá la condición de ortogonalidad si:

$$\frac{A_c^2}{2} \left[ \frac{\text{sen}[2\pi h + (\phi_1 - \phi_2)] - \text{sen}(\phi_1 - \phi_2)}{2\pi h} \right] = 0 \quad (4.15)$$

y como  $\phi_1 = \phi_2$ , por lo tanto la condición mínima de ortogonalidad se cumple al tener  $h = 0.5$ , lo que implica que deberá tenerse  $\Delta f = \frac{1}{4Tb}$ , para conseguir una modulación MSK.

Según la referencia [2] para modulación con discontinuidad de fase la condición mínima de ortogonalidad es  $h=1$ .

Con la condición mínima de ortogonalidad ya demostrada y con la expresión 4.11 una señal MSK se puede representar con la ecuación 4.16 de la siguiente manera:

$$s_{MSK}(t) = A_C \cos \left[ 2\pi \cdot f_C + \frac{\pi \cdot t}{2Tb} \cdot p_k + \gamma_k \right] \quad (4.16)$$

utilizando una identidad trigonométrica, una señal MSK puede expresarse como términos de componentes en cuadratura como en la ecuación 4.17 a continuación:

$$s_{MSK}(t) = A_C \left\{ \cos \left[ 2\pi \cdot f_C + \frac{\pi \cdot t}{2Tb} \cdot p_k + \gamma_k \right] \cos(2\pi f_C t) - \right. \\ \left. - \text{sen} \left[ 2\pi \cdot f_C + \frac{\pi \cdot t}{2Tb} \cdot p_k + \gamma_k \right] \text{sen}(2\pi f_C t) \right\} \quad (4.17)$$

La señal MSK puede generarse utilizando un modulador FSK con  $\Delta f = 1/4Tb$  con los procesos y bloques de modulación ya descritos en 4.2.1 y 4.2.2. Otra manera de obtener una señal MSK es un método con moduladores balanceados basándose en la ecuación 4.17, este último método puede verse en la referencia [2].

De la referencia [1] o [2] se obtiene la siguiente expresión 4.18 para la Densidad Espectral de Potencia de la modulación MSK.

$$S(f) = \frac{16 \cdot A_C^2 \cdot Tb}{\pi^2} \left( \frac{\cos^2(2\pi Tbf)}{[1 - (4Tbf)^2]^2} \right) \quad 4.18.$$

La ecuación 4.18 es la densidad espectral de potencia pero en banda base es decir no afectada por la portadora. De la ecuación 4.18 se puede observar que el primer valor de frecuencia donde la expresión se hace cero

es en  $f=1/4T_b$ , es decir es el lugar de frecuencias donde se concentra la mayor cantidad de potencia de la señal.

### 4.3 MODULACIÓN PSK (*PHASE SHIFT KEYING*).

Este tipo de modulación lleva la información en el cambio de fase de la señal portadora; en la sección de modulación ASK se trató sobre la modulación PRK que corresponde a una modulación de fase binaria propiamente.

En general la señal modulante puede ser multinivel, lo que quiere decir que puede tomar diferentes niveles para representar los dígitos binarios deseados. En el caso de dígitos binarios ( $1_L$  y  $0_L$ ) el número de niveles  $M$ , es igual a  $2^L$  donde  $L$  es el número de dígitos que pueden ser representados con cada nivel de señal; por ejemplo, si  $L = 2$ ,  $M = 4$ .

Si la señal modulante es multinivel se puede entonces hablar de que la modulación PSK puede ser 2-PSK, 4-PSK (también denominada QPSK), 8-PSK, 16-PSK, etc.

En general una señal PSK puede representarse por la ecuación 4.19

$$S_{PSK}(t) = \cos\left(2\pi \cdot f_c \cdot t + \frac{g_L(t) \cdot \delta\phi}{2}\right) \quad (4.19)$$

Donde:

$L$ , es el número de fases, generalmente potencias de 2.  $f_c$  es la frecuencia de la señal portadora.

$\delta\phi$  es la separación entre fases adyacentes  $\Rightarrow \delta\phi = \frac{2\pi}{L}$

$g_L(t)$  es la señal modulante del tipo NRZ, simétrica y de L niveles que pueden tomar los valores:  $\pm 1, \pm 3, \pm 5, \dots$  etc.

La expresión 4.19 indica que se emplea el 100% de modulación, es decir que la variación de fase de un intervalo a otro puede estar entre  $-\pi$  y  $\pi$ . En la figura 4.25 se encuentra un ejemplo de las señales 2-PSK y 4-PSK, además se puede observar una representación que suele llamarse diagrama fasorial o diagrama de constelación ver referencias [1] y [6].

Se puede representar la ecuación 4.19 expandiendo los términos como en la ecuación 4.20, con ayuda de igualdades trigonométricas.

$$\begin{aligned}
 \text{si } \phi &= \frac{g_L(t) \cdot \delta\phi}{2} : & (4.20) \\
 S_{PSK}(t) &= \cos\phi \cdot \cos(2\pi \cdot f_c \cdot t) - \text{sen}\phi \cdot \text{sen}(2\pi \cdot f_c \cdot t)
 \end{aligned}$$

Se suele llamar a esta ecuación como representación en cuadratura; ya que  $\cos\phi$  y  $\text{sen}\phi$  son constantes durante un intervalo de señal y son los coeficientes para expresar  $\cos(2\pi f_c t + \phi)$  como una combinación lineal de las señales  $\cos(2\pi f_c t)$  y  $\text{sen}(2\pi f_c t)$ . Nótese que esta expresión tiene la misma forma que una modulación lineal; es decir, una modulación digital en fase se puede analizar y generar con los mismos métodos utilizados para las modulaciones lineales.

Debido a que  $\cos(2\pi f_c t)$  y  $\text{sen}(2\pi f_c t)$  están desfasados el uno respecto del otro  $\pi/2$ , se dice que estos dos términos son ortogonales en un diagrama fasorial y de este hecho se tiene la expresión "en cuadratura".

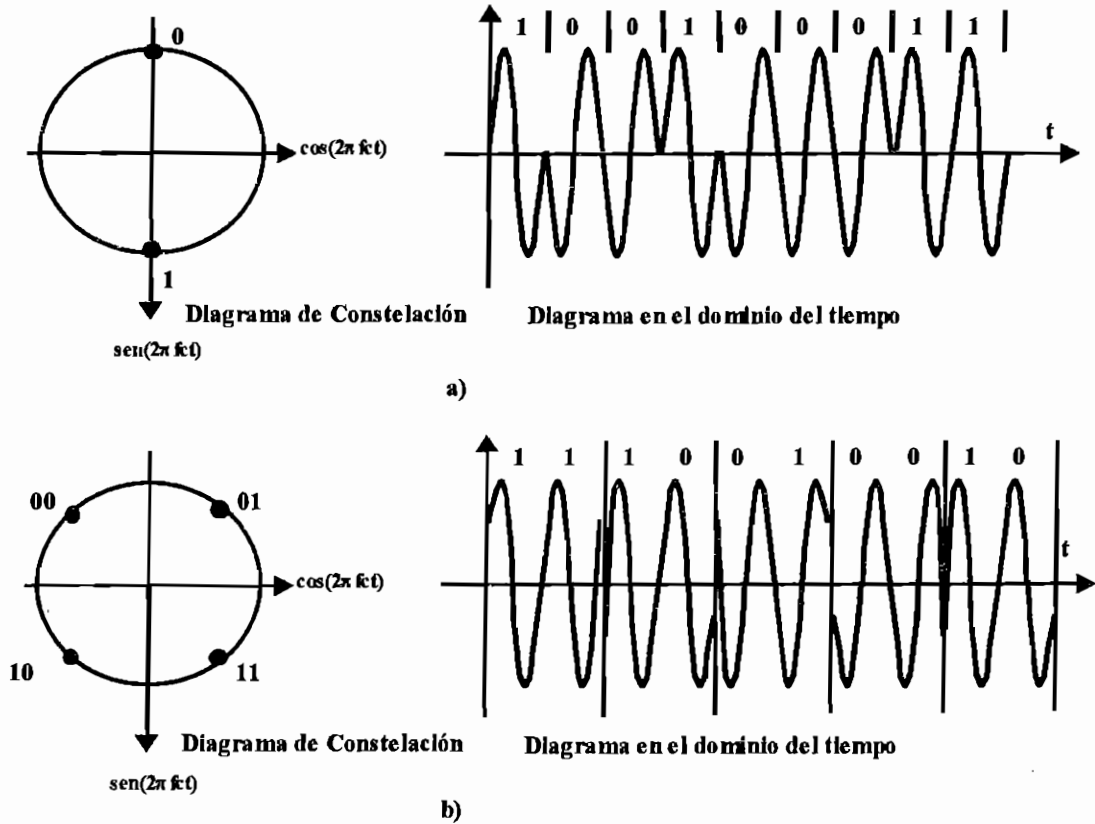


Figura 4.25 Modulación PSK: a) 2-PSK b) 4-PSK

Esencialmente  $\cos(2\pi fct)$  y  $\text{sen}(2\pi fct)$  representan la base de los vectores del diagrama fasorial o de constelación. La señal coseno se suele denominar como referencia o señal I y a la señal seno que se encuentra en desfase con respecto a la primera como señal Q. En las tablas 4.1, 4.2 y 4.3 se indican ejemplos con valores de los coeficientes de las señales I y Q para modulaciones 4-PSK, 8-PSK y 16-PSK. Además en la figura 4.26 se tiene el diagrama de constelación para 8-PSK y 16-PSK.

Digitos Binarios	Coeficientes en Cuadratura		Portadora Modulada
	$\cos\phi$	$-\text{sen}\phi$	
01	0.707	-0.707	$\cos(2\pi fct + \pi/4)$
00	-0.707	-0.707	$\cos(2\pi fct + 3\pi/4)$
10	-0.707	0.707	$\cos(2\pi fct - 3\pi/4)$
11	0.707	0.707	$\cos(2\pi fct - \pi/4)$

Tabla 4.1 Coeficientes en cuadratura para modulación 4-PSK

Dígitos Binarios	Coeficientes en Cuadratura		Portadora Modulada
	$\cos\phi$	$-\text{sen}\phi$	
011	0.924	-0.383	$\cos(2\pi fct + \pi/8)$
010	0.383	-0.924	$\cos(2\pi fct + 3\pi/8)$
000	-0.383	-0.924	$\cos(2\pi fct + 5\pi/8)$
001	-0.924	-0.383	$\cos(2\pi fct + 7\pi/8)$
101	-0.924	0.383	$\cos(2\pi fct - 7\pi/8)$
100	-0.383	0.924	$\cos(2\pi fct - 5\pi/8)$
110	0.383	0.924	$\cos(2\pi fct - 3\pi/8)$
111	0.924	0.383	$\cos(2\pi fct - \pi/8)$

Tabla 4.2 Coeficientes en cuadratura para modulación 8-PSK

Dígitos Binarios	Coeficientes en Cuadratura		Portadora Modulada
	$\cos\phi$	$-\text{sen}\phi$	
0000	0.980	-0.195	$\cos(2\pi fct + \pi/16)$
0001	0.831	-0.555	$\cos(2\pi fct + 3\pi/16)$
0011	0.555	-0.831	$\cos(2\pi fct + 5\pi/16)$
0010	0.195	-0.980	$\cos(2\pi fct + 7\pi/16)$
0110	-0.195	-0.980	$\cos(2\pi fct + 9\pi/16)$
0111	-0.555	-0.831	$\cos(2\pi fct + 11\pi/16)$
0101	-0.831	-0.555	$\cos(2\pi fct + 13\pi/16)$
0100	-0.980	-0.195	$\cos(2\pi fct + 15\pi/16)$
1000	-0.980	0.195	$\cos(2\pi fct - 15\pi/16)$
1001	-0.831	0.555	$\cos(2\pi fct - 13\pi/16)$
1011	-0.555	0.831	$\cos(2\pi fct - 11\pi/16)$
1010	-0.195	0.980	$\cos(2\pi fct - 9\pi/16)$
1110	0.195	0.980	$\cos(2\pi fct - 7\pi/16)$
1111	0.555	0.831	$\cos(2\pi fct - 5\pi/16)$
1101	0.831	0.555	$\cos(2\pi fct - 3\pi/16)$
1100	0.980	0.195	$\cos(2\pi fct - \pi/16)$

Tabla 4.3 Coeficientes en cuadratura para modulación 16-PSK

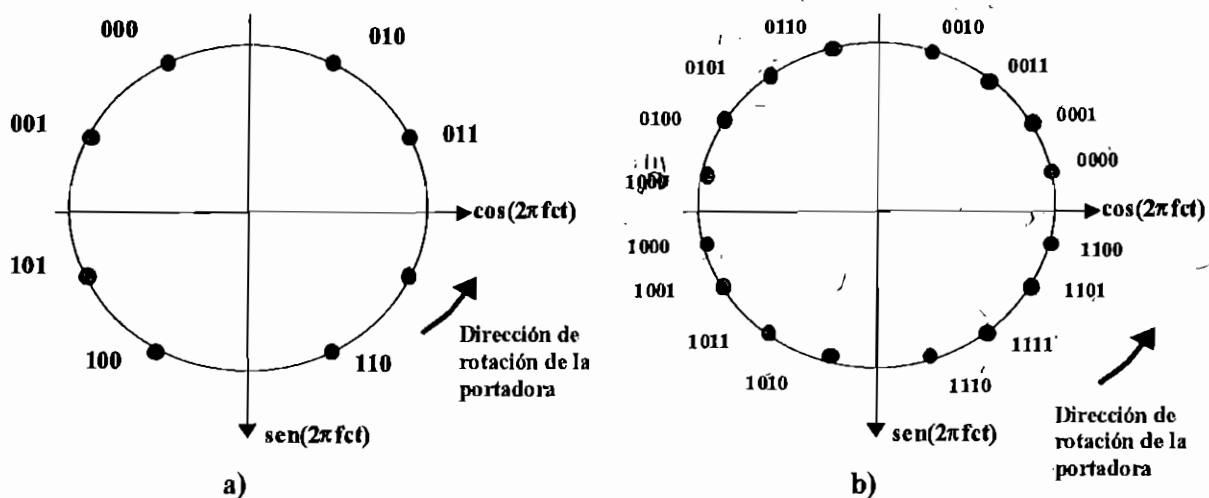


Figura 4.26 Diagrama de constelación de: a) Modulación 8-PSK y b) Modulación 16-PSK

### 4.3.1 Moduladores PSK.

De las varias técnicas para generar señales PSK, se emplearán la generación de múltiples fases y la combinación lineal de las componentes en cuadratura. A continuación se presentan las técnicas aplicadas para modular señales del tipo BPSK, 4-PSK y 16-PSK.

#### 4.3.1.1 Modulación PSK mediante generación de múltiples fases.

Esta técnica consiste en generar cada una de las fases necesarias para satisfacer la ecuación 4.19, dependiendo la elección de cada fase de los datos binarios. En la figura 4.27 se encuentra un diagrama de bloques de este tipo de modulación.

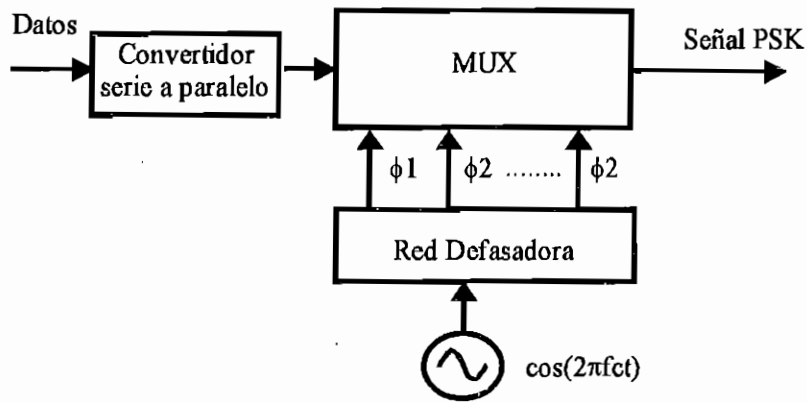


Figura 4.27 Esquema de Modulador PSK con red defasadora o generación de multiples fases.

En la figura 4.27 se tiene un bloque llamado MUX, que es la representación de un multiplexor analógico, luego un bloque denominado red de defasaje que permite obtener la señal  $\cos(2\pi fct + \phi)$ . La señal de datos controla la señal de salida del mux analógico, es decir que permite que pase una de las entradas de acuerdo a los datos binarios ingresados.

Los datos binarios antes de controlar el mux deben pasar por un convertidor serie paralelo, para cumplir con las tablas antes mencionadas. Evidentemente en el caso de 2-PSK no se necesita este bloque.

Este esquema puede extenderse para la modulación multinivel, ya que sólo se necesita generar las fases de acuerdo con la ecuación 4.19. Para los diferentes tipos de modulación las fases a generar se pueden encontrar en la columna denominada Portadora Modulada, de las tablas 4.1, 4.2 y 4.3.



#### 4.3.1.2 Modulación PSK como combinación lineal de señales en cuadratura.

Este método consiste en generar los coeficientes  $-\text{sen}\phi$  y  $\text{cos}\phi$ , necesarios para cumplir con la ecuación 4.20, estos valores varían en función del número de niveles para cada tipo de modulación PSK.

En la figura 4.28 se encuentra el diagrama de bloques de este tipo de generación de señales PSK. Los datos binarios son convertidos de serie a paralelo, luego se generan los coeficientes  $\text{cos}\phi$  y  $-\text{sen}\phi$  y luego se multiplica estas señales con  $\text{cos}(2\pi fct)$  y  $\text{sen}(2\pi fct)$  respectivamente. Finalmente el resultado se suma, cumpliéndose de esta manera con la ecuación general 4.20.

A este tipo de moduladores se lo suele también denominar moduladores balanceados.

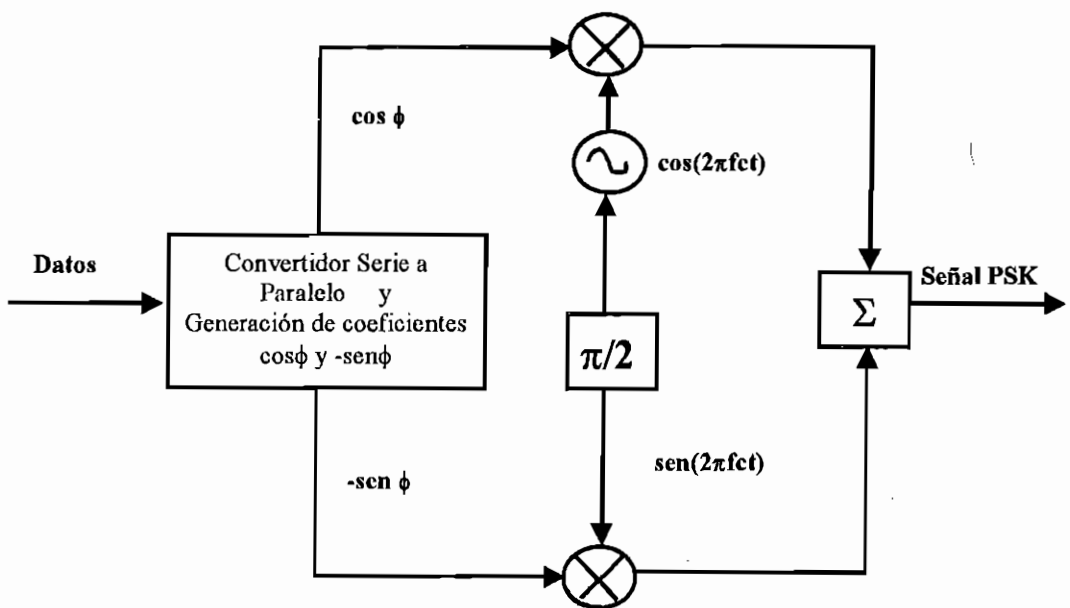


Figura 4.28 Generación de PSK mediante componentes en cuadratura

#### 4.3.1.3 Simulación del Modulador 2-PSK por red de defasaje.

En la sección 4.1.1.2 ya se indicó como se puede generar 2-PSK a partir de un modulador del tipo OOK, en esta sección se incluyen dos bloques adicionales de simulación que permiten generar señales PSK, de acuerdo a los métodos generales antes mencionados.

El primer método es la generación de dos fases,  $\pi/2$  y  $-\pi/2$ , para que según el dato binario de entrada, la señal a la salida sea  $\cos(2\pi fct + \pi/2)$  o  $\cos(2\pi fct - \pi/2)$ .

La señal de datos ingresa al bloque de simulación, ésta controla la salida del multiplexor analógico y genera la salida de fase correspondiente según el dato de entrada. En la figura 4.29 se tiene el bloque de simulación, en la parte b de esta figura se tiene los bloques que se utilizaron para la generación de la señal BPSK. Como multiplexor analógico se utilizó el bloque *Multiport Switch* de la librería *no linear*, y, como la red defasadora (figura 4.30) se diseñó con dos bloques *Fcn*, los cuales tienen como dato una señal del tiempo de simulación que proviene del bloque *clock* de la librería *source*.

El bloque *Multiport Switch* necesita un nivel de señal de control mayor a 1.5 para que pase la primera entrada, 2.5 para la segunda, 3.5 la tercera y así sucesivamente. Se diseña este bloque para varias dos entradas, debido a los requerimientos la señal de control, el bloque *Fcn* ayuda a adaptar a la señal de control de tal forma que cuando el dato sea uno la respuesta sea 2 y cero cuando el dato sea un cero. Así cuando el dato sea cero se tendrá la señal de fase  $\pi/2$  y si el dato es uno la de fase  $-\pi/2$ .

La red defasadora, es simplemente una generación directa de  $\cos(2\pi fct+\pi/2)$  y  $\cos(2\pi fct-\pi/2)$  con los bloques  $Fcn$ , donde  $t$  es la señal proveniente del bloque *clock*.

El usuario puede modificar la frecuencia de portadora en la respectiva plantilla de datos que se muestra en la figura 4.29c.

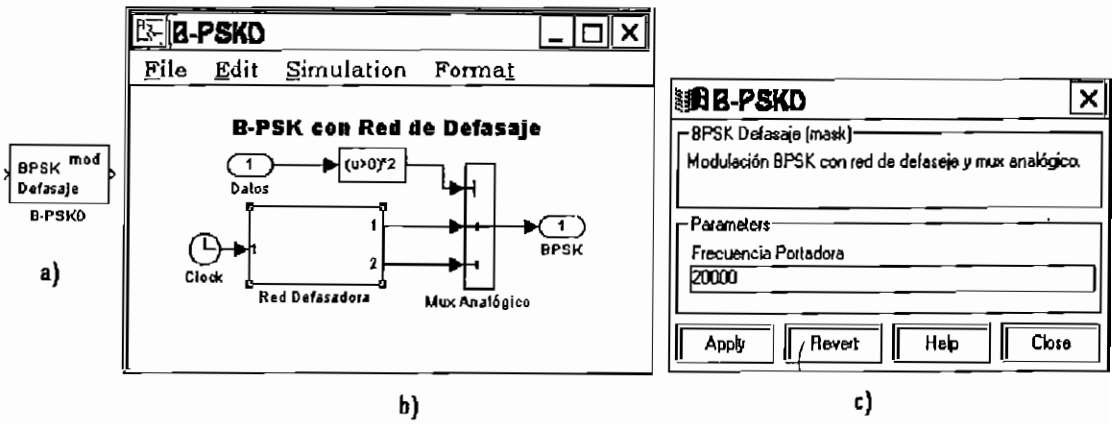


Figura 4.29 Simulación BPSK por red de Defasaje:  
 a) Bloque de Simulación b) Subsistema de Simulación c) Plantilla de datos.

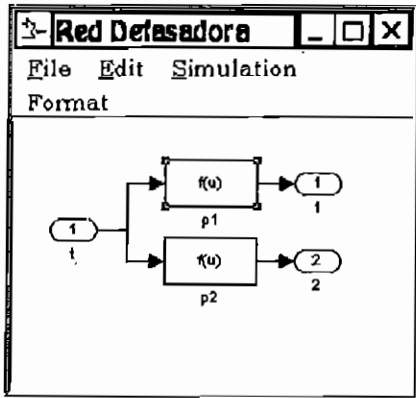


Figura 4.30 Contenido de la Red de Defasaje.

#### 4.3.1.4 Simulación del modulador 2-PSK como combinación lineal de señales en cuadratura.

Los datos binarios ingresan al bloque de simulación, si el dato es cero lógico se tiene una salida de 0 para la señal  $\cos\phi$  y  $-1$  para  $-\sin\phi$ ; si el dato es uno la salida es 0 para la señal  $\cos\phi$  y  $1$  para  $-\sin\phi$ . Estos valores se multiplican con las funciones  $\cos(2\pi fct)$  y  $\sin(2\pi fct)$  respectivamente, para finalmente sumarse y de esta manera generar la señal 2-PSK.

El bloque de simulación se encuentra representado en la figura 4.31. Se le da al usuario la posibilidad de cambiar el valor de la frecuencia portadora. En la parte b de la figura 4.31 se puede observar el diseño del bloque modulador, el cual consta de un bloque generador de coeficientes de dos generadores de las señales  $\cos(2\pi fct)$  y  $\sin(2\pi fct)$  y de un bloque sumador a la salida.

