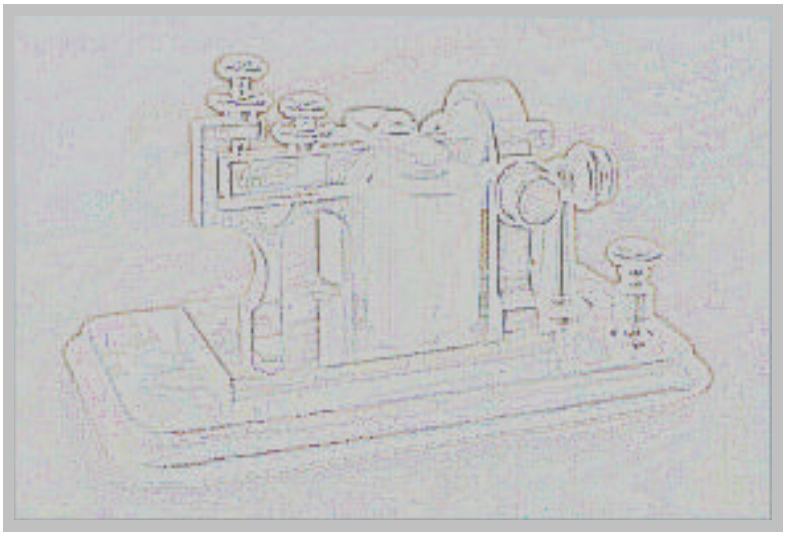
PROYECTO

DECODIFICADOR AUTOMÁTICO DE CÓDIGO MORSE



PABLO FERREIRA **PABLO SALVÁTICO**

Ing. Juan Pechiar

AÑO 2002

ÍNDICE TEMÁTICO

PREFACIO

Desde nuestra entrada a Facultad de Ingeniería, y quizás también desde antes, esperábamos el día en que pudiéramos presentar nuestro proyecto de final de carrera y marcar así el final de nuestra formación. Luego de transitar varios años por esta institución - que no dió muchas satisfacciones y (no vamos a negarlo) nos costó mucho esfuerzo- logramos comprender la esencia de la profesión que elegimos: sin dudas podemos afirmar que va más halla de las fórmulas o de los teoremas, de los circuitos o de la Internet, sino que es en el campo de las ideas donde nuestra profesión se desarrolla. Es en la pasión por lograr la solución, es en lo abstracto de la intuición, es en lo atento de la observación, es en la paciente investigación y en el global entendimiento de la física de nuestro mundo- entre otras cosas - donde un Ingeniero hace valer su título.

Más hallá de la complejidad y del frenétismo del mundo tecnológico que nos tocará vivir a lo largo de nuestra vida profesional, es en la formación básica que hemos recibido donde están todas las claves para desarrollarnos completamente en nuestra profesión. Desde éstas bases que vamos a poder seguir creciendo, evolucionando e innovando para contribuir a ser mejor a nuestra sociedad. Por tanto estamos convencidos de que ésto es recién el comienzo y que muchos desafíos nos esperan a la vuelta de la esquina de ahora en más. Pero tenemos confianza en que poseemos las herramientas suficientes como para tener éxito en el ejercicio de la Ingeniería.

En éste, que formalmente es nuestro último trabajo en esta casa de estudios, tuvimos la suerte de poder ahondar en varios temas y experimentar el desafío de buscar la solución a un problema en particular.

La idea para nuestro proyecto surgió como respuesta a un e-mail de una persona que, desde España, se interesó por un proyecto nuestro anterior que había sido publicado en la página web del Instituto referido a un modulador-demodulador de FM de banda angosta construido sobre la base de un DSP 56300. En este cruce de mensajes electrónicos pudimos saber que se trataba de Jabi Aguirre, vicepresidente de AMSAT, acérrimo radioaficionado y apasionado del los DSP. Él nos propuso entonces construir un detector de código Morse lo más automátizado posible. A primera vista nos pareció un proyecto simple y hasta pasado de moda, sin embargo, al estudiar la propuesta, vimos que se nos estaba presentando un desafío interesante y con la anuencia de Juan Pechiar comenzamos a desarrollarlo.

1 DESCRIPCIÓN DEL PROYECTO

1.1 Definiciones y Requerimientos

El objetivo del proyecto que se nos planteó era construir un decodificador de código Morse basado en DSP que requiriera la mínima intervención del operador en la tarea de ajuste del mismo, y que debía cumplir con los siguientes requisitos:

1. Minimizar la intervención del operador.
2. Ser capaz de operar en un rango amplio de frecuencias.
3. Decodificar a velocidades de transmisión normales en CW (hasta 25WPM).
4. Presentar estabilidad y detectar señales – aún las más ruidosas- en forma correcta.
5. Contar con una interfase humana simple y de fácil uso.
6. Decodificar las señales de igual o mejor forma que un operador experto.

Para la implementación del sistema contábamos con los siguientes materiales: un Kit de desarrollo basado en el DSP 56002 de Motorola (donado por AMSAT), herramientas de simulación y tratamiento de señales de audio.

1.2 Plan de Trabajo

El proyecto fue dividido en varias etapas con el fin encausar todos los esfuerzos a soluciones puntuales, y cada fase estuvo bien definida presentando claros requerimientos y metas a alcanzar.

1.2.1 Investigación

En esta primera instancia del proyecto comenzamos a tomar contacto con lo que era el código Morse, con su historia y con su seguidores. Analizamos distintas técnicas y problemas que deberíamos enfrentar respecto a cosas que en un comienzo parecían tan simples como detectar presencia o ausencia de señales.

1.2.2 Análisis de Soluciones

En lo que refiere al análisis de las soluciones planteamos distintos modelos y formas de enfrentar los problemas, y fue de dicho análisis que surgió un modelo ideal y distintas variantes que fueron estudiadas a fondo – éramos plenamente conscientes que una mala decisión en esta intancia comprometería seriamente los

tiempos del proyecto -. En suma definimos el modelo de bloques y de capas que usaríamos durante todo el proyecto.

1.2.3 Simulación y Evaluación

Los distintos bloques funcionales pasaron a ser simulados en Matlab y buena parte del tiempo se dedicó a caracterizar la señal y su efecto sobre los distintos bloques. Fue aquí donde evaluamos disversos detectores y filtros hasta hallar aquel de cada especie que extrajera el máximo de información de la señal.

1.2.4 Desarrollo de Bloques Básicos

Una vez simulado el funcionamiento de los módulos y definidas sus entradas y salidas pasamos al desarrollo del código en assembler. Cada bloque fue probado y optimizado para cumplir con los requerimientos particulares y globales.

1.2.5 Integración

Creamos una superestructura que combinara y gerenciara el funcionamiento de los módulos en su interacción en tiempo real.

1.2.6 Ensayos y Depuración

Esta instancia resultó una de las más largas del proyecto en la cual se nos presentaron la mayor cantidad de cuestionamientos del proceso. Esto se debió a que los bloques funcionaban correctamente pero el funcionamiento global no cumplia con las metas del proyecto. Fue entonces el momento de rediseñar parte de los módulos, tomando en cuenta patrones de señal que inducían a fallas graves de detección que hasta el momento no surgían como relevantes. Fue un proceso de varias semanas de simulaciones , ajustes y pruebas de campo.

1.2.7 Evaluación de Performance

Para este momento el decodificador funcionaba correctamente y ya se lo había probado en señales reales con distintos patrones de distorsión, superando todas las pruebas.
Luego llegó el día en que el sistema fue sometiudo a la prueba de fuego: enfrentarlo a un operador humano con señales reales obtenidas directamente del receptor del radioaficionado. Como esperábamos la codificación fue la correcta en todas las pruebas realizadas.

1.2.8 Documentación

La documentación fue un proceso constante durante todo el proyecto: se llevaron apuntes de las pruebas, problemas y soluciones. Dado que programar en assembler es complejo fue vital registrar las distintas versiones para poder ir depurando un código que superó las mil quinientas (1.500) líneas.

2 INTRODUCCIÓN AL CÓDIGO MORSE

2.1 Un Poco de Historia

Se puede afiemar que la modulación de onda continua (CW en sus siglas en inglés) surge en forma natural con el descubrimiento de la electricidad ya que resulta ser la forma de modulación más simple de generar: sólo hace falta un interruptor que se abra y y que se cierre para generarla.

Los sistemas de transmisión de mensajes a través de presencia o ausencia de una señal están presentes desde mucho antes de la invención del telégrafo. Podemos citar como ejemplo a los mensajes que se transmitían entre barcos encendiendo o apagando una fuente de luz, o a los mensajes que intercambiaban indígenas, tanto de América como de África, por medio de señales de humo o del sonido de sus tambores.

Lo realmente revolucionario del CW fue la posibilidad de transmitir mensajes, no ya a pocos kilómetros sino a miles de kilómetros, a una velocidad asombrosa que dejaba atrás cualquier otro sistema utilizado hasta el momento. Fue de esta forma que se comenzaron a unir ciudades y luego países por cables que corrían al costado de vías o por debajo de los ríos y que atravesaban montañas.

Cuando resultó necesario crear un código que hiciera posible la comunicación en forma eficiente, fue que Samuel Morse desarrolló el código que lleva su nombre, continuando trabajando en su perfeccionamiento luego hasta llegar al código tal como se lo conoce y urtiliza en la actualidad.

2.2 El Morse Como Lenguaje

El código Morse asigna una combinación de puntos y de rayas a cada letra del alfabeto. Como forma de hacerlo más eficiente las letras que más se repiten en el idioma inglés son representadas con menos símbolos, lo que hace que el código sea de largo variable.

Cuando decimos que una letra ésta formada por puntos y rayas estamos representando gráficamente la forma de generar el código, es decir, el punto es presencia de señal durante un intervalo breve y la raya durante un intervalo más largo de tiempo. A su vez, las rayas y los puntos están separados por silencios del largo de un (1) punto. La duración del punto y la raya está prefijado siendo el largo de una raya tres (3) veces el del punto.

CW Típico	25 WPM	
Fast CW	50 WPM	Meteor scater
SSB	250 WPM	
Slow HSCW	300 WPM	
Kinda slow HSCW	400 WPM	Usada en EEUU para CQ
Faster HSCW	800 WPM	Uso común
Very Fast HSCW	1600 WPM	
Ultra Fast HSCW	3200 WPM	

Para transmitir un mensaje es necesario transmitir en principio cada una de las letras que lo componen, por lo que debemos tener una forma de separar letras y palabras. Se establece que las letras se separarán por un silencio del largo de una (1) raya y las palabras por un silencio equivalente al largo de siete (7) puntos.

Como se puede apreciar, de la forma de transmitir surge que la velocidad también es variable y dependerá de la pericia de cada operador. Se establece una forma de medir la velocidad de transmisión por medio del método PARIS, donde a ésta palabra se la toma como el largo base para la medición de velocidad.

```
   P           A          R          I       S
.  - - .     . -        . - .      . .     . . .
11 31311  3  113    3   11311  3   111  3  11111   7
```

En total el largo es equivalente a cincuenta (50) puntos.

Entonces alguien que, por ejemplo, pueda transmitir a 5 WPM (cinco palabras por minuto) será capaz de generar Morse con puntos de largo 60/(5WPM*50 puntos)= 240 ms. Esta modalidad es conocida como “Slow Code” ya que suena realmente lento.

Una variante de trasmisión es el modo "Farnsworth”, donde se acorta la duración del punto y la raya pero se aumenta el espacio entre caracteres y palabras de forma de mantener la velocidad global de transmisión.

Las velocidades usuales de transmisión están entre las 15 y 25 WPM. Velocidades mucho mayores se pueden obtener en el llamado HSCW (High Speed CW o CW de alta velocidad) donde se transmite a más de 300 WPM utilizando software y técnicas de transmisión más sofisticadas, que incluyen la compresión del Morse. El otro extremo lo ocupa el CW de muy baja velocidad donde un punto puede durar varios minutos y es utilizado en comunicaciones de muy baja potencia, por ejemplo la técnica denominada EME que utilizan la reflexión sobre la Luna para comunicar dos puntos sobre la tierra.

En las tablas mostradas a continuación se puede apreciar la combinación asignada a cada letra.

Letra	Morse	Letra	Morse	Dígito	Morse	Letra	Morse	Puntuación	Morse
A	.-	N	-.	0	-----	Ä	.-.-	.	.-.-.-
B	-...	O	---	1	.----	Á	.--.-	,	--..--
C	-.-.	A	.--.	2	..---	Å	.--.-	:	---...
D	-..	Q	--.-	3	...--	Ch	----	?	..--..
E	.	R	.-.	4	....-	É	..-..	Apóstrofe	.----.
F	..-.	S	...	5	.....	Ñ	--.--	Tilde	-....-
G	--.	T	-	6	-....	Ö	---.	/	-..-.
H	....	U	..-	7	--...	Ü	..--	Paréntesis	-.--.-
I	..	V	...-	8	---..			Comillas	.-..-.
J	.---	W	.--	9	----.				
K	-.-	X	-..-						
L	.-..	Y	-.--						
M	--	Z	--..						

El código Morse como lenguaje tiene una serie de abreviaciones de palabras que surgieron del uso. Muchas de ellas se refieren a parámetros de la transmisión y se engloban dentro del código Q, otras forman parte del protocolo de la comunicación, y otras tantas están referidas a aplicaciones particulares. A continuación se presentan algunas de estas abreviaciones:

Código Q	Pregunta	Respuesta
QRA	What is the name of your station ?	The name of my station is...
QRB	How far approximately are you from my station ?	The approximate distance between our stations is.....nautical miles (or.... kilometers)
QRG	Will you tell me my exact frequency (or that of....) ?	Your exact frequency (or that of...) is ...Khz (or Mhz)
QRH	Does my frequency vary ?	Your frequency varies.
QRI	How is the tone of my transmission ?	The tone of your transmission is ... 1 = good. 2 = variable. 3 = bad

QRK	What is the readability of my signals ?	The readability of your signals is..... 1 = bad. 2 = poor. 3 = fair. 4 = good. 5 = excellent.
QRL	Are you busy ?	I am busy
QRM	Are you being interfered ?	I am being interfered with: 1 = nil. 2 = slightly. 3 = moderately. 4 = severely. 5 = extremely.
QRN	Are you troubled by static ?	I am troubled by static 1 = nil. 2 = slightly. 3 = moderately. 4 = severely. 5 = extremely.
QRO	Shall I increase transmitter power ?	Increase transmitter power.
QRP	Shall I decrease transmitter power ?	Decrease transmitter power.
QRQ	Shall I send faster ?	Send faster (or ... words per minute).
QRS	Shall I send more slowly ?	Send more slowly (or ... words per minute).
QRT	Shall I stop sending ?	Stop sending.
QRU	Have you anything for me ?	I have nothing for you.
QRV	Are you ready ?	I am ready.
QRW	Shall I inform...that you are calling him on...khz (or...Mhz)?.	Please inform...that I am calling him on...khz(or...Mhz)
QRX	When will you call me again ?	I will call you again at...hours (on ...khz (or ...Mhz)).
QRY	What is my turn ?	Your turn is number....(or according to any other indication).
QRZ	Who is calling me ?	You are being called by...(on ...khz (or ...Mhz)).
QSA	What is the strength of my signals ?	The strength of your signals (or those of ...) is... 1 = scarcely perceptible. 2 = weak. 3 = fairly good. 4 = good. 5 = very good.
QSB	Are my signals fading ?	Your signals are fading.
QSD	Is my keying defective ?	Your keying is defective.
QSK	Can you hear me between your signals and if so can I break in on your transmission ?	I can hear you between my signals;break in on my transmission
QSL	Can you acknowledge receipt ?	I am acknowledging receipt.
QSO	Can you communicate with...?	I can communicate with...direct (or by relay through...).

QSP	Will you relay to...?	I will relay to...
QST	Is there any message for radio-hams ?	Here follows a message for radio-hams
QSU	Shall I send or reply on this frequency (or on ...khz (or ...Mhz))?	Send or reply on this frequency ? (on ...khz (or ...Mhz)).
QSV	Shall I send a series of V's on this frequency (or on ...khz (or ...Mhz))?	Send a series of V's on this frequency (or on ...khz (or ...Mhz)).
QSW	Will you send on this frequency (or on ...khz (or ...Mhz))?	I am going to send on this frequency (or on ...khz (or ...Mhz)).
QSX	Will you listen to...?	I am listening to ...(call signs) on ...khz (or Mhz).
QSY	Shall I transmit on an other frequency ?	Transmit on an other frequency. (or on ...khz (or ...Mhz)).
QTC	How many messages have you for me ?	I have...messages for you.
QTH	What is your position ?	My position is...

1-9
33 fondest regards
55 best succes
73 best regards
88 love and kisses

A
abt about
ads adress
agn again
ani any
ans answer
ant antenna
awdh auf wiederhoeren

B
b4 before
bcnu be seeing you
bcp beaucoup (french)
bd bad
bjr bonjour (french)
bk break
bth both
buro bureau

C
c yes
cb (1) callbook
cb (2) citizensband

M
mci merci (french)
mgr manager
mi my
mni many
mom moment
msg message
mult multiplier

N
n no
n 9 (as in RST-report)
nfd national fieldday
nil nothing
nr number
nr near
nw now

O
ob old boy
oc old chap
ok correct
om old man
onli only
op operator
opr operator
ot old timer
ow old woman

cba callbook address
cfm confirm
cl closing
clg calling
clix key-clicks
cmg coming
condx conditions
congrats congratulations
cpi copy
cq general call
crd card
cs callsign
cuagn call you again
cud could
cul call you later
cuz because
cw continuous wave

D

de from
dr dear
dsw see you again (russian)
dwn down
dx long distance

E

el element
enuf enough
es and
eu europe
eve evening

F

fb fine business
fd fieldday
fer for
fm from
fone telephony
fq frequency
freq frequency
fwd forward

G

ga go ahead
ga good afternoon
gb goodbye

P

pse please
pwr power
px prefix
px press

Q

Q-Code

R

r received OK
rcvd received
rcvr receiver
rig radio equipment
rprt report
rpt repeat
rx receiver

S

sa say
sed said
shud should
sig signature
sigs signals
sk silent key
sked schedule
sn soon
sp short path
sri sorry
stn station
sum some
swl short wave listener

T

temp temperature
test test
test contest
tfc traffic
thru through
til until
tks thanks
trbl trouble
trx transceiver
tt that
tu thank you
tx transmitter

gd good day
ge good evening
gg going
gld glad
gm good morning
gn good night
gnd ground
gp ground plane
gs green stamp
gud good

H
ham radio amateur
hi high
hi laughter
hpe hope
hq headquarters
hr here
hr hour
hrd heard
hrs hours
hv have
hvg having
hvy heavy
hw how
hw? how did you copy?

I
ii I repeat
info information
inpt input

J
ja japan

K
k invitation to transmit
klix keyclicks

L
lid poor operator
lis liscensed
lp long path
lsn listen
lw long wire
txt text

U
u you
ufb ultra fine business
unlis unlicensed
up upward
ur your
urs yours

V
vert vertical
vx vieux (french)
vy very

W
watsa what do you say
wid with
wkd worked
wkg working
wl will
wpm words per minute
wrk work
wud would
ww would
wx weather

X
xcvr transceiver
xmas christmas
xmtr transmitter
xtal crystal
xyl wife

Y
yf wife
yl young lady
yr year

Z
z zulu-time (=UTC)

Como se puede observar el código Morse tiene variadas combinaciones de caracteres que simplifican la tarea de transmisión y agilitan el envio de mensajes. Sin embargo, se pierde la relación con el mensaje original tanto en su estructura, en la estadística de caracteres y en la coherencia entre palabras, por lo que se está en presencia de un nuevo lenguaje.