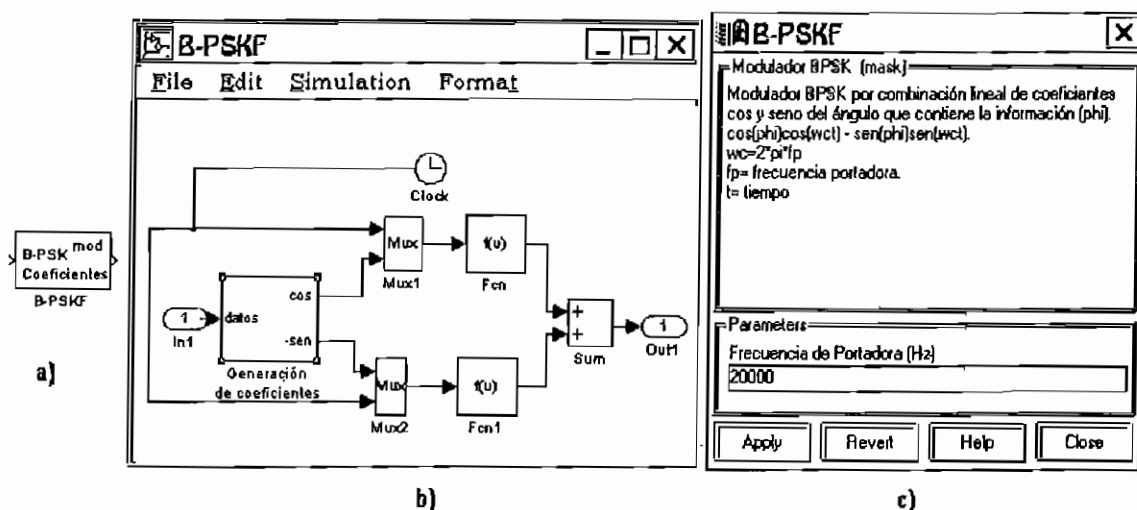


Figura 4.31 Modulador BPSK por coeficientes en cuadratura: a) Bloque de Simulación b) Subsistema de modulación, c) Plantilla de datos.

La generación de coeficientes se representa en la figura 4.31, aquí se identifica si la señal de entrada de datos es uno o cero y luego se asigna los valores de los coeficientes que le corresponden a la modulación 2-PSK. La identificación de los datos, a la entrada del bloque de simulación se la realiza a través del bloque *switch* que tiene como entrada de control los datos binarios y como salidas posibles, dos constantes 1 y 0 de acuerdo a que la señal de control sea mayor que cero o cero respectivamente. Según la señal del bloque *switch*, se dan los valores de [0 1] y [0 -1] mediante el bloque *Combinatorial Logic*. La salida de este último bloque es un vector cuya primera componente es el coeficiente  $\cos\phi$  y la segunda  $\sin\phi$ , así  $[\cos\phi \sin\phi]$ . Por la naturaleza de la salida se debe utilizar un bloque *Demux* para separar las salidas respectivas del bloque de Generación de coeficientes.

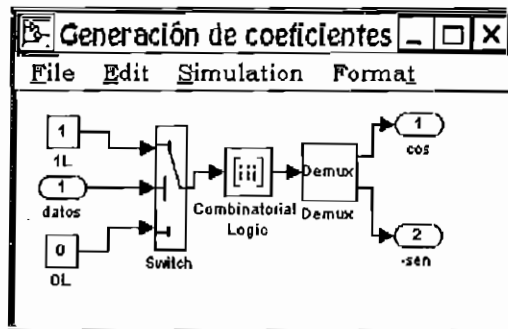


Figura 4.31 Generación de coeficientes

Los generadores de las funciones  $\cos\phi\cos(2\pi fct)$  y  $-\sin\phi\sin(2\pi fct)$  son dos bloques *Fcn* que tienen como variable  $t$  una de las señales de entrada de su *Mux* respectivo, la cual a su vez proviene del bloque *clock*. La otra entrada del *Mux* es una de las señales de salida del generador de coeficientes. Se requiere de este bloque *Mux* para que el bloque *Fcn* pueda operar con dos entradas ( $u[1]$  y  $u[2]$  en el orden con que ingresan al *Mux*). Cuando se tienen las dos señales en el *Mux* se especifica en forma de ecuación la multiplicación de coseno o seno con su correspondiente coeficiente.

#### 4.3.1.5 Simulación del Modulador 4-PSK por red de defasaje.

La simulación de la modulación QPSK por el método de las redes de defasaje tiene el mismo proceso general descrito anteriormente. Lo que cambia para el desarrollo del modulador QPSK con el diseño de los bloques de conversión serie a paralelo y el multiplexor analógico.

El usuario puede modificar la frecuencia portadora y el tiempo de bit, que son datos principales para la modulación. El bloque de simulación y la plantilla de datos se encuentran en la figura 4.32, en la parte b se encuentra como está constituido el bloque de simulación.

La señal de entrada es llevada a la forma paralela para controlar el multiplexor analógico, el cual da paso a la señal con la fase según la tabla 4.1.

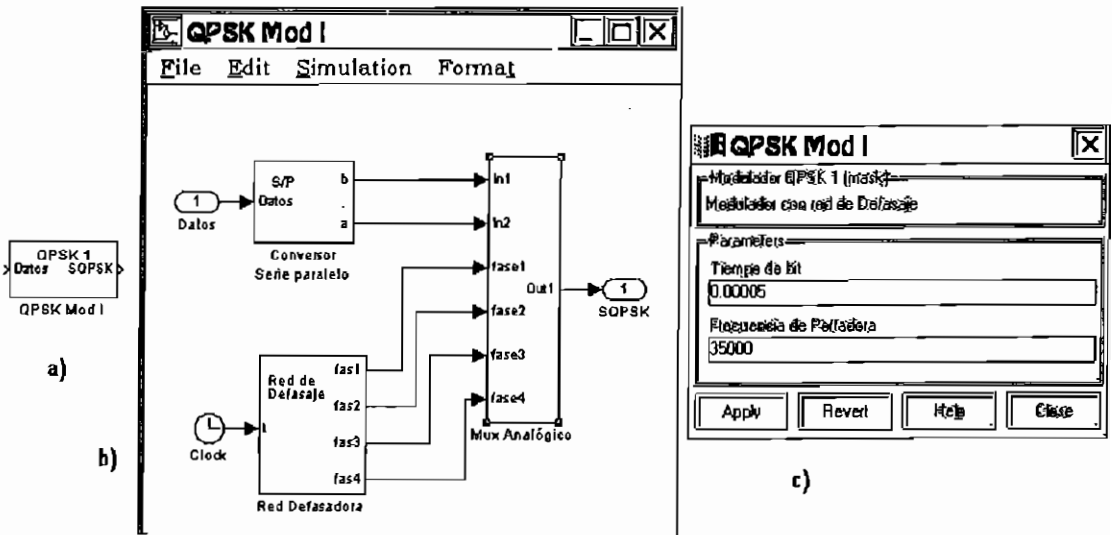


Figura 4.32 Modulador QPSK por red de defasaje: a) Bloque de Simulación b) Subsistema de modulación c) Plantilla de datos.

El diseño del convertidor serie paralelo se muestra en la figura 4.33 a. La conversión serie-paralelo toma dos entradas binarias consecutivas

para tener como salida dos señales binarias con los valores respectivos de cada una de las entradas. La duración de los datos paralelos es el doble que la duración de los datos en serie. Este proceso se consigue con dos bloques de retardo de la librería fuentes, un bloque Reloj de la misma librería y un bloque denominado de Habilitación. Los bloques de la librería fuentes se explican en el capítulo 2, el bloque Habilitación es un subsistema que sólo tiene los bloques de entrada y salida (*in* y *out* de la librería *connections*) y un bloque *Trigger* de la librería *connections*, el cual permite el paso de la señal que ingresa al bloque Habilitación cuando la señal de control pasa de cero a un valor positivo. Entonces la señal de datos ingresa a los bloques de retardo manteniendo el valor del primer dato binario hasta tener el segundo dato binario, es en este instante que la señal de reloj pasa de cero a uno y permite que las salidas del bloque Habilitación tengan los valores paralelos requeridos y los mantiene hasta que los dos siguientes datos lleguen a la entrada de este bloque.

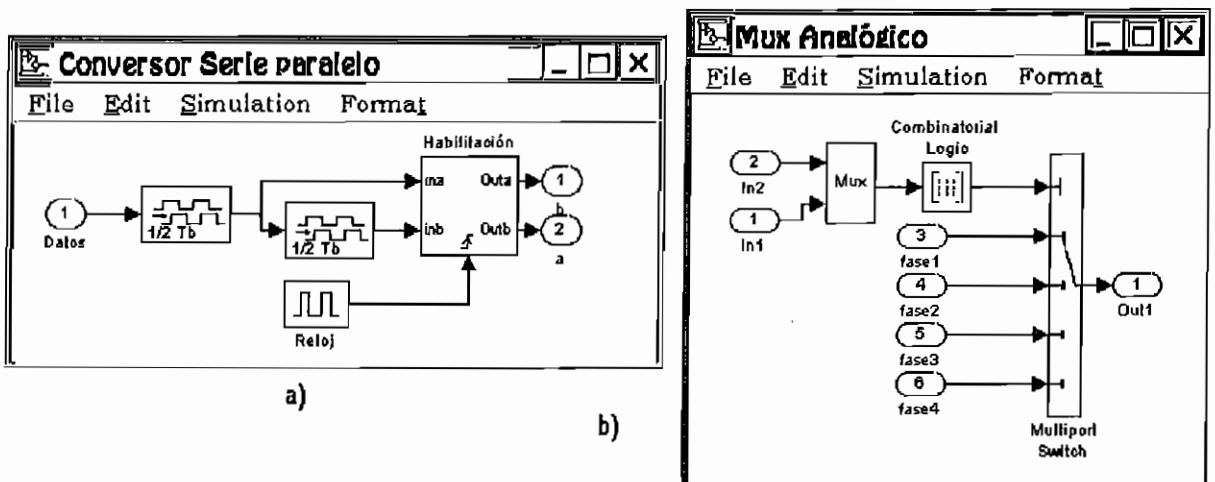


Figura 4.33 Componentes del Modulador QPSK : a) Conversor Serie Paralelo  
b) Multiplexor Analógico

El multiplexor analógico se desarrolló a partir del bloque *Multiport Switch* que es el mismo que se trató en la modulación 2-PSK con dos entradas adicionales. El bloque *Multiport Switch* requiere que la señal de

entrada tenga valores de 1.5, 2.5, 3.5, etc. para tener a la salida la primera, la segunda, la tercera, etc., entradas. Para dar a las señales de control los niveles adecuados se dispone de un bloque de lógica combinacional.

4.3.1.6 Simulación del modulador 4-PSK como combinación lineal de señales en cuadratura.

Se diseñó el bloque modulador QPSK con las mismas características que el modulador 2-PSK (por el método de la generación de coeficientes) como se muestra en la figura 4.34. La diferencia se presenta en la manera de procesar los datos de entrada, que son convertidos de serie a paralelo de la misma forma que el modulador QPSK anterior.

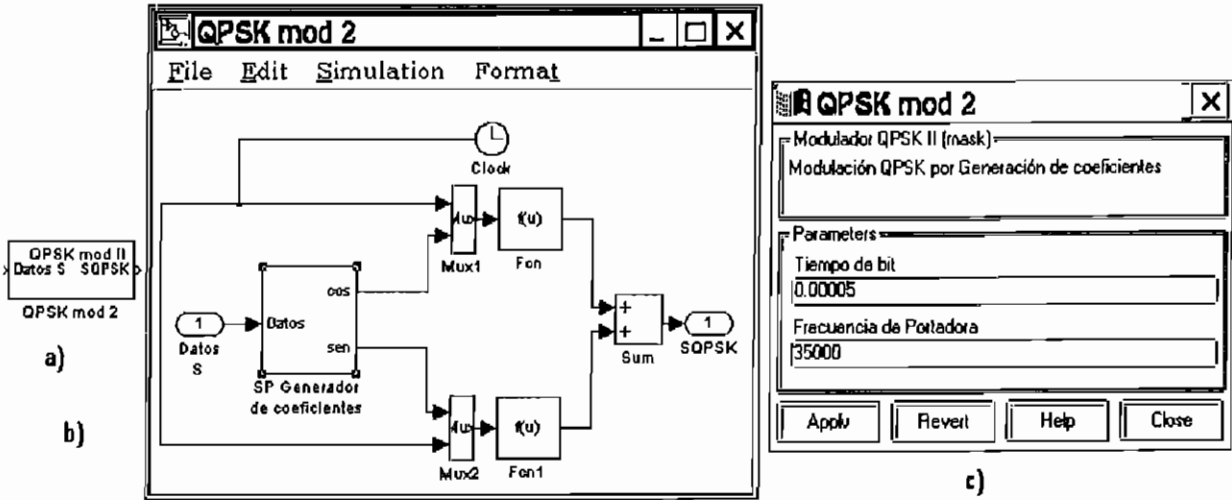


Figura 4.34 Modulador QPSK por generación de coeficientes: a) Bloque de Simulación b) Subsistema de modulación c) Plantilla de datos.



#### 4.3.1.7 Simulación del modulador 16-PSK como combinación lineal de señales en cuadratura.

Este modulador se desarrolló solamente para este método de modulación; el usuario, como en los moduladores anteriores, puede modificar desde la plantilla de datos (figura 4.35b) los datos de tiempo de bit y frecuencia de portadora.

El diseño es esencialmente el mismo que el descrito para el caso 2-PSK, solo difiere en que los datos binarios deben agruparse en número de cuatro para hacerles corresponder los valores respectivos de los coeficientes  $\cos\phi$  y  $-\sin\phi$ , de acuerdo a la tabla 4.3.

Para la agrupación de los datos binarios se utiliza un convertidor serie paralelo conformado por cuatro bloques de retardo, un bloque de Reloj de la librería "fuentes" y un bloque similar al bloque Habilidadación que se trató en el caso de la modulación 2-PSK. Los datos binarios agrupados tienen cuatro veces el tiempo de bit con relación a los datos originales, lo que debe ser tomado en cuenta para la demodulación.

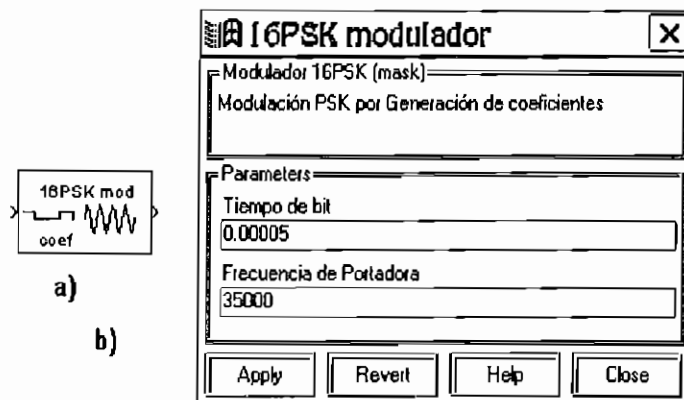


Figura 4.35 Modulador 16-PSK: a) Bloque de Simulación  
b) Plantilla de Datos

### 4.3.2 Demoduladores PSK.

La detección de señales PSK se realiza con demodulación coherente, es decir que se necesita recuperación de portadora. De manera general la señal modulada ingresa a un proceso de detección de frecuencia, luego de la cual la señal resultante es proporcional a los coeficientes  $\cos\phi$  y  $-\text{sen}\phi$  de la ecuación 4.20 y finalmente mediante un proceso de comparación de valores se tiene los datos binarios. Un esquema general de la demodulación PSK que se utilizará para la simulación se representa en la figura 4.36.

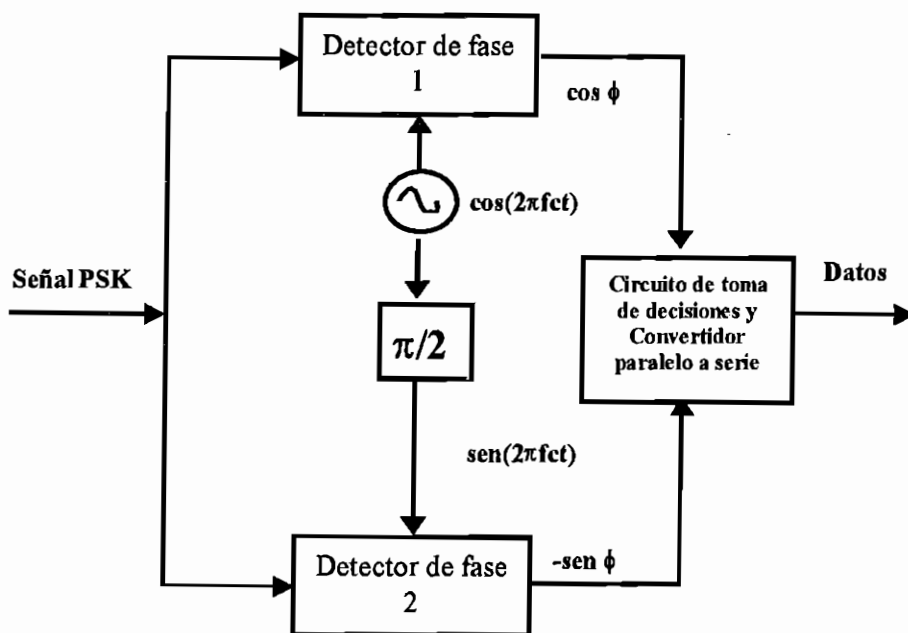


Figura 4.36 Esquema General de la demodulación PSK

Los detectores de fase son moduladores de producto seguidos de un filtro pasabajos, como se puede observar en la figura 4.37. La representación matemática del resultado del primer detector se muestra en la ecuación 4.21, donde  $\text{pasabajos}\{\cdot\}$  es la función del filtro diseñada para eliminar los términos del doble de la frecuencia portadora.

$$\begin{aligned}
 y_1(t) &= \text{pasabajos}\{\cos(2\pi fct + \phi) \cdot \cos(2\pi fct)\} \\
 &= \text{pasabajos}\left\{\frac{\cos \phi}{2} + \frac{\cos \phi \cos(4\pi fct)}{2} - \frac{\text{sen } \phi \text{sen}(4\pi fct)}{2}\right\} \\
 &= \frac{\cos \phi}{2}
 \end{aligned}
 \tag{4.21}$$

En la ecuación 4.22 se tiene la expresión resultante del segundo detector de fase.

$$\begin{aligned}
 y_2(t) &= \text{pasa bajos}\{\cos(2\pi fct + \phi) \cdot \cos(2\pi fct)\} \\
 &= -\frac{\text{sen } \phi}{2}
 \end{aligned}
 \tag{4.22}$$

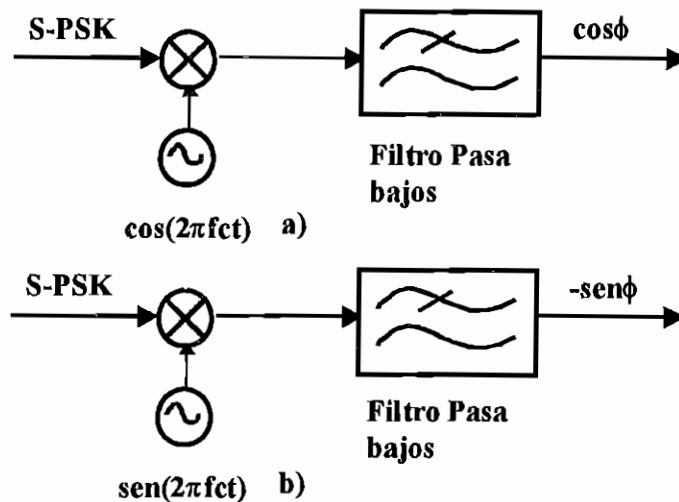


Figura 4.38 Detectores de fase: a) Detector de fase 1 b) Detector de fase 2

Luego de la detección de fase, el siguiente paso a seguir es un circuito de toma de decisiones el cual permite, mediante el reconocimiento de signos y magnitudes identificar si el dato respectivo es uno o cero. En la detección de la señal 2-PSK es necesario únicamente un detector de fase, ya que el signo de la señal que da como resultado el filtro pasabajos permite distinguir entre uno o cero. Para modulación 4-PSK, la información

del primer detector de fase no es suficiente y se complementa con la información que proporciona el segundo detector de fase. Un ejemplo de la demodulación 4-PSK (QPSK), como se puede observar en la tabla 4.1 donde el signo del resultado del primer detector determina si el primer bit de dato es uno o cero (uno si es positivo). Del mismo modo el signo de la señal del segundo detector de fase, determina si el segundo dato es uno o cero (uno si es positivo).

Finalmente después de identificar los datos binarios se tiene un convertidor paralelo a serie para la recuperación de la señal de datos originalmente transmitida.

#### 4.3.2.1 Simulación del demodulador 2-PSK.

El bloque desarrollado consta de un modulador de producto, un filtro pasabajos y un bloque de toma de decisiones. La señal ingresa al modulador de producto para luego ser filtrada y si la señal filtrada es mayor que cero se tiene que el dato original es uno; y, si es menor que cero el dato original es cero. El usuario debe proporcionar los datos de frecuencia de portadora y tiempo de bit en la plantilla de datos como se indica en la figura 4.39c.

Se puede observar en la figura 4.39 b que se diseñó el modulador de producto mediante un bloque *producto* de la librería *no linear* y un bloque *Fcn* de la misma librería que proporciona la función  $\text{sen}(2\pi fct)$ , donde  $t$  es la señal de salida del bloque *clock*. El bloque de toma de decisiones es un bloque *Fcn* que cumple la función de un comparador, cuando la señal proveniente del filtro es mayor que cero su resultado es uno y cero en caso contrario.

El filtro pasabajos es del tipo *Butterworth*, de cuarto orden con una frecuencia de corte mayor o igual al ancho de banda de la señal modulante y menor que el doble de la frecuencia portadora. Es necesario indicar que no se garantiza en el bloque demodulador una buena recepción si la frecuencia de portadora no es mayor que el ancho de banda de la señal modulante.

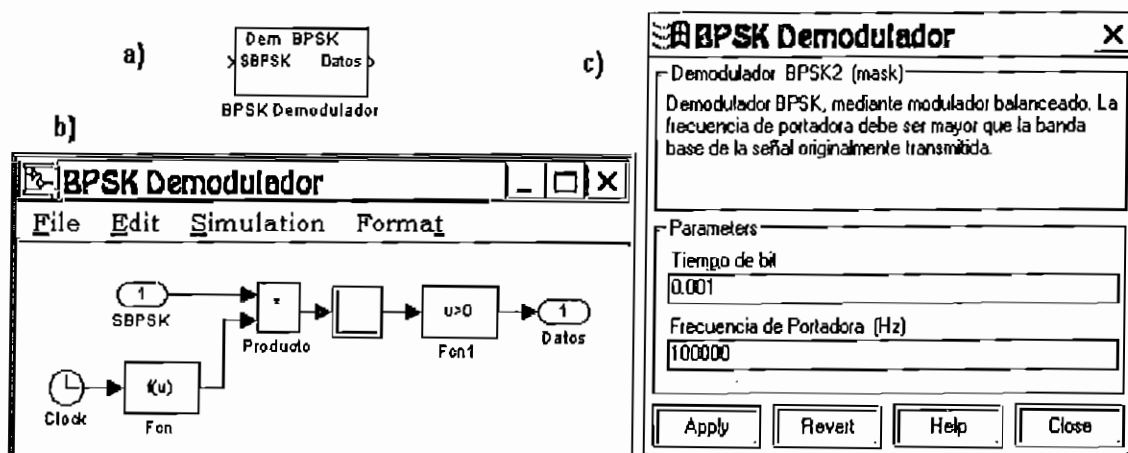


Figura 4.39 Simulación del demodulador BPSK: a) Bloque de Simulación b) Subsistema de simulación y c) Plantilla de datos.

#### 4.3.2.2 Simulación del demodulador QPSK.

La demodulación se diseñó a partir de dos detectores de fase, dos moduladores de producto, una sección lógica y un convertidor serie a paralelo. La señal QPSK ingresa a los detectores de fase y su resultado son dos señales proporcionales a  $\cos\phi$  y  $-\sin\phi$ , dependiendo del signo de estas señales se determinan los valores del bit más significativo y del menos significativo mediante la sección lógica. Una vez que se tienen los datos binarios por pares, éstos ingresan al bloque diseñado para cumplir la función de convertidor paralelo a serie.

En la figura 4.40 se tiene el bloque diseñado para la demodulación QPSK. Los detectores de fase son dos moduladores de producto como el que se diseñó para la demodulación 2-PSK y están compuestos de un bloque *producto*, un bloque *Fcn* y un filtro del tipo *Butterworth* de orden cuatro. Luego a la sección lógica se le denominó circuito lógico y está conformada de dos secciones, la primera de dos registros de desplazamiento y la segunda de un circuito de sincronización. Finalmente se desarrolló un bloque que permite convertir dos datos en paralelo a dos datos en serie.