3 EL KIT DESARROLLO Y EL DSP 56002

3.1 Descripción General

El DSP56002EVM Evaluation Module (EVM) es una plataforma de bajo costo con la cual los usuarios se pueden familiarizar con el uso del Procesador de Señal (DSP) Motorola DSP56002.

En la placa se combinan: un procesador 24-bit con 32kbytes de memoria on-board SRAM y un codec Cristal 4215 que maneja calidad de audio de CD. Estas y varias cualidades más hacen de esta placa una herramienta muy interesante para demostraciones y prueba de nuevas aplicaciones.

Si bien el procesador que la integra está por ser discontinuado es el hermano menor de la familia de DSP56XXX, por lo que, conocido el funcionamiento de éste y su arquitectura, es muy fácil utilizar las nuevas versiones como ser el DSP56300. Hay que considerar que no todas las aplicaciones diseñadas para DSP56002 correrán directamente en versiones posteriores.

El módulo de evalución consta de:

1 Placa de desarrollo
2 Software de programación : compilador y ensamblador
3 Software para depuración (Debugger)
4 Manuales en CDROM

Sólo son necesarias una fuente de 7-9 volts , un cable serial con conectores DB9 y un PC para comenzar a usar la placa.

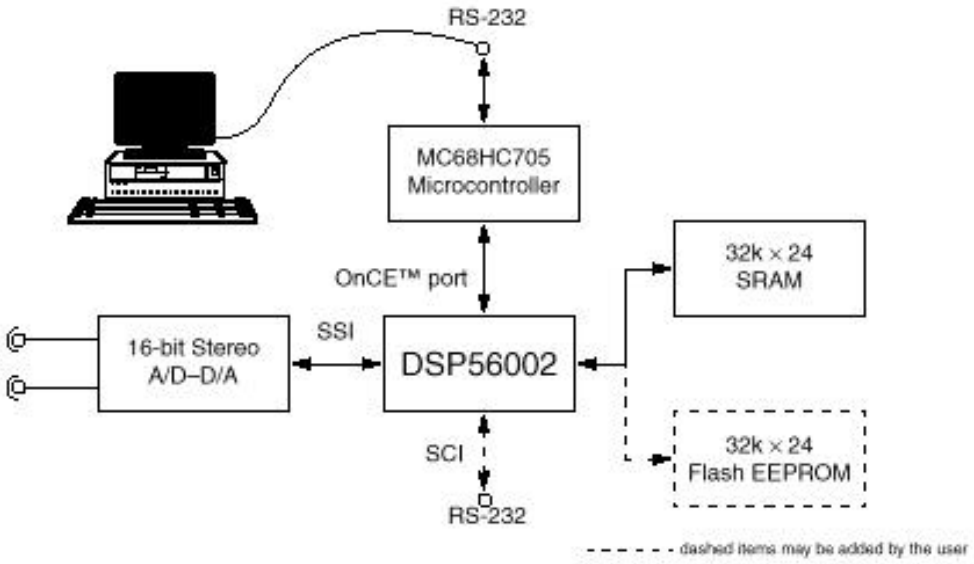
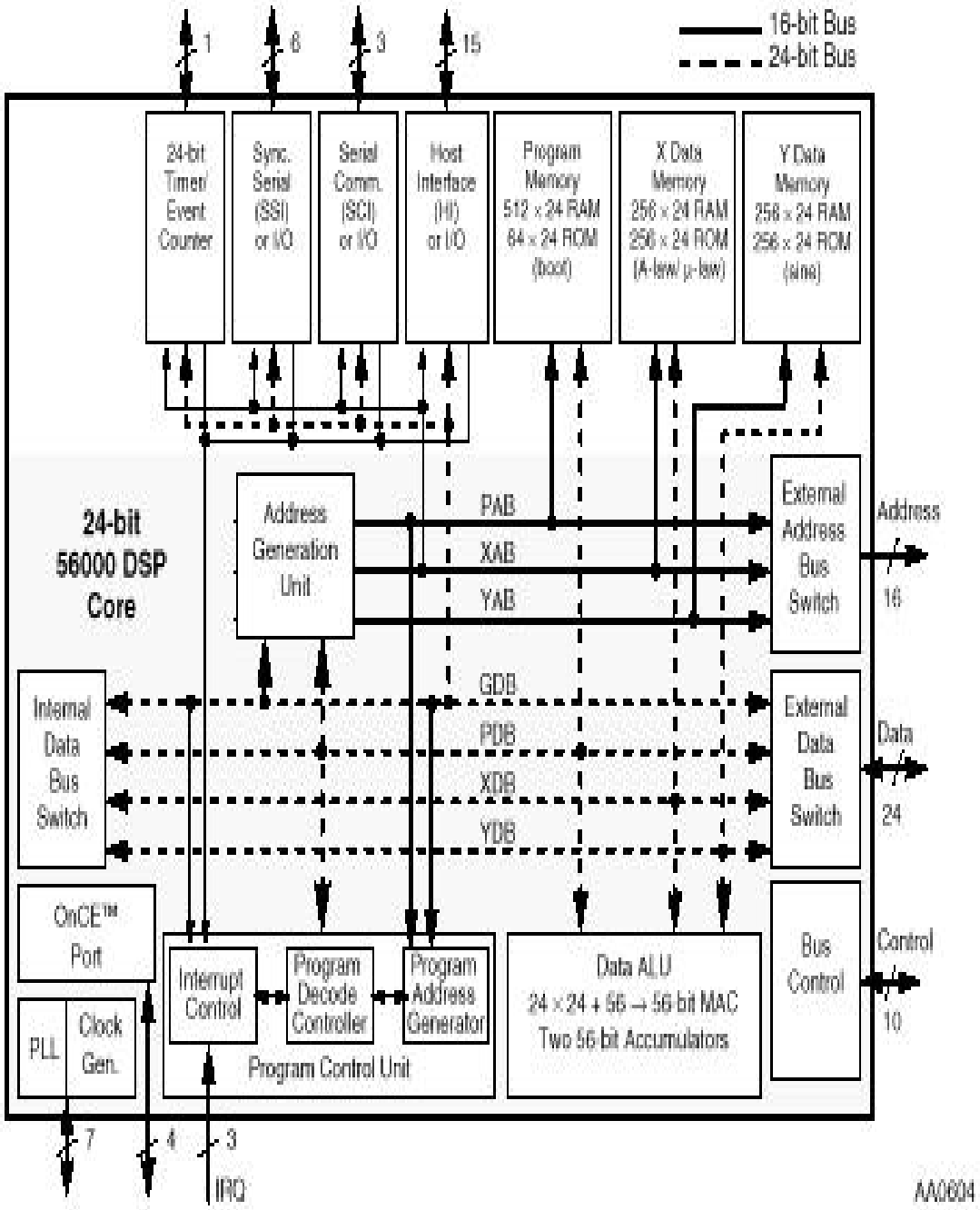


Figure 1 DSP56002EVM Block Diagram

3.2 El Hardware

3.2.1 Arquitectura del DSP56002

A continuación presentamos la estructura interna del procesador. Creemos que de ésta forma se hace más fácil comprender la lógica de los programas y cómo operan las instrucciones que los componen.



DSP56002 Block Diagram

Sus componentes principales son:

a. Buses de datos.
b. Buses de direcciones.
c. Unidad Aritmético-Lógica -Data Arithmetic Logic Unit (ALU).
d. Unidad Generadora de Direcciones - Address Generation Unit (AGU).
e. Unidad de Control de Programa -- Program Control Unit (PCU).
f. Expansión de Memoria (Port A).
g. Puerto para Debugger --- On-Chip Emulator (OnCE™).
h. Circuito Phase-locked Loop (PLL).

a. *Buses de Datos*

El módulo procesador DSP56K está organizado alrededor de tes unidades independientes: PCU, AGU y la ALU. Los movimiento de datos entre la distintas unidades se da a través de cuatro Buses de datos de 24-bits: bus de datos X (XDB), bus de datos Y (YDB), el bus de datos de programa (PDB) y bus global datos(GDB).

La transferencia de datos entre la memoria Y o X y la ALU se realizan dentro del chip para maximizar la velocidad y minimizar la disipación de potencia. Las otras transferencias de datos, por ejemplo hacia a los periféricos, se realiza por el GDB. El prefetch de las instrucciones del programa se realiza en forma paralela sobre el bus PDB.

La estructura del bus soporta movimiento de datos de:

1 registro a registro
2 registro a memoria
3 de memoria a registro.

Puede transferir hasta dos (2)palabras de 24-bits o una de 56-bit durante un ciclo de instrucción. La transferencia entre buses se realiza en el Bus switch.

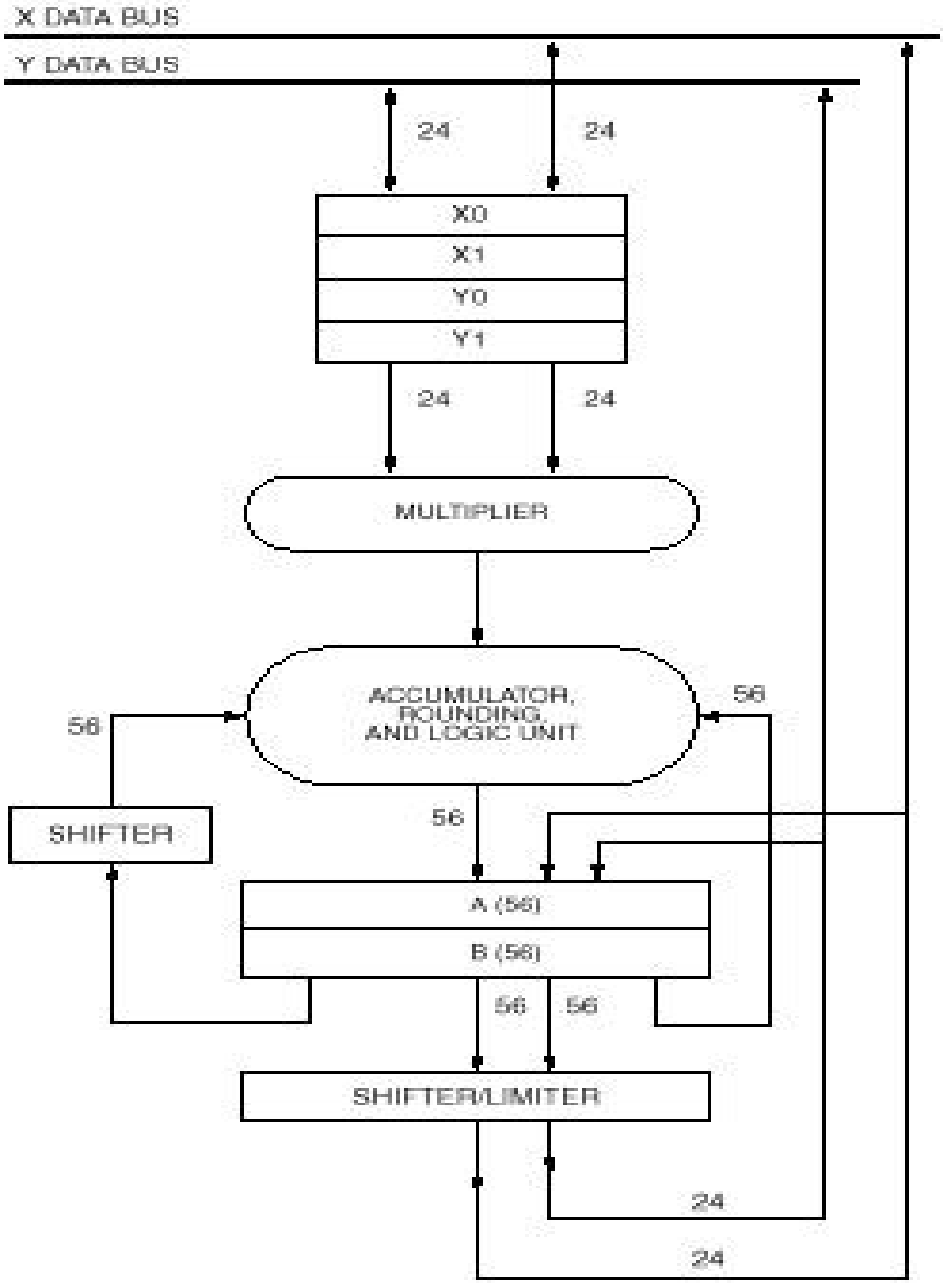
b. *Buses de Direcciones*

Las direcciones son especificadas para la memoria interna X e Y sobre dos Buses unidirecionales de 16-bits (XAB y YAB). Las direcciones de memoria correspondientes a programas son especificadas sobre un bus bidirecional de 16-bits (PAB). Las direcciones de memoria externas son cubiertas por un bus de 16 bit unidireccional manejado por un multiplexor que puede seleccionar XAB, YAB o PAB. Sólo un acceso a una dirección externa puede hacerse durante un ciclo de instrucción.

c. *ALU*

La ALU lleva a cabo todas las operaciones lógicas y matemáticas. Está formada por:

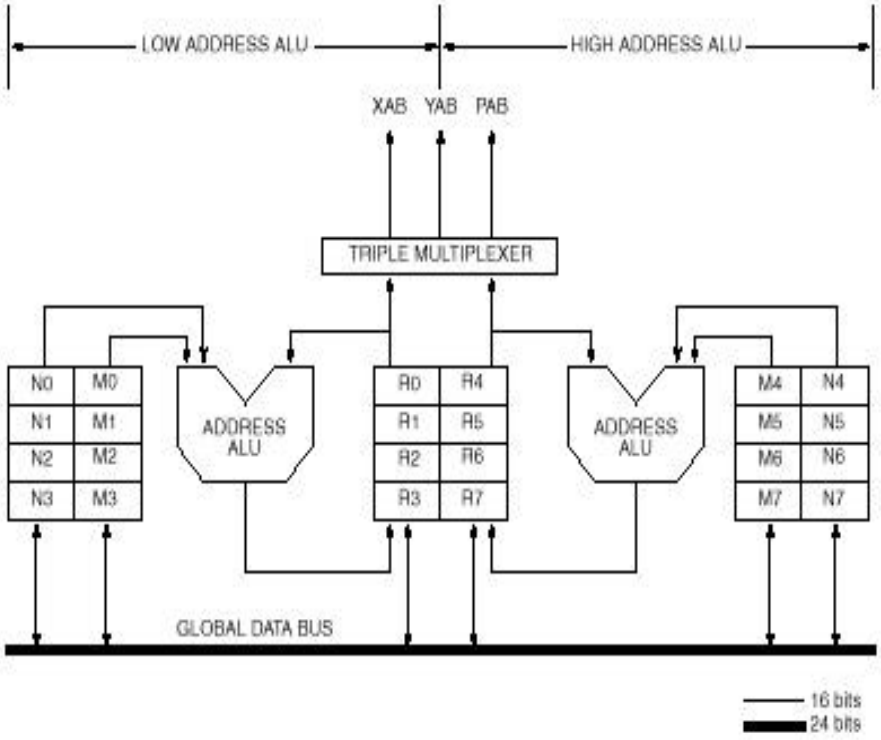
- cuatro (4) registros de 24 Bits.
- dos (2) acumuladores de 48-bit
- dos (2) registros de extensión del acumulador 8bit a un (1) shifter del acumulador
- un (1) limitador
- un (1) Multiplicador-Acumulador (MAC) paralelo de un (1) ciclo.



Data ALU

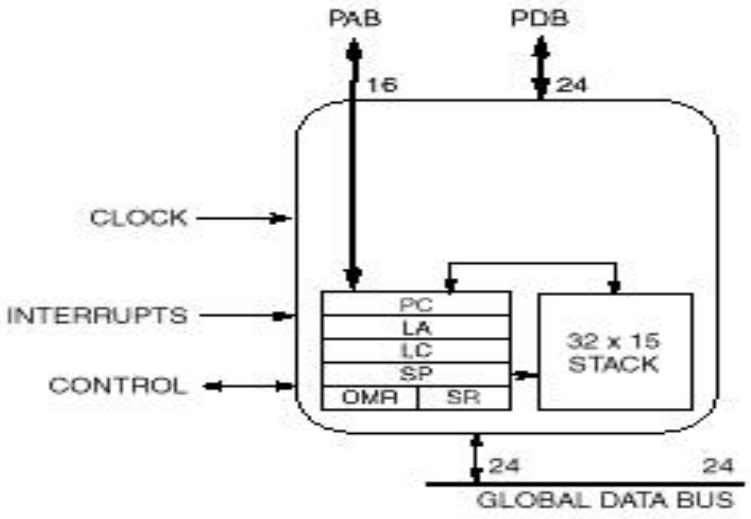
d. *AGU*

La AGU lleva a cabo el almacenamiento de las direcciones y los cálculos necesarios para el direccionamiento indirecto. Tiene dos (2) unidades idénticas para generar direcciones de 16-bit en cada ciclo de instrucción y además puede llevar a cabo operaciones de aritmética lineal, en módulo, reverse-carry.



PCU

La Unidad de Control de Programa realiza el prefetch de las instrucciones, la decodificación, el control del hardware DO loop, y el procesamiento de las interrupciones.



Program Address Generator

Está formada por:

- Un (1) generador de direcciones de programa.
- Un (1) controlador de decodificación.
- Un (1) controlador de interrupciones.
- Registros: Program counter (PC), Loop Address (LA), Loop Counter (LC), Status Register (SR), Operating Mode Register (OMR), and Stack Pointer (SP).

f. *Puerto A*

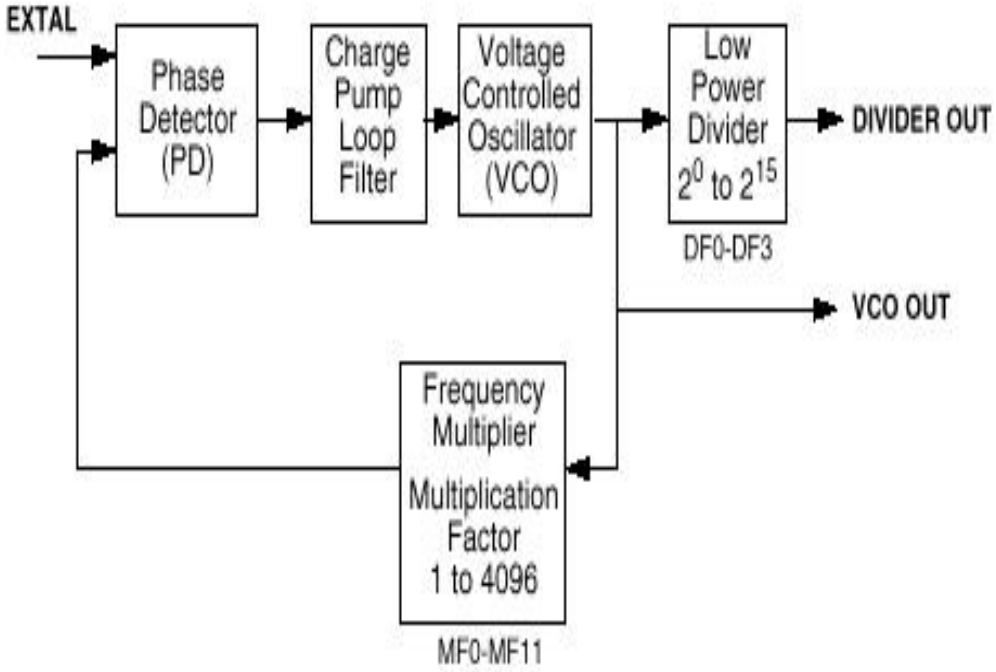
El puerto A posibilita la comunicación sincrónica con dispositivos tales como RAMs estáticas de alta velocidad, otros DSPs o MPUs en configuración master/slave.

g. *Puerto OnCE*

Este puerto exclusivo permite el acceso a registros, memoria y periféricos on-chip mediante una interfase simple y económica sin sacrificar recursos del DSP. Es utilizado para las tareas de Debugging.

h. *PLL*

El PLL permite usar casi cualquier fuente de reloj para operar al máximo de velocidad. Lleva a cabo la multiplicación de frecuencia y la división de baja potencia.