La función de los moduladores de producto en los detectores de fase es multiplicar la señal QPSK por  $\cos(2\pi fct)$  y  $\sin(2\pi fct)$  respectivamente; y, luego filtrar la señal resultante con una adecuada frecuencia de corte. Para la simulación se generan las funciones senoidales a la frecuencia de portadora para realizar el producto con la señal modulada.

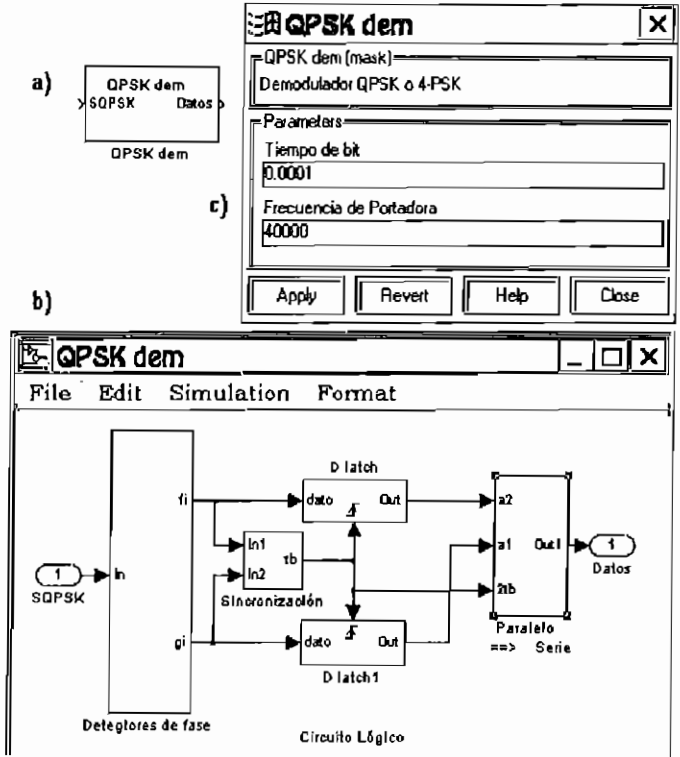


Figura 4.40 Demodulador QPSK: a) Bloque de Simulación b) Subsistema demodulador c) Plantilla de datos.

La sincronización se desarrolla a partir de un VCO de onda cuadrada, habilitado el momento que empiezan los datos. El VCO se describe en el capítulo 2 pero no es más que un generador de onda cuadrada a partir de una onda triangular (Ver referencia [9]). La habilitación se consigue mediante un comparador que detecta el inicio de datos permaneciendo desde ese instante en un nivel de uno, ésta es la señal de control para un bloque que está formado por el VCO y un bloque denominado *enable*<sup>1</sup> de la librería *connections*. Como resultado se tiene un generador de onda cuadrada con un período igual a dos veces el tiempo de bit desde el inicio de datos válidos y es esta señal la que controla los registros de desplazamiento y el convertidor serie a paralelo.

Los registros de desplazamiento son esencialmente bloques desarrollados a partir de un bloque que contiene los bloques *input*, *output* y *Trigger*; y, se diseñó de tal forma que cuando la señal de sincronización pasa de un nivel bajo a un nivel alto el bloque permite la lectura del dato de entrada y se actualiza el valor de la respectiva salida. Utilizando este comportamiento sincronizado de este tipo de registro se diseñó de tal forma que entre el bloque *input* y *output* estén comparadores de nivel hechos con bloques *Fcn*, los cuales indican si el bit correspondiente es uno o cero lógicos.

Por último el bloque convertidor paralelo a serie está conformado por un bloque *switch* cuya señal de control es la señal de sincronización y permite alternar entre el valor de un bit y el otro en el instante respectivo.

---

<sup>1</sup> Para su modo de empleo ver referencia [3].

#### 4.3.2.3 Simulación del demodulador 16-PSK.

El esquema a seguir para la demodulación de la señal 16-PSK es prácticamente el mismo, es decir la señal modulada ingresa a los detectores de fase, luego a una sección lógica y por último a la conversión paralelo a serie.

El bloque desarrollado se representa en la figura 4.41b. La detección de fase es la misma que para el demodulador anterior, es decir con los moduladores de producto. Luego viene una sección que ahora se divide en tres: sincronización, registros de desplazamiento y circuito lógico. Al final se encuentra el convertidor de paralelo a serie.

Se diseñó la sincronización con dos VCO habilitados mediante bloques *enable* y su respectiva señal de control que proviene de un bloque *Relay* que a su vez cumple la función de comparador, el cual se sitúa en un nivel 1 cuando el primer dato es recibido. Las frecuencias de los dos VCO son: la mitad de la frecuencia de datos ( $1/2T_b$ ) y la cuarta parte de dicha frecuencia ( $1/4T_b$ ). Las dos señales del VCO se utilizan en la conversión paralelo a serie y la de cuarta parte de la frecuencia de datos en los bloques de registro de desplazamiento.

Los registros de desplazamiento siguen conteniendo los bloques *input*, *output* y el bloque *Trigger*. Sin embargo entre los bloques de entrada y salida se encuentra un conjunto de comparadores que permiten precisar valores de las señales que provienen de los detectores de fase, esto debido a que para la demodulación se requiere el conocimiento de las magnitudes de  $\cos\phi$  y  $-\sin\phi$  y no solamente el signo. La lectura de datos a la entrada del registro se diseñó con el cambio de alto a bajo.



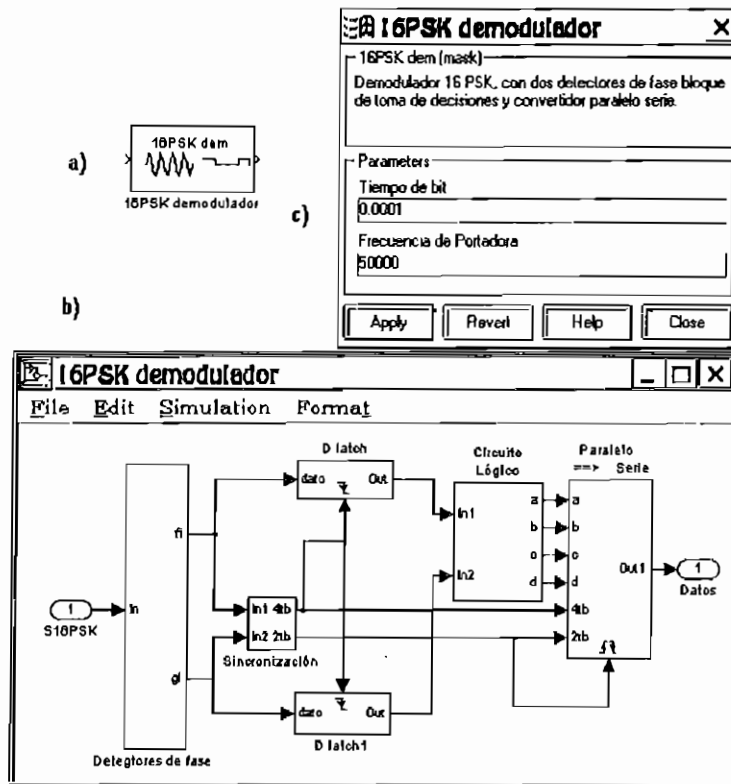


Figura 4.41 Demodulador 16-PSK: a) Bloque de Simulación b) Subsistema demodulador c) Plantilla de datos.

#### 4.4 MODULACIÓN D-BPSK(DIFERENTIAL PHASE SHIFT KEYING BINARY).

En este tipo de modulación la información se codifica utilizando las diferencias de fases entre dos bits, existe cambio de fase si el dato de entrada es cero lógico y no existe cambio de fase si se tiene uno lógico, en el ejemplo que se ilustra en la figura 4.42, se toma  $\pi$  como el ángulo de referencia.

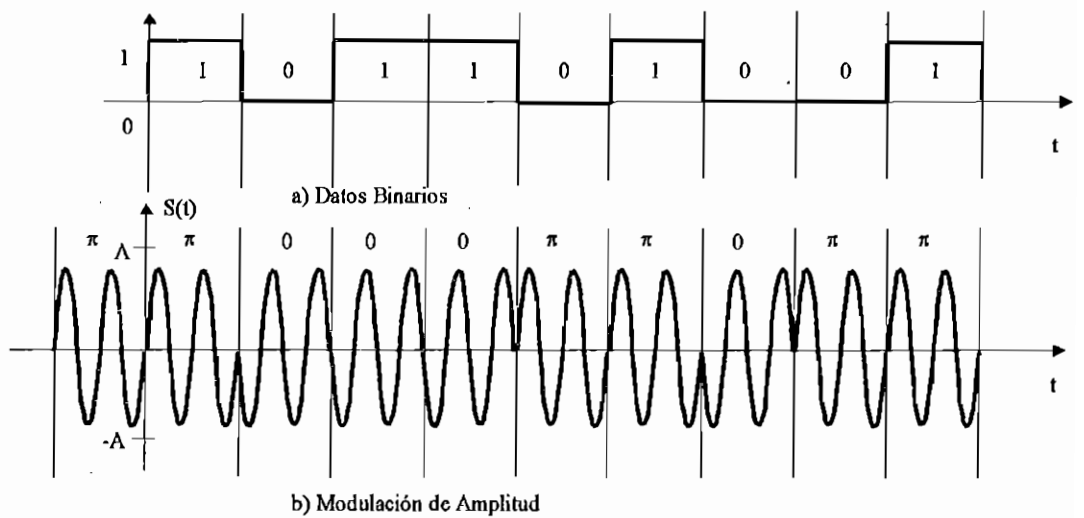


Figura 4.42 Ejemplo de Señal B-DPSK

En la figura 4.42 puede observarse como cuando se tiene un dato de cero la fase cambia con respecto a la anterior; y, cuando se tiene uno la fase es la misma que la anterior.

En la detección de la señal se toma como referencia la fase inmediatamente anterior y no una fase fija como el caso BPSK, en el cual puede darse el caso ambiguo de no saber si la portadora recuperada es coseno o seno.

#### 4.4.1 Modulador B-DPSK.

Un método para la modulación B-DPSK consiste en realizar antes de la modulación BPSK ya conocida, una codificación diferencial a partir de la secuencia de datos inicial. Esta secuencia diferencial se genera con ayuda de una compuerta lógica XOR complementada ( $\overline{XOR}$ ) y un bloque de retardo como se muestra en la figura 4.43.

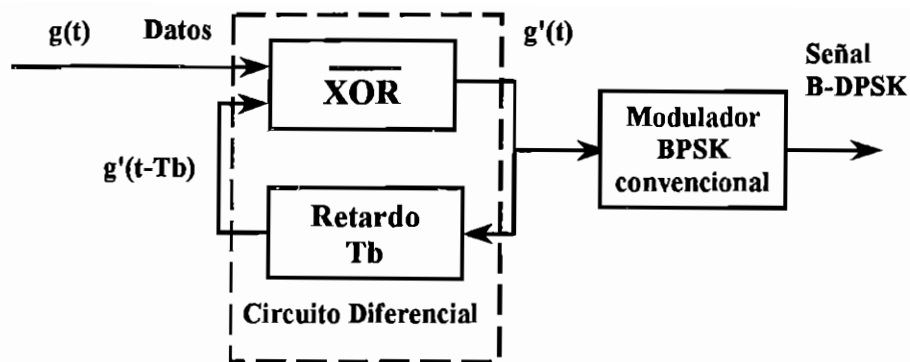


Figura 4.43 Modulador B-DPSK

La señal de datos se denomina  $g(t)$ ,  $g'(t)$  es la salida de la compuerta lógica  $\overline{XOR}$  y  $g'(t-T_b)$  es la señal  $g'(t)$  retardada un tiempo de bit  $T_b$ . Se asume un valor inicial para  $g'(t-T_b)$  igual a cero, con lo cual se evita el hecho de no saber que valores están a la entrada de la compuerta lógica. La señal diferencial  $g'(t)$  es la señal de datos para un modulador BPSK como los ya estudiados.

La estructura mostrada en la figura 4.43 puede comprobarse en el siguiente ejemplo de la Tabla 4.4. Donde se tiene la señal BPSK de salida del modulador luego de haber aplicado la operación  $\overline{XOR}$  y la respuesta en la parte denominada señal BPSK. En esta tabla se presenta una fila denominada Teórico donde a partir de los datos se tiene el resultado luego de aplicar las reglas generales con un ángulo de referencia de  $\pi$ .

t		0- $T_b$	1 $T_b$ -2 $T_b$	2 $T_b$ -3 $T_b$	3 $T_b$ -4 $T_b$	4 $T_b$ -5 $T_b$	5 $T_b$ -6 $T_b$	6 $T_b$ -7 $T_b$	7 $T_b$ -8 $T_b$
$g(t)$		0	1	1	0	0	1	0	1
$g'(t)$	0	1	1	1	0	1	1	0	0
Señal BPSK		0	0	0	$\pi$	0	0	$\pi$	$\pi$
Teórico	$\pi$	0	0	0	$\pi$	0	0	$\pi$	$\pi$

Tabla 4.4 Ejemplo de Modulación B-DPSK con el modulador y comparación con el resultado teórico.

#### 4.4.1.1 Simulación del Modulador B-DPSK.

Para desarrollar la simulación del modulador B-PSK se diseñó un bloque denominado circuito diferencial, el cual debe ir antes del bloque modulador BPSK, que puede ser el de la sección 4.3.1.3 o 4.3.1.4.

En la figura 4.44a se muestra un bloque modulador BPSK por defasaje y cómo debe utilizarse el bloque Diferencial, las letras Cod en la esquina superior derecha quiere decir codificador para distinguirlo del decodificador diferencial que se tratará en la sección 4.4.2.1. En la parte b se tiene la plantilla de datos del modulador PSK donde el usuario puede modificar la frecuencia de portadora. Luego en la parte c se encuentra el contenido desarrollado para el bloque Circuito Diferencial. Por último se tiene la plantilla de datos desarrollada para el Circuito Diferencial, dentro de la cual el usuario debe indicar el tiempo de bit de la señal de datos inicial.

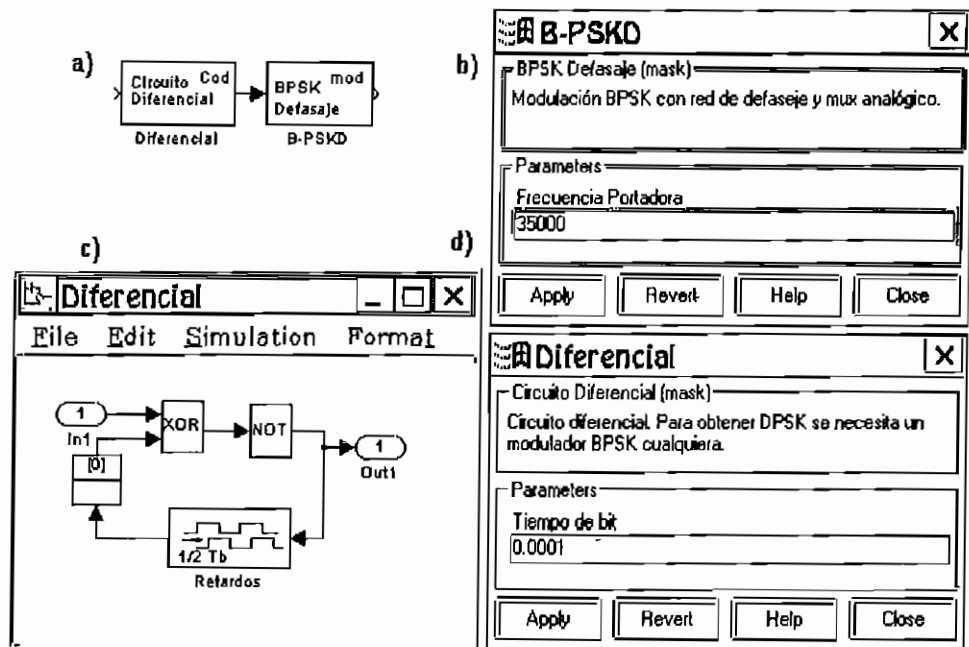


Figura 4.44 Modulador B-DPSK: a) Bloques que conforman un Modulador B-DPSK b) Plantilla de datos del modulador BPSK c) Subsistema del Bloque Diferencial d) Plantilla de datos del Bloque Diferencial.

El bloque Circuito Diferencial se diseñó a partir de dos bloques llamados *Logical operator* de la *librería no linear*, un bloque de retardos y un bloque denominado *IC (Initial Conditions)* de la librería *conections*. En el bloque *Logical operator* de la parte superior izquierda se tomó la opción XOR de dos entradas y en el de la derecha se tomó la opción not de una sola entrada. En el bloque de retardos se especifica el tiempo de bit y que el retardo es por dos semiperiodos de la señal, es decir un tiempo de bit. Finalmente el bloque *IC* contiene el valor inicial de cero para evitar que en un inicio no exista respuesta de los bloques lógicos.

#### 4.4.2 Demodulador B-DPSK.

Para demodular la señal B-DPSK se puede utilizar un demodulador BPSK seguido de un circuito decodificador diferencial, tal como se muestra en la figura 4.45. La señal ya demodulada ingresa al circuito diferencial y entrega como resultado la señal original de datos.

En la figura antes mencionada se puede ver que la señal demodulada  $g'(t)$  ingresa al proceso  $\overline{XOR}$  con su respectiva señal retardada en un tiempo de bit denominada  $g'(t-T_b)$ . La señal resultante del bloque  $\overline{XOR}$  corresponde a los datos originalmente enviados. Evidentemente  $g'(t-T_b)$  tiene un valor inicial de cero de acorde con el primer valor tomado para la modulación.

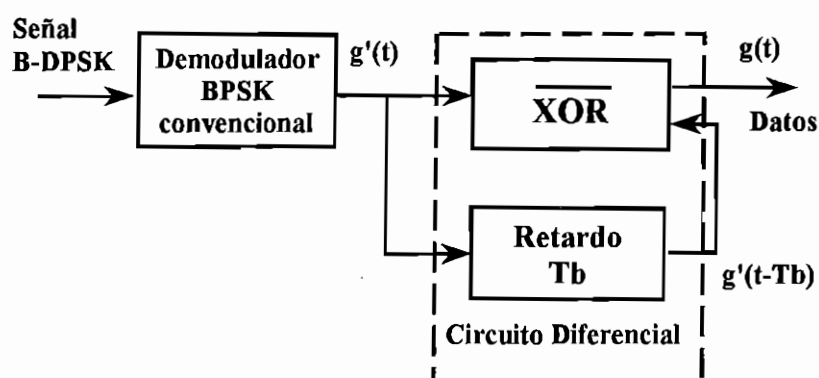


Figura 4.45 Demodulador B-DPSK

Mediante un ejemplo se puede comprobar el funcionamiento del circuito decodificador diferencial. Se tiene la señal demodulada  $g'(t)$ , la señal  $g'(t-T_b)$  y la señal  $g(t)$  como resultado de la operación lógica  $\overline{XOR}$  entre las dos primeras señales. La señal  $g'(t)$  es la misma de la tabla 4.4 y corresponde a la demodulación de este ejemplo. Se puede ver como la señal  $g'(t-T_b)$  se encuentra retardada un tiempo de bit con respecto a  $g'(t)$ , con un valor inicial igual a cero. La señal  $g(t)$  resultante corresponde a los datos iniciales antes de la modulación como puede comprobarse en la tabla 4.4, con lo cual se ha mostrado el funcionamiento de la decodificación diferencial.

t	0-T <sub>b</sub>	1T <sub>b</sub> -2T <sub>b</sub>	2T <sub>b</sub> -3T <sub>b</sub>	3T <sub>b</sub> -4T <sub>b</sub>	4T <sub>b</sub> -5T <sub>b</sub>	5T <sub>b</sub> -6T <sub>b</sub>	6T <sub>b</sub> -7T <sub>b</sub>	7T <sub>b</sub> -8T <sub>b</sub>
$g'(t)$	1	1	1	0	1	1	0	0
$g'(t-T_b)$	0	1	1	1	0	1	1	0
$g(t)$	0	1	1	0	0	1	0	1

Tabla 4.5 Ejemplo de Demodulación B-DPSK.

#### 4.4.2.1 Simulación del Demodulador B-DPSK.

Se diseñó la demodulación en dos bloques, el primero es el demodulador 2-PSK de la sección 4.3.2.1 y el segundo se desarrolló para obtener la decodificación.

El diseño del bloque decodificador consta de un bloque de retardos de la librería de fuentes y dos bloques *Logical operator* (librería *no linear*) que se utilizan para cumplir con la operación lógica  $\overline{XOR}$ . En la figura 4.46c se muestra como está diseñado el bloque decodificador denominado Diferencialdec o Circuito Diferencial dentro del gráfico del bloque, además

se tiene las siglas Dec que señalan que el bloque mencionado realiza la decodificación. Adicionalmente se utiliza un bloque de retardos antes de la salida del bloque Diferencialdec para evitar unos pequeños pulsos que se presentan por la falta de sincronía entre las señales que ingresan a los bloques lógicos.

En la figura 4.46a se presenta la conexión de los bloques y en las partes b y d las respectivas plantillas de datos que el usuario debe llenar con los datos respectivos de tiempo de bit y frecuencia de portadora.

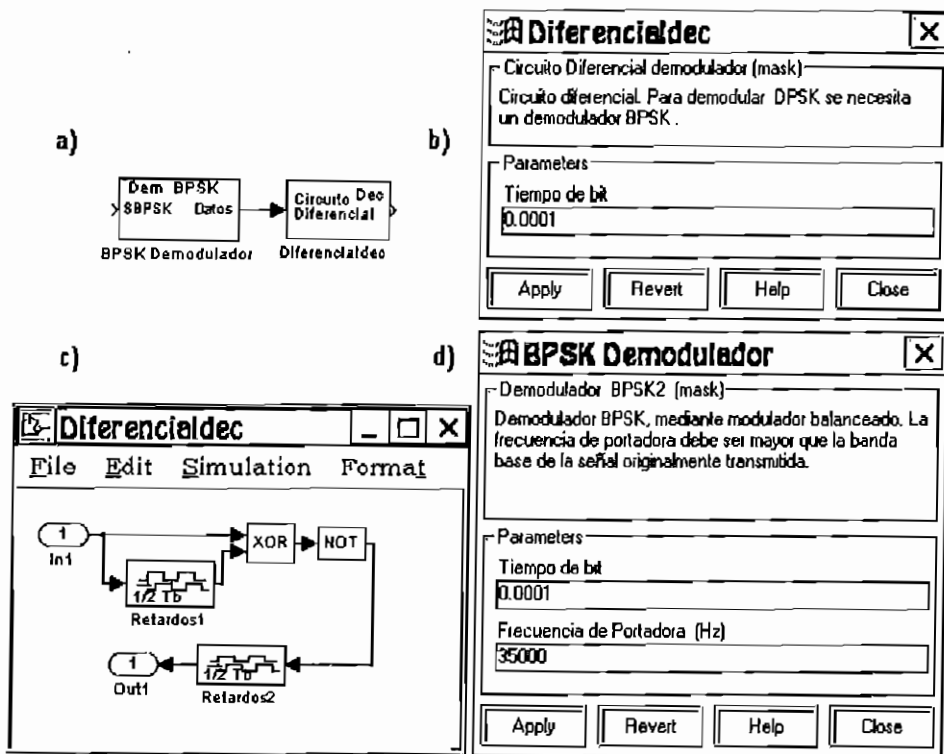


Figura 4.46 Bloque Demodulador B-DPSK: a) Bloques que se agrupan para la demodulación. b) Plantilla de datos del Circuito Diferencial decodificador. c) Bloques que consituyen el bloque diseñado Circuito Diferencial. d) Plantilla de datos del Bloque Demodulador BPSK.

#### 4.5 MODULACIÓN QAM (QUADRATURE AMPLITUDE MODULATION).

La modulación QAM puede verse como una extensión de la modulación PSK, pero las señales en banda base, es decir los coeficientes de la ecuación 4.20 son generados de manera independiente, en tanto que en la modulación PSK están relacionados entre sí. La ecuación 4.23 representa una señal QAM en general

$$s_{QAM}(t) = x(t)\cos(2\pi f_c t) - y(t)\sen(2\pi f_c t) \quad (4.23)$$

en donde  $x(t)$  y  $y(t)$  toman en forma independiente los valores discretos previstos según el número de niveles establecidos, siendo  $M=2^L$  ( $L$  es el número de niveles de cada canal en cuadratura).

Para el caso especial de dos niveles ( $\pm 1$ ), el sistema no es más que el 4-PSK. En la figura 4.47 se representa el diagrama de constelación para el caso del 16-QAM donde las variables  $x(t)$  y  $y(t)$  pueden tomar los valores comprendidos entre  $(-3,-1,+3,+1)$ , cada valor de  $x(t)$  y  $y(t)$  es asociado con un par de bits. Se asume la regla de codificación de Gray que dice que dos puntos en el diagrama de constelación adyacentes<sup>1</sup> deben variar en un solo bit, este criterio se tomó de la referencia [10].

---

<sup>1</sup> Son puntos adyacentes a aquellos que difieren un solo bit entre sí



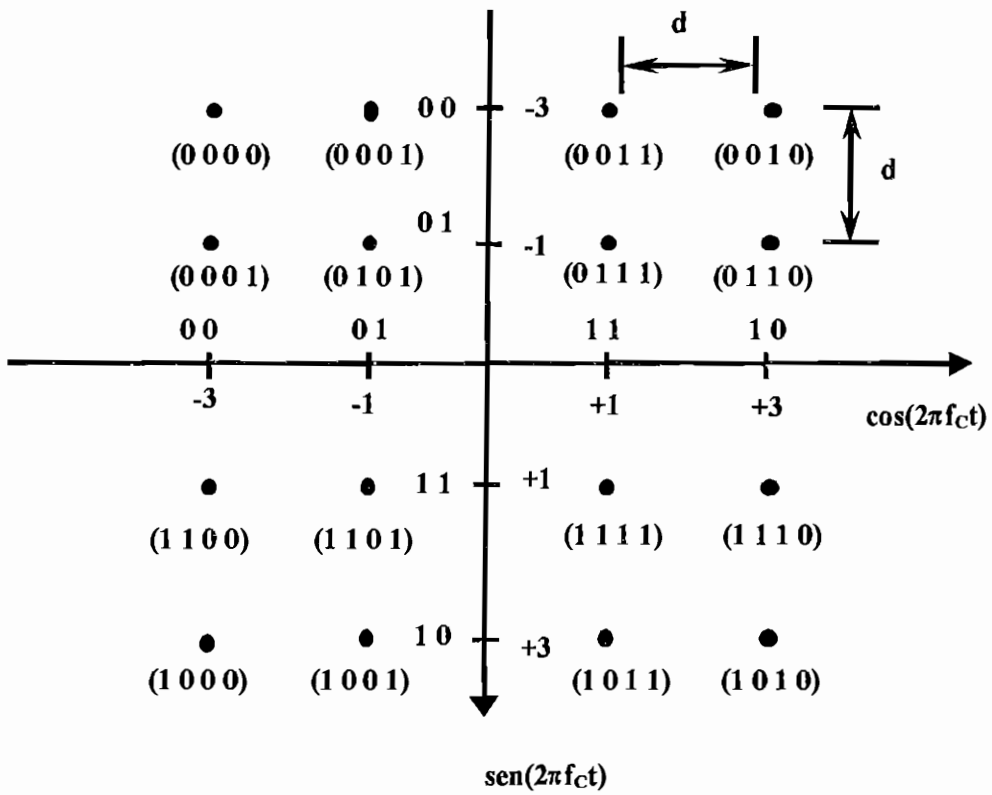


Figura 4.47 Diagrama de Constelación de la modulación 16-QAM

Como se observó en los diagramas de constelación de la modulación PSK, todos los puntos se encuentran sobre la circunferencia y mientras más se incrementa el orden de la PSK la distancia entre estos puntos es cada vez menor, a menos que se incremente el radio de la circunferencia lo que implica un incremento en la potencia de transmisión.

Por otro lado, la distancia entre puntos en un esquema QAM es siempre mayor que en un esquema PSK equivalente para  $M > 4$ . Esto puede verificarse a partir de la ecuación 4.24 que determina la distancia entre puntos adyacentes para la modulación PSK:

$$d = 2 \cdot \text{sen}\left(\frac{\pi}{M}\right) \quad (4.24).$$

y la ecuación 4.25 que representa la distancia entre puntos adyacentes de la modulación QAM.

$$d = \frac{\sqrt{2}}{(L-1)}, \quad L = \sqrt{M} \quad (4.25).$$

#### 4.5.1 Modulador QAM.

La modulación QAM es similar a la modulación PSK, con componentes en cuadratura según la ecuación 4.23. Los datos binarios son ingresados a un convertidor serie a paralelo, luego se les da el valor de  $x(t)$  y  $y(t)$ , que son constantes para cada grupo de  $\ell$  bits y estas señales son ingresadas a un modulador balanceado como se muestra en la figura 4.48.

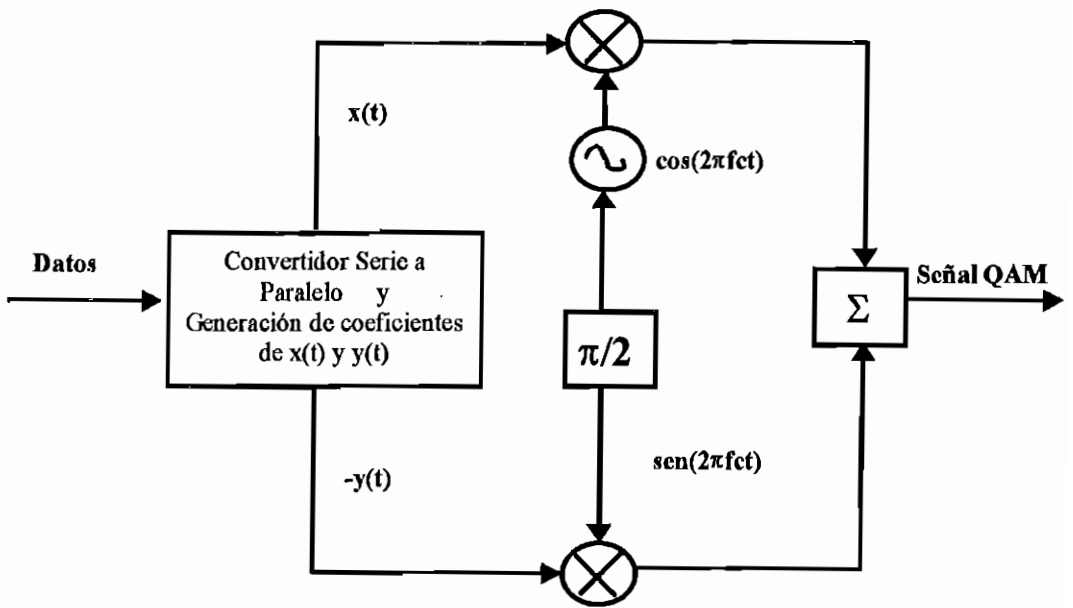


Figura 4.48 Generación de QAM mediante componentes en cuadratura

#### 4.5.1.1 Simulación de Modulación 16-QAM.

El bloque de simulación de la modulación 16-QAM se diseñó a partir del esquema de la figura 4.48. Los valores de  $x(t)$  y  $y(t)$  correspondientes para cada grupo de cuatro bits son los del diagrama de constelación de la figura 4.47, es decir por ejemplo si los primero cuatro bits son 1011, entonces  $x(t)=+3$  y  $y(t)= +1$ . Los datos binarios que ingresan al bloque modulador, se convierten de serie a paralelo con una salida de cuatro bits, luego se asigna los valores respectivos de  $x(t)$  y  $y(t)$ , para posteriormente multiplicar estas constantes por  $\cos(2\pi f_c t)$  y  $-\text{sen}(2\pi f_c t)$  respectivamente; y, sumar estos resultados al final obteniéndose así la señal 16-QAM.

En la figura 4.49 se muestra el bloque diseñado y su respectiva plantilla de datos que le permite al usuario modificar el tiempo de bit y la frecuencia de portadora.

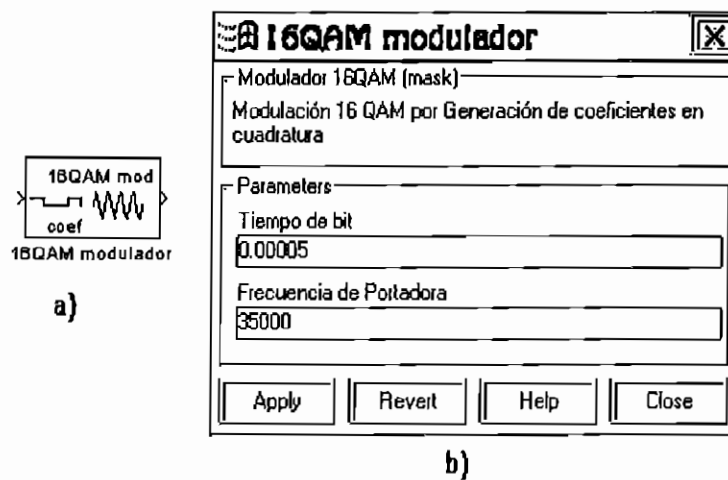


Figura 4.49 Modulador 16-QAM: a) Bloque de Simulación b) Plantilla de Datos

El desarrollo de este bloque es el mismo que el bloque modulador 16-PSK, como se puede observar en la figura 4.50b, donde se encuentra el esquema general y el contenido del bloque de generación de coeficientes

en la figura 4.50a. La única variación se encuentra en los datos asignados a  $x(t)$  y  $y(t)$ .

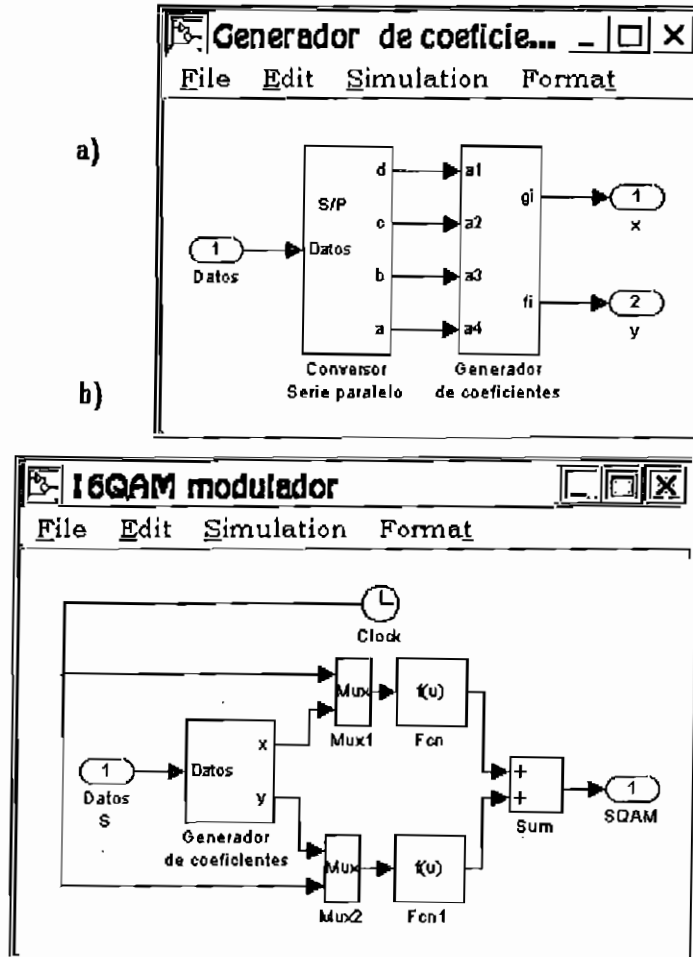


Figura 4.50 Contenido del Bloque Modulador 16-QAM: a) Bloques que forman el Bloque de Simulación 16-QAM. y b) Bloque Generador de coeficientes.

#### 4.5.2 Demodulación QAM.

En la figura 4.52 se encuentra un esquema de la demodulación QAM. Se utilizan dos detectores de fase los cuales entregan los valores de  $x(t)$  y  $y(t)$ , posteriormente en el circuito de toma decisiones se comparan estos niveles y se entregan los datos correspondientes de manera serial, luego

de un convertidor paralelo serie. Los detectores de fase son exactamente los mismos que para la modulación PSK.

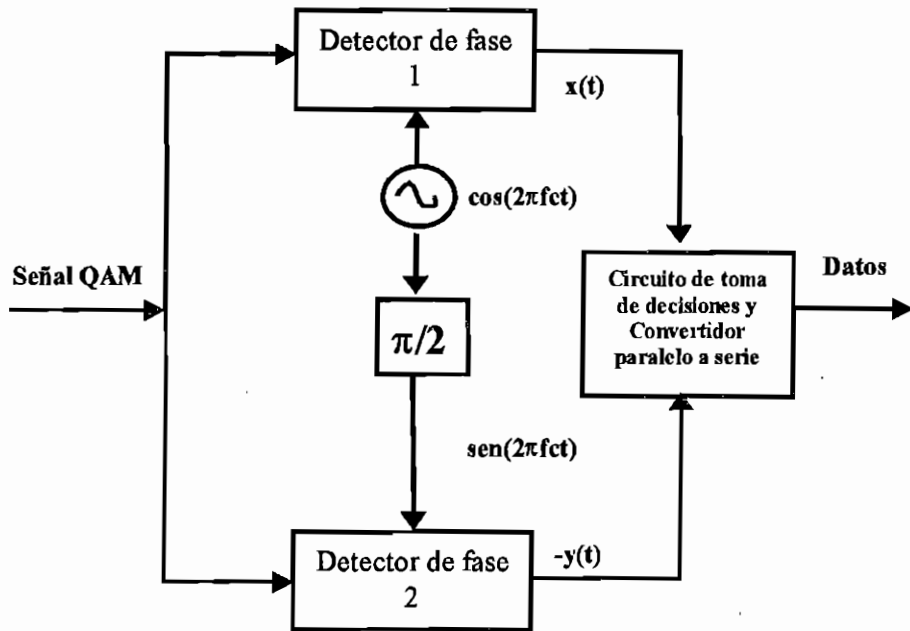


Figura 4.51 Esquema General de la demodulación QAM

#### 4.5.2.1 Simulación de Demodulación 16-QAM.

La demodulación se realiza mediante dos detectores de fase como en el caso de la demodulación 16-PSK, con el cuidado que se debe detectar los niveles previstos para las señales multinivel  $x(t)$  y  $y(t)$ . Una vez obtenidos los valores de  $x(t)$  y  $y(t)$ , según su equivalencia se asigna los valores de cuatro bits correspondientes y luego se convierten los datos de paralelo a serie.

El bloque diseñado se encuentra representado en la figura 4.52, el usuario debe ingresar los datos de tiempo de bit y frecuencia de portadora, como lo indica la plantilla de datos de la parte b. Los detectores de fase son los mismos que los de la demodulación PSK, de igual manera la conversión paralelo a serie, solo se desarrolló de manera diferente el circuito lógico.

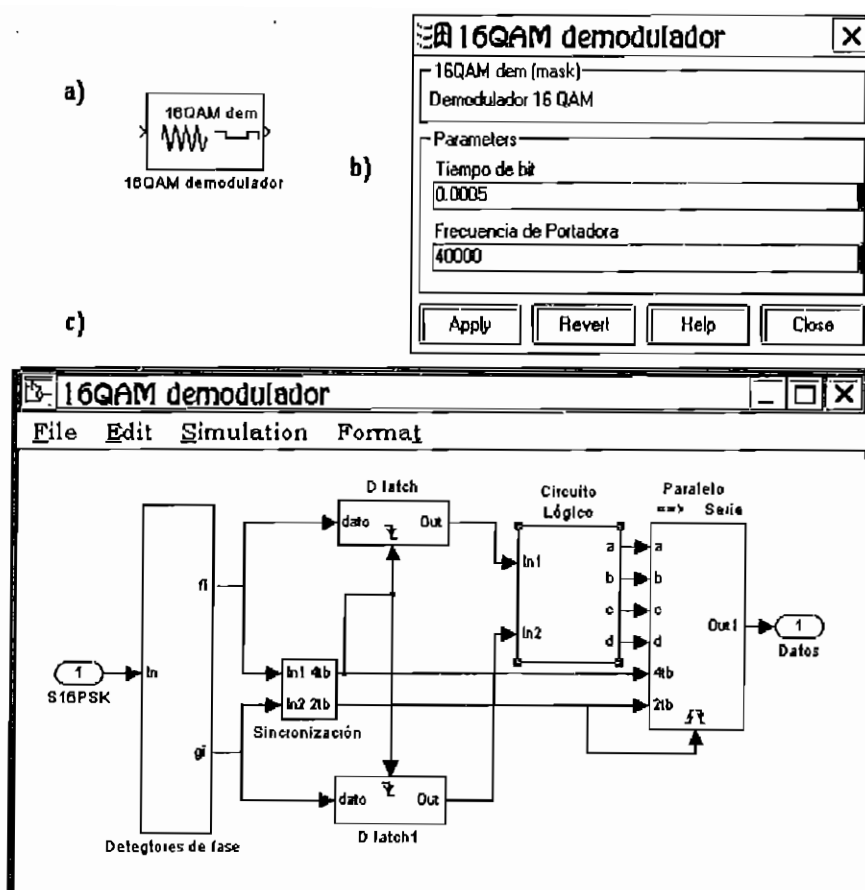


Figura 4.52 Demodulador 16-QAM: a) Bloque de Simulación b) Plantilla de Datos c) Bloques que forman el Bloque de Simulación 16-QAM.

En la figura 4.53 se tiene el detalle del circuito lógico el cual es mucho más simple que el utilizado en la demodulación 16-PSK. Aquí se puede observar que la comparación se la realiza con los bloque Fcn, los cuales van a entregar el valor de 1 o cero de acuerdo a cada condición de la variable de entrada.

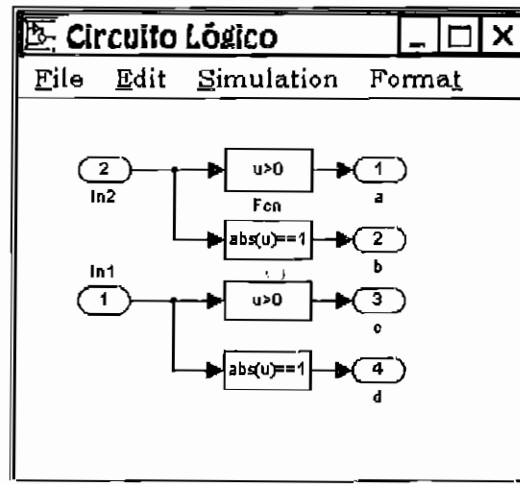


Figura 4.53 Circuito Lógico del Demodulador 16-QAM.

Adicionalmente los bloques D latch de la figura 4.52 tienen comparadores de nivel que permiten determinar los valores de las señales que se obtienen a partir de los detectores de fase.

#### 4.6 DENSIDAD ESPECTRAL DE POTENCIA DE LA MODULACIÓN PSK Y QAM.

Tanto para las señales PSK como para las QAM, en la referencia [2] se encontró la expresión 4.26 para la densidad espectral de potencia para datos NRZ.

$$S(f) = \frac{T_s}{4} \left( \frac{\text{sen}(\pi(f - f_c)T_s)}{\pi(f - f_c)T_s} \right) \quad (4.26).$$

donde  $T_s$  es el tiempo de duración del símbolo y es igual a  $\ell \cdot T_b$ .

## 4.7 EJEMPLOS DE SIMULACIÓN.

En esta sección se tratará acerca de la utilización de los bloques de simulación diseñados. Dentro del programa de inicio (cuyo comando es la palabra "comdig") se encuentran disponibles las librerías de modulación digital y algunos ejemplos útiles para el usuario. En las secciones 4.7.1 se tendrá un ejemplo práctico para la modulación digital y en 4.7.2 se mostrará como se puede tener una estimación de la Densidad Espectral de Potencia con el bloque desarrollado en la sección 2.2.4.

El programa de inicio es una ventana que se presenta con el comando comdig (figura 4.54), ésta se diseñó para que presente las librerías desarrolladas en la tesis de una manera gráfica y el usuario no tiene que escribir varios comandos para encontrar el bloque de simulación o el ejemplo deseado.

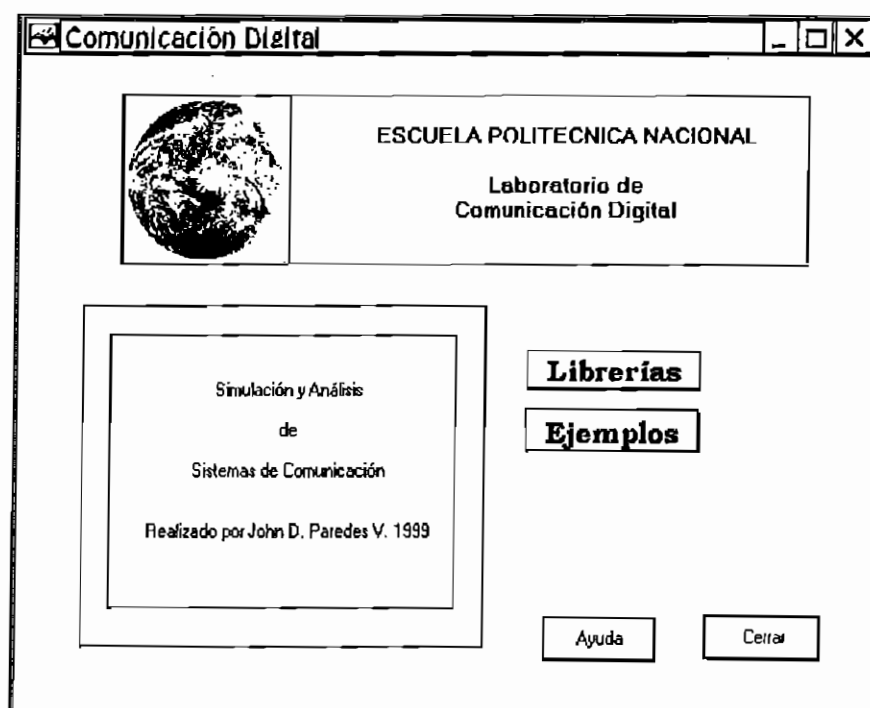


Figura 4.54 Ventana General del comando 'comdig'



Una vez que se tiene la ventana de la figura 4.54 se pueden elegir las opciones de librerías o ejemplos. Al seleccionar el botón librerías se tiene la figura 4.55, luego con ayuda del ratón se puede disponer de la librería deseada. Con la opción de Nuevo modelo se activa un modelo con el nombre de *untitled*, que quiere decir sin título, el cual se encuentra vacío y con los valores por defecto de los parámetros de simulación.

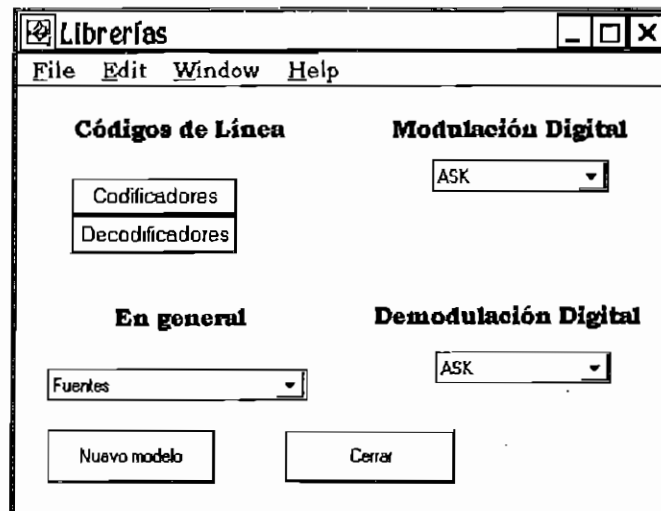


Figura 4.55 Librerías Desarrolladas en esta Tesis

Entonces para la simulación de un nuevo modelo de modulación digital se toma la opción de nuevo modelo y los bloques necesarios que se encuentran en las librerías de Modulación Digital, Demodulación Digital y las librerías que se encuentran bajo el título de "En general", donde se tienen las fuentes digitales, filtros y otras utilidades bajo la librería análisis.

Se diseñaron varios ejemplos de modulación y el usuario puede disponer de ellos bajo la opción ejemplos de la ventana principal de la figura 4.54. Cuando se toma la selección ejemplos aparece la ventana desarrollada de la figura 4.56 y el usuario puede elegir el tipo de modulación bajo el menú Modulación Digital, el tipo de ejemplo disponible

aparecerá en el menú denominado Ejemplos en la ventana y por último una vez seleccionado el ejemplo de este menú el usuario puede llamar a un ejemplo a través del botón Ejecutar.

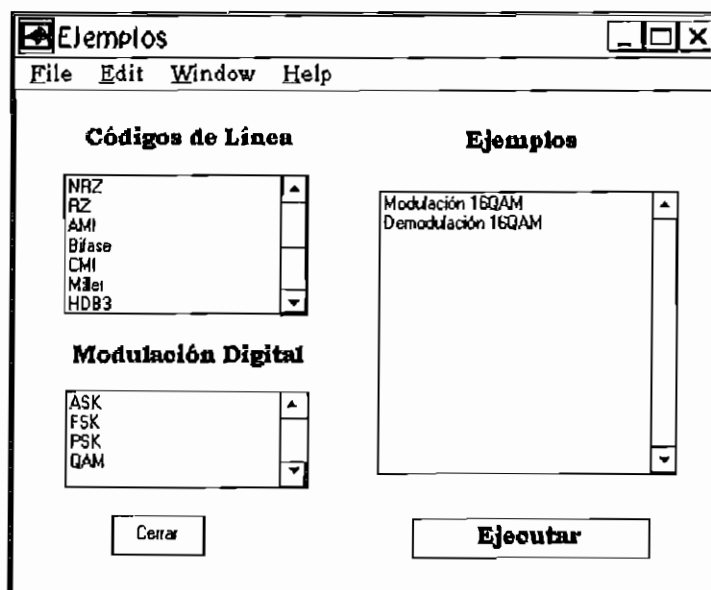


Figura 4.56 Ejemplos Desarrollados en esta Tesis

Estos son los pasos generales que el usuario puede utilizar para no escribir en la línea de comandos el nombre de cada librería o el nombre de cada modelo de ejemplo para realizar consultas de su propio interés. Los ejemplos desarrollados permiten al usuario realizar cualquier tipo de cambio, por lo que se recomienda tener un respaldo.