PLL Block Diagram

3.2.2 La Placa de Desarrollo

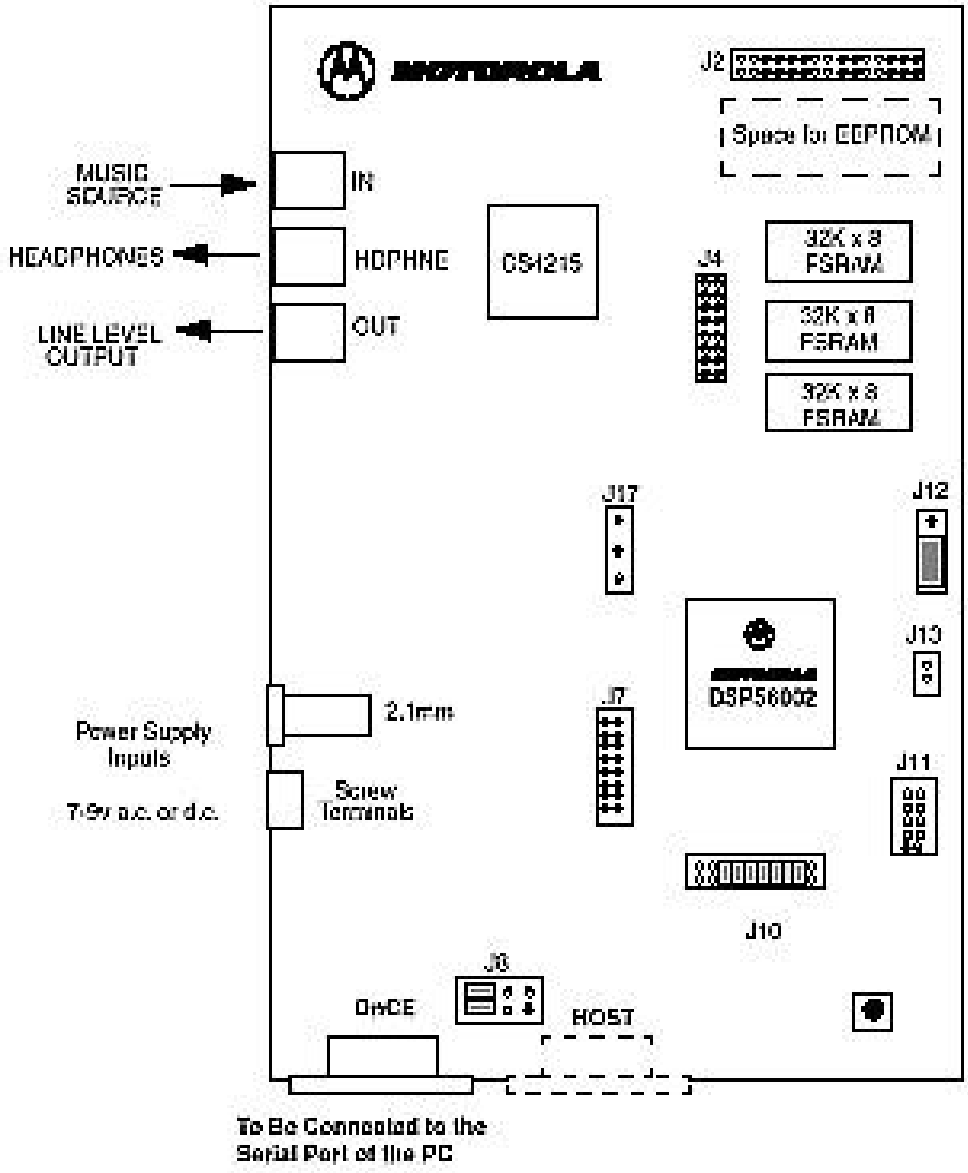
Las características técnicas más destacables de la placa de desarrollo se enumeran a continuación:

1. Posee un Procesador Digital de señal de 24-bit DSP56002 con un clock de 40 MHz.
2. Realiza hasta 20 milliones de instrucciones por Segundo (MIPS)– ciclo de instrucción de 50 ns a 40 MHz-.
3. Realiza hasta 120 millones de operaciones por Segundo (MOPS) a 40 MHz.
4. Ejecuta una Fast Fourier Transform (FFT) 1024-puntos en 59,898 clocks.
5. Contiene cuatro (4) Buses de Datos de 24-bit y tres (3) Buses de Direcciones de 16-bit para acceso simultáneo a una (1) memoria de programa y dos (2) de datos de 512 .
6. Posee RAM on-chip 24 bit programable con bootstrap ROM.
7. Posee dos (2) 256 x 24-bit on-chip RAMs para datos.
8. Incluye dos (2) 256 x 24-bit on-chip ROMs de datos conteniendo tabla del seno , Ley-A y Ley-?.
9. Posibilidad de expansión de su memoria por bus de direcciones de 16-bit y bus de datos de 24-bits.
10. Byte-wide Host Interface (HI) que soporta acceso directo a memoria.
11. Synchronous Serial Interface (SSI) para comunicarse con el codec y dispositivos seriales sincronos.
12. Serial Communication Interface (SCI) para comunicación serial asincrona full-duplex.
13. Contiene 24-bit Timer/Event Counter.
14. Contiene On-Chip Emulation (OnCE™) Puerto para debugging.
15. PLL (Phase-Locked Loop) programable por Software.
16. Tiene 32k x 24-bit zero-wait-state RAM estática externa para memoria de expansión.
17. Tiene opción para 32k x 8 bits flash EEPROM para programar boot strapping and stand-alone operation.
18. Posee conversor de alta calidad Análogo-Digital / Digital-Análogo estéreo con calidad de CD, Crystal modelo CS4215.
19. Cristal de 24.576 MHz para frecuencias de muestreo de: 48, 32, 16, 9.6 y 8 kHz.
20. Puede realizar los siguientes formatos de codificación: 16-bit linear, 8-bit Ley-mu, 8-bit Ley-A y 8-bit linear.
21. Regulador de voltaje MC7805ACT.
22. Controlador RS-232 para el puerto OnCE™ y SCI.
23. Presenta opción para Segundo conector RS-232 para acceso serial al DSP56002 por SCI.
24. Mediante el microcontrolador MC68HC705K1 realizando la conversión de comandos RS-232-to-OnCE.
25. MC33078 pre-amp para buffering analógico.

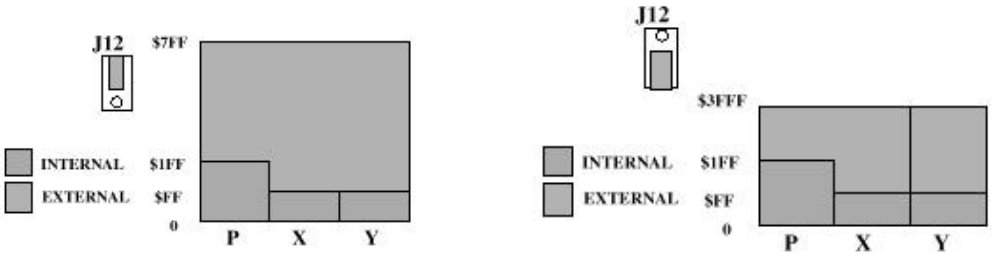
26. Contiene Jacks para entrada MIC, salida LINE y para auriculares.
27. Posee jack 2.1 mm y terminal de 2 tornillos para conexión de alimentación.

El siguiente gráfico ilustra la distribución real de los componentes sobre la placa. Destacamos la presencia de los siguientes componentes:

1. Los Jumpers que se utilizan para configurar algunas de las facilidades del DSP.
2. El conector que aparece punteado se agregó, ya que viene como opcional. Se utiliza para la conexión serie por el puerto SCI.



Seguidamente mostramos cómo afecta la configuración de la memoria a la posición de J12.

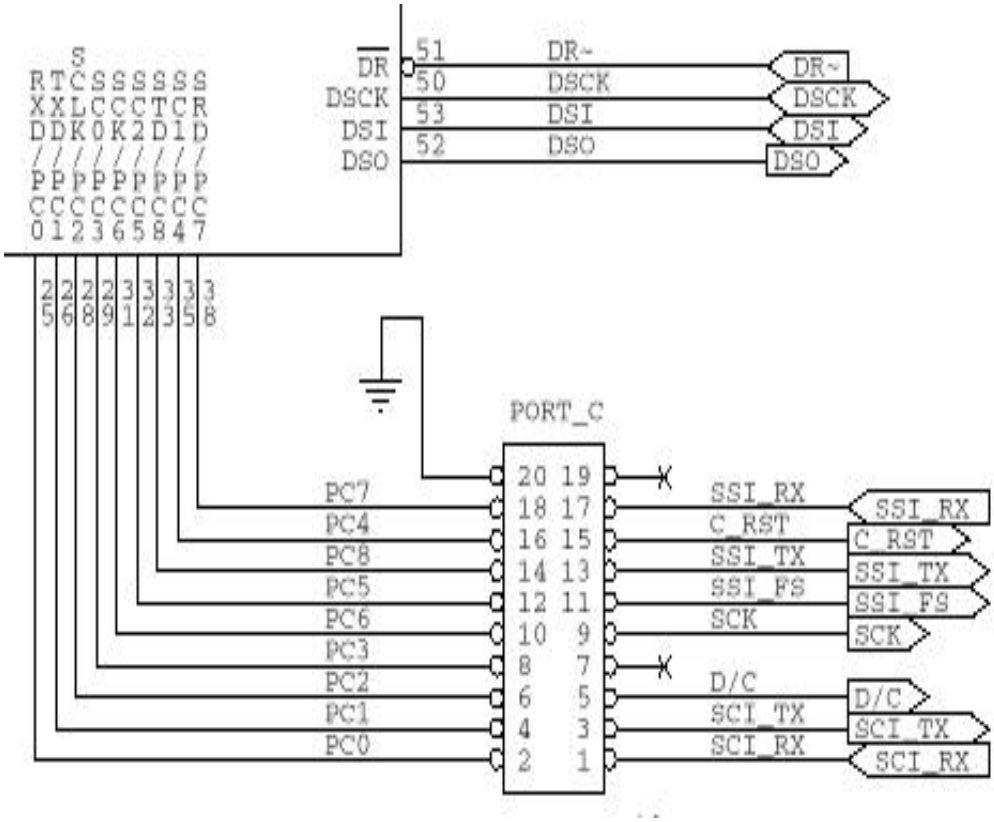


A B

La configuración usada en nuestro proyecto se indica con la letra B ya que potencia la operación en paralelo.

Otro Jumper útil es J8 que permite cruzar los puertos serie OnCE y SCI. Podemos utilizar un cable derecho estándar para comunicar el DSP con el PC.

Por último el Jumper J10 es el que habilita o deshabilita físicamente las salidas y entradas del puerto C. Se encuentran habilitadas cuando se ponen en corto circuito 1-2 , 3-4, etc.



El CODEC (codificador-decodificador) de la empresa Crystal Semiconductor modelo CS4215, realiza las conversiones A/D y D/A Este chip está diseñado para aplicaciones donde se necesiten conversiones de alta calidad de audio y bajo precio. Es capaz de convertir dos señales analógicas a valores digitales de 16 bits (A/D). También contiene dos conversores digitales analógicos de 16 bits (D/A). Estos conversores A/D y D/A forman las puertas de entrada y salida analógicas del EVM .

El CS4215 es un chip flexible, capaz de realizar muestreos programables por el usuario; tiene asimismo programable la ganancia de entrada y la atenuación de salida. El CODEC se comunica con el DSP por medio de la interfase SSI.. Una característica particular del EVM es que utiliza una sola entrada de audio del CS4215, la entrada de micrófono (mic). La entrada de línea está deshabilitada. Otra particularidad es que solamente se usa un cristal de referencia, de frecuencia 24.576 MHz, el cual está conectado a la posición "XTAL2" del CS4215 (que normalmente para otras aplicaciones está reservada para un cristal de 16.9344 Mhz). La referencia XTAL2 debe ser siempre elegida cuando se inicializan las velocidades de muestreo. Se puede elegir entre las siguientes velocidades de muestreo de : 8, 9.6 , 16, 32 y 48 mil muestras por segundo.

Un microprocesador en SMD de la familia 6805 simplifica el interfase Interface On-Chip Emulation (OnCE) para realizar las tareas de "debugging" (depuración del software). Este chip no solamente libera espacio en la placa, sino que también rebaja el precio del EVM. Su misión es convertir los comandos que recibe en serie provenientes del software de debugging desde el host (PC), a comandos de control en paralelo para controlar la operación del OnCE. La lógica de bajo nivel de la interfase OnCE es muy intrincada y consiste en diversos algoritmos de muy difícil programación y uso para los no expertos. El 6805 realiza toda esta labor pero solamente con una limitación: la velocidad ya que esta puerta serie, en el kit actual, es sólo capaz de operar hasta una velocidad de 19200 baudios. A esta velocidad, los programas muy largos pueden tardar un poco en ser cargados en la EVM. No es crucial para un desarrollador de programas para el DSP56002 el conocer el protocolo de bajo nivel de la interfase para la depuración de software, sí puede ser interesante el conocer las capacidades de la interfase OnCE a la hora de escribir otros programas o hacer posible el usar el EVM con otro hardware, por ejemplo en plataformas que no soporten el sistema operativo DOS.

Destacamos que las comunicaciones con el micro 6805 se hacen a 8 bits y 19200 baudios. El host -PC, en nuestro caso- envía un código de comando que puede consistir en un único byte o

varios bytes, dependiendo del comando. La respuesta del micro 6805 está también codificada, y consiste así mismo, en un único o en varios bytes.

El DSP 56002 tiene una memoria estática (SRAM) de 1024 palabras de 24 bits en el propio chip, y que a su vez está dividida en tres partes : 512 palabras de memoria de programa, "P"; 256 palabras de memoria, "X" y otras 256 palabras de memoria "Y". Ésta memoria incluida en el chip funciona a la velocidad del procesador y normalmente está reservada para ser usada por el código que necesite mayor velocidad de respuesta (por ejemplo interrupciones críticas ó variables que sean de acceso muy frecuente).

Además existe otra SRAM exterior de 32K en tres (3) chips 6206, cada uno de 8K. La lógica necesaria para la decodificación de direcciones es complicada debido a los pequeños tiempos de acceso y retrasos de propagación de los circuitos.

3.3 El Software de Tiempo Real

El DSP56002 presenta una variedad de herramientas para la programación de software orientado al tratamiento de señales de audio y video. Como parte de esta especialización nos encontramos con instrucciones de código y estructuras de registros que posibilitan el uso eficiente de los recursos al momento de calcular FFT, filtros, transformadas, etc.

El procesamiento paralelo se hace evidente al observar la estructura del Hardware que ofrece espacios de memoria y buses independientes para datos y para programas. Esta separación de datos e instrucciones se llama arquitectura Harvard y es diferente a la arquitectura Von Neumann usada, por ejemplo, en procesadores del tipo 80x86.

La arquitectura Harvard es típicamente usada en la tecnología DSP, básicamente debido a que utiliza el criterio del paralelismo. La potencia de esta arquitectura en paralelo se ve resaltada al considerar que la principal característica de los DSPs es su capacidad para realizar multiplicaciones y sumas (MAC) en forma rápida. El 56002 puede multiplicar dos (2) operandos, acumular (sumar) el resultado, tomar dos (2) nuevos operandos de dos (2) direcciones de datos independientes, realizar el ajuste de las direcciones y tomar la próxima instrucción, todo ello en un ciclo de reloj.

El manejo de memoria se realiza en forma eficiente por la posibilidad que ofrece el DSP de contar con multiples punteros y modos de direccionamiento. Estos modos se obtienen en formatos de instrucciones de una (1) palabra e instrucciones de dos (2) palabras. Otra diferencia es el rango dinámico adicional de los datos a 24 bits (144.5 dB) frente a

los 96.3 dB con los chips de 16 bits. El rango dinámico de 96 dB es suficiente para la calidad CD (Compact Disc) ya que está lo suficientemente cerca del rango dinámico del oído humano (100 dB.) y es la que normalmente se utiliza en los productos comerciales. Pero ésta diferencia de rango dinámico es fundamental en aplicaciones tales como la detección de señales débiles en ambientes ruidosos.

Otro factor destacable desde el punto de vista del desarrollo de aplicaciones interactivas - o que dependen de procesos externos no predecibles - es la facilidad para manejar interrupciones. El DSP56002 da la posibilidad de controlar interrupciones provenientes de distintos periféricos: cada periférico tiene un lugar asignado en memoria donde espera la subrutina que atiende a la interrupción. Además, es posible cambiar las prioridades o inhibir completamente la interrupciones mientras se ejecuta una rutina. Como puede apreciarse es la plataforma ideal para desarrollar aplicaciones donde el tiempo está acotado y la eficiencia del código es fundamente.

Cuando hablamos de software de tiempo real nos referimos a aplicaciones que interactúan con el medio en forma dinámica y con tiempos estrictos para dar un resultado. Pensemos, por ejemplo, en el software que controla el piloto automático de un avión. Debe evaluar las variables que le llegan de los sensores y llevar a cabo una acción correctiva en fracciones de segundo. Por otro lado un simulador de vuelo que busca probar como reaccionaría el sistema expuesto a los mismos estimulos está exigido a dar un resultado, pero la diferencia radica en que tiene el tiempo que necesite para presentarlo (sólo importa el resultado).De este ejemplo queda claro cuán importante es que las rutinas no sobrepasen los tiempos máximos de ejecución en aplicaciones de tiempo real.

La implementación del decodificador Morse precisamente explota las características antes mencionadas buscando cumplir las rutinasen un tiempo dado.

Por la forma en que se estructuró la solución ninguna rutina puede demorar en completarse más que el tiempo que hay entre una muestra y otra. Ésto nos llevó a crear una estructura que hiciera simple la tarea de organizar la ejecución de las subrutinas que comprenden el programa. Varias de las rutinas utilizan muchas muestras antes de completarse y a su vez otras subrutinas dependen de eso resultados. Al mismo tiempo se deben atender interrupciones y procesos que no siguen una ejecución lineal. Atendiendo estos requerimientos implementamos un sistema de flags que al ser verificado por un gestor de procesos organiza el flujo del programa.

Cada subrutina tiene asignados dos (2) flags lógicos : uno que indica si el proceso está activo y otro que indica si ya está disponible el resultado. Cada rutina para poder ejecutarse debe tener encendido el

flag activo. A éste flag lo enciende la rutina que precedió a ésta como forma de habilitarla a correr. Al tener un resultado enciende el flag dato válido y apaga su flag de activo.

Esta forma de organizar el flujo probó ser muy útil a la hora de realizar el debugging, además de generar un código claro de leer.

4 CARACTERIZACIÓN DE LA SEÑAL

Las transmisiones de larga distancia (DX) están afectadas por perturbaciones asociadas a las bandas en las que se realizan las comunicaciones. Presentamos a continuación un cuadro con los distintos órdenes de frecuencias utilizados en las transmisiones deradio, y en particular de radioaficionados, transmitiendo Morse (CW).

Bandas de Frecuencia

Banda	Frecuencias	Longitud de onda
VLF	3 a 30 Khz	Miriamétricas (Very Low Frequency)
LF	30 a 300 Khz	Kilométricas (Low Frequency)
MF	300 a 3000 Khz	Hectométricas (Medium Frequency)
HF	3 a 30 Mhz	Decamétricas (High Frequency)
VHF	30 a 300 Mhz	Métricas (Very High Frequency)
UHF	300 a 3000 Mhz	Decimétricas (Ultra High Frequency)
SHF	3 a 30 Ghz	Centimétricas (Super High Frequency)
EHF	30 a 300 Ghz	Milimétricas (Extra High Frequency)
-	300 a 3000 Ghz	Decimilimétricas

Frecuencias Autorizadas para Radioaficionados

- Banda de 136 Khz, banda de experimentación autorizada
- Banda de 160 metros (1.810 - 2.000 kHz)
- 1.800 - 1.838 CW y RTTY
- 1.830 - 1.840 CW DX intercontinental
- 1.840 - 1.850 LSB, CW
- 1.840 - 2.000 LSB, SSTV (sólo Regiones 2 y 3)
- Banda de 80 metros (3.500 - 3.800 kHz)
- 3.500 - 3.510 sólo CW (ventana DX intercontinental)
- 3.510 - 3.560 sólo CW (recomendado para concursos)
- 3.560 - 3.580 sólo CW
- - 3.590 RTTY, AMTOR, PACTOR, CW
- 3.590 - 3.600 RTTY, Radiopaquete (packet), CW
- 3.600 - 3.620 LSB, Radiopaquete, CW
- 3.620 - 3.650 LSB (recomendado para concursos), CW
- 3.650 - 3.730 LSB, CW
- 3.730 - 3.740 SSTV, Fax, Fonía, CW
- 3.740 - 3.790 LSB (recomendado para concursos), CW
- 3.790 - 3.800 LSB (ventana DX intercontinental), CW
- 3.800 - 3.900 LSB (sólo en Región li, ver nota)
- Banda de 40 metros (7.000 a 7.100 kHz en Región 1)
- 7.000 - 7.035 Sólo CW
- 7.035 - 7.040 RTTY, AMTOR, PACTOR, Packet, CW
- 7.040 - 7.045 RTTY, AMTOR, PACTOR, Packet, SSTV, Fax, LSB, CW
- 7.045 - 7.100 LSB, CW
- Banda de 30 metros (1.100 a 10.150 kHz)

- 10.100 - 10.140 Sólo CW
- 10.140 - 10.150 RTTY, AMTOR, PACTOR, Packet, CW, (LSB sólo en emergencias)
- Banda de 20 metros (14.000 a 14.350 kHz)
- 14.000 - 14.060 Sólo CW (recomendado para concursos)
- 14.060 - 14.065 Sólo CW (segmento recom. para QRP)
- 14.065 - 14.070 Sólo CW
- 14.070 - 14.089 RTTY, AMTOR, PACTOR, CW
- 14.089 - 14.099 Radiopaquete, CW
- 14.099 - 14.101 Reservado para balizas mundiales
- 14.101 - 14.112 Radiopaquete, USB, CW
- 14.112 - 14.125 USB, CW
- 14.125 - 14.225 USB (recomendada. para concursos), CW
- 14.225 - 14.235 SSTV, Fax, USB, CW
- 14.235 - 14.300 USB (recomendada para concursos), CW
- 14.300 - 14.350 USB, CW
- Banda de 17 metros (18.068 - 18.168 kHz)
- 18.068 - 18.100 Sólo CW
- 18.100 - 18.109 RTTY, AMTOR, PACTOR, Radiopaquete,cw
- 18.109 - 18.111 Reservado para balizas mundiales
- 18.111 - 18.168 USB, CW
- Banda de 15 metros (21.000 a 21.450 kHz)
- 21.000 - 21.080 Sólo CW
- 21.080 - 21.100 RTTY, AMTOR, PACTOR, Radiopaquete, cw
- 21.100 - 21.149 Sólo CW
- 21.149 - 21.151 Reservado para balizas mundiales
- 21.151 - 21.335 USB, CW
- 21.335 - 21.345 SSTV, Fax, USB, CW
- 21.345 - 21.450 USB, CW
- Banda de 12 metros (24.890 a 24.990 kHz)
- 24.890 - 24.920 Sólo CW
- 24.920 - 24.929 RTTY, AMTOR, PACTOR, Radbpaquete, cw
- 24.929 - 24.931 Reservado para balizas mundiales
- 24.931 - 24.990 USB, CW
- Banda de 11 metros (26960 a 27410 Khz) Solo autorizados con licencia ECB
- 26960 - 27410 AM,FM,SSB Banda Cebeista.
- 27410 - 28000 Solo experimentación SSB
- Banda de 10 metros (28.000 a 29.700 kHz)
- 28.000 - 28.050 Sólo CW
- 28.050 - 28.120 RTTY, AMTOR, PACTOR, CW
- 28.120 - 28.150 Radiopaquete, RTTY, AMTOR, PACTOR, cw
- 28.150 - 28.190 Sólo CW
- 28.190 - 28.255 Reservado para balizas mundiales
- 28.255 - 28.675 USB, CW
- 28.675 - 28.685 SSTV, Fax, USB, CW
- 28.685 - 29.200 USB, CW
- 29.200 - 29.300 Radiopaquete-NBFM, SSB, FM, CW
- 29.300 - 29.550 Reservado para bajada Satélites

- 29.550 - 29.700 Fonía, Repetidores, CW
- Banda de 6 metros (50 Mhz)
- 50.000 - 50200 AM-SSB
- Plan de Banda de 2 metros (144 - 146 MHz)
- 144,000 - 144,150 CW
- 144,000 - 144,035 Rebote Lunar
- 144,050 Llamada CW (DX)
- 144,100 Llamada CW Random MS (Meteor-Scatter)
- 144,140 - 144,150 CW FAI (Propagación. esporádica)
- 144,150 - 144,500 SSB, CW
- 144,150 - 144,160 SSB FAI (Propagación esporádica)
- 144,195 - 144,205 SSB Random MS
- 144,300 Llamada SSB (DX)
- 144,395 - 144,405 SSB Random MS
- 144,500 - 144,845 Todos los modos
- 144,500 Llamada SSTV
- 144,625 - 144,675 Tráfico digital (Radiopaquete)
- 144,700 Llamada Fax
- 144,750 Tráfico de servicio TVA
- 144,845 - 144,990 Reservado a balizas
- 145,000 - 145,175 Entrada repetidores RO-R7 (8an.dúplex)
- 145,200 - 145,575 FM S8-S23 (16 canales símplex)
- 145,300 Tráfico digital local (RTTY)
- 145,500 Llamada móviles
- 145,600 - 145,775 Salida repetidores RO-R7
- 145,800 - 146,000 Servicio de Satélites

Notas:

- El "tráfico digital" comprende las modalidades de RTTY (Baudot, ASCII, AMTOR, PACTOR, etc.).
- "MS" significa «Meteor Scatter. o dispersión por meteoritos.
- El "Tráfico de servicio TVA" engloba sólo el tráfico auxiliar en FM relativo a la actividad reseñada.
- En 144,300 se encuentran asimismo comunicaciones digitales locales en radiopaquete (Packet Radio).

- Banda de 70 cm (430 - 440 MHz
- 430,000 - 432,000 Plan de banda nacional
- 430,025 - 430,375 Salida de repetidores (FRUI- FRU15)
- 430,400 - 430,575 Enlaces digitales
- 430,600 - 430,925 Repetidores digitales (R52-R65)
- 430,950 - 431,025 Canales multimodo (R66-R69)
- 431,050 - 431,825 Entrada repetidores (R70-R101)
- 432,000 - 432,800 Segmento de DX, banda estrecha
- 432,000 - 432,150 CW
- 432,000 - 432,025 Rebote lunar
- 432,050 Llamada CW
- 432,150 - 432,500 SSB, CW
- 432,200 Llamada SSB

- 432,350 Llamada-respuesta SSB
- 432,500 SSTV (banda estrecha)
- 432,500 - 432,600 Entrada transpondedores lineales
- 432,600 RTTY (FSK/PSK)
- 432,600 - 432,800 Salida transpondedores lineales
- 432,700 Fax (FSK)
- 432,800 - 432,990 Balizas
- 433,000 - 433,375 Entrada repetidores (RUO-RU15)
- 433,420 - 434,420 Asignación libre (Norma UN-30)
- 433,400 - 434,575 Canales símplex (SU16-SU23)
- 433,400 SSTV(FM/AFSK)
- 433,500 Llamada móvil FM
- 433,600 RTTY FM
- 433,625 - 433,775 Comunicaciones digitales
- 433,700 Fax (FM/AFSK)
- 434,000 - 440,000 TV Aficionados
- 434,600 - 434,975 Salida de repetidores (RUO-RU15)
- 435,000 - 440,000 Plan de banda nacional
- 438,025 - 438,175 Comunicaciones digitales
- 438,200 - 438,525 Repetidores digitales (R52-R65)
- 438,550 - 438,625 Canales multimodo (R66-R69)
- 438,650 - 439,425 Salida repetidores (R70-R101)
- 439,800 - 439,975 Enlaces de comunicaciones digitales

Notas:

- En las transmisiones de TVA la portadora de vídeo debería estar por debajo de 434,500 o por encima de 438,500 MHz.
- En caso de interferencias entre estaciones de TVA y el servicio de satélites, éste tiene prioridad.
- En concursos y aperturas de banda, el tráfico local en banda estrecha deberá ceñirse al segmento comprendido entre 432,500 y 432,800 MHz.
- Región 1 comprende Euroasia y Africa.
- Región 2 comprende el continente americano.
- Región 3 comprende parte del sur de Asia y el resto del mundo.

Como se puede advertir el uso del CW está extendido en toda las frecuencias. Cada banda es afectada o es inmune a una serie de agentes que inevitablemente perturban las señales en el camino desde el emisor al receptor. El origen de éstas perturbaciones puede ser debido a factores tales como:

1. Atmosféricos (tormentas eléctricas, manchas solares, auroras boreales, meteoritos).
2. Humanos (motores, líneas de alta tensión, transmisores mal calibrados, etc.).
3. De equipamiento (antenas, conectores, fuentes, amplificadores, etc.).

Todas estas posibles fuentes de ruido se suman a la señal original produciendo una nueva señal que trataremos de caracterizar en función de sus efectos sobre la forma de onda y comportamiento en frecuencia.

Analizando muestras de señales recibidas en distintos momentos- y procedentes de transmisiones de bandas también diferentes- nos encontramos con una serie de distorsiones que podemos agrupar de la siguiente manera:

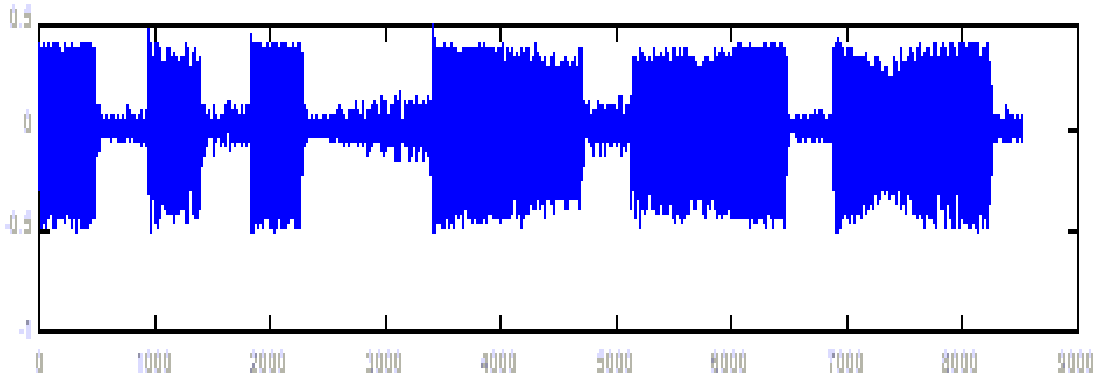
1. Ruido blanco : ruido con densidad de potencia uniforme en todo el espectro.
2. Ruido coloreado: ruido con densidad de potencia uniforme en una banda.
3. Ruido impulsivo: perturbación de muy corta duración y gran amplitud.
4. Distorsión de amplitud: modificación en la forma del pulso.
5. Deriva de frecuencia: variación de la frecuencia del tono.

El último punto, es causado por : inestabilidades en el oscilador del transmisor, fading multicamino o por el movimiento relativo del emisor y el receptor (Efecto Doppler) . Es usual encontrar éste problema por ejemplo en vehículos en movimiento o cuando se usa la luna como punto de rebote de las señales de CW (Earth-Moon-Earth). En las pruebas realizadas no se encontraron variaciones de frecuencias apreciables en una misma transmisión. Lo que si se encontró fueron señales de Morse muy próximas entre sí pero de potencias distintas, por lo que se fijó como parámetro de diseño el centrado del notch a una frecuencia fija (para no incurrir en comportamientos erráticos de pasar de una transmisión a otra si se producían silencios prolongados).

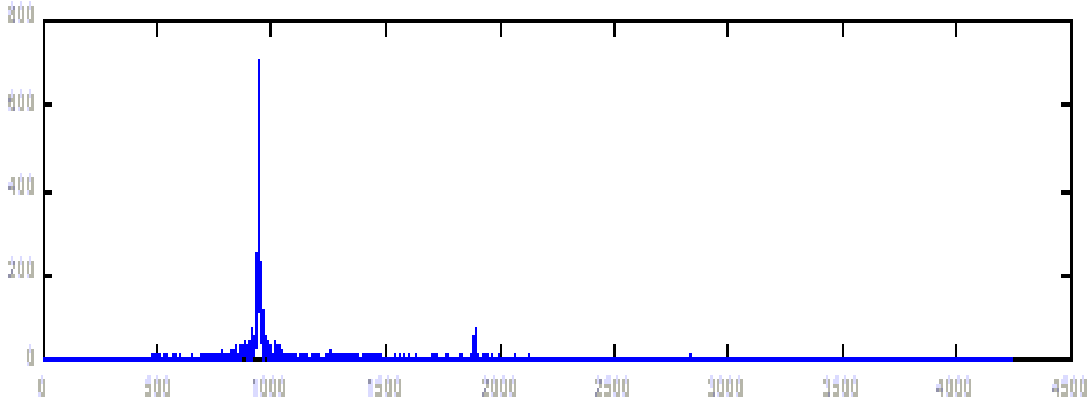
A modo de ejemplo, veamos muestras representativas de señales:

a- Señal con poca distorsión.

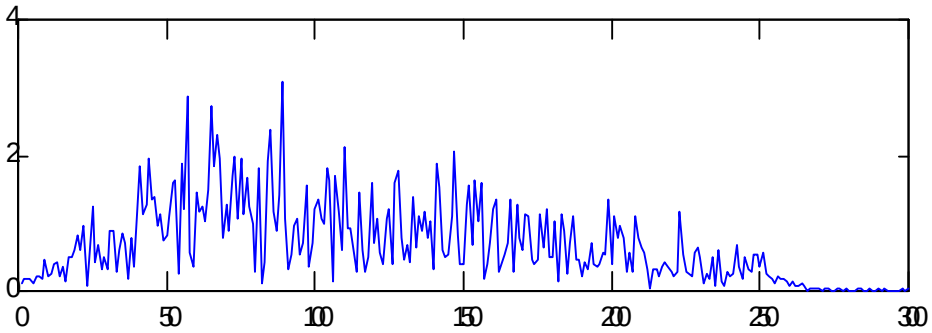
Se observa en el espacio del tiempo claramente el Morse.



En lo que refiere a la frecuencia se aprecia una buena relación señal-ruido.

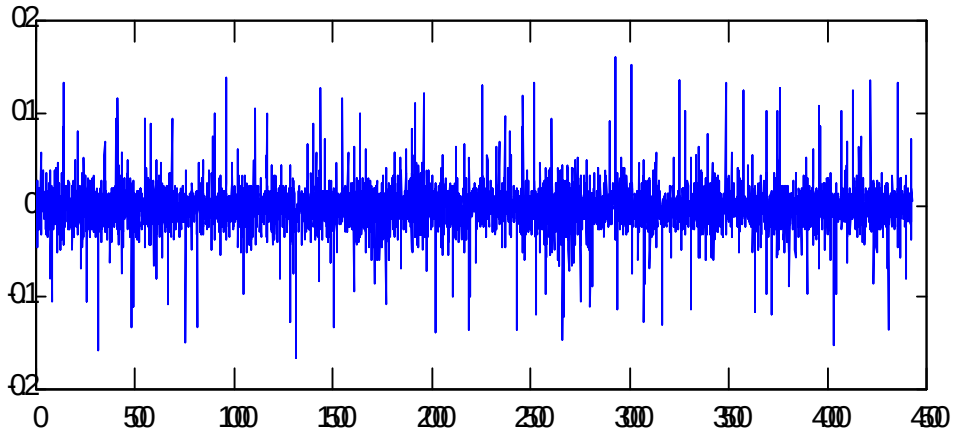


Espectro de la señal entre silencios de Morse cuando no hay portadora. Se observa una densidad de potencia más o menos uniforme en todo el espectro.