#### 4.7.1 Ejemplo de Simulación de Modulación Digital.

El ejemplo que se desarrollará a continuación será muy simple con el fin de ilustrar al usuario y no perder el objetivo de mostrar la utilización de las librerías. Los ejemplos disponibles son mucho más complejos y de ayuda al usuario cuando ya esté familiarizado con todas las herramientas

disponibles, tanto las desarrolladas en esta Tesis como las disponibles en el *SIMULINK*.

El proceso de creación de modelos de simulación es el mismo que el utilizado para la simulación de códigos (Sección 3.3.1), donde se vio cómo se empieza un nuevo modelo, cómo se realiza el llenado del modelo con los bloques deseados, las conexiones respectivas y la forma que debe llenarse los datos y parámetros de simulación. En esta sección se creará el modelo de una manera simple sin entrar en detalles tales como los parámetros de simulación o procesos de conexión los cuales se encuentran muy bien descritos en la referencia [3].

El ejemplo elegido para esta sección es la modulación BPSK por el método de la generación de coeficientes, en este ejemplo se desea presentar la señal de reloj, los datos digitales, la señal modulada y la señal demodulada. Siguiendo los pasos descritos en la introducción a la sección 4.7 se empieza un nuevo modelo, el cual en un inicio se denomina por defecto *untitled* y se encuentra desde luego vacío; es sobre este modelo vacío donde se van a colocar los bloques necesarios para la simulación.

De la librería fuentes se toma el bloque Datos digitales secuenciales y el bloque reloj. El bloque modulador se toma del menú de librerías bajo el nombre de Moduladores Digitales de la figura 4.55, aquí se desplegará una serie de opciones de las cuales se toma la selección denotada por PSK, aparecerá un modelo donde se encuentran los bloques diseñados para la modulación PSK en este sitio se encuentra el bloque denominado B-PSKF. De la misma manera se busca el bloque demodulador. Finalmente en la librería *connections* y en la librería *no linear* se tiene los elementos para generar el subsistema o bloque muxm, compuesto de cuatro bloques *Fcn* y un bloque *mux*.

Al trasladar los bloques de las respectivas librerías y realizar las conexiones se tiene el modelo para la simulación como el de la figura 4.57. El bloque Muxm solo sirve para adaptar las señales al bloque *scope* que se utiliza para visualizar los resultados. Cuando se han llenado todos los datos requeridos para cada bloque y los parámetros de simulación se aconseja guardar el modelo, en este caso se lo hizo con el nombre de b-psk2dem.mdl.

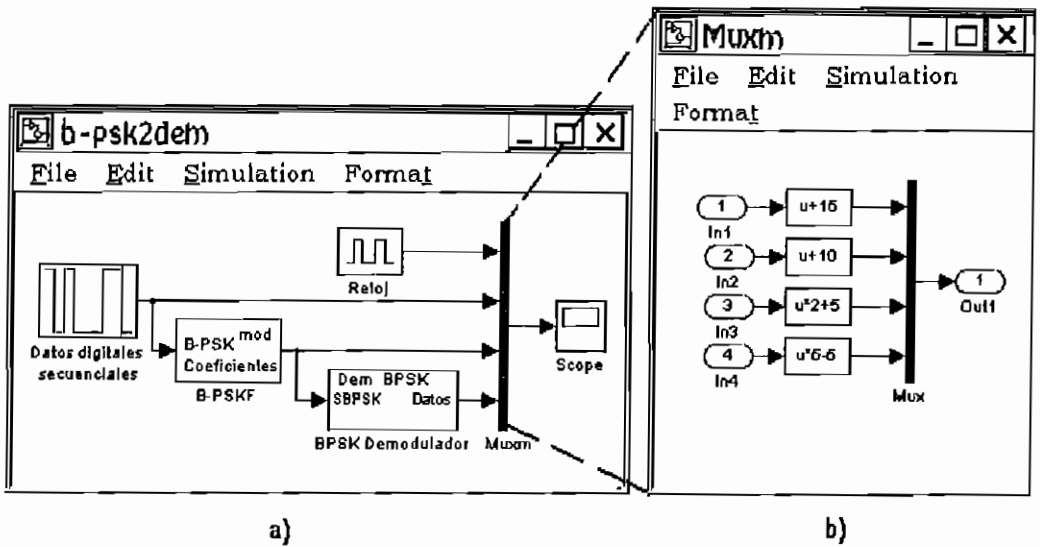


Figura 4.57 Ejemplo de Modulación BPSK: a) Modelo de Simulación y b) Detalle del bloque Muxm.

En la plantilla de datos del bloque de Datos secuenciales se ingresa los valores de 0.1 en el "Tiempo de bit", [1 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1] en "Secuencia de datos" y uno en "Amplitud de Salida". La plantilla de datos del bloque modulador y del demodulador se llenan con los datos de "tiempo de bit" de 0.001 y una frecuencia de portadora de 10000. Se debe recordar que para obtener las plantillas de datos debe realizarse sobre cada bloque un doble "click".

Dentro del bloque muxm que se puede observar en la parte derecha de la figura 4.57 los contenidos de los bloque Fcn y la presencia del

bloque *Mux*, en conjunto cumplen con la función de dar a las señales unos niveles adecuados para la visualización de resultados en el bloque *Scope*.

EL último bloque que requiere de datos es el denominado *Scope* donde se tiene los datos de la figura 4.58.

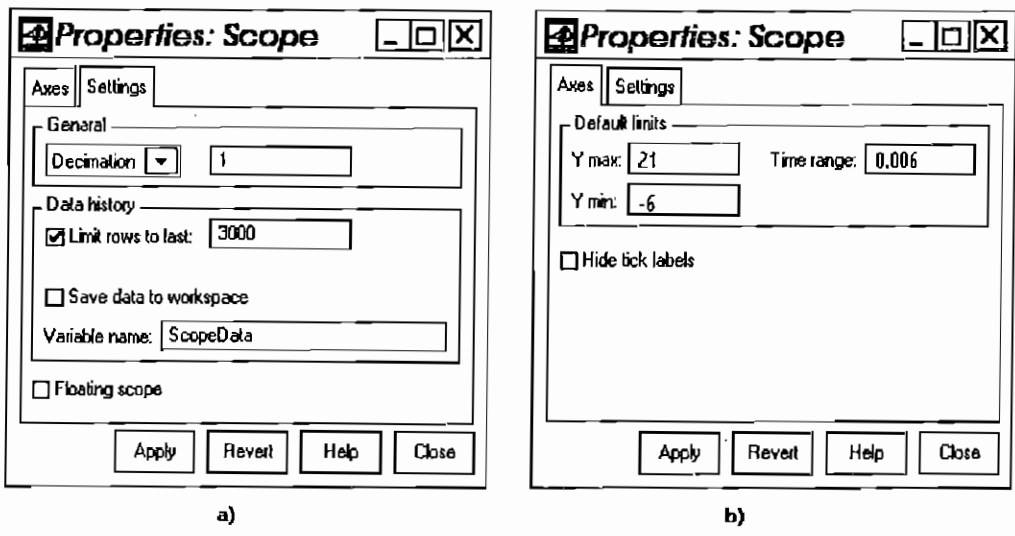


Figura 4.58 Plantillas de datos del bloque *Scope*: a) Opción *Setting* b) Opción *Axes*

Si ya se han cumplido todos los pasos anteriores se deben dar valores adecuados en la sección "*simulation parameters*" de la opción *simulation* del menú donde se encuentra el modelo a simular. Los datos requeridos en esta sección son los mismos que para el ejemplo de la sección 3.3.1.

Finalmente cuando se han llenado los parámetro de simulación se puede ya ejecutar la simulación desde el menú del modelo bajo la opción *simulation*. En la figura 4.59 se encuentra el resultado de la simulación, con ayuda del bloque muxm se pueden tener las señales ordenadas de esta forma, de otro modo estarían sobrepuestas. Desde la parte superior a la inferior del gráfico, se tiene las señales de reloj, datos, señal modulada y señal demodulada.

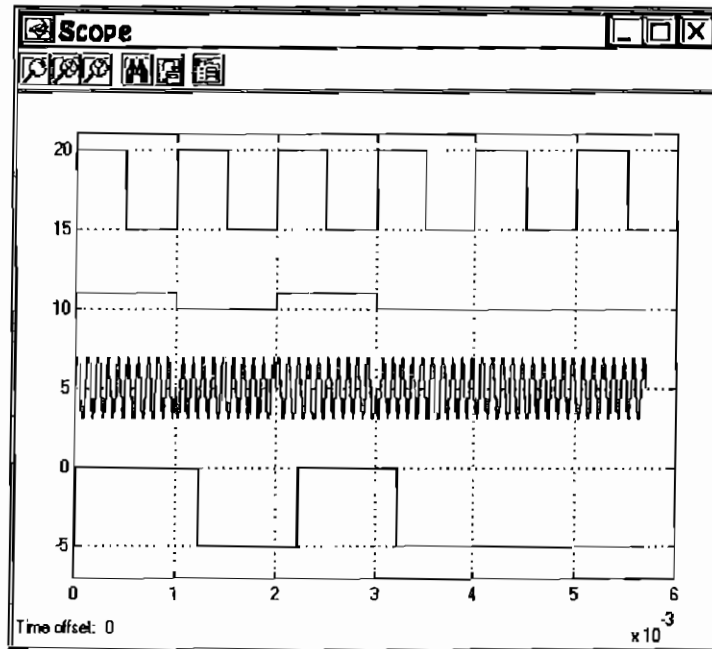


Figura 4.59 Resultados de la Simulación del modelo b-psk2dem.mdl

#### 4.7.2 Ejemplo de la obtención de la Densidad Espectral de Potencia de una señal Modulada.

Siguiendo los mismos pasos que para la creación del modelo del ejemplo de la sección anterior se creó el modelo denominado b-psk2psd. Un nuevo bloque se ha añadido, el denominado Densidad Espectral de Potencia II, el cual nos permitirá tener una medida gráfica de la Densidad Espectral de potencia de la señal BPSK.

Después de haber colocado los bloques y realizado las conexiones necesarias se tiene el modelo de la figura 4.60. Aquí se puede observar que la fuente de datos ha cambiado con relación a la anterior ya que ésta permite tener un buen número de datos aleatoriamente pero con una distribución gaussiana.

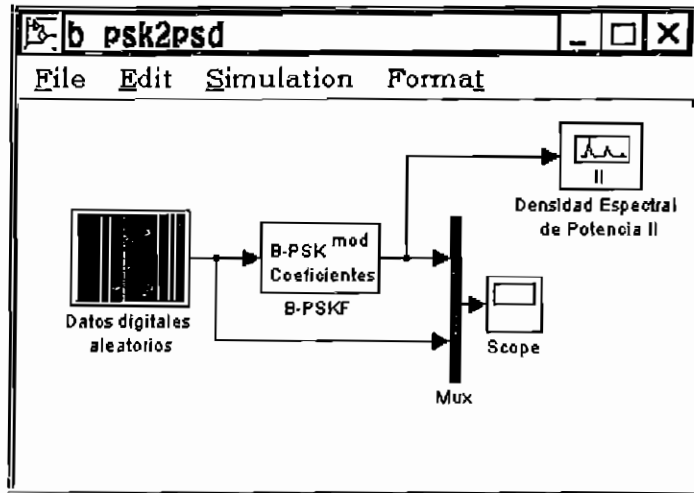


Figura 4.60 Modelo de Simulación para obtener la Densidad Espectral de Potencia de una señal BPSK, cuyo nombre es b-psk2psd.mdl

El bloque de mayor importancia en este modelo es el bloque de Densidad Espectral de potencia, cuya plantilla de datos es la de la figura 4.61.

**Densidad Espectral de Pot...** [X]

Analizador de Espectros II. (mask)

Analizador de Espectros en una ventana gráfica. Eje horizontal frecuencia por tiempo de bit y el Eje vertical potencia.

---

Parameters:

Número de Datos tomados del muestreo  
4096

Número de puntos para la fit  
4096

Número de puntos para graficar:  
4096

Tiempo de Muestreo (recomendación  $T_b/100$ ):  
1/10000

Eje horizontal ( $f \cdot T_b$ ) de la PSD (xmin xmax)  
{6 14}

Eje vertical la PSD (ymin ymax)  
{0 60}

Tiempo de bit (sec)  
1/1000

Figura 4.61 Plantilla de datos del bloque de Densidad Espectral de Potencia.

En la figura 4.62 se tiene el resultado gráfico, que coincide con lo esperado para una señal BPSK, es decir la forma del Densidad espectral de Potencia es la misma que para el código NRZ unipolar.

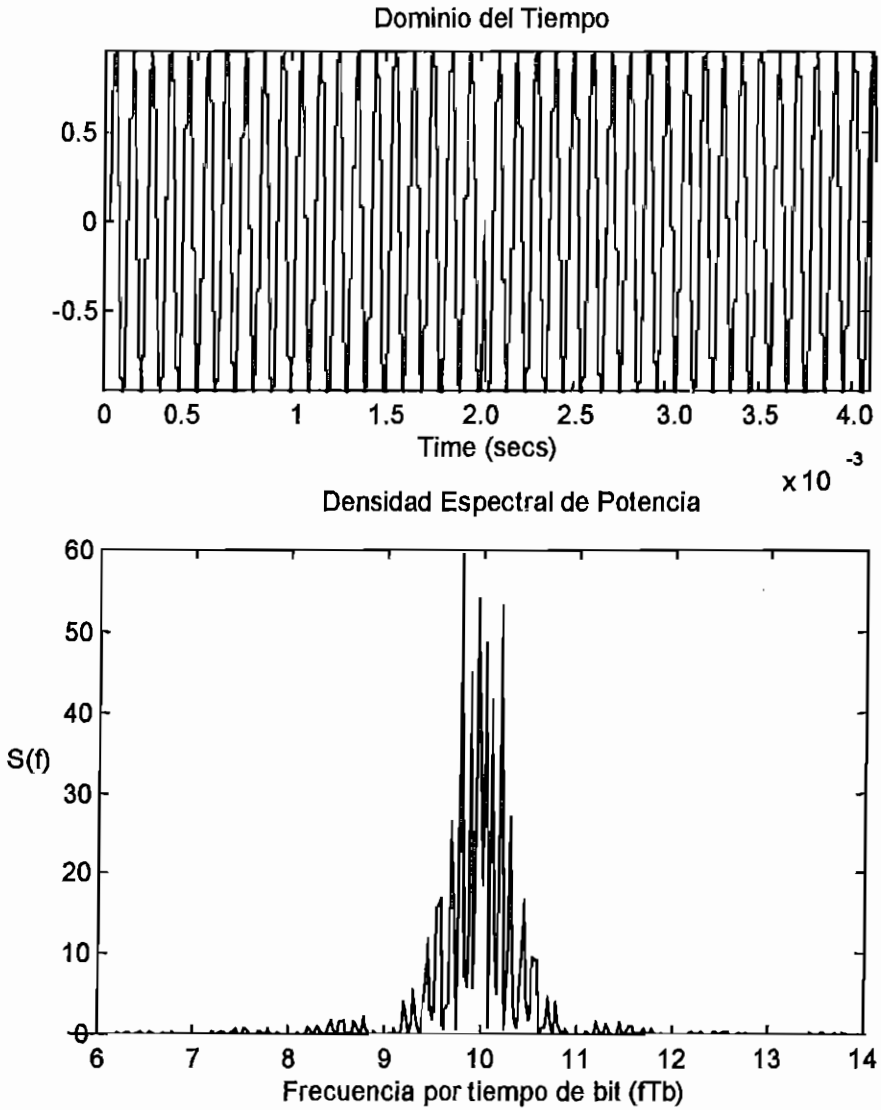


Figura 4.62 Resultados del bloque de Densidad Espectral de Potencia.



#### 4.8 REFERENCIAS.

1. Strembler F.G, Introducción a los Sistemas de Comunicación, Addison Wesley, México 1993
2. Leon W. Couch II, Digital and Analog Communication Systems, four edition, Macmillan Publishing Company New York 1993.
3. James B. Dabney y Thomas L. Harman. "The Student Edition of SIMULINK® User's Guide", Prentice Hall, 1997.
4. Nelson Avila, "Diseño y Construcción de un Codec Didáctico", Tesis de Grado previa a la Obtención del Título de Ingeniero en Electrónica y Telecomunicaciones, EPN. 1994.
5. Bellamy J., "Digital Telephony", John Wiley & Sons, Nuew York, 1991.
6. Vidaller J. y Otros, "Transmision de Datos", E.T.S. Ingenieros Telecomunicaciones Universidad Politécnica de Madrid, Madrd 2ª ed., 1979.
7. Math Works INC. "Signal Proccesing Toolbox", 1997.
8. Savant C. J. y Otros, Diseño Electrónico, Ed. Addison Wesley, 1992.
9. Coughlin R. y Driscoll F. Amplificadores Operacionales y Circuitos Integrados Lineales, Ed. Prentice Hall, 1993.
10. M.T. Hill y A. Cantoni, "A Frecuency Steered Phase Locked Loop", IEEE Transactions on Comunications, vol 45, Junio de 1997 págs, 737-743.
11. Castro y Orbe, "Análisis y Simulación de un Sistema de Transmisión Digital Utilizando un Microcomputador", Tesis de Grado previa a la Obtención del Título de Ingeniero en Electrónica y Telecomunicaciones, EPN. 1995.

## CAPÍTULO 5 CONCLUSIONES

La simulación ha adquirido cada vez más importancia en el diseño de sistemas de comunicación, la herramienta desarrollada presenta una alternativa didáctica para el estudio de los códigos de línea, modulación digital y análisis de sus correspondientes densidades espectrales de potencia.

Un proceso de simulación digital que está representando señales analógicas, lo hace siempre mediante una representación discreta. En consecuencia se está hablando de señales "pseudo-analógicas", que han sido muestreadas con la frecuencia suficiente para representarse como analógicas<sup>1</sup>; los bloques de simulación de esta tesis lo hace de igual modo. En la simulación se debe considerar que existe un compromiso entre la capacidad de datos muestreados y la precisión deseada.

Durante la simulación se puede considerar que los bloques discretos tienen internamente un convertidor analógico digital para la señal de entrada y un convertidor digital/analógico para la respuesta del bloque en el modelo o sistema en ejecución. En el manual del *SIMULINK*, se encuentran las consideraciones que deben hacerse para trabajar en un modelo de simulación con bloques que representen sistemas discretos y sistemas continuos<sup>2</sup>.

Otro aspecto importante de la simulación es, que dentro del proceso se pueden cambiar algunas variables y ver en ese momento su efecto, lo

---

<sup>1</sup> Tomar en cuenta el teorema del muestreo de las referencias [1] y [2] del capítulo 4.

<sup>2</sup> Se consideran sistemas continuos los representados por ecuaciones diferenciales y discretos aquellos que se representan por ecuaciones en diferencias.

que permite adaptar, por ejemplo, la frecuencia de corte de un detector ASK o la velocidad de la transmisión, etc.

En la simulación de los códigos de línea fue de mucha utilidad el bloque *S-function*, en su forma discreta y se recomienda un estudio más detallado sobre este bloque, ya que permite generar un sinnúmero de nuevos bloques de simulación.

Para la modulación digital del tipo PSK fue necesario generar bloques que permitan sincronizar los datos binarios, para esto se diseñaron bloques que simulan el comportamiento de un oscilador generado por voltaje o VCO. Este VCO se desarrolló a partir del esquema de un generador de onda cuadrada.

Dentro de la demodulación FSK, se desarrolló un bloque de simulación de un PLL (*Phase Locked Loop*), además se diseñaron bloques como el repetidor regenerativo (a partir de la creación de un bloque monoestable), los convertidores serie a paralelo y paralelo a serie, lo que demuestra la versatilidad de la herramienta que representa el *SIMULINK*.

La demodulación digital en esta tesis no considera recuperación de portadora, ni recuperación de sincronía, por cuanto el objetivo es mostrar didácticamente el proceso demodulación; consecuentemente se recomienda un tema de tesis que abarque estos dos conceptos, ya que existen para su desarrollo varios métodos que podrían ser investigados.

Se desarrolló en el capítulo tercero un bloque para la simulación de un canal contaminado con ruido blanco gaussiano, a partir de esto se observa la posibilidad de modelar canales reales de transmisión tales como los de atenuación multicamino. En este mismo capítulo se vió una

manera de estimar la densidad espectral de potencia, existen otros métodos que de igual forma pueden ser simulados.

Cuando se trabaja con *MATLAB* y *SIMULINK*, se comprueba que una simulación solo en la línea de comandos, difiere de una simulación en el ambiente , ya que esto implica tener las señales que están presentes durante todo el tiempo de simulación de una forma latente mientras que en , la simulación se va realizando poco a poco conforme se vayan dando las instrucciones. Por esta razón se recomienda también el desarrollo de funciones de *MATLAB* que permitan, bajo la línea de comandos, la simulación de sistemas de transmisión.

No se han tratado todos los métodos de modulación digital, por ejemplo las modulaciones tradicionales combinadas con TCM (*Trellis Coded Modulation*) y se recomienda seguir ampliando la librería con nuevos métodos de modulación que sean más utilizados a partir de los ya creados.

Tomando como base los bloques desarrollados se pueden simular sistemas de comunicaciones más complejos, como por ejemplo modems, y desarrollar más bloques para simular transmisión de datos por radiofrecuencia, sistemas telefónicos, enlaces PCM (*Pulse Coded Modulation*), redes LAN (*Local Area Network*), TDMA (*Time Division Multiplex Access*), CDMA (*Code Division Multiplex Access*) y otros.

Con el *Signal Processing Toolbox* y el *Image Processing Toolbox* de *MATLAB*, se pueden tratar señales de audio e imágenes de alta resolución, que pueden utilizarse también para simular sistemas de comunicación muchos más complejos, como por ejemplo una red ISDN (*Integrated Service Digital Network*).

Se ha tratado en esta tesis sobre simulación de códigos de línea y modulación digital y se recomienda con la herramienta desarrollada el estudio de errores de transmisión, como motivo para otro tema de tesis donde se pueden añadir nuevos modelos de canales y de esta forma analizar la calidad de un sistema de transmisión.

Empleando la técnica para genera aplicaciones *GUI (Graphical Unit Interface)* de *MATLAB* se pueden combinar funciones *"\*.m"* con modelos *"\*.mdl"* para hacer una herramienta de análisis de sistemas de transmisión de una manera más amigable para el usuario, un ejemplo de esto es la ventana de inicio del programa con el comando "comdig".

Con ayuda del *Real-time Workshop* de *Mathworks*, que permite manejar los puertos paralelo y serie del computador y de esta manera realizar transmisiones reales entre computadores o con tarjetas especializadas, como por ejemplo *DSPs (Digital Signal Processors)*. La carencia de este *Toolbox* puede suplirse con conocimiento de lenguaje C o Visual C.

## BIBLIOGRAFÍA.

- Strembler F.G, Introducción a los Sistemas de Comunicación, Addison Wesley, México 1993.
- Leon W. Couch II, Digital and Analog Communication Systems, four edition, Macmillan Publishing Company New York 1993.
- James B. Dabney y Thomas L. Harman. "The Student Edition of SIMULINK® User's Guide", Prentice Hall, 1997.
- Nelson Avila, "Diseño y Construcción de un Codec Didáctico", Tesis de Grado previa a la Obtención del Título de Ingeniero en Electrónica y Telecomunicaciones, EPN. 1994.
- Bellamy J., "Digital Telephony", John Wiley & Sons, Nuew York, 1991.
- Vidaller J. y Otros, "Transmision de Datos", E.T.S. Ingenieros Telecomunicaciones Universidad Politécnica de Madrid, Madrd 2ª ed., 1979.
- Math Works INC. "Signal Proccesing Toolbox", 1997
- Math Works INC. "Matlab accelerating inovation and development", 1997.
- Math Works INC. "SIMULINK® Simulación de Sistemas Dinámicos", 1997.
- Math Works INC. "Instalation for PC and Macintosh 1996.
- Savant C. J. y Otros, Diseño Electrónico, Ed. Addison Wesley, 1992.
- Coughlin R. y Driscoll F. Amplificadores Operacionales y Circuitos Integrados Lineales, Ed. Prentice Hall, 1993.
- M.T. Hill y A. Cantoni, "A Frecuency Steered Phase Locked Loop", IEEE Transactions on Communications, vol 45, Junio de 1997 págs, 737-743.
- Castro y Orbe, "Análisis y Simulación de un Sistema de Transmisión Digital Utilizando un Microcomputador", Tesis de Grado previa a la

Obtención del Título de Ingeniero en Electrónica y Telecomunicaciones, EPN. 1995.