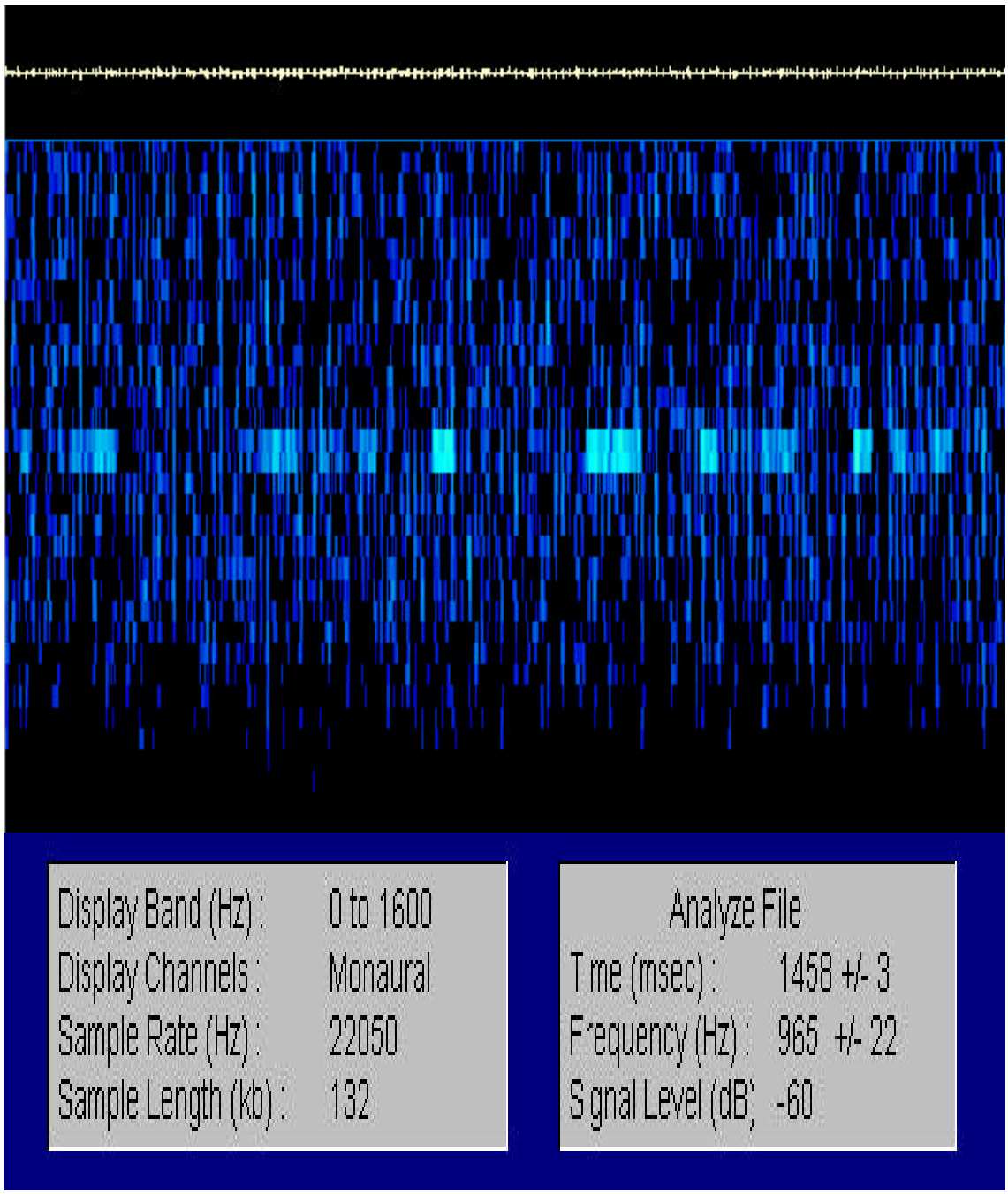


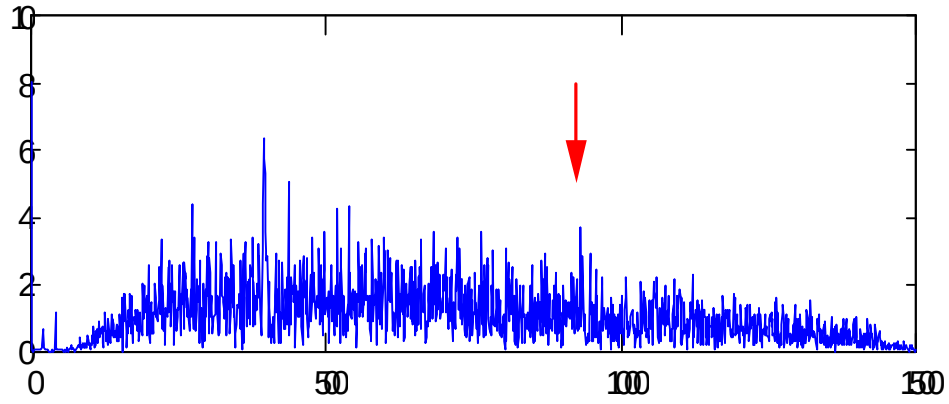
b- Señal con fuerte presencia de ruido:

Esta señal ya no es posible de interpretar en el espacio del tiempo debido a la baja relación señal a ruido. Es apreciable la fuerte componente de ruido impulsivo que no incidirá mucho sobre la detección final si el largo de los pulsos es mucho mayor que la duración del pico impulsivo.

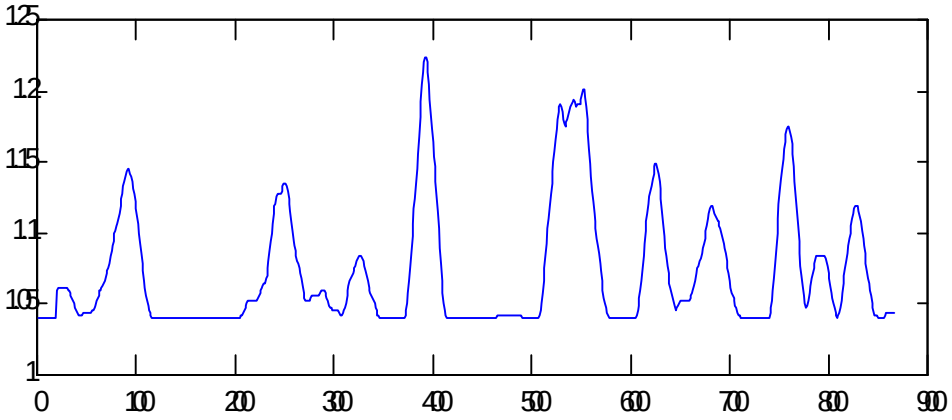


El siguiente espectrograma revela la presencia de CW, un tono centrado a 965Hz. Sin embargo, la potencia del tono no es constante y se desvanece rápidamente evidenciando distorsión de amplitud.





Cuando realizamos una FFT para un número dado de muestras se observa que la componente del tono no resalta, sino que se pierde en el ruido. Por ésto se utilizó un análisis estadístico en el tiempo de la FFT de manera que, mediante sumas ponderadas, se decanta la frecuencia persistente del tono de CW.



Al aplicar un filtro notch en la frecuencia de la portadora de CW verificamos lo antes expuesto; hay una variación importante de la forma y potencia de los pulsos.

5 MODELADO DEL DECODIFICADOR

5.1 Introducción

Para el modelado del decodificar pensamos que era necesario dividir el problema en capas y a su vez cada capa en bloques funcionales.

Las capas resuelven problemas puntuales a partir de interfases bien definidas.

Cada capa actuará en forma independiente, lo que pensamos es fundamental para acotar la complejidad del problema.

Los bloques funcionales especializados comparten información y recursos entre bloques integrantes de la misma capa.

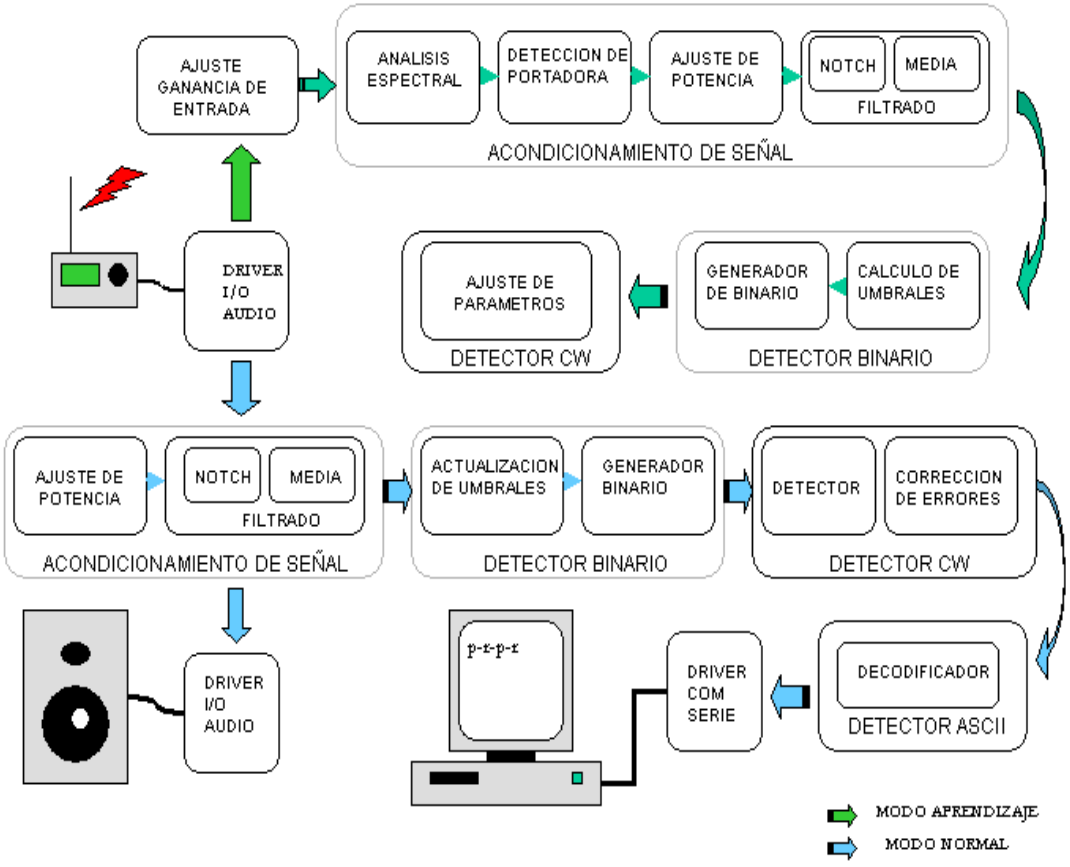
De esta forma se reduce el tiempo de prueba del software ya que se construyen pequeñas estructuras de código que luego se van combinando hasta formar la capa.

A través de las capas fluye el mensaje Morse, que va cambiando su estructura de ser una señal de audio a la entrada del decodificador a ser una cadena de caracteres ASCII a la salida.

A este flujo es al que nos referiremos como señales físicas, en contraposición con las lógicas – flags - que estarán asociadas con la estructura de gestión de procesos.

El decodificador por ser autoajustable debe tener un proceso de aprendizaje durante el cual el detector ajusta sus parámetros para luego pasar al modo normal donde se realiza la decodificación.

5.2 Diagrama de Flujo de Señales Físicas



5.3 Gestión de Procesos

Una vez que se realiza la inicialización del DSP queda definida la forma de operación de éste y se pasa a la etapa de funcionamiento normal.

Bajo el nombre “gestión de procesos” englobamos a la estructura que hace posible el funcionamiento armonioso de las distintas rutinas que integran el decodificador. La base de la rutina de gestión de procesos es la utilizacion de flags , para lo cual definimos: flag módulo activo, flag de dato válido y flag de error. Cada proceso importante cuenta con estos tres flags y todos los flags de las distintas capas estan ordenados en una variable que oficia de registro (por lo que basta consultar esta variable para saber el estado del programa).

Con el Flag de módulo activo indicamos si está completo o no un proceso. Por ejemplo, el cálculo de la FFT necesita de muchas muestras para completarse antes de sacar un dato válido. Mientras tanto se ejecuta el módulo de filtrado y acondicionamiento de señal para cada

muestra. Por lo que el cálculo de la FFTseguirá en segundo plano hasta tener datos para procesar. Como se puede ver el sistema de flags permite manejar procesos que corren a distintas velocidades en forma transparente para el programador.

El dato válido nos indica que el dato contenido en las variables de salida de la rutina ya se pueden leer y corresponden a datos correctos. En algunos casos, los resultados son descartados por lo que la rutina termina poniendo el flag activo off y datos validos también off. En otros casos, se requiere de la corrección del error por lo que se activa el flag de error y esto hace que el flujo del programa se desvie para atender casos particulares de error.

```
;act_ ON -> tarea activa
;act_ OFF y cs_ON->datos prontos nuevos y tarea completa
;
;Control de activacion de tareas 1=activa   0=no activa
act_boot      equ     0         ;booteo en proceso
act_fft       equ     1         ;calculo fft,coef de filtro noch
act_histo     equ     2         ;modulo histograma
act_vol       equ     3         ;modulo ajuste inicial de volumen in
act_noch      equ     4         ;modulo noch
act_bin       equ     5         ;modulo detector binario
act_mensj     equ     6         ;despligue de mensaje en proceso
act_cw        equ     7         ;modulo detector cw
act_asci      equ     8         ;modulo det ascii
act_sci       equ     9         ;modulo sci
act_reboot    equ    10         ;rebooteo en proceso

;Tarea completa datos prontos,clear to send
cs_fft        equ    12
cs_histo      equ    13
cs_bin        equ    14
cs_cw         equ    15
cs_vol        equ    16          ;adjuste de ganacia realizado
cs_tger       equ    17          ;modulo triger
cs_ascii      equ    18          ;modulo det ascii
err_tger      equ    20          ;error en modulo triger
err_vol       equ    21          ;error en modulo triger
```

Como complemento de los flags se encuentraN los comparadores, encargados de gestionar el flujo del programa en funció n del estado de los flags.

```
Loop_main

;=============================================

sys_admin
                    jset #act_reboot,y:flags,sys_reboot
                    jset #act_menu,y:flags,sys_menu
                    jclr #act_boot,y:flags,sys_norm
```

```
sys_boot ; sistema booteando
=====================================================

                jset #cs_tger,y:flags,fin_boot
                jset #act_histo,y:flags,histo_boot
                jset #cs_histo,y:flags,trigr_boot
                jset #act_vol,y:flags,vol_boot
                jset #act_fft,y:flags,fft_boot
                jset #act_buffer,y:flags,buffers_fill
```

Al combinar los flags y el conjunto de comparadores logramos centralizar el flujo del programa, lo que presenta numerosos ventajas como ser: mejorar de la trazabilidad , simplificar el agregado de rutinas, simplificar las pruebas del conjunto, agregar claridad al código.

Es claro que este sistema de gestión, se apoya en la modularidad del programa e independencia de las rutinas que lo componen.

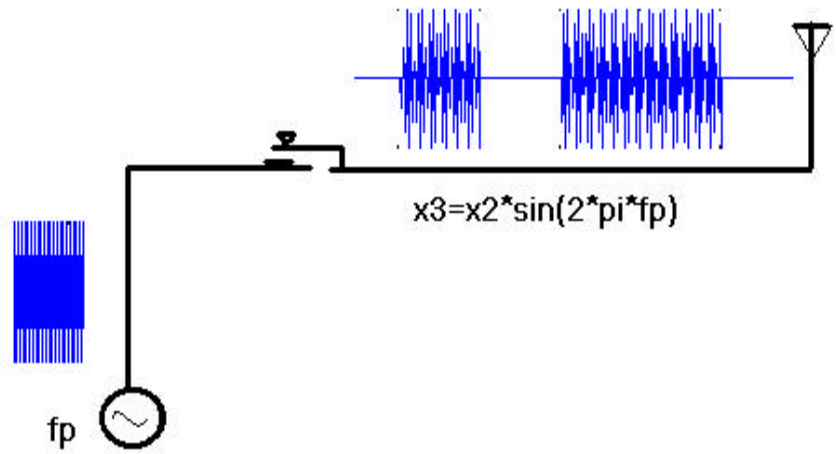
5.4 Modelo de Capas

5.4.1 CAPA 1 – Acondicionamiento de Señal

5.4.1.1 Detección de Portadora

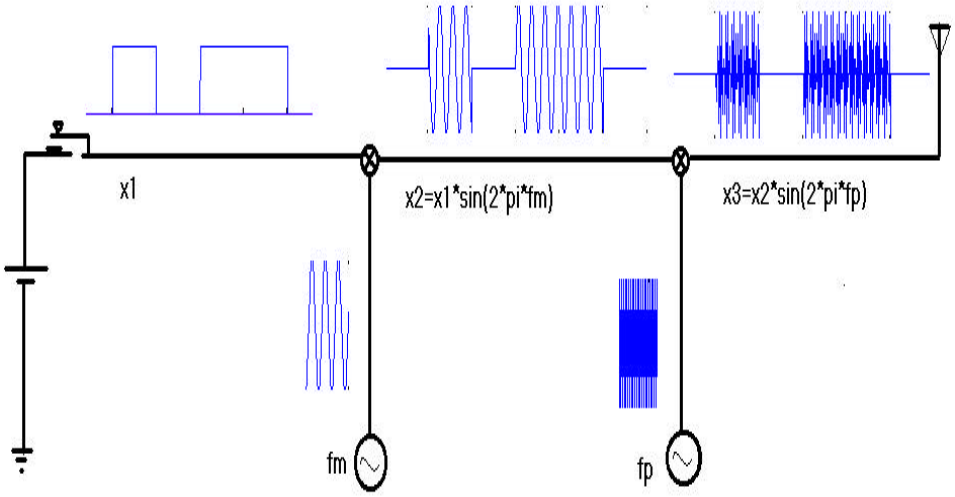
Transmisor

Antes de entrar en los detalles de cómo funciona la detección de portador vamos a realizar una breve reseña de cómo se genera la señal que pretendemos detectar. En el caso más simple se modula la portadora directamente según el siguiente modelo:



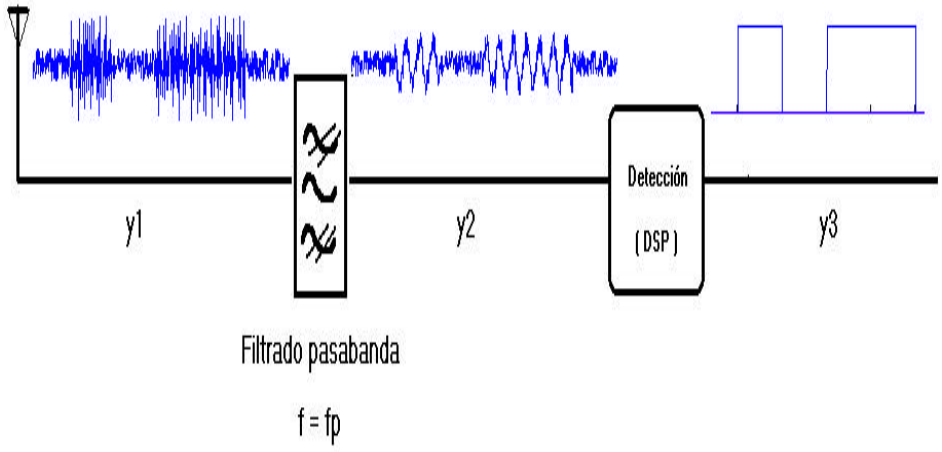
Existe una limitación física al trabajar directamente sobre el transmisor, lo que hace que, en el promedio de los equipos, no se superen las 100 WPM. Por ésto surgieron métodos

alternativos, que modulan un tono y luego le aplican la portadora, como ser transmisiones en SSB, cuyo modelo es el siguiente:



Pasemos ahora a describir las etapas que componen el receptor de radio:

Receptor



La señal recibida **y1** es filtrada por el receptor de RF, dando lugar a una señal **y2** de audio -conteniendo el Morse original transmitido más el ruido introducido por el canal-. Básicamente el receptor se sintoniza para escuchar una transmisión a una frecuencia dada, variando la posición del filtro pasabanda .

La señal **y2** usualmente pasa a la etapa de salida de audio y de ésta a los parlantes. El decodificador Morse toma su

entrada de la etapa de salida de audio, por lo que utiliza la misma información que recibe el operador.

Con el nombre de "Detección de portadora" nos referimos al bloque encargado de sintonizar el tono y reconstruir la señal original transmitida

Analizaremos ahora cómo se implementó en el decodificador esta etapa:

El DSP recibe los datos del CODEC el cual se ha programado para tomar muestras a 8 Khz. y 16 bits de precisión por su entrada MIC. Este decisión resulta de:

1. Asumir que la señal de audio que contiene el Morse tiene un ancho de banda máximo de 4Khz. Según la opinión de varios operadores es entre 800 Hz y 2Khz que resulta agradable escuchar el Morse.

2. El hecho que el uso de frecuencias de muestreo superiores (oversampling) tiene la ventaja de ofrecer más muestras sobre las cuales operar, lo que - pensado en el espacio del tiempo - implica menos incertidumbre sobre el estado de la señal en un momento dado. Sin embargo, para nuestra solución de tiempo real, el tiempo entre muestras es vital debido a que todo el procesamiento se realiza en este intervalo, por lo que nos interesa que sea lo más largo posible.

3. Tomar el máximo de precisión que maneja el CODEC para minimizar el ruido de cuantización y tener más cifras significativas para poder operar.

Es posible encontrar el valor de la frecuencia **fm** haciendo un análisis espectral de la señal **y2**. Como estamos trabajando en tiempo discreto debemos apelar al cálculo de la FFT (Fast Fourier Transform). La FFT se implementó con un algoritmo - optimizado para operar en el DSP - llamado Radix 2 Decimation in Time In-Place Fast Fourier Transform. Como es sabido, la resolución (Rs) del análisis espectral usando la FFT depende de la frecuencia de muestreo (Fs) y del número de muestras usadas para su calculo (m) según la siguiente relación :

$$Rs = Fs/m$$

Claramente un mayor número de muestras implica una mayor resolución y mayores tiempos de cálculo. Por tanto llegamos a un valor de compromiso de m=256 que nos dá una resolución de 31,25 Hz. Más adelante veremos que este valor es compatible con el ancho de banda de notch.

Como forma de resaltar la presencia del tono de CW que buscamos detectar, se realiza durante ésta etapa la ponderación en el tiempo de varios espectrogramas tomados a intervalos regulares. Este bloque busca minimizar errores al hora de decidir la ubicación del tono, dándole más importancia a la historia de la señal que a perturbaciones puntuales.

Luego de analizarse la señal durante unos segundos, se elige la frecuencia donde la potencia es máxima, pasando a ser la candidata a **fm**. Se centra el filtro notch a la frecuencia **fm** con el cual se filtrará la señal **y2** logrando una primer aproximación a **x2**. Los coeficientes del filtro se recalculan según el valor de **fm** en forma automática.

5.4.1.2 Filtrado de Señal

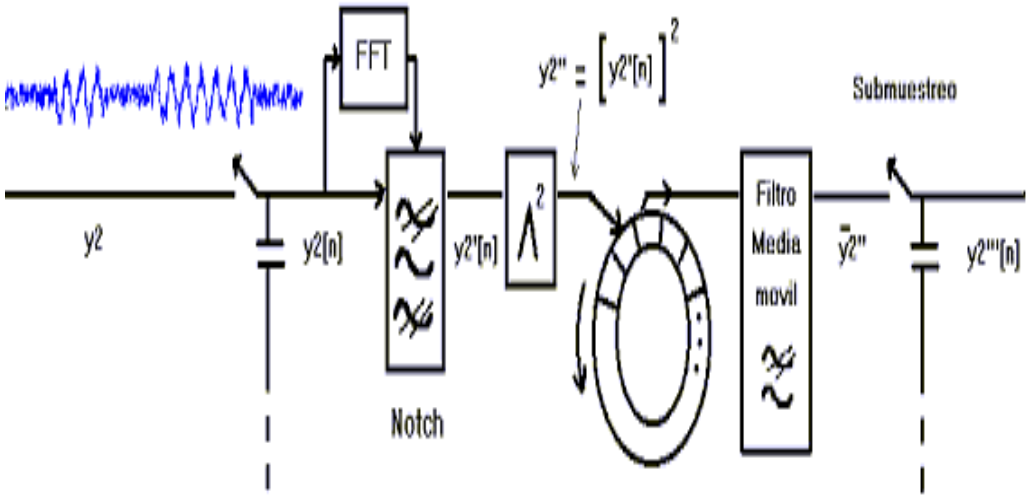
Como mencionamos en la sección anterior la señal y2 de entrada al DSP debe ser filtrada con la intención de obtener una señal lo más parecida a x2. Para ésto se ha implementado en el DSP un filtrado en dos bloques:

1. Un filtro notch.
2. Un algoritmo basado en la potencia media cuadrática y un filtro de media móvil.

La meta de este módulo es recoger la mayor cantidad de información útil, eliminando el ruido y artefactos de la señal de entrada, de forma de minimizar los errores transferidos a capas superiores.

La descripción de cómo se llegó a esta solución se detallará en el capitulo Ensayos.

Diagrama de Bloques del filtro:



5.4.1.3 Descripción del notch

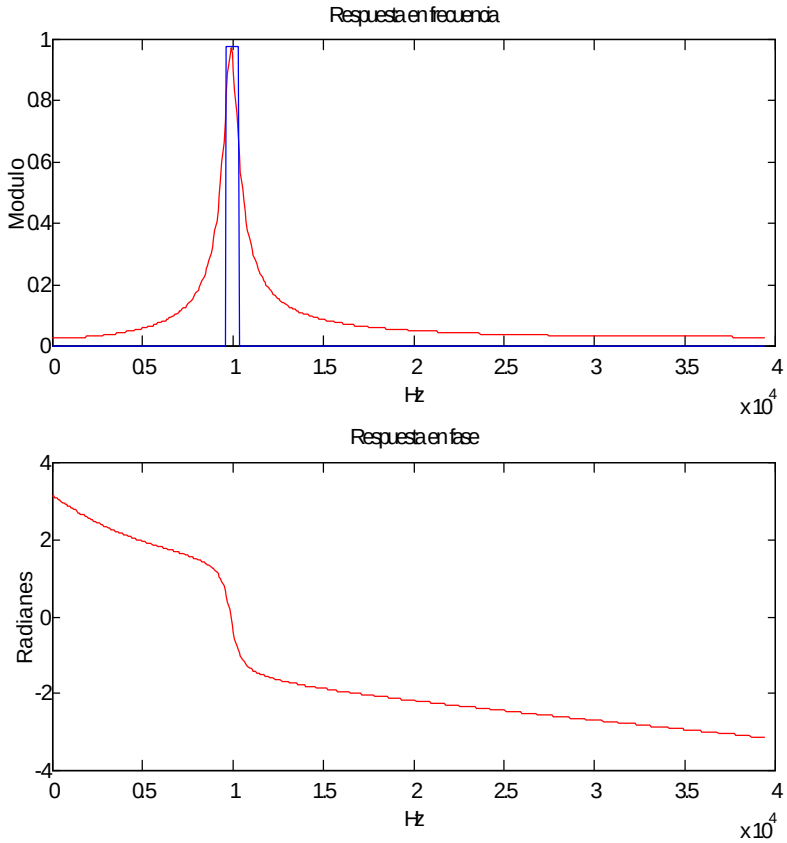
La elección del filtro juega un papel importantísimo en el desempeño final del decodificador, por lo que mereció un estudio detallado y una revisión contínua de su desempeño durante el desarrollo del proyecto.

La premisa de partida necesaria, para cualquier implementación de filtro que quisiéramos realizar, debía contemplar que el filtro fuera lo suficientemente angosto como para filtrar el ruido que rodea al tono de CW, y lo necesariamente ancho como para dejar pasar Morse a un frecuencia razonable de transmisión. Con esto se vuelven a plantear dos metas contrapuestas:

1- Si el filtro es muy ancho, se agregan al tono en el que está centrado el notch componentes de frecuencias cercanas que distorsionan la señal (en otras palabras suman ruido), por lo que parecería lógico lograr un filtro lo más angosto posible.

2- La onda cuadrada que modula el tono tiene componentes de baja frecuencias que van aumentando cuanto más rápido el operador transmite. Esto se refleja en que si el filtro es muy angosto perderíamos el Morse que queremos detectar. Otra forma de verlo es pensar que el filtro tiene que ser lo suficientemente rápido como para seguir las variaciones impuestas por el operador.

El filtro notch utilizado para el filtrado de la señal es un IIR de segundo orden de 70 Hz de ancho de banda, que es suficiente para copiar correctamente un operador que transmita hasta hasta 25 WPM.



Respuesta en frecuencia y fase del filtro notch centrado a 1 Khz, en rojo. Respuesta de filtro notch ideal del mismo ancho de banda, en azul.

Estudiemos brevemente cómo surge la expresión para este filtro. Comencemos expresándolo en transformada Z:

$$Y = \frac{(z^{-1} - a.e^{jw0}) \,.\, (z^{-1} - a.\, e^{-jw0})}{(z^{-1} - b.e^{jw0}) \,.\, (z^{-1} - b.\, e^{-jw0})} \;.\, X \;-\; 1$$

a y b reales , tales que a < b próximos a 1.

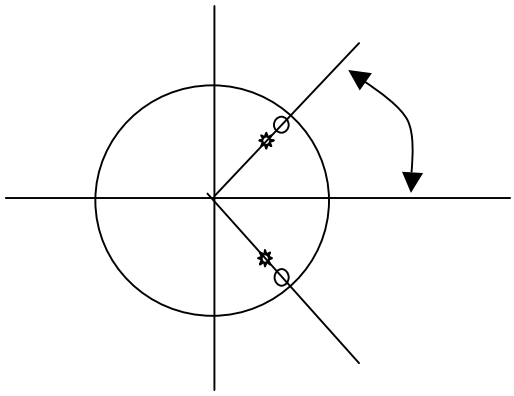
$$w0 = \frac{2.pi \,.\, ft}{Fs}$$

ft = frecuencia del tono de CW
Fs = frecuencia de muestreo

Observamos que si a > b entonces se obtiene un bloqueador de banda en vez de un pasa banda.

Los polos y los ceros son dobles conjugados, ubicados a la frecuencia donde se requiere el máximo de ganancia y de módulo cercano a 1. Su ubicación respecto del círculo unidad se puede apreciar en la figura:

Cero ○
Polo ✡



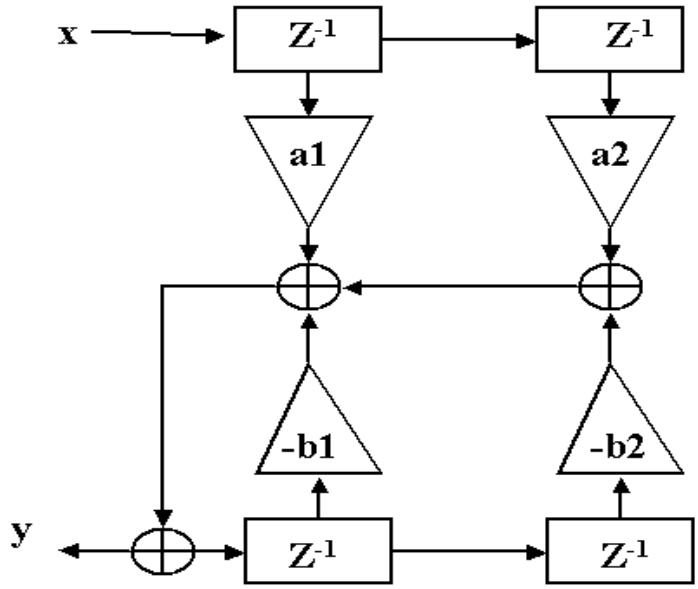
Llevado a la forma canónica se obtiene:

$$H(z)= \frac{-2 \cdot (a-b) \cdot \cos(w0) \cdot z^{-1} + (a^2-b^2) \cdot z^{-2}}{1 - 2 \cdot b \cdot \cos(w0) \cdot z^{-1} + b^2 \cdot z^{-2}}$$

Los coeficientes elegidos para cumplir los requerimientos son:

$$a = 0.94 \quad y \quad b = 0.97$$

De la fórmula anterior se observa que todos los datos son conocidos, excepto w0 que surge del análisis de la señal de entrada, por lo que los términos en z^{-1} deberán ser calculados en el DSP. El resto pueden ser guardados como constantes.



La forma canónica se puede expresar como muestra el gráfico anterior. Esta manera de representar el filtro ayudará a comprender la rutina que implementa el DSP.

Como es sabido Z^{-1} representa un retardo de una muestra, por lo que es necesario tener un buffer circular para guardar las últimas dos (2) muestras de la entrada y la salida. La utilización de punteros, combinada con esta forma de direccionamiento, simplifica la implemenntacíon del filtro digital.

Los datos se encuentran ordenados y son procesados de forma de minimizar los movimientos de memoria. Para utilizar la capacidad de procesamiento paraleb colocamos los coeficientes en la memoria **X** y los buffers de datos -tanto de la entrada como de la salida- en la memoria **Y.**

Módulo filtrado rutina notch:

```
;Noch ------------------------------------------------------------------------

iir_pasabanda

move #iirn_cc,r4              ;puntero a coef
move y:p_iirn,r1              ;puntero a buffer datos
move #3,m1
move #2,n1

move x:(r5),x0                ;x(n) ✍ x0
move x0,x1
move y:(r4)+,y0               ;a0

clr a                         ;cálculo el filtro
rep #5
macr +x0,y0,a x:(r1)+,x0 y:(r4)+,y0

rep #2                        ;por ajuste de escala coef y(n-2)*4
asl a

move a,x:(r1)-                ;y(n)
move x1,x:(r1)+n1             ;x(n)

move a,y:iir_noch             ;salida filtro
```

La rutina que ubica el tono conteniendo el Morse es la encargada de calcular el valor de cos(w0) mediante la búsqueda en una tabla de coseno (de 256 valores) guardada en memoria. Por esto, una vez encontrado el tono de la portadora, el notch queda centrado y fijo para su operación hasta ser reseteado. En la etapa de

modelado pensamos en construir un algoritmo que siguiera al tono en posibles desplazamientos. Durante el periodo de caracterización de la señal quedo claro que, salvo excepciones, las variaciones de frecuencias no eran asiduas y se daban en casos particulares. En la práctica, el hecho de que la frecuencia de centrado del notch permaneciera fija evitó que el decodificador pudiera mezclar transmisiones de Morse cercanas al producirse silencios prolongados.

5.4.1.4 Descripción de Filtrado por Media Móvil

Como se observa en el diagrama de bloques, luego de pasar la señal por un filtro notch se procede a un nuevo filtrado -para eliminar el ruido remanente en el Morse recibido-, y se elevan las muestras al cuadrado -para obtener un valor proporcional a la potencia cuadrática de la señal-. De aquí en adelante, la detección se realiza sobre ésta nueva señal, la que tiene como ventajas ser positiva y tener valor medio.

Ambos aspectos son importantes para la detección por dos razones:

1- Simplifica los cálculos en etapas que utilicen umbrales de comparación.

2- Al tener valor medio se puede comenzar a utilizar la historia de la señal.

Tiene, por otra parte, la desventaja de que se pierden cifras significativas por el echo de multiplicar dos números menores que 1 contando con un número fijo de cifras decimales.

El resultado de la operación anterior es guardado en un buffer circular de 256 muestras, sobre las que se aplica el filtro de media móvil. La respuesta en frecuencia de este filtro es un sinc cuyo lóbulo principal tiene ancho:

$$2 \times F_{muestreo}/256 = 2 \times 8000\ Hz/256 = 2 \times 31.25\ Hz$$

La señal que genera el operador, que modula la portadora, es de baja frecuencia, por lo que el filtro de media móvil elimina las perturbaciones no deseadas que el Notch deje pasar.

Por los parametros de diseño que elegimos (25 WPM máximo) la frecuencia de la señal que buscamos decodificar no superará los 23 Hz.

En este punto el submuestreo de la señal resulta una herramienta eficiente para disminuir la carga de procesamiento de las muestras que no serán utilizadas por los algoritmos que corren a continuación. Luego de varias pruebas, un submustreo de veinticinco (25) muestras- o lo que es equivalente a cambiar la frecuencia de muestreo de 8000 Hz a 320 Hz - fue suficiente para lograr un desempeño adecuado.

5.4.1.5 Ajuste de Ganancia

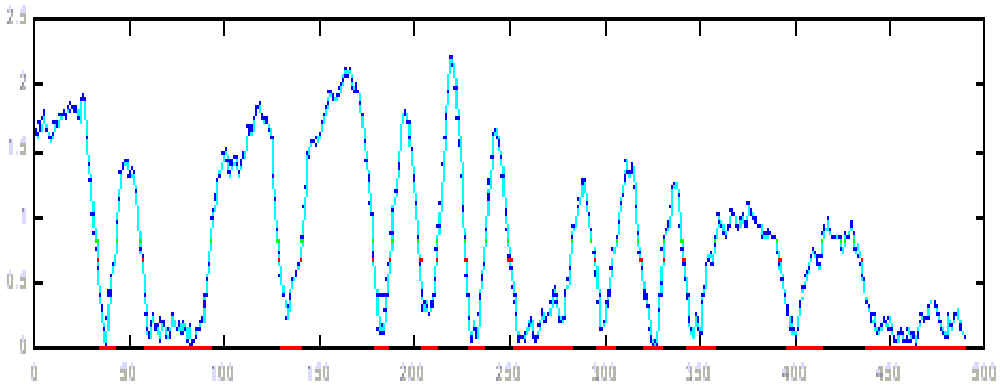
El módulo de ajuste de ganancia es un componente básico de todo sistema de comunicación. Su función es mantener la estabilidad de la señal recibida minimizando las variaciones de potencia.

En nuestro caso, utilizamios el ajuste de ganancia para mantener una señal dentro de un rango prefijado, donde se garantice el normal funcionamiento de todo el sistema. Para estos valores maximizamos el funcionamiento de los algoritmos de punto fijo que utilizamos en la detección. De no tomar esta precaución nos enfrentaríamos a errores erráticos muy difíciles de combatir, causados por:

1 Overflow al realizar los cálculos: el DSP solo puede manejar valores menores que la unidad.
2 Pérdida de la señal por falta de cifras significativas en alguna etapa de la detección.

Por lo expuesto, de no tomar en cuenta las limitaciones que nos impone el hardware, estaríamos introduciendo perturbaciones serias que comprometerían el funcionamiento confiable del decodificador.

En la práctica, el ajuste de ganancia actúa distinguiendo la componente de señal útil del ruido y escalando la señal, de tal forma de cumplir requerimientos mínimos y máximos de potencia.



Variaciones de potencia de una señal típica.