- Math Works INC. "Using SIMULINK®", 1996.
- Matolak David y Wilson Stephen, "Detection for a Statically Known, Time - Varing Dispersive Channel", IEEE Transactions on Comunications Vol 44 No.12 December 1996.
- O.M. Zeytinoglu & N.W Ma, "Communication Systems II ELE 045", Communication Toolbox and Manual, Departament of Electrical and Computer Engineering , Ryerson Polytechnic University, Canada,1994.

---

# Anexos

---



# Anexo A

# Anexo A

---

## **Bloque *S-function*(Función de Sistema)**

---

Es un bloque que permite extender las capacidades del SIMULINK, es decir con el propósito de generar nuevos bloques de simulación que respondan a los requerimientos del usuario, dando versatilidad al proceso de simulación.

Una *S-function* nos es más que un programa escrito en lenguaje MATLAB (archivos .m) o en lenguaje C (archivos .MEX<sup>1</sup>), que le permite al usuario describir un sistema dinámico de simulación. Evidentemente debe cumplirse con una determinada sintaxis y orden lógico de dicho lenguaje de programación, ya que la *S-function* es la esencia de como trabaja el SIMULINK.

Lo que se pondrá en consideración en este apéndice, es la manera en que se empleó esta *S-function*, describiendo algunas características que no se encuentran en las guías de usuario, es decir para una descripción más completa e ilustrativa de todas las características y posibilidades de uso de la *S-function* se recomienda el capítulo 8 del manual de Usuario del SIMULINK ("Using SIMULINK"), que se encuentra en formato PDF bajo el subdirectorio *Matlab/help/techdoc/pdfsdocs* .

Básicamente la simulación de un sistema en el SIMULINK, se hace mediante el cálculo de las ecuaciones diferenciales (sistema continuo) o ecuaciones de diferencias (sistema discreto), también existen sistemas que son combinaciones de los dos anteriores. Entonces en una simulación será

---

<sup>1</sup> Para los archivos de extensión MEX, escritos en lenguaje C, mayor información se encuentra en el capítulo 8, del "Using Simulink" de Mathworks Inc. 1996.

necesario indicar si el sistema a simular tiene estados discretos o continuos, las condiciones iniciales y las características de la señal de salida en función de todos estos parámetros.

En general un sistema de simulación tiene: señal de entrada, estados o condiciones iniciales y la señal de salida (figura A.1). Entonces la señal de entrada es un vector " $u$ ", con los valores de amplitud y tiempo a cada momento de la simulación, el vector " $y$ " de las salidas y por otra parte se tiene la variable de estados en el vector " $x$ " compuesta por dos partes principalmente: los valores que corresponden a la derivada de la entrada (por tratarse de sistemas de ecuaciones diferenciales) y, por las actualizaciones de las variables de estado de las ecuaciones en diferencias.

Durante la simulación existen varias llamadas que el sistema de simulación hace a cada bloque, con el fin de que a cada fracción de tiempo se verifiquen las siguientes tareas: lectura de condiciones iniciales (al inicio de la simulación), cálculo del siguiente valor del tiempo (tiempo de muestreo) en que actuara el bloque<sup>1</sup>, cálculo de la salida (si se trata de un sistema discreto), actualización de las variables de estado (vector  $x$ ), cálculo de las salidas en función de la integración de la ecuación diferencial (si se trata de un sistema continuo), cálculo de las derivadas para la ecuación diferencial y por último la finalización del proceso que puede realizar tareas complementarias (como presentación de gráficos o secuencias de simulación gráfica). El proceso descrito se encuentra representado en la figura A.2.

---

<sup>1</sup> Este paso se toma en consideración, si el bloque de simulación requiere de un tiempo de muestreo, para mayor detalle de los tiempos de muestreo refiérase a la página 10-11 (*Sample time colors*) del *Using SIMULINK* (Diciembre 1996).



$$y = f_o(t, x, u) \quad \text{Salida}$$

$$\dot{x}_c = f_d(t, x, u) \quad \text{Derivada}$$

$$x_{t+\Delta t} = f_u(t, x, u) \quad \text{Actualización de estados}$$

Figura A.1 Elementos de un sistema general

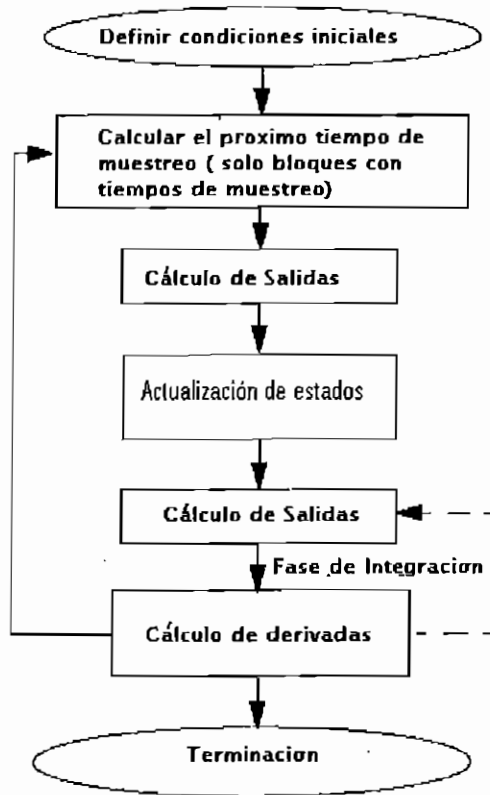


Figura A.2 Diagrama de flujo de un bloque de simulación

Se ha hablado de llamadas, que pueden considerarse como tiempos dentro de la simulación, en los cuales se hacen requerimientos a los

bloques de simulación, estas llamadas se identifican mediante variables de control, denominadas banderas. Dependiendo del valor que tenga la bandera, se debe cumplir con una subrutina (las tareas que se explicó en el párrafo anterior) en el programa de la S-function, así en la tabla A.1 se tiene los diferentes tipos de banderas existentes y sus tipos de subrutinas.

**Tabla A.1 Estados de Simulación**

<b><i>Estado de la Simulación</i></b>	<b><i>Subrutina de la s-function</i></b>	<b><i>Bandera (M-file-SFunctions)</i></b>
Inicialización	mdlInitializeSizes	flag = 0
Cálculo Próximo tiempo de muestreo	mdlGetTimeOfNextVarHit	flag = 4
Cálculo de las salidas	mdlOutputs	flag = 3
Actualización de estados discretos	mdlUpdate	flag = 2
Cálculo de derivadas	mdlDerivatives	flag = 1
Terminación de las tareas	mdlTerminate	flag = 9

Dentro de este grupo de subrutinas, se desarrollarán los procesos necesarios para definir un bloque de simulación. En el caso del desarrollo de la fuente de datos digitales (capítulo 2) y de los codificadores con memoria (capítulo 3), se emplea un bloque con características de un sistema discreto, con un tiempo de muestreo fijo de la señal de entrada, un modelo básico de la S-function se representa en el ejemplo A.1, donde se indica la S-function utilizada para la codificación AMI.

**Ejemplo A.1:** Se pide elaborar una S-function que permita la codificación AMI, a partir de una señal de entrada, se tiene como datos el tiempo de bit, el tipo de marca anterior del uno binario y la amplitud de salida.

Primeramente se elige el bloque de simulación S-function, posteriormente se especifica en la plantilla de datos respectivo(figura A.3),

tanto el nombre del archivo .m<sup>1</sup> como los parámetros adicionales requeridos (datos adicionales), separados con comas<sup>2</sup>. En este caso el nombre del archivo es amicod.m, y las variables: samp, um, ampp, respectivamente: tiempo de bit, tipo de marca anterior y magnitud de salida de la señal codificada.

### Sintaxis del ejemplo A.1

```
function [sys,x0,str,ts] = amicod(t,x,u,flag,samp,um,ampp)

% Amicod M-File S-function codificador AMI
% Este M-file realiza la función de codificar bits de
% entrada con un retraso inicial de medio tiempo de bit
% permite la elección del tiempo de bit, si el uno anterior fue positivo o
% negativo
% Copyright (c) 1997-98 by John Paredes, Inc.
% $Revision: 0.1 $

switch abs(flag)

    %%%%%%%%%%%
    % Initialization %
    %%%%%%%%%%%
    case 0,
        [sys,x0,str,ts] = mdlInitializeSizes(samp,um,ampp);

    %%%%%%%%%%%
    % Update %
    %%%%%%%%%%%
    case 2,
        sys = mdlUpdate(t,x,u,samp,um,ampp);

    %%%%%%%%%%%
    % Output %
    %%%%%%%%%%%
    case 3,
        sys = mdlOutputs(t,x,u,samp,um,ampp);

    %%%%%%%%%%%
    % Terminate %
    %%%%%%%%%%%
    case 9,
        sys = [];
    otherwise
        error('unhandled flag = ',num2str(flag));
end

%end amicod%
%=====
% mdlInitializeSizes
% Entrega los valores de inicio de la simulación, tiempo de muestreo y ofset de la S-function.
%=====
%
function [sys,x0,str,ts]=mdlInitializeSizes(samp,um,ampp)    %Iniciación de Variables
cs=0;buffi=0;dat=0;marca=um;tb2=samp/2;
```



Figura A.3 plantilla de datos de la Sfunction

<sup>1</sup> Solo se tratará el tipo de archivos .m, es decir en lenguaje MATLAB.

<sup>2</sup> Si se desea elaborar una plantilla de datos más personalizada, se recomienda las técnicas de enmascaramiento Mask, descritas en la Guía de Usuario "User Guide" SIMULINK Student Edition de Mathworks INC, 1996.

```

sizes = simsizes; % Vector de las variables de estado

sizes.NumContStates = 0; % Número de estados continuos (parte del número de columnas del vector x)
sizes.NumDiscStates = 4; % Número de estados discretos (parte del número de columnas del vector x)
sizes.NumOutputs = 1; % Dimensión del vector de salida ó número de salidas
sizes.NumInputs = 1; % Número de elementos de entradas
sizes.DirFeedthrough = 0; % Realimentación directa de la salida en la entrada: 1 si es si, 0 si no.
sizes.NumSampleTimes = 1; % Número de tiempos de muestreo (pueden ser dos o más)

sys = simsizes(sizes); % Vector del sistema al inicio, es decir definición del tipo de bloque.

x0 = [cs buffi dat marca]; % Condiciones iniciales x0, misma dimensión que el vector x
str = []; % Variable empleada para aplicaciones gráficas de Usuario (GUI)
ts = [tb2,0]; % Sample period of samp seconds % Tiempo de muestreo en segundos (tb2) y variación (o denominado offset)

% x(1) o cs : indica si es medio tb o tb (tb: tiempo de bit),
% x(2) o buffi : dato a transmitirse
% x(3) o dat : dato al tiempo de medio bit
% x(4) o marca : marca de signo alternado 1 significa que fue +
% end mdlInitializeSizes Término de esta subrutina o función de inicio.
%
%=====
% mdlUpdate
% Actualización de datos discretos , cumplimiento de tiempos de muestreo y otras operaciones
%=====
%
function sys = mdlUpdate(t,x,u,samp,um,ampp)
if u >= 1, % Asignación de valores, de la entrada u, según su voltaje y almacenando en la variable de estado x(3).
    x(3)=1;
else
    x(3)=0;
end

x(1)=not(x(1));

if x(1)==1, % toma del valor de bit a mitad de periodo y mantiene
    x(2)=x(3); % si el valor es cero se mantiene en la salida x(2) el valor asignado de 0 voltios a la salida
    if x(3)==1, % caso del uno lógico es alternado el valor de +1 y -1 de acuerdo a la marca anterior (x(4))
        x(4) = (-1)*x(4);
        x(2) = x(3)*x(4);
    end
end

end

sys=x; % La salida evidentemente es la actualización del vector de estados x, única variable que se mantiene sin inicializarse.

%end mdlUpdate fin de la actualización

%=====
% mdlOutputs
% Nos da el vector de salida para la S-function
%=====
%
function sys = mdlOutputs(t,x,u,samp,um,ampp)

sys=x(2)*ampp; % la salida esta en el elemento x(2) y puede realizarse operaciones matemáticas para manipular la salida.

%end mdlOutputs

```

Según este ejemplo desarrollado la s-function tiene básicamente tres partes, una declaración del nombre principal de la función (amicod(a,b,c,..)) con sus variables(o parámetros de entrada y salida), un esquema estructurado que lleva a realizar una subrutina de acuerdo al

valor de la bandera y por último la definición de las subrutinas. Es necesario indicar que la sintaxis del archivo amicod.m es en lenguaje MATLAB y para mejor entendimiento se han hecho comentarios en el mismo listado del programa, estos comentarios van después de un símbolo % y no son tomados en cuenta al momento de interpretar el programa en la simulación.

### **Bandera 0 (subrutina mdlInitializeSizes).**

Es necesario subrayar la importancia de la bandera de valor 0, aquí es donde se definen (al inicio de la simulación únicamente) las características del bloque a simular mediante tres vectores principales:  $x_0$ ,  $sys$  y  $ts$ .

$x_0$ .- el vector de las variables de estado al tiempo cero segundos de la simulación, es decir los valores iniciales del vector  $x$ .

$sys$ .- compuesto de los siguientes elementos:

```
sizes.NumContStates = 0; % Número de estados continuos (parte del número de columnas del vector x)
sizes.NumDiscStates = 4; % Número de estados discretos (parte del número de columnas del vector x)
sizes.NumOutputs    = 1; % Dimensión del vector de salida ó número de salidas
sizes.NumInputs     = 1; % Número de elementos de entradas
sizes.DirFeedthrough = 0; % Realimentación directa de la salida en la entrada: 1 hay realimentación directa y 0 si no la hay.
sizes.NumSampleTimes = 1; % Número de tiempos de muestreo (pueden ser dos o más)
```

es decir  $sys = [0 \ 4 \ 1 \ 1 \ 0]$ .

$ts$ .- el tiempo de muestreo del bloque cuyos elementos son: el tiempo de muestreo( $tsam$ )en segundos y su tolerancia( $of$ ), entonces se puede describir este vector de la siguiente manera  $ts=[tsam, of]$  (en el ejemplo  $ts=[tb_2,0]$ , donde  $tb_2=samp/2=tsam$  y  $of=0$ ).



## **Bandera 2 (Subrutina mdlUpdate).**

Luego esta el proceso de programación (Actualización de variables de estado), esta subrutina es llamada cada tiempo de muestreo, de manera análoga a un proceso de interrupciones en un microprocesador. Es decir que al final del tiempo de muestreo el SIMULINK entrega datos a esta subrutina y solicita los resultados respectivos.

Las variables  $t$ ,  $x$ ,  $u$ ,  $samp$ ,  $uma$ ,  $ampp$  son los datos entregados por la simulación a la subrutina y los resultados correspondientes se entregan en la misma variable  $x$ . Estos datos requeridos por la función `mdlUpdate` tienen los siguientes significados: la variable  $t$  corresponde al tiempo vigente dentro del proceso de la simulación, la variable  $x$  es la variable de estado, la variable  $u$  es la magnitud de la señal de entrada al bloque `S-function` al tiempo  $t$  de simulación y las variables siguientes son los parámetros generales ya descritos. Con este concepto entonces es posible manipular la información, la misma que utilizará la siguiente bandera para entregar la señal de salida del bloque.

Esta subrutina se diseñó para que la señal de entrada se interprete de acuerdo a una estimación de su magnitud; es decir si el valor  $u$  de entrada es mayor o igual que 1 se interpreta como un lógico y si es menor que 1 como cero lógico. El dato estimado entonces se almacena en la columna 3 del vector de la variable de estado, es decir  $x(3)$  en lenguaje MATLAB.

El tiempo de muestreo de datos es de medio tiempo de bit ( $tb/2$ ), es decir se ha planificado cada medio tiempo de bit la lectura del vector  $u$ , pero el dato de  $u$  que es de interés (denominado dato válido) para la

codificación se da cuando el tiempo es:  $tb2, 3tb2, 5tb2\dots$ ; lo que significa que al primer tiempo de muestreo se debe procesar la información, el siguiente no, el siguiente sí y así sucesivamente; por esta razón se utiliza la variable  $x(1)$  ( de valor inicial 0) que va a complementar su valor cada que la subrutina es llamada. Dentro de este tiempo de lectura del dato valido ( $x(1)=1$ ) se ha dispuesto que en  $x(2)$  se almacene el valor obtenido de  $x(3)$ . Si el valor de  $x(3)$  es 1 se multiplica por -1 el valor de  $x(4)$ (valor inicial igual al valor de la variable  $uma$ ) que es la variable que representa la marca anterior(la marca anterior significa la polaridad del uno precedente, 1 si es positiva y -1 si es negativa); luego dentro de esta condición del dato valido igual a uno se multiplica  $x(4)$  con  $x(2)$  y se almacena este resultado en  $x(2)$ . En el caso de que  $x(3)$  sea igual a cero solo existe el almacenamiento del dato en  $x(2)$ .

Finalmente se envía como resultado el vector  $x$  en la variable  $SYS$  que es la variable de respuesta de la subrutina, esperando la siguiente bandera.

### **Bandera 3 (Subrutina mdlOutputs).**

Los mismos parámetros son requeridos en esta función, es aquí donde se puede manipular el valor de salida requerido. Se utiliza el valor almacenado en  $x(2)$  como valor de salida pero antes de entregar este valor como respuesta se multiplica por  $ampp$ , para obtener la amplitud deseada; entonces este resultado es enviado en la variable  $SYS$  constituyendose en la señal de salida del bloque.

Durante la simulación se repetirán solamente las subrutinas cuyas banderas son la 2 y 3 para esta *S-function*.

Es necesario indicar que las subrutinas tienen la sintaxis del lenguaje MATLAB, que son funciones definidas por el usuario. En la programación se requieren de variables que mantengan su valor durante la simulación, esto se consigue mediante los elementos del vector  $x$ , esta parte no se encuentra especificada en ningún manual y en un inicio para guardar estas variables fue necesario un archivo de lectura escritura (\*.mat) que hacía más lento el proceso.

Tanto las fuentes de datos, como los codificadores tienen el mismo tipo de archivos .m, para realizar cambios en los bloques que tengan como subsistema la S-function, debe revisar la programación en estos archivos "\*.m".

#### Nota.

En cuanto a los bloques con tiempo de muestreo, como el de este ejemplo, es muy importante indicar, que cualquier proceso bajo la bandera 2 (actualización) se lleva a cabo transcurridos  $t_s$  segundos (más el tiempo de *offset* o de histéresis). Como ejemplo, si se utilizara el bloque del código AMI, solamente para lectura y retransmisión de los datos de la variable  $u$ , se tendría un retraso entre la señal de datos y la señal de salida del bloque igual al tiempo de muestreo de este bloque.

# Anexo B

## ECEN4618: Experiment #3 Design of a Voltage-Controlled Waveform Generator

© 1997 Dragan Maksimović  
Department of Electrical and Computer Engineering  
University of Colorado, Boulder

In this lab assignment your task is to design a waveform generator shown in Fig. 1.

The generator is to produce two periodic output waveforms: a triangular-wave output  $v_{ot}(t)$  and a square-wave output  $v_{os}(t)$ . There is one input voltage  $v_m$  that controls the frequency  $f$  of the output waveforms. The duty ratio  $D$  of the output is to be controlled by an adjustable potentiometer  $R_1$ . The two controls should be independent: when  $R_1$  is turned, only the duty ratio should change while the frequency should remain constant; when  $v_m$  changes, only the frequency should vary, while the duty ratio should remain constant. Both outputs should have constant amplitudes independent of the duty ratio or frequency settings.

A laboratory pulse generator can perform functions similar to what is described above. Frequency can be adjusted by an external knob, or modulated by an external input voltage. The duty ratio can also be adjusted independently. The lab generator usually allows independent control of the waveform amplitudes, and has a much wider frequency range than what we can achieve in the lab (given generic components, time and breadboard constraints).

The waveform generator belongs to a class of circuits called voltage-controlled oscillators (VCO), where frequency of the generated waveforms is proportional to a voltage input. The VCO is a key component in phase-locked loops (PLL) which in turn are the key components in communication systems. PLL and its applications will be the topic of the lab experiments #4 and #5.

### 1 Design Specifications

Here is the set of design specifications and constraints for the waveform generator shown in Fig. 1:

1. Only one supply voltage (+15V) is available.
2. Frequency  $f$  of the output waveforms is related to the input voltage  $v_m$  as:

$$f = K_o v_m$$

$$K_o = 5\text{kHz/V}$$

$$0.5\text{V} \leq v_m \leq 10\text{V}$$

Therefore, the generator should produce the outputs at the frequency  $f$  in the range  $2.5\text{kHz} \leq f \leq 50\text{kHz}$ . Constant  $K_o$  is called the gain of the VCO.

3. By turning  $R_1$ , the duty ratio can be adjusted in the range  $0.1 \leq D \leq 0.9$  at any frequency  $f$  in the specified range.
4. Turning  $R_1$  should not affect the frequency  $f$ . Changing  $v_m$  should not affect the duty ratio  $D$ .

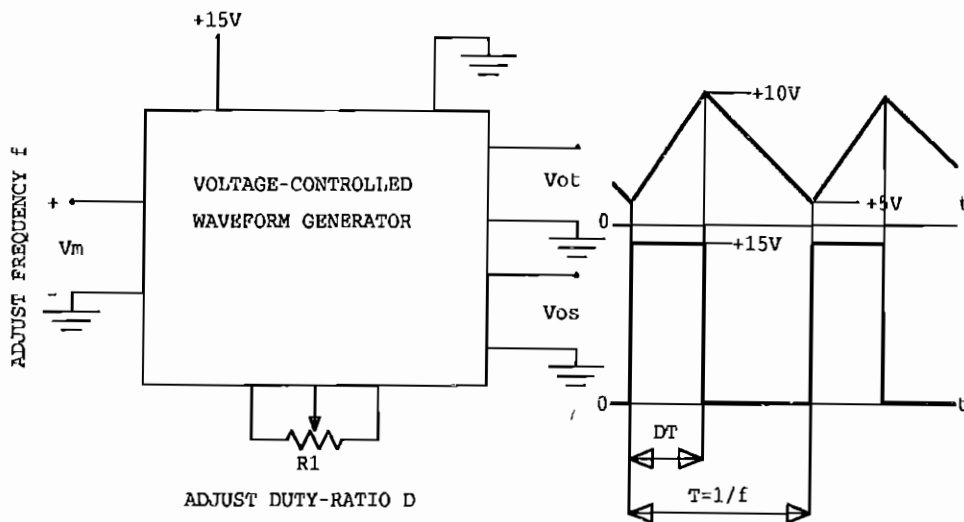


Figure 1: A voltage-controlled waveform generator.

5. The square-wave  $v_{os}$  amplitude is 0V to +15V.
6. The rise and the fall times of  $v_{os}$  are less than  $1\mu s$ .
7. The triangular-wave  $v_{ot}$  is between +5V and +10V.

## 2 An Approach to Constructing the Voltage-Controlled Waveform Generator

There are many possible approaches to constructing the generator. This handout outlines only one of the approaches. You are welcome and encouraged to explore other options using general-purpose integrated and discrete components.

The solution outlined in this section can be implemented using a voltage comparator (such as LM311), operational amplifiers (such as LF353) and discrete MOS transistors (such as NMOS ZVN3310A and PMOS ZVP3310A), resistors and capacitors.

The solution is based on the use of a voltage comparator with hysteresis, and an integrator built around an op-amp. Similar circuits were used in the pulse-width modulator design.