Del gráfico se desprenden las siguientes observaciones:

1 Se ve claramente que la potencia de los pulsos no es siempre la misma

2 A pesar de las variaciones de potencia es posible aún distinguir la presencia de la señal binaria

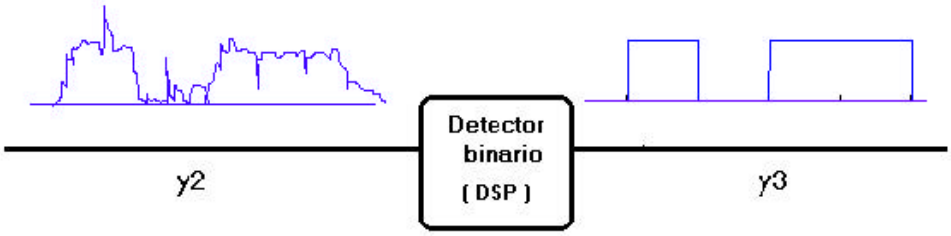
Lo que busca el ajuste de ganancia implementado es atacar el primer punto para mantener la amplitud de los pulsos dentro de un rango conocido, facilitando de esta forma la determinación de los umbrales de comparación.

5.4.2 CAPA 2- Detector Binario

Esta etapa se encarga de obtener una señal digital binaria pura partiendo de la señal procedente de la etapa anterior. Es en este punto, en que cambiamos la forma de interpretar las señales, dejamos de representarlas por extensión para pasar a describirlas por comprensión, es decir, que la señal no es más descripta punto a punto en el espacio del tiempo sino que se pasa a un nivel mayor de abstracción, describiéndose la señal por sus características. Una señal digital binaria se puede interpretar como pulsos de valor unidad separados por espacios o pulsos de amplitud nula. Es entonces que bastará para describir la señal indicar en cada caso :

1. Si se está en presencia de un pulso o un espacio (pulso nulo).

2. El largo del pulso.



En resúmen, esta etapa dá como resultado la descripción por extensión de la señal binara pura. Pasemos ahora a describir cómo logramos este cometido.

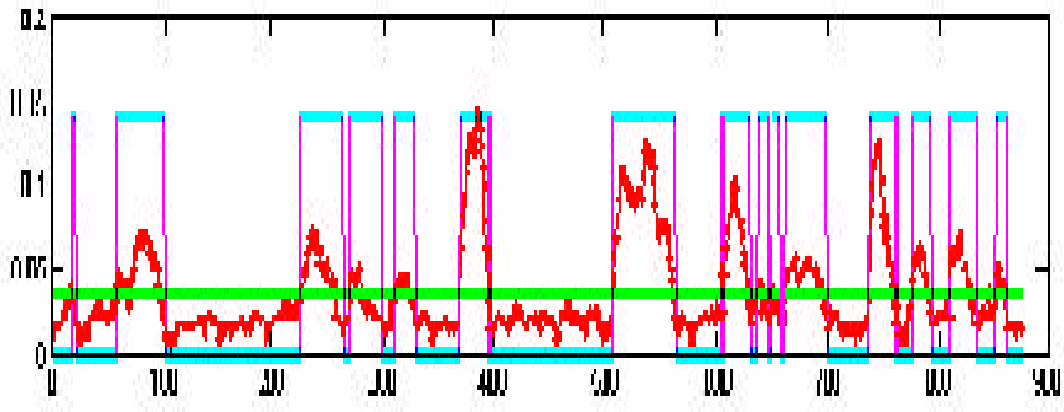
5.4.2.1 Umbrales de Detección

Es común en los sistemas análogo-digitales encontrarnos con el bloque encargado de reconstruir señales del tipo **y3** binarias a partir de señales del tipo de **y2. ,** por lo cual existen múltiples ejemplos y soluciones.

El problema que se nos planteó fue encontrar algún criterio con el cual decidir dónde comienza y dónde termina el pulso a detectar. Si bien el problema parece simple, se torna difícil al tener pulsos en **y2** de amplitud y forma variable.

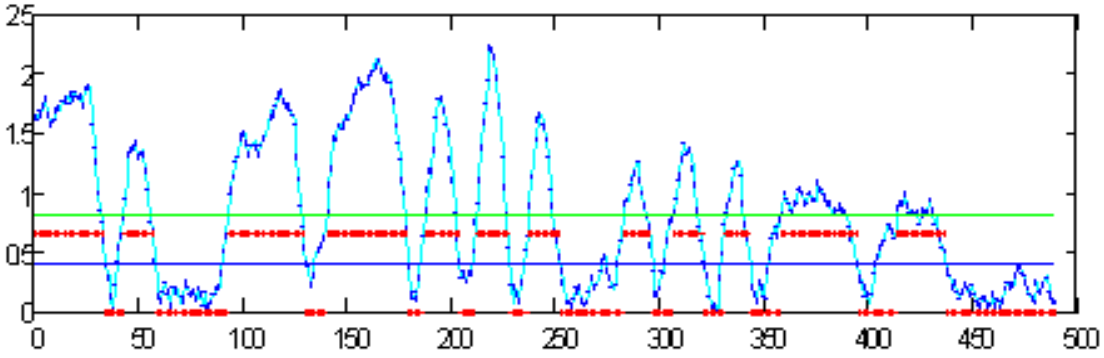
Buscando un detector binario se evolucionó desde un detector básico, y mediante simulaciones y ensayos se llegó a la solución definitiva. Veamos rápidamente el proceso de depuración que sufrió este módulo.

1 Detector binario de umbral unico fijo.
Cuando el valor de las muestras supere el umbral, la salida del detector pasa a valer '1', en caso contrario vale '0'. Mediante este procedimiento obtenemos una señal binaria pura. Si bien su implementación es simple su performance no es buena frente a señales ruidosas, principalmente las de carácter impulsivo y amplitud variable.



2 Detector binario de doble umbral fijo.

Este detector presenta histeresis, es decir la salida no pasa al estado anterior al atravesar un umbral .



Como se desprende del gráfic,o la señal binaria pasa a tener amplitud unidad cuando se superan ambos umbrales y amplitud, o cuando es menor a ambos umbrales. La ventaja de este método está en la zona neutra que se crea entre ambos umbrales. En esa franja las variaciones de la señal de entrada no causan variaciones en la salida, en otras palabras la salida sólo seguirá los comportamientos marcados de la señal de entrada y no las pequeñas perturbaciones. Por lo que éste método corrige una de las fallas del método anterior. Este sistema es conocido como comparador con ventana Schmitt Trigger.

3 Detector de doble umbral adaptativo.

Este detector tiene las mismas ventajas que el anterior, más el hecho de que neutraliza errores de detección debidos a las variaciones de la amplitud de la señalde entrada. Sin embargo, de la forma en que los umbrales siguen las variaciones de amplitud dependerá el éxito o fracaso del método. Por lo tanto vamos presentar los procedimientos experimentados:

a- Adaptación de umbrales por histograma.
Esta forma de adaptar los umbrales a la señal se basa en ubicar las muestras de la señal en un histograma de amplitudes. Logramos así obtener la distribución de la amplitud de las muestras en la que aparecerán dos picos:

1. Un Pico cercano a 0 formado por las muestras de poca amplitud (ruido de fondo).

2. Un Pico cercano a la unidad integrado por las muestras que integran las mesetas de los pulsos.

Como forma de minimizar el error de decisión, ubicamos en el punto medio entre ambos picos el umbral primario.

En caso de usar un solo umbral ésta sería la ubicación. En el caso de utilizar dos umbrales, a partir del umbral primario se crearían simétricamente los dos umbrales, a una amplitud que dependerá de la inmunidad al ruido que se quiera lograr. La adaptación se haría dinámicamente ajustando según la distribución de las muestras.

Esta solución tiene el problema de no comportase en forma aceptable para señales en las cuales :

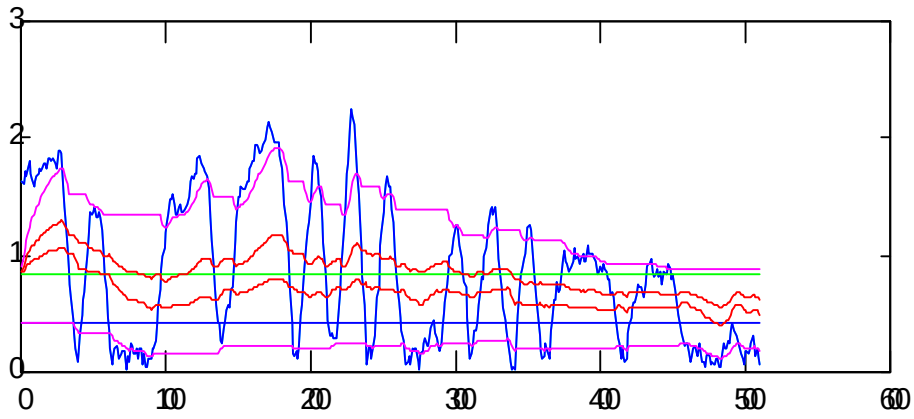
1 Hay variaciones de potencia importantes.
La falla es producida por la inercia del método al usar muchas muestras para tener un histograma medianamente confiable reacciona lentamente a los cambios.

2 Los flancos, tanto de subida como de bajada de los pulsos, no tienen pendientes pronunciadas. Esto hace que el histograma sea más "llano", siendo dificultoso encontrar los picos.

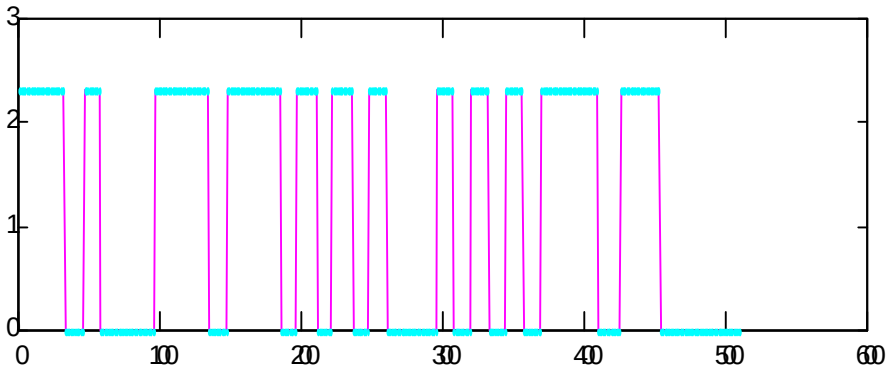
b- Adaptación de umbrales por potencia media.
Calculando la potencia media de la señal en forma continua ajustamos los umbrales de forma tal que el umbral primario se fije en el valor correspondiente a la potencia media. En esta variante usamos el comparador con ventana Schmitt Trigger con algunas modificaciones, tales como que la ventana de detección tiene sus umbrales variables en el

tiempo. La variación en el tiempo permite que la ventana de detección se encuentre siempre entre las máximas amplitudes de cada pulso y el nivel de ruido.

Con este fin es que se calculan los promedios de las muestras, que son mayores que la potencia media instantánea, y también las menores al 50% de la potencia media. Con el primer promedio, se determina la primer medida de las amplitudes máximas de los pulsos. De la misma manera el segundo promedio mide el nivel de ruido. Partiendo de éstas dos medidas se construye la ventana de detección adaptiva. Esta alternativa cumple con las exigencias planteadas por su rápida adaptación al cambio de volumen. Las siguientes imágenes ilustran los resultados obtenidos:

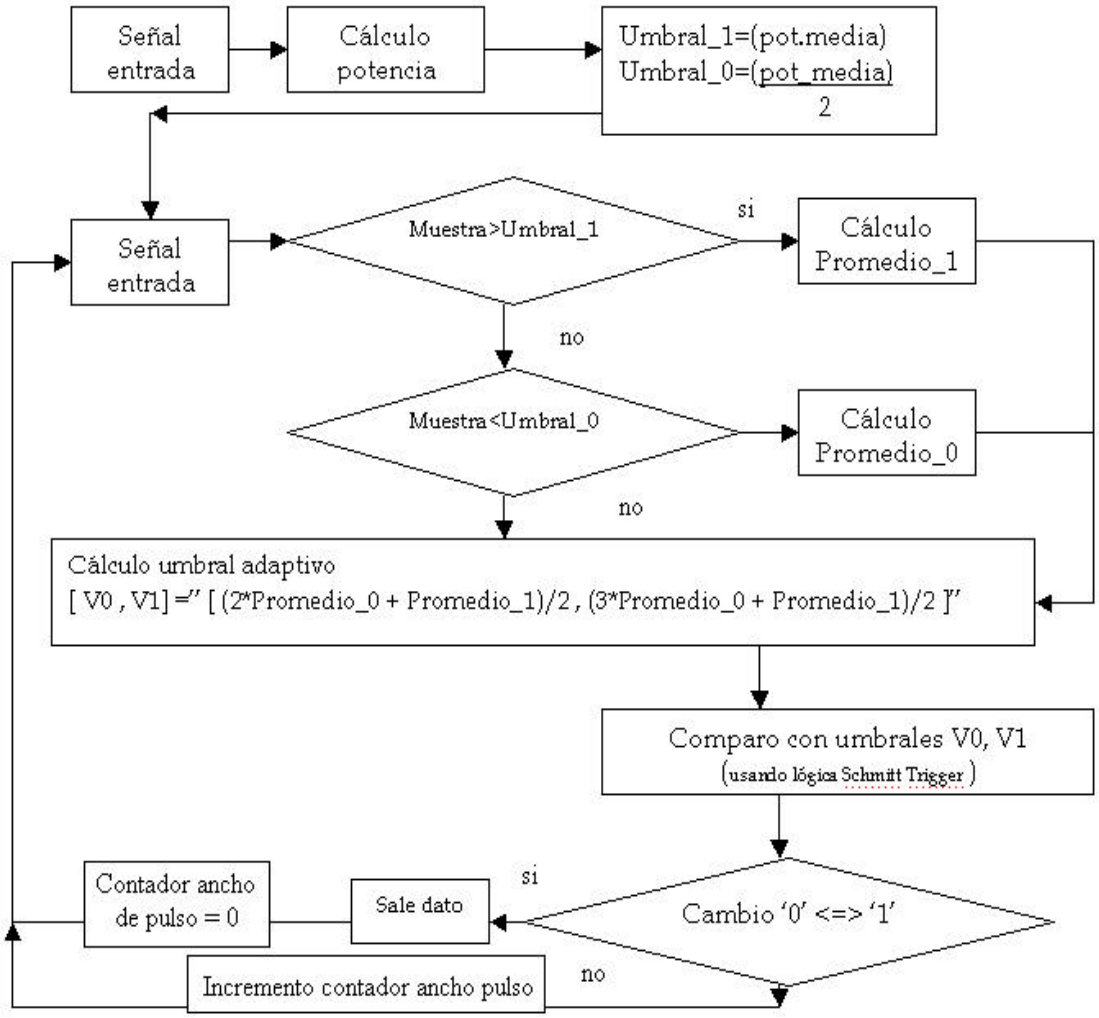


Señal filtrada y umbrales. Se grafica: en verde, la potencia media; en lila, los umbrales máximos y de ruido, y en rojo, los umbrales delventana Schmitt trigger.



Señal binaria resultante

5.4.2.2 Generación de Onda Cuadrada



Una vez que se tienen los umbrales de comparación sólo resta verificar en cuál zona se encuentra la muestra y así, según el algoritmo expuesto en el gráfico, obtenemos una señal pura binaria.

Como ya adelantamos, la salida del detector binario está formada por un valor numérico, correspondiente al largo del pulso, y un valor lógico, que indica si es pulso o espacio. A este conjunto pasaremos a denominarlo paquete. Este será tomado por el detector CW para recuperar el mensaje Morse.

5.4.2.3 Corrección de Errores

Esta etapa no tiene un método para corregir errores propios, más hallá de los previstos en el buen diseño del ajuste de los umbrales. Pero sí presenta un mecanismo que, ante el requerimiento de una capa superior, ayuda a la

corrección. Como se verá más adelante, que el detector binario guarde los datos de la salida anterior ayuda a corregir los espureos en una forma simple.

5.4.3 CAPA 3- Detector Morse

5.4.3.1 Decodificación Punto-Raya

En ésta etapa del proceso de decodificación realizamos la decodificación del mensaje en su modo más básico, es decir en puntos y rayas, como lo haría un registrador automático sobre papel de los que se usaban en los comienzos del Morse. También generamos otros símbolos que ayudan a la posterior corrección de errores y dan más robustez a la decodificación. Estos símbolos indican los espacio entre caracteres y palabras, los espureos, los silencios prolongados o rayas muy largas que se pueden considerar como una portadora no modulada.

Dado que una de las premisas del proyecto es hacer el decodificador automático e independiente del operador, es necesario volver a definir umbrales y criterios para identificar los símbolos antes mencionados, teniendo en cuenta una serie de factores que pasaremos a detallar:

1. No conocemos la velocidad de transmisión, lo que nos genera la incertidumbre del largo de los puntos, rayas, espacios, etc.Pero si conocemos la relación entre estos elementos.

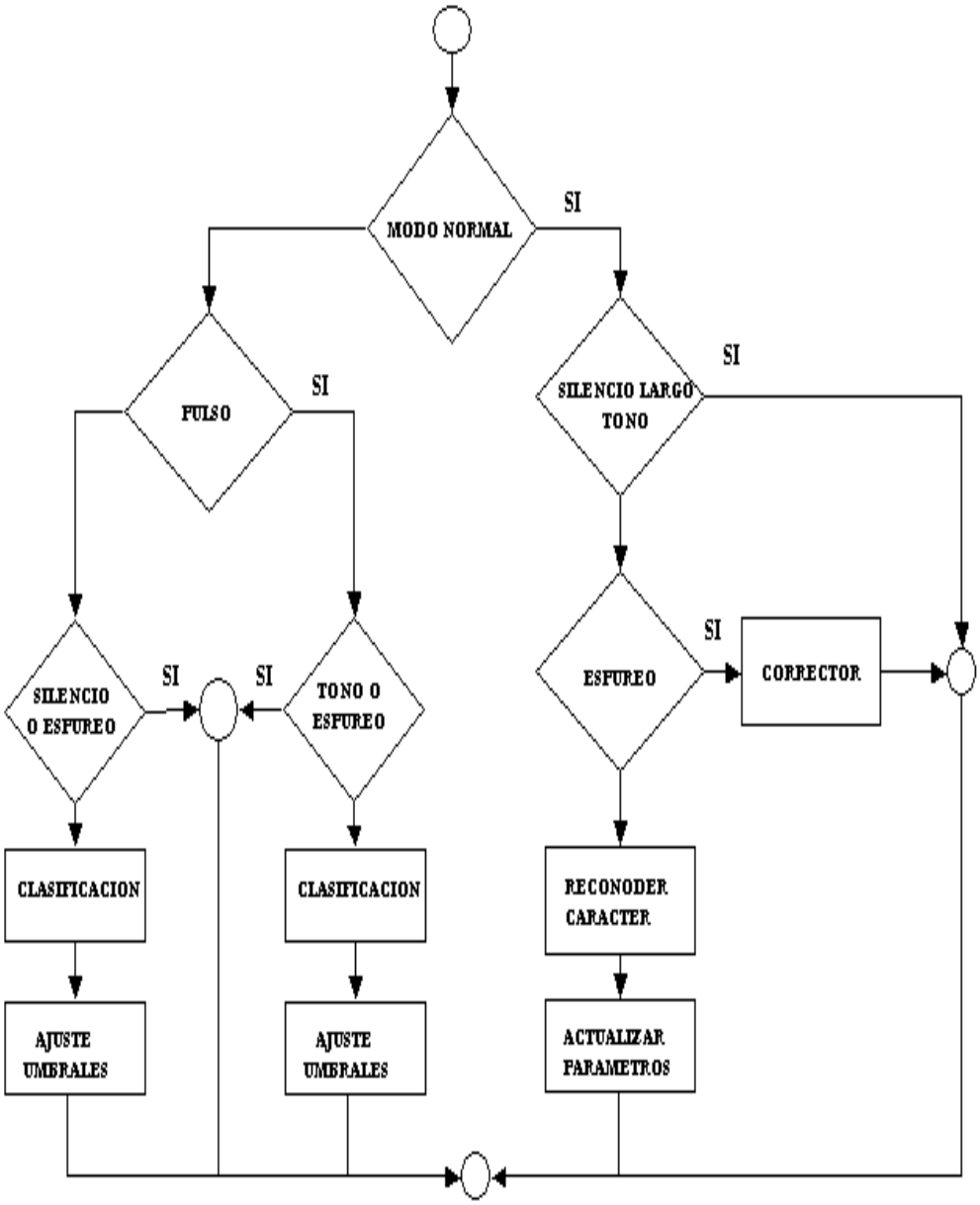
2. El Morse recibido no es ideal. Con esto queremos indicar que la relación de duración entre puntos y rayas no es estrictamente la misma siempre, debido a que el operador que genera, en la mayoría de las veces, es humano.

3. La existencia de pulsos espureos, cuyo tratamiento es el que se tratará en la sección siguiente por ser una fuente importante de incertidumbre.

4. Los silencios y los tonos no modulados, que le quitan a la señal correlación. Por esto, si no son detectados a tiempo, afectarán la estadística de la señal.

Por la autonomía que le imponemos al decodificador es imprescindible tener una etapa de aprendizaje, es decir, el

detector necesita ser entrenado para ajustarse, de modo de reconocer puntos y rayas de una forma efectiva (partiendo solamente de la relación existente entre los elementos que componen el Morse). Es por esta razón que encontramos tres (3) etapas bien definidas dentro del detector CW: modo aprendizaje, modo normal y detección de errores básicos.



En el diagrama se puede observar, en grandes bloques, la estructura del detector CW.

5.4.3.1.1 Detección de Errores Básicos

La detección de errores básicos la encontramos al comienzo de la rutina cw_det_cw.asm. Mediante este algoritmo se verificará si el paquete enviado por el detector binario corresponde a un paquete válido, o si está dentro de las categorías espureo, tono no modulado o silencio prolongado.

Para llevar a cabo la verificación nos valemos de umbrales predefinidos, que salen de las especificaciones del decodificador, en cuanto a las velocidades máximas y mínimas de decodificación. Por esto un pulso no podrá ser más corto que el correspondiente al punto a la velocidad máxima de operación. De igual forma se llega al valor para tonos prolongados no modulados (ráfagas) o silencios.

Al detectar los errores en este punto evitamos pasar información que altere la estadística de la señal que contiene el mensaje.

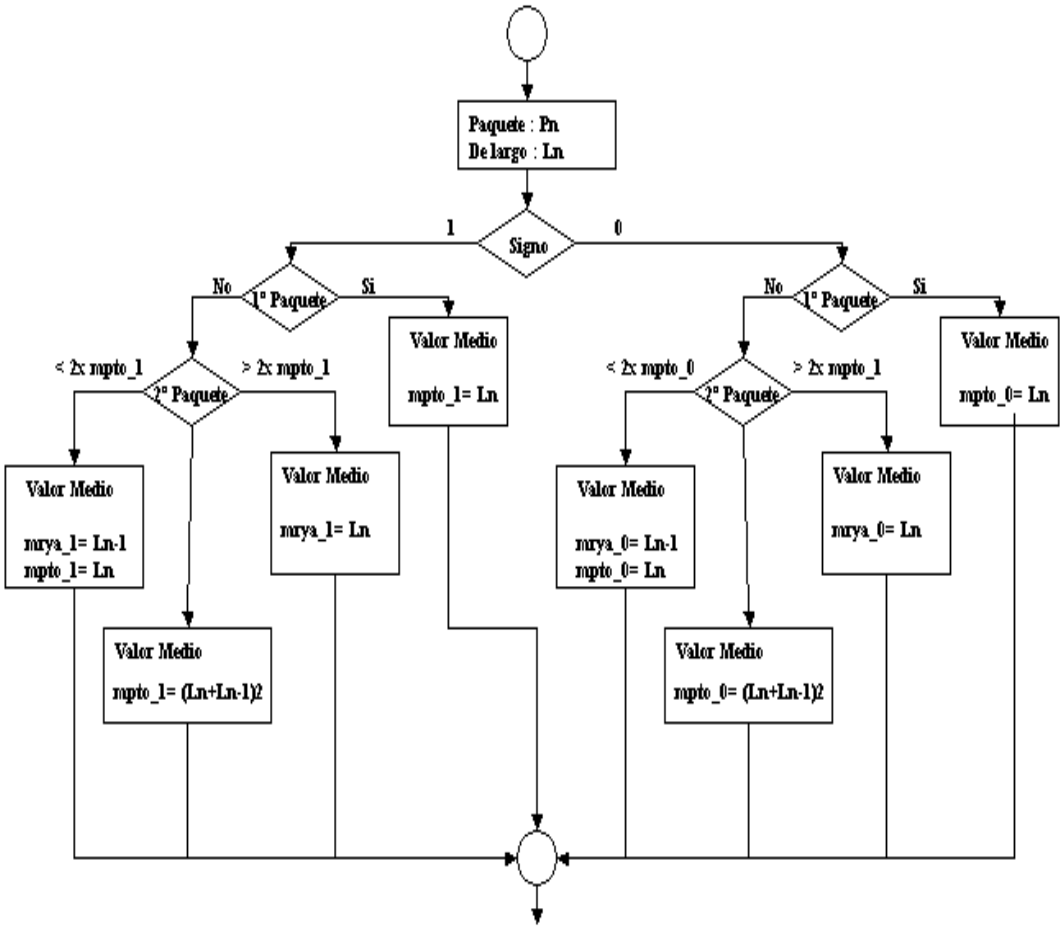
```
;==============Detector de CW =================

    clr  a
    move  y:packet_cnt,a1
    bclr    #cs_bin,y:flags

 ;Chequeo espurios ----------------------------------------------

    clr  b   ;si esta en función normal impongo el espurio
             ;cuando es menor que (y:mpto_1)/4
    jset     #cw_norm,y:flags_cw,espp
    move  #>min_pto,b1
    jmp     espboot

    espp
    clr  b   ;con esto  distingo espurios de puntos o rayas
    jset     #cw_type,y:flags_cw,espureo_1
    move  y:mpto_0,b
    asr b
    asr b
    jmp     verif

    espureo_1
    move  y:mpto_1,b
    asr b
    asr b

    verif
```

```
move  b,y:y_min_pto
clr  b
move  y:y_min_pto,b1
espboot
cmp    b,a
jgt  det_rafaga

move  y:pack_old,x0      ;corrector de espureos
move  y:packet_cnt,a
add    x0,a
move  a,y:sub_pkt
move  a,y:packet_cnt

jmp     end_det_cw2

det_rafaga     ;-------------------------------------------------------

 clr b
;si esta en función normal impongo la rafaaga cuando es
mayor
; que 2*(y:mrya_1)
jset      #cw_norm,y:flags_cw,esp2
move  #>max_rya,b1
jmp     espboot2

esp2
clr  b
;con esto  distingo máximos de puntos o rayas
 jset     #cw_type,y:flags_cw,raf_1

move  y:mrya_0,b
asl  b
jmp     endraf
raf_1
move  y:mrya_1,b
asl  b
endraf
move  b,y:y_max_rya
clr  b
move  y:y_max_rya,b1
espboot2
cmp    b,a
 jle  deteccion

 jset     #cw_type,y:flags_cw,Frafaga_1
Frafaga_0
move  #rafaga_0,r0
move  r0,x:cw_simb
jmp     end_det_cw
Frafaga_1
move  #rafaga_1,r0
move  r0,x:cw_simb
jmp     end_det_cw
```

Una vez que el paquete procedente del detector binario pasa este clasificador, estamos en condiciones de comenzar la etapa de aprendizaje.

5.4.3.1.2 Modo Aprendizaje

El fin de esta parte del código es hallar los umbrales que nos permitan clasificar los paquetes de unos como puntos o rayas y los paquetes de ceros como separación entre caracteres (puntos de cero) o entre palabras (rayas de cero).



Vamos a realizar una detección estadística, para la cual los umbrales se deberán ajustar en forma dinámica, siguiendo las fluctuaciones de velocidad del operador que genera el Morse. Es entonces vital hallar el valor medio de los paquetes de uno y de cero para luego construir los

umbrales. La forma en que actúa éste algoritmo se expresa en el diagrama y en el código que presentamos a continuación:

```
cw_boot ;-------------------------------------------------------------

        jclr #cw_1pkt,y:flags_cw,cw_bootpkt
        jset    #cw_type,y:flags_cw,cw_boot1
        jmp     cw_boot0

    cw_bootpkt
        bset    #cw_1pkt,y:flags_cw          ;descarta 1
    packet
        clr  b
        move  b,y:pack_old

        jmp     end_det_cw2


cw_boot0 ;-----------------------------------------------------------

        jset    #cw_mpto0,y:flags_cw,cw_boot0_2do
        move  a,y:mpto_0
        bset    #cw_mpto0,y:flags_cw
        jmp     end_det_cw2

cw_boot0_2do
        move  y:packet_cnt,b

        move  y:mpto_0,a
        rep     #3
        asl  a
        cmp    a,b ; Chequeo si el 2do packet > 8 veces el
    pto -> rafaga0
        jgt  cw_boot0_rafaga
        asr a
        cmp    a,b                  ; Chequeo si el 2do packet
    > 2 veces el pto
        jgt  cw_boot0_1rya

        move  y:mpto_0,a    ; Chequeo si el 2do packet < 1/2
    veces el pto
        asr a
        cmp    a,b      ; Si es < .5*mpto -> interambio valor de
    pto x rya
        jle  cw_boot0_pxr

        move  y:packet_cnt,a         ;Si otro pto -> promedio
    valor del pto
        move  y:mpto_0,x0
        add    x0,a
        asr a
        move  a,y:mpto_0
        jmp     end_det_cw2

    cw_boot0_pxr
```

```
    move  y:mpto_0,a
    move  a,y:mrya_0
    move  b,y:mpto_0
    bset    #cw_mrya0,y:flags_cw
    jmp     calc_hum_0

cw_boot0_1rya
    move  b,y:mrya_0
    bset    #cw_mrya0,y:flags_cw
    jmp     calc_hum_0

cw_boot0_rafaga
    jmp     end_det_cw2

cw_boot1 ;-------------------------------------------------------------

    jset    #cw_mpto1,y:flags_cw,cw_boot1_2do
    move  a,y:mpto_1
    bset    #cw_mpto1,y:flags_cw
    jmp     end_det_cw2

cw_boot1_2do
    move  y:packet_cnt,b

    move  y:mpto_1,a
    rep     #3
    asl a
    cmp     a,b ; Chequeo si el 2do packet > 8 veces el
pto -> rafaga0
    jgt  cw_boot1_rafaga

    move  y:mpto_1,a
    asl a
    cmp     a,b                 ; Chequeo si el 2do packet
> 2 veces el pto
    jgt  cw_boot1_1rya

    move  y:mpto_1,a    ; Chequeo si el 2do packet <1/2
veces el pto
    asr a
    cmp     a,b     ; Si es < .5*mpto -> interambio valor de
pto x rya
    jle   cw_boot1_pxr

    move  y:packet_cnt,a        ;Si otro pto -> promedio
valor del pto
    move  y:mpto_1,x0
    add     x0,a
    asr a
    move  a,y:mpto_1

    jmp     end_det_cw2

cw_boot1_pxr
    move  y:mpto_1,a
    move  a,y:mrya_1
    move  b,y:mpto_1
```

```
        bset    #cw_mrya1,y:flags_cw
        jmp     calc_hum_1

    cw_boot1_1rya
        move  b,y:mrya_1
        bset    #cw_mrya1,y:flags_cw
        jmp     calc_hum_1

    cw_boot1_rafaga
            jmp     end_det_cw2
```

Una vez que se tienen los valores medios se calculan los umbrales, como el promedio de los valores medios de puntos y rayas. Debemos observar que se toman precauciones extras en ésta etapa con las rayas, que pasan el control de ráfagas, pero son aún más largas que lo esperado dado el largo del punto. Esto lo hacemos para evitar que en el aprendizaje rayas largas generadas por el operador eleven el promedio fuera del valor esperado llevando a un error en la fijación de los umbrales.

La existencia de los umbral se manifiesta mediante flags, una vez que se tienen los umbrales tanto para paquetes de ceros como de unos se entra en el modo normal.

5.4.3.1.3 Modo Normal

En esta etapa ya contamos con los umbrales, por lo que sólo es necesario realizar la comparación del paquete contra el umbral correspondiente para luego dar como salida alguno de los símbolos antes mencionados.

```
cw_normal ;-------------------------------------------------------
    jset    #cw_type,y:flags_cw,cw_norm1

    move y:mpto_0,b          ;descarto rayas largas de
0, que hallan pasado
    rep     #3               ;deteccion de rafagas0
por humb_0 muy alto
    asl b
    cmp   b,a                ;> 8 veces el pto ->
rafaga0
    jgt Fraf_0
```

```
    move y:mhumb_0,b
    cmp   b,a                   ;Detecto Ptos o Ryas
de cero
    jgt Frya_0

    move #punto_0,r0
    move r0,x:cw_simb       ; escribo en buffer de
salida

    move #pesob,x0 a,y0         ; recalculo valor
medio del pto_0
    mpy   x0,y0,a
    move #pesoa,y0
    move y:mpto_0,x0
    mac   x0,y0,a

    move a,y:mpto_0
    jmp    calc_hum_0

Frya_0

    move #raya_0,r0
    move r0,x:cw_simb
    move #pesob,x0 a,y0                 ; recalculo
valor medio del

rya_0
    mpy   x0,y0,a
    move #pesoa,y0
    move y:mrya_0,x0
    mac   x0,y0,a

    move a,y:mrya_0
    jmp    calc_hum_0

Fraf_0
    move #rafaga_0,r0
    move r0,x:cw_simb
    jmp    calc_hum_0
cw_norm1
    move y:mpto_1,b         ;descarto rayas largas de
1, que hallan pasado
    rep    #3               ;deteccion de rafagas1
por humb_1 muy alto
    asl b
    cmp   b,a               ;> 8 veces el pto ->
rafaga1
    jgt Fraf_1
    move y:mhumb_1,b
    cmp   b,a               ;Detecto Ptos o Ryas de
unos
```

```
    jgt Frya_1
    move #punto_1,r0
    move r0,x:cw_simb      ; escribo en buffer de salida
    move #pesob,x0 a,y0       ;recalculo valor medio
del

pto_1
    mpy   x0,y0,a
    move #pesoa,y0
    move y:mpto_1,x0
    mac   x0,y0,a
    move a,y:mpto_1
    jmp    calc_hum_1

Frya_1
    move #raya_1,r0
    move r0,x:cw_simb
    move #pesob,x0 a,y0                    ; recalculo
valor medio del rya_1
    mpy   x0,y0,a
    move #pesoa,y0
    move y:mrya_1,x0
    mac   x0,y0,a
    move a,y:mrya_1
    jmp    calc_hum_1
Fraf_1
    move #rafaga_1,r0
    move r0,x:cw_simb
    jmp    calc_hum_1


calc_hum_0 ;-------------------------------------------------------
    clr a
    move y:mrya_0,x0
    move y:mpto_0,a1
    add   x0,a
    asr    a

    jclr #cw_hum0,y:flags_cw,passhumb_0
    move y:mhumb_0,x0
    add   x0,a
    asr a

passhumb_0

  move   a,y:mhumb_0
    bset   #cw_hum0,y:flags_cw
    jmp    end_det_cw

calc_hum_1 ;-------------------------------------------------------
    move y:mrya_1,x0
```

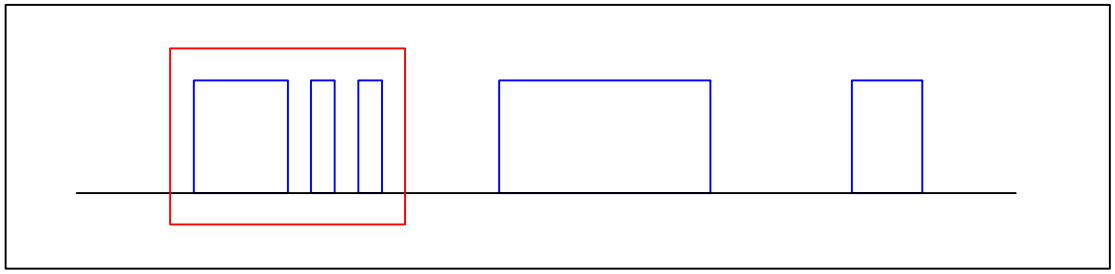
```
    move y:mpto_1,a1
    add  x0,a
    asr  a

    jclr #cw_hum1,y:flags_cw,passhumb_1
    move y:mhumb_1,x0
    add  x0,a
    asr  a

passhumb_1
    move a,y:mhumb_1
    bset #cw_hum1,y:flags_cw

end_det_cw

;correccion simb repetido por espureo----------------------
--------

    clr a
    clr b
    move y:cw_simb_old,a
    move x:cw_simb,b
    cmp  b,a
    jeq  end_det_cw2

    clr a
    move #raya_1,r0
    move r0,a
    cmp  b,a
    jne cw_bien0
    clr a
    move #punto_1,r0
    move r0,a
    move y:cw_simb_old,x0
    cmp  x0,a
    jeq end_det_cw3

cw_bien0

    clr a
    move #punto_1,r0
    move r0,a
    cmp  b,a
    jne cw_bien
    clr a
    move #raya_1,r0
    move r0,a
    move y:cw_simb_old,x0
    cmp  x0,a
    jeq end_det_cw3
```

```
cw_bien

    move y:cw_simb_old,a
    move a,x:cw_simbout

    move x:cw_simb,b
    move b,y:cw_simb_old
    bset   #cs_cw,y:flags       ; con esto habilito que
salga el dato
    jmp    end_det_cw2

end_det_cw3

    move y:cw_simb_old,a
    move a,x:cw_simbout

end_det_cw2

    move y:packet_cnt,b       ;actualizo valor anterior
contador
    move b,y:pack_old
    bclr   #act_cw,y:flags

end_ajuste

    rts
```

Además de la detección se realizan otras tareas como ser la actualización de los umbrales. Los valores medios se actualizan mediante suma ponderada, dándose más peso a la historia que al dato nuevo. Esto nos permite seguir las variaciones del operador al generar el Morse y rechazar variaciones rápidas asociadas a paquetes con comportamientos particulares. El uso de esta forma de actualización sumó estabilidad a la detección al eliminar eficazmente las fluctuaciones.

5.4.3.2 Errores Causados por Espureos

Los espureos pueden llevar a errores de decodificación como expondremos a continuación.



Luego de repetidas observaciones, y una vez que se logró un buen algoritmo para el ajuste de los umbrales del detector binario, notamos que los espureos no se daban en forma aislada de un punto o de una raya, sino que estaban estrechamente cercanos en el tiempo