Simplified circuit diagram of a basic triangle-wave and square-wave generator is shown in Fig. 2, together with typical waveforms  $v_s(t)$  (square wave) and  $v_t(t)$  (triangle wave). At  $t = 0$ ,  $v_t(0)$  is equal to the upper threshold voltage  $V_H$  of the comparator with hysteresis. The comparator output  $v_s$  jumps to  $+V_X$ . As a result, constant current  $+V_X/R$  through  $R$  discharges the capacitor  $C$ , so that

$$v_t(t) = V_H - \frac{V_X}{RC}t, \quad t \geq 0. \quad (1)$$

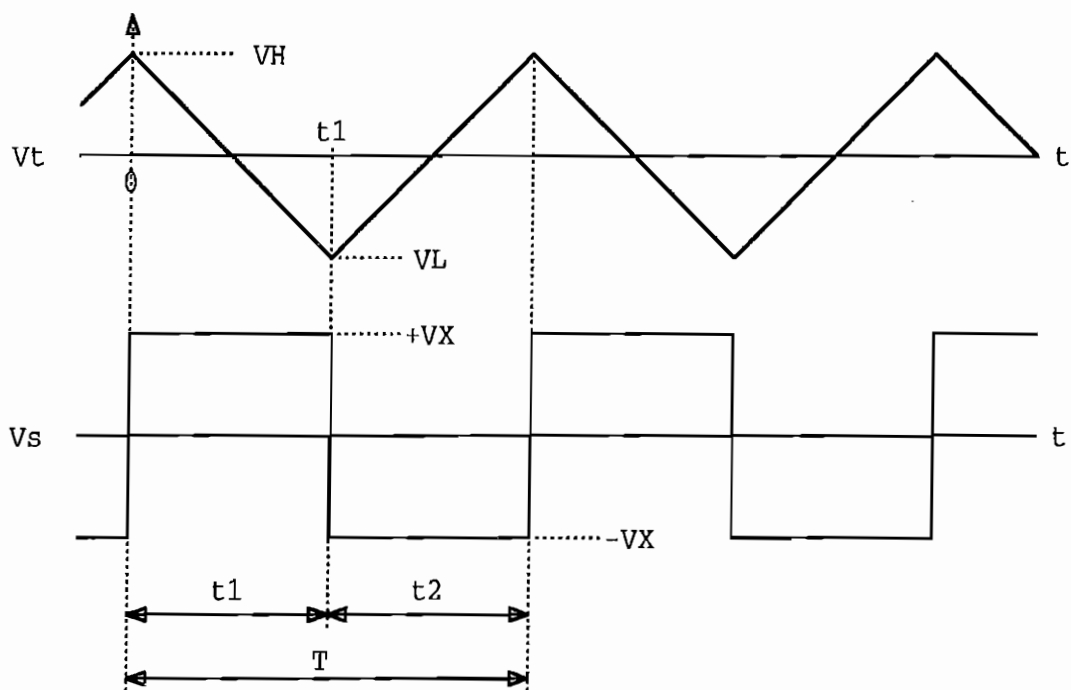
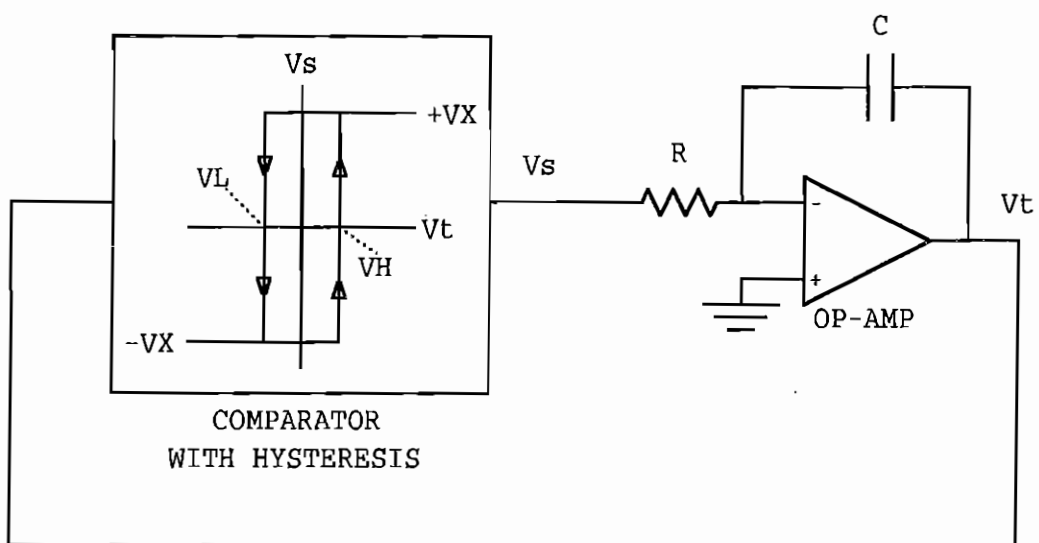


Figure 2: Simplified circuit diagram of a basic triangle-wave/square-wave waveform generator.

The comparator output stays at  $v_s = +V_X$  as long as the comparator input voltage  $v_t$  is greater than the lower threshold  $V_L$ . In the circuit of Fig. 2 it is assumed that the threshold voltages are symmetrical around zero,  $V_L = -V_H$ . At  $t = t_1$ ,  $v_t(t_1) = V_L$ , and the output  $v_s$  of the comparator with hysteresis jumps to  $-V_X$ . The time interval  $t_1$  can be found from

$$v_t(t_1) = V_H - \frac{V_X}{RC}t_1 = V_L = -V_H, \quad (2)$$

$$t_1 = RC \frac{2V_H}{V_X}. \quad (3)$$

After the comparator output  $v_s$  jumps to  $-V_X$ , the current through  $R$  becomes  $-V_X/R$ , and this current charges the capacitor. As a result, the op-amp output  $v_t(t)$  increases as a linear function of time:

$$v_t(t) = V_L + \frac{V_X}{RC}(t - t_1), \quad t \geq t_1. \quad (4)$$

At  $t = T$ , the triangle-wave output  $v_t$  reaches the upper threshold  $V_H$  again, and the comparator output  $v_s$  returns to  $+V_X$ , starting a new period. Period  $T = 1/f$  can be found from the condition  $v_t(T) = V_H$ , or

$$T - t_1 = t_2 = RC \frac{2V_H}{V_X} = t_1, \quad (5)$$

$$T = RC \frac{4V_H}{V_X}. \quad (6)$$

The frequency of the output waveforms is given by

$$f = \frac{1}{T} = \frac{1}{RC} \frac{V_X}{4V_H}. \quad (7)$$

If we compare the operation of the basic triangle-wave/square-wave waveform generator with the actual design requirements, we have the following conclusions:

- The frequency  $f$  is *linearly* proportional to the voltage  $V_X$  that determines the charging/discharging current through the capacitor. Therefore,  $V_X$  could be used to achieve the desired voltage control  $f = K_o v_m$ . Note, however, that in the circuit of Fig. 2 both  $+V_X$  (for discharging) and  $-V_X$  (for charging) are needed.
- The amplitude of the triangle wave  $v_t$  is determined directly by the threshold voltages  $V_H$  and  $V_L$  of the comparator with hysteresis.
- The square-wave amplitude is  $\pm V_X$ , and frequency is proportional to  $V_X$ , while the specifications require a constant-amplitude square wave  $v_{os}$ .
- The basic circuit has no provisions for changing the output duty ratio. Since the charging and the discharging currents are the same,  $t_1 = t_2$ , and the output duty ratio is always equal to 50%. To control the duty ratio, the circuit should be modified to include a potentiometer  $R_1$  that would change the ratio of the charging/discharging currents.
- Positive and negative supply voltages are needed to implement the circuit of Fig. 2, while the specifications allow for only one +15V supply voltage.



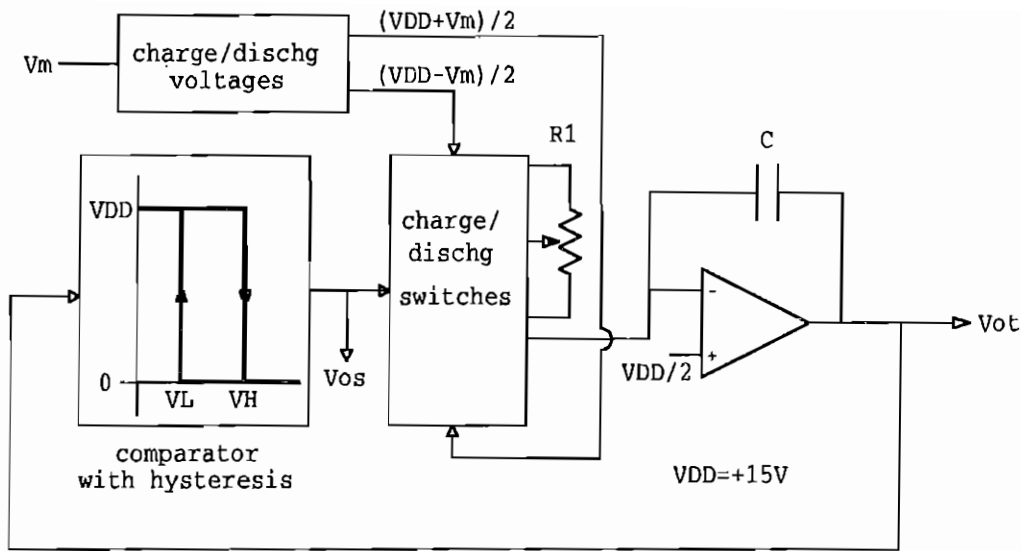


Figure 3: A block diagram of the voltage-controlled triangle-wave/square-wave waveform generator.

To proceed with the design, the basic circuit of Fig. 2 should be modified to generate the required charging/discharging voltages from the input  $v_m$ , allow for the duty ratio control using  $R_1$ , and operate from a single +15V supply. A block diagram of one possible solution is shown in Fig. 3.

With  $V_{DD} = +15V$ , charging and discharging voltages are generated as  $(V_{DD} + v_m)/2$  and  $(V_{DD} - v_m)/2$ , symmetrical around  $V_{DD}/2$ . These are the voltages that replace  $\pm V_X$  in Fig. 2. Note that the + input of the op-amp is also set to  $V_{DD}/2$ .

The characteristic of the comparator with hysteresis is constructed so that the output triangle wave is between  $V_L = +5V$  and  $V_H = +10V$ . LM311 can be used to implement the comparator with hysteresis.

A block “charge/discharge switches” is inserted between the comparator output and the op-amp integrator, to allow for switching of the charging and discharging voltages and to allow for duty ratio adjustments. Discrete MOS transistors can be used to implement this block. The potentiometer  $R_1$  should be connected so that turning  $R_1$  changes the *balance* of the charging and the discharging current through the capacitor.

### 3 Experiment

Your main task in the lab is to design and verify operation of the voltage-controlled waveform generator according to the specifications in Section 1. The next task is to make an improvement or modification of the circuit, as discussed in Section 3.1.

Try to follow the modular design/verification approach as in Lab 2: start with constructing and verifying operation of simpler blocks and proceed only when the simpler blocks work as expected. Do not forget about proper power supply decoupling !

In your report you should describe how you arrived at your final circuit: the experiments you performed to verify operation of the blocks and any design modifications you made. Include a complete and labeled circuit diagram of your final solution.

Once you are satisfied with operation of the circuit, do the following experiments:

1. Set the duty ratio at 50% and measure the output frequency  $f$  as a function of the input voltage  $v_m$ . Include a plot of  $f(v_m)$  in your report. Find the actual  $K_o$  of your circuit. Test and explain what happens if  $v_m$  goes below or above the specified limits.
2. Measure and report the actual amplitudes of  $v_{ot}$  and  $v_{os}$  at the extremes of the frequency range, for  $v_m = 0.5V$ , and for  $v_m = 10V$ , with the duty ratio equal to 50%. Measure and report the actual rise and fall times of  $v_{os}$ .
3. Set the frequency to the minimum value, for  $v_m = 0.5V$ . Measure and report the actual range of duty-ratios that you can obtain by turning  $R_1$ . Repeat the measurement at the highest frequency, for  $v_m = 10V$ . Include *labeled* sketches or printouts of the triangular and square-wave output waveforms at the extremes of the frequency and duty-ratio settings (total of 4 sets of waveforms).

In the report, include the corresponding PSpice simulation results and compare the simulation waveforms to the actual measured waveforms. If necessary, make appropriate simplifications of the simulated circuit so that the results can be obtained with the available evaluation version.

4. In (3), you may notice that adjusting the duty ratio introduces some (undesirable) frequency variations and variations in the amplitude of  $v_{ot}$ , especially at the highest frequency setting. If this is so, explain why it happens, measure and report how much the frequency and the amplitude change while the duty ratio is adjusted from the minimum to the maximum value. Suggest how this error could be reduced.

### 3.1 Modifications

After you complete and verify your design, do at least one modification of the waveform generator as described below, or suggest and pursue a modification on your own. Extra-credit will be given for particularly original/good solutions.

1. Modify the generator to allow independent control of the triangle-wave peak-to-peak amplitude, while keeping all other design specifications the same. The triangle wave should be centered around  $V_{DD}/2$  but the amplitude should be adjustable by another potentiometer. Adjusting the amplitude should not affect the duty ratio nor the frequency of the output waveforms. Design, on paper, the additional circuitry and experimentally verify its operation. Include a complete, labeled circuit diagram and results of experimental verification in your report.
2. Modify the original design to increase the VCO gain to  $K_o = 15\text{kHz/V}$ , while keeping all other specifications the same. Include a complete labeled circuit diagram, and test results that illustrate performance of the modified generator.

3. Modify the generator so that the triangle-wave output swings between 0V and +5V, while keeping all other design specifications the same. Include a complete labeled circuit diagram, and test results that illustrate performance of the modified generator.
4. Modify the generator so that it also has a sinusoidal output centered around  $V_{DD}/2$ , with 2V peak-to-peak amplitude. Include a complete labeled circuit diagram, and test results that illustrate performance of the modified generator.

## 4 Prelab Assignment

The prelab assignment is due in the lab on the day when you start working on the experiment.

Read the complete Lab 3 handout.

Design (on paper) the voltage-controlled waveform generator shown in Fig. 1, according to the design specifications given in Section 1.

Turn in the complete circuit diagram with all component values. Label the pin numbers on all active devices. Show how you found the component values.

Do PSpice simulations to verify your design. If your design exceeds the number of components allowed by the PSpice evaluation version, you will need to make appropriate simplifications of the simulated circuit. Turn in a sketch of the simulated circuit and the plots of the output waveforms at the maximum output frequency and the two extreme values of the duty ratio.

**Make a copy of your prelab work so that you can use it during the Lab sessions.**

# Anexo C

## Experiment #4 CMOS 4046 Phase-Locked Loop

© 1997 Dragan Maksimović  
Department of Electrical and Computer Engineering  
University of Colorado, Boulder

The purpose of this lab assignment is to introduce operating principles and characteristics of a phase-locked loop (PLL) built around CMOS 4046 integrated circuit.

In the lab assignment #5, this PLL will be used to design a data modem based on a digital frequency modulation technique called frequency-shift keying (FSK). Before approaching the design problem, it is necessary to understand principles of operation and characteristics of the PLL.

This handout includes: a brief summary of the theory of phase-locked loops in Section 1, description of the PLL components on the 4046 chip in Section 2, experiments you need to perform in the lab in Section 3, and the prelab assignment in Section 4.

### 1 Phase-Locked Loop Concepts

Phase-locked loop is a feedback loop where a voltage-controlled oscillator can be automatically synchronized (“locked”) to a periodic input signal. The locking property of the PLL has numerous applications in communication systems (such as frequency, amplitude, or phase modulation/demodulation, analog or digital), tone decoding, clock and data recovery, self-tunable filters, frequency synthesis, motor speed control, etc.

The basic PLL has three components connected in a feedback loop, as shown in the block diagram of Fig. 1: a voltage-controlled oscillator (VCO), a phase detector (PD), and a low-pass loop filter (LPF).

The VCO is an oscillator whose frequency  $f_{osc}$  is proportional to input voltage  $v_o$ . The voltage at the input of the VCO determines the frequency  $f_{osc}$  of the periodic signal  $v_{osc}$  at the output of the VCO. In the lab #3 you have designed one VCO.

The output of the VCO,  $v_{osc}$ , and a periodic incoming signal  $v_i$  are inputs to the *phase detector*. When the loop is *locked* on the incoming signal  $v_i$ , the frequency  $f_{osc}$  of the VCO output  $v_{osc}$  is *exactly equal* to the frequency  $f_i$  of the periodic signal  $v_i$ ,

$$f_{osc} = f_i. \quad (1)$$

It is also said that the PLL is in the *locked condition*. The phase detector produces a signal proportional to the phase difference between the incoming signal and the VCO output signal.

The output of the phase detector is filtered by a low-pass *loop filter*. The loop is closed by connecting the filter output to the input of the VCO. Therefore, the filter output voltage  $v_o$  controls the frequency of the VCO.

A basic property of the PLL is that it attempts to maintain the frequency lock ( $f_{osc} = f_i$ ) between  $v_{osc}$  and  $v_i$  even if the frequency  $f_i$  of the incoming signal varies in time. Suppose that the PLL is in the locked condition, and that the frequency  $f_i$  of the incoming signal increases slightly. The phase difference between the VCO signal and the incoming signal will begin to increase in time.

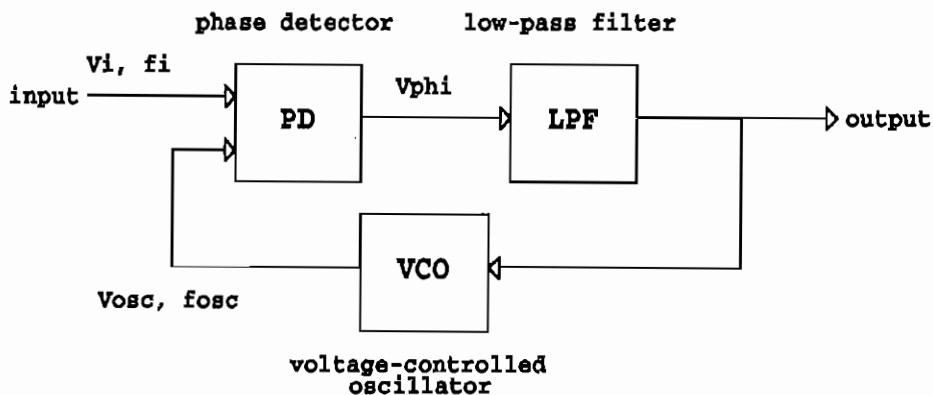


Figure 1: Block diagram of a basic phase-locked loop (PLL).

As a result, the filter output voltage  $v_o$  increases, and the VCO output frequency  $f_{osc}$  increases until it matches  $f_i$ , thus keeping the PLL in the locked condition.

The range of frequencies from  $f_i = f_{min}$  to  $f_i = f_{max}$  where the locked PLL *remains* in the locked condition is called the *lock range* of the PLL. If the PLL is initially locked, and  $f_i$  becomes smaller than  $f_{min}$ , or if  $f_i$  exceeds  $f_{max}$ , the PLL fails to keep  $f_{osc}$  equal to  $f_i$ , and the PLL becomes unlocked,  $f_{osc} \neq f_i$ . When the PLL is unlocked, the VCO oscillates at the frequency  $f_o$  called the *center frequency*, or the *free-running* frequency of the VCO. The lock can be established again if the incoming signal frequency  $f_i$  gets close enough to  $f_o$ . The range of frequencies  $f_i = f_o - f_c$  to  $f_i = f_o + f_c$  such that the initially unlocked PLL becomes locked is called the *capture range* of the PLL.

The lock range is wider than the capture range. So, if the VCO output frequency  $f_{osc}$  is plotted against the incoming frequency  $f_i$ , we obtain the PLL steady-state characteristic shown in Fig. 2. The characteristic simply shows that  $f_{osc} = f_i$  in the locked condition, and that  $f_{osc} = f_o = const.$  when the PLL is unlocked. A hysteresis can be observed in the  $f_{osc}(f_i)$  characteristic because the capture range is smaller than the lock range.

We can now examine how the PLL is actually implemented on the CMOS 4046 integrated circuit.

## 2 The 4046 Phase-Locked Loop

A diagram of the 4046 PLL is shown in Fig. 3.

A single positive supply voltage is needed for the chip. The positive supply voltage  $V_{DD}$  is connected to pin 16 and the ground is connected to pin 8. In the lab we will use  $+V_{DD} = +15V$ .

The incoming signal  $v_i$  goes to the input of an internal amplifier at the pin 14 of the chip. The internal amplifier has the input biased at about  $+V_{DD}/2$ . Therefore, the incoming signal can be capacitively coupled to the input, as shown in Fig. 3. The incoming ac signal  $v_i$  of about one volt peak-to-peak is sufficient for proper operation. The capacitor  $C_i$  together with the input resistance  $R_i \approx 100k\Omega$  at the pin 14 form a high-pass filter.  $C_i$  should be selected so that  $v_i$  is in the pass-

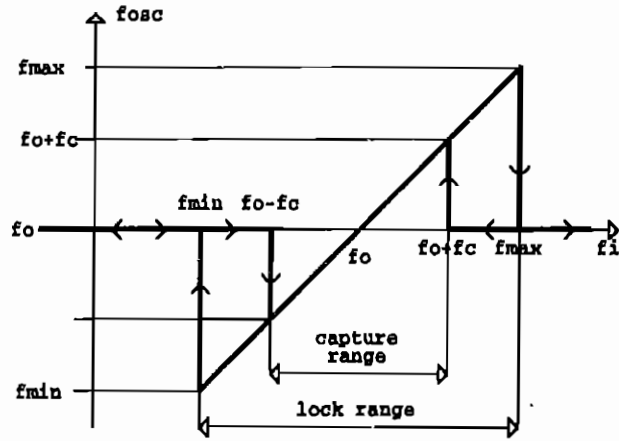


Figure 2: Steady-state  $f_{osc}(f_i)$  characteristic of the basic PLL.

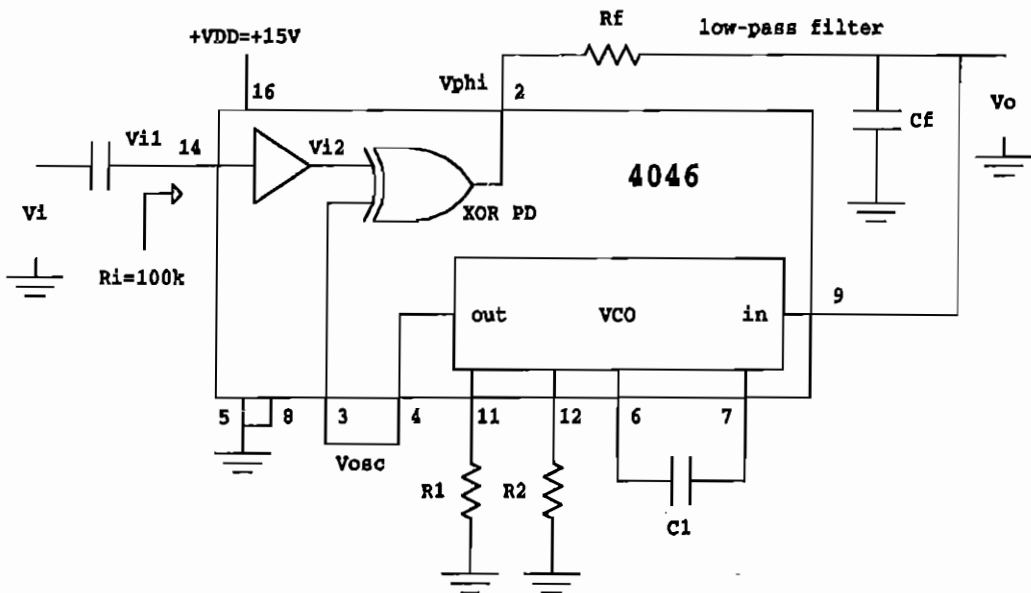


Figure 3: CMOS 4046 PLL: basic connection diagram.

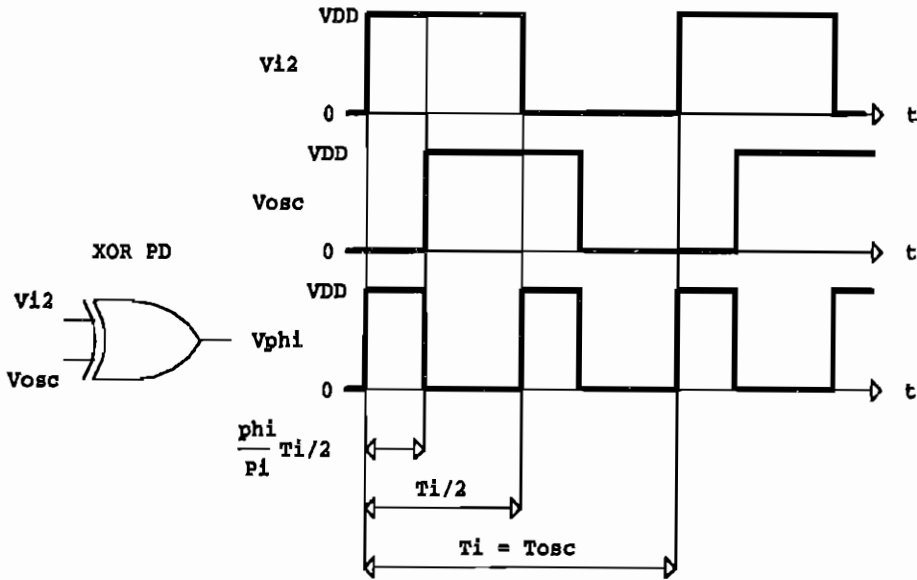


Figure 4: Operation of the phase detector with XOR gate.

band of the filter, i.e., so that  $f_i > 1/(2\pi R_i C_i)$  for the lowest expected frequency  $f_i$  of the incoming signal. The output  $v_1$  of the internal amplifier is internally connected to one of the two inputs of the phase detector on the chip.

## 2.1 Phase Detector

The phase detector on the 4046 is simply an XOR logic gate, with logic low output ( $v_\phi = 0V$ ) when the two inputs are both high or both low, and the logic high output ( $v_\phi = V_{DD}$ ) otherwise. Fig. 4 illustrates the operation of the XOR phase detector when the PLL is in the locked condition.  $v_1$  (the amplified  $v_i$ ) and  $v_{osc}$  (the VCO output) are two phase-shifted periodic square-wave signals at the same frequency  $f_{osc} = f_i = 1/T_i$ , and with 50% duty ratios. The output of the phase detector is a periodic square-wave signal  $v_\phi(t)$  at the frequency  $2f_i$ , and with the duty ratio  $D_\phi$  that depends on the phase difference  $\phi$  between  $v_i$  and  $v_{osc}$ ,

$$D_\phi = \frac{\phi}{\pi}. \quad (2)$$

The periodic signal  $v_\phi(t)$  at the output of the XOR phase detector can be written as the Fourier series:

$$v_\phi(t) = v_o + \sum_{k=1}^{k \rightarrow \infty} v_k \sin((4k\pi f_i)t - \theta_k), \quad (3)$$

where  $v_o$  is the dc component of  $v_\phi(t)$ , and  $v_k$  is the amplitude of the  $k^{th}$  harmonic at the frequency  $2kf_i$ . The dc component of the phase detector output can be found easily as the average of  $v_\phi(t)$



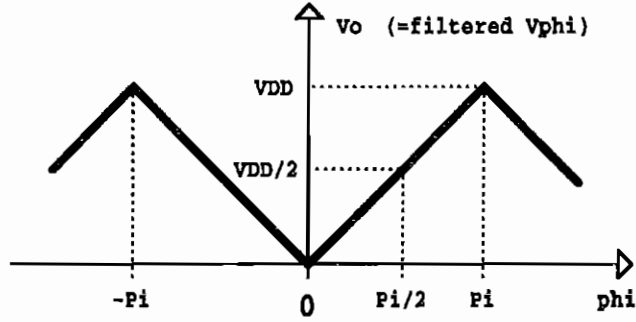


Figure 5: Characteristic of the phase detector.

over a period  $T_i/2$ ,

$$v_o = \frac{1}{T_i/2} \int_0^{T_i/2} v_\phi(t) dt = \frac{2}{T_i} \int_0^{D_\phi T_i/2} V_{DD} dt = \frac{V_{DD}}{\pi} \phi. \quad (4)$$

## 2.2 Loop Filter

The output  $v_\phi$  of the phase detector is filtered by an external low-pass filter. In Fig. 3, the loop filter is a simple passive  $RC$  filter. The purpose of the low-pass filter is to pass the dc and low-frequency portions of  $v_\phi(t)$  and to attenuate high-frequency ac components at frequencies  $2kf_i$ . The simple  $RC$  filter has the transfer function:

$$F(s) = \frac{1}{1 + sR_f C_f} = \frac{1}{1 + s/\omega_p}, \quad (5)$$

where

$$f_p = \frac{\omega_p}{2\pi} = \frac{1}{2\pi R_f C_f} \quad (6)$$

is the cut-off frequency of the filter. If  $f_p \ll 2f_i$ , i.e., if the cut-off frequency of the filter is much smaller than twice the frequency  $f_i$  of the incoming signal, the output of the filter is approximately equal to the dc component  $v_o$  of the phase detector output. In practice, the high-frequency components are not completely eliminated and can be observed as high-frequency ac ripple around the dc or slowly-varying  $v_o$ .

Eq. 4 shows that  $v_o$  is proportional to the phase difference  $\phi$  between the incoming signal  $v_i(t)$  and the signal  $v_{osc}(t)$  from the VCO. The constant of proportionality,

$$K_D = \frac{V_{DD}}{\pi} \quad (7)$$

is called the *gain or the sensitivity* of the phase detector. This expression is valid for  $0 \leq \phi \leq \pi$ . In general, the filter output  $v_o$  as a function of the phase difference  $\phi$  is shown in Fig. 5. Note that  $v_o = 0$  if  $v_i$  and  $v_{osc}$  are in phase ( $\phi = 0$ ), and that it reaches the maximum value  $v_o = V_{DD}$  when

the two signals are exactly out of phase ( $\phi = \pi$ ). From Fig. 4 it is easy to see that for  $\phi < 0$ ,  $v_o$  increases, and for  $\phi > \pi$ ,  $v_o$  decreases. Of course, the characteristic is periodic in  $\phi$  with period  $2\pi$ . The range  $0 \leq \phi \leq \pi$  is the range where the PLL can operate in the locked condition.

### 2.3 Voltage Controlled Oscillator

As shown in Fig. 3, the filter output  $v_o$  controls the VCO, i.e., determines the frequency  $f_{osc}$  of the VCO output  $v_{osc}$ . The VCO inside the 4046 chip is based on the same operating principles as the VCO you built in the lab #3. The voltage  $v_o$  controls the charging and discharging currents through an external capacitor  $C_1$ , and therefore determines the time needed to charge (discharge) the capacitor to a pre-determined threshold level. As a result, the frequency  $f_{osc}$  of the VCO output is determined by  $v_o$ . The VCO output  $v_{osc}$  is a square wave with 50% duty ratio and frequency  $f_{osc}$ .

As shown in Fig. 3, the VCO characteristics are user-adjustable by three external components:  $R_1$ ,  $R_2$  and  $C_1$ . When  $v_o$  is at zero, the VCO operates at the minimum frequency  $f_{min}$  given approximately by:

$$f_{min} = \frac{1}{R_2(C_1 + 32pF)}. \quad (8)$$

When  $v_o = V_{DD}$ , the VCO operates at the maximum frequency  $f_{max}$  given approximately by

$$f_{max} = f_{min} + \frac{1}{R_1(C_1 + 32pF)}. \quad (9)$$

Unfortunately, the actual operating frequencies can differ significantly from the values predicted by the above equations. As a result, you may need to tune the component values by experiment.

For proper operation of the VCO, the components  $R_1$ ,  $R_2$  and  $C_1$  should satisfy:

$$1M\Omega \geq R_1 \geq 10k\Omega, \quad 1M\Omega \geq R_2 \geq 10k\Omega, \quad 100nF \geq C_1 \geq 100pF. \quad (10)$$

For  $f_{osc}$  between the minimum  $f_{min}$  and the maximum  $f_{max}$ , the VCO output frequency  $f_{osc}$  is ideally a linear function of the control voltage  $v_o$ . The slope  $K_o = \Delta f_{osc} / \Delta v_o$  of the  $f_{osc}(v_o)$  characteristic is called the *gain or the frequency sensitivity* of the VCO, in Hz/V.

Operation of the 4046 PLL can now be summarized as follows. Suppose that initially there is no incoming signal at all,  $v_i = 0$ . The VCO output  $v_{osc}$  is a square-wave signal with 50% duty ratio. Since  $v_i = 0$  by assumption, the output of the XOR phase detector is exactly the same as the VCO output,  $v_\phi = v_{osc}$ . The low-pass filter attenuates ac components of  $v_\phi$  so that the filter output is approximately equal to the dc component of  $v_\phi$ . In this case, the dc component  $v_o$  is equal to  $V_{DD}/2$  because  $v_\phi$  is  $V_{DD}$  for one half of the period and 0 for the other half of the period. The filter output voltage  $v_o$  is the input of the VCO. Therefore, we conclude that the VCO operates at the frequency that corresponds to the voltage  $v_o = +V_{DD}/2$  at the VCO input. This frequency is called the *center*, or the *free-running* frequency  $f_{osc} = f_o$  of the PLL.

Suppose now that the incoming signal  $v_i$  at frequency  $f_i$  is brought to the input of the PLL, i.e., to one of the two inputs of the XOR phase detector. If the frequency  $f_i$  is sufficiently close to the free-running frequency  $f_o$ , the PLL will “capture” the incoming signal. During the capture process, the phase difference between  $v_i$  and  $v_{osc}$  changes in time, and therefore  $v_o$  changes in time until  $v_o$  reaches the value required to lock the VCO frequency  $f_{osc}$  to the frequency  $f_i$  of the incoming signal. In the locked condition  $f_{osc} = f_i$ , and  $v_i$  and  $v_{osc}$  are phase shifted by  $\phi$ . The

phase difference  $\phi$  between  $v_i$  and  $v_{osc}$  depends on the value of the incoming signal frequency  $f_i$ . If  $f_i = f_o$ , the phase difference must be exactly  $\phi = \pi/2$  so that, from Eq. 4,  $v_o = V_{DD}/2$  and the VCO indeed operates at  $f_o$ . If, for example,  $f_i = f_{max}$ , the phase difference  $\phi$  must be equal to  $\pi$  in order to obtain  $v_o = V_{DD}$  that results in the VCO output frequency equal to  $f_{max}$ . It is important to note here that in order to keep  $f_{osc} = f_i$  in the locked condition, the filter output voltage  $v_o$  must vary together with the frequency  $f_i$  of the incoming signal. Therefore, a change in  $f_i$  causes a proportional change in  $v_o$ , as long as the PLL stays in the locked condition with  $f_{osc} = f_i$ . As a result, if the incoming signal is frequency modulated, with  $f_i(t)$  varying in time, the PLL behaves as an FM demodulator with  $v_o$  as the demodulated output.

## 2.4 Lock and Capture Ranges

Once the PLL is in the locked condition, it remains locked as long as the VCO output frequency  $f_{osc}$  can be adjusted to match the incoming signal frequency  $f_i$ , i.e., as long as  $f_{min} \leq f_i \leq f_{max}$ . When the lock is lost, the VCO operates at the free-running frequency  $f_o$ , which is between  $f_{min}$  and  $f_{max}$ . To establish the lock again, i.e. to *capture* the incoming signal again, the incoming signal frequency  $f_i$  must be close enough to  $f_o$ . Here “close enough” means that  $f_i$  must be in the range from  $f_o - f_c$  to  $f_o + f_c$ , where  $2f_c$  is called the capture range. The capture range  $2f_c$  is an important PLL parameter because it determines whether the locked condition can be established or not. Refer again to Fig. 2 that shows a typical steady-state PLL characteristic. Note that the capture range  $2f_c$  is smaller than the lock range  $f_{max} - f_{min}$ .

The capture range  $2f_c$  depends on the characteristics of the loop filter. For the simple RC filter, a very crude, approximate implicit expression for the capture range can be found as:

$$f_c \approx \frac{V_{DD}}{2} \frac{K_o}{\sqrt{1 + (f_c/f_p)^2}}, \quad (11)$$

where  $f_p$  is the cut-off frequency of the filter,  $V_{DD}$  is the supply voltage, and  $K_o$  is the VCO gain. Given  $K_o$ , and  $f_p$ , this relation can be solved for  $f_c$  numerically, which yields an approximate theoretical prediction for the capture range  $2f_c$ .

If the capture range is much larger than the cut-off frequency of the filter,  $f_c/f_p \gg 1$ , the expression for the capture range is simplified,

$$2f_c \approx \sqrt{2K_o f_p V_{DD}}. \quad (12)$$

Note that the capture range  $2f_c$  is smaller if the cut-off frequency  $f_p$  of the filter is lower. It is usually desirable to have a wider capture range, which can be accomplished by increasing the cut-off frequency  $f_p$  of the filter. On the other hand, a lower cut-off frequency  $f_p$  is desirable in order to better attenuate high-frequency components of  $v_\phi$  at the phase detector output, and improve noise rejection in general. In Lab #5, we will discuss other aspects of the filter design in more details.

## 3 Experiments

The purpose of the lab experiments is to verify the basic PLL operation in practice and to determine or experimentally confirm the 4046 PLL characteristics needed for the design in Lab #5.

### 3.1 VCO Component Selection

The theoretical expressions for the VCO minimum frequency  $f_{min}$  and the maximum frequency  $f_{max}$  are unfortunately not very reliable. Experimentally determine  $R_1$ ,  $R_2$  and  $C_1$  so that  $f_{min} = 8\text{kHz}$  and  $f_{max} = 12\text{kHz}$ . To adjust for  $f_{min}$  you can simply connect the VCO input (pin 9) to ground. To adjust for  $f_{max}$ , you can connect the pin 9 to  $V_{DD}$ . The loop filter is not needed in this part.

From the experiment, determine factors  $k_1$  and  $k_2$  in

$$f_{min} = \frac{k_1}{R_2(C_1 + 32pF)}, \quad (13)$$

$$f_{max} = f_{min} + \frac{k_2}{R_1(C_1 + 32pF)}, \quad (14)$$

so that the modified expressions can be used to estimate the required parameter values more accurately for a different set of specifications.

### 3.2 VCO Characteristics

Measure and plot the VCO characteristic  $f_{osc}(v_o)$  for  $0 \leq v_o \leq V_{DD}$ . The loop filter is not needed in this part.

In the range  $(v_o)_{min} \leq v_o \leq V_{DD}$ , the VCO output frequency  $f_{osc}$  should change as a linear function of  $v_o$ . For  $v_o < (v_o)_{min}$ , the frequency  $f_{osc}$  does not depend on  $v_o$ .

Determine:

1.  $(v_o)_{min}$ ,
2. the free-running VCO frequency  $f_o$  for  $v_o = V_{DD}/2$ , and
3. the gain  $K_o$  of the VCO.

If the VCO characteristic is linear,  $K_o$  can be determined from the end-points of the linear range,

$$K_o = \frac{f_{max} - f_{min}}{V_{DD} - (v_o)_{min}}. \quad (15)$$

Slope of a straight-line interpolation through the measured points for  $v_o \geq (v_o)_{min}$  should give the same result.

### 3.3 Operation of the PLL

In this part you should close the loop by inserting the  $RC$  filter between the phase detector output and the VCO input, as shown in Fig. 3. Use a laboratory pulse generator as the source of the incoming signal  $v_i$ . Adjust frequency  $f_i$  of  $v_i$  to approximately match the free-running frequency  $f_o$  of the VCO. When  $v_i$  is applied, the PLL should operate in the locked condition, with  $f_{osc}$  exactly equal to  $f_i$ . The locked condition can be easily verified by observing  $v_i$  and  $v_{osc}$  *simultaneously* on a dual-trace oscilloscope. If  $f_i = f_{osc}$ , stable waveforms of *both*  $v_i$  and  $v_{osc}$  can be observed. Otherwise, one of the waveforms on the scope screen is blurred or is moving with respect to the other.

By changing  $f_i$  of the incoming signal, determine the actual lock range of the PLL, i.e., determine the maximum and the minimum frequency  $f_i$  such that starting from the locked condition the PLL remains in the locked condition. The lock range should be equal to  $f_{max} - f_{min}$ .

Determine the minimum peak-to-peak amplitude of the incoming signal  $v_i$  such that the PLL remains locked.

Get a hard copy or sketch the waveforms  $v_i$ ,  $v_{osc}$  and  $v_\phi$  for the PLL in the locked condition, at three frequencies  $f_i$ :

1.  $f_i = f_o$ ;
2.  $f_i =$  the lowest frequency of the lock range (should be equal or very close to  $f_{min}$ );
3.  $f_i =$  the highest frequency of the lock range (should be equal or very close to  $f_{max}$ ).

In the report, compare the phase difference  $\phi$  between  $v_i$  and  $v_{osc}$  to the theoretical prediction for the three frequencies of the incoming signal.

### 3.4 Characteristic of the PLL

Measure and sketch the characteristic  $v_o(f_i)$  of the PLL. Note that  $v_o$  has a dc component with some high-frequency ac ripple around it. In this measurement we are interested in the dc component, which means that  $v_o$  can be measured using a dc voltmeter. Since  $f_{osc}$  is directly determined by  $v_o$ , this characteristic should have the same shape as the theoretical characteristic  $f_{osc}(f_i)$  of Fig. 2. In particular, note the hysteresis in the characteristic, which means that you need to change  $f_i$  in both directions in order to obtain all parts of the characteristic correctly. During the measurement, it helps to monitor whether the PLL is locked or not. When the PLL is not in the locked condition, the voltage  $v_o$  should be equal to  $V_{DD}/2$ . Make sure that you determine the end points of the capture range and the lock range correctly. In your report, find a theoretical prediction for the  $v_o(f_i)$  characteristic, and compare it to the measurement.

### 3.5 Effects of Changing the Loop Filter

In this part, you need to investigate effects of changing the cut-off frequency  $f_p$  of the loop filter on the capture range of the PLL and the high-frequency ripple in  $v_o$ . You can change  $f_p$  simply by changing one of the filter components ( $C_f$  or  $R_f$ ). Cover the range  $100\text{Hz} < f_p < 10\text{kHz}$ . Measure and plot, as functions of  $f_p$ :

1. The end points of the capture range, and the capture range  $2f_c$ ;
2. The peak-to-peak ac ripple in the filter output voltage  $v_o$ , when the PLL is locked, and  $f_i = f_o$ .

In your report, compare the measured results with theoretical predictions.

Results of this lab assignment will be used for design purposes in the lab #5. Therefore, keep a copy of your report as a reference.

## 4 Prelab Assignment

The prelab assignment is due in the lab on the day when you start working on the experiment.

Note: Lab 4 is a **one-week** lab but you have two weeks to complete the report. See the due date on the Syllabus page.

Read the complete Lab 4 handout.

1. Select  $C_1$ ,  $R_1$ , and  $R_2$  in Fig. 3 of the handout, so that the VCO operates from  $f_{min} = 8\text{kHz}$  to  $f_{max} = 12\text{kHz}$ . Find the VCO frequency sensitivity  $K_o$  if the VCO characteristic  $f_{osc}(v_o)$  is linear for  $0 \leq v_o \leq V_{DD}$ . The supply voltage is  $V_{DD} = 15\text{V}$ .
2. Select  $C_f$  and  $R_f$  so that the cut-off frequency of the low-pass filter is  $f_p = 1\text{kHz}$ .
3. Select  $C_i$  if the incoming signal frequency  $f_i$  is in the 5kHz to 15kHz range.
4. Assume that  $v_i(t)$  is a square-wave signal at the frequency  $f_i$ , that the PLL is in the locked condition ( $f_{osc} = f_i$ ), and that  $v_o$  is a dc voltage with negligible ac component. Sketch and label  $v_i(t)$  and  $v_{osc}(t)$ , and determine voltage  $v_o$ , for:
  - a)  $f_i = 9\text{kHz}$
  - b)  $f_i = 10\text{kHz}$
  - c)  $f_i = 11\text{kHz}$

**Make a copy of your prelab work so that you can use it during the Lab session.**

---

# Anexo D














---

# Anexo D

## Instalación del sistema de simulación

El sistema de bloques de simulación se encuentra en un directorio denominado Comdigital y distribuido en cinco subdirectorios denominados: General, Codec, Modem, Ejcodif y Ejmod. Siendo su contenido el siguiente:

General: Son archivos de propósito general

-  ayudacomdig.m
-  comdig
-  comlibs
-  ejemplost
-  muestraparam
-  ayudacomdig
-  comdig.m
-  comlibs.m
-  ejemplost.m
-  gendatos.m
-  analisis.mdl
-  filtros.mdl
-  fuentes.mdl
-  muestraparam.m
-  ospv.m
-  retardos.m
-  retardosmul.m
-  sfunpsd2.m
-  sfunxyjp.m

Codec: Archivos de los codificadores de línea.

 amicod.m	 manchsdif.m
 b3zscod.m	 manchsdifdec.m
 b3zsdec.m	 millercod.m
 bifasemscod.m	 millerdec.m
 bifasemsdec.m	 ms43cod.m
 cb6bcod.m	 ms43dec.m
 cb6bdec.m	 pstcod.m
 cmicod.m	 pstdec.m
 cmidec.m	 qb3tcod.m
 db1qcod.m	 qb3tdec.m
 db1qdec.m	 qb5bcod.m
 hdb3cod.m	 qb5bdec.m
 hdb3dec.m	 codecs.mdl
 manchsm	 decods.mdl
 manchsddec.m	



Modem: Archivos que utilizan para modulación digital.

- conypseriem.m
- askdemodular.mdl
- askmodular.mdl
- demodular.mdl
- fskdemodular.mdl
- fskmodular.mdl
- modular.mdl
- pskdemodular.mdl
- pskdsdemodular.mdl
- pskdsmodular.mdl
- pskmodular.mdl
- qamdemodular.mdl
- qammodular.mdl

Ejcodif: Ejemplos de simulación de códigos de línea

amidec.mdl	manchesterpsd.mdl
amirzdec.mdl	millerdecod.mdl
b3zsdecodific.mdl	ms43decodif.mdl
bifase_msdec.mdl	nrzdec.mdl
cb6bdecodif.mdl	nrzpolardec.mdl
cmidecodif.mdl	pstdecodif.mdl
db1qdecodif.mdl	qb3tdecodif.mdl
hdb3.mdl	qb5bdecodif.mdl
manchdifdec.mdl	rzpolardec.mdl
manchesterdec.mdl	rzunpdec.mdl

Ejmod: Ejemplos de simulación de Modulación Digital

- askdemmul.mdl
- fskdemodcruces.mdl
- fskdemp11sfmsen.mdl
- b\_psk2dem.mdl
- b\_psk2psd.mdl
- dpskdem.mdl
- prkdem.mdl
- psk16dem.mdl
- qpskdemsk.mdl
- qpskmod1.mdl
- qpskmod2.mdl
- qam16dem.mdl

## D.1 Requerimientos de Hardware.

Este sistema de simulación, al utilizar las Student Edition de MATLAB y SIMULINK, necesita las siguientes condiciones de trabajo:

- IBM or 100% compatible con Intel 486 procesador más 487 coprocesador (excepto 486DX, que lo incluye), Pen-tium, o Pentium Pro.
- Microsoft Windows 95 o Windows NT.
- CD-ROM drive.
- Suficiente espacio en disco solo para el MATLAB y SIMULINK.
- Memoria:
- Microsoft Windows 95: 8 MB mínimo; 16 MB recomendado.
- Microsoft Windows NT 3.51 or 4.0: 12 MB mínimo; 16 MB recomendado.

## D.2 Instalación del sistema de simulación.

La instalación de estos programas es muy sencilla, consiste en copiar estos archivos de la manera que se encuentran organizados y luego se debe hacer que el MATLAB, tenga presente donde esta cada subdirectorio.

En el menú general el usuario encontrará un botón denominado *pathbrowser*, tal como se indica a continuación en la figura D.1.

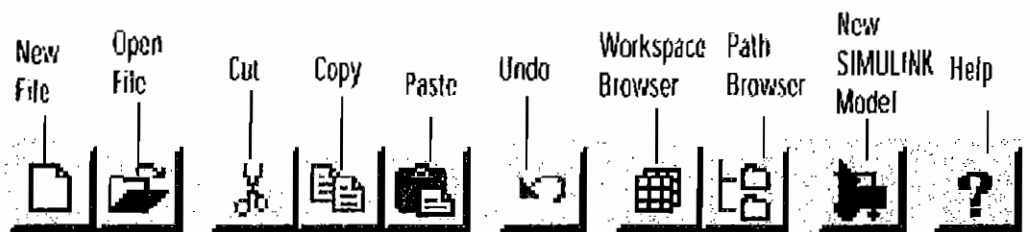


Figura D.1 Botones del menú principal de *MATLAB*.

Al tomar la opción de *path Browser* aparecerá la siguiente ventana de la figura D.2, donde se tomará el botón *Add to Path*, luego se indica donde se encuentran cada subdirectorio. Una vez terminado este proceso se cierra esta ventana con el botón *Save Setup* finalizándose de esta manera la instalación.

Una vez finalizada la instalación digita "comdig" para tener acceso a las librerías y ejemplos.

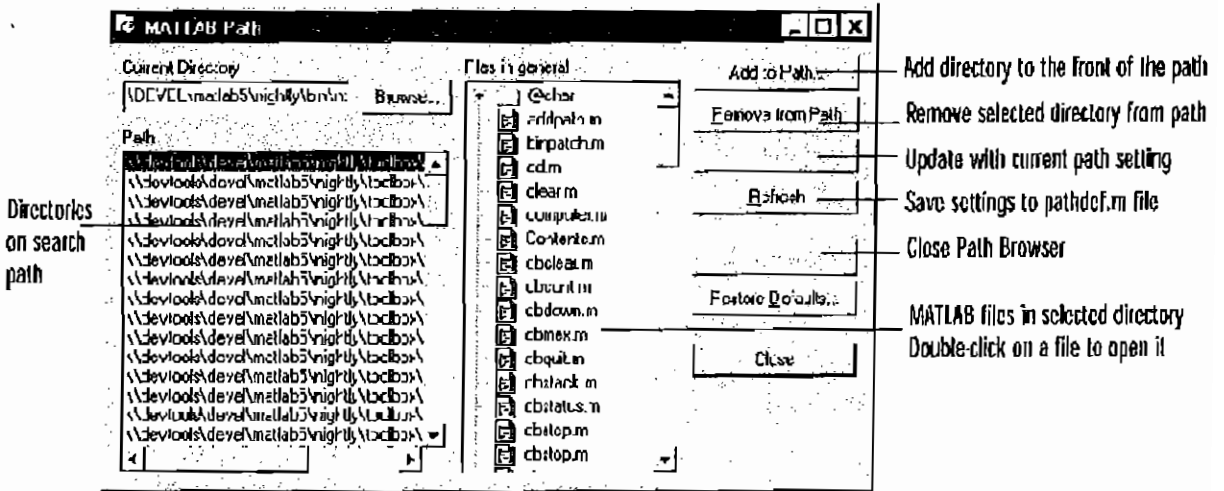


Figura D.2 Ventana del Path Browser de *MATLAB*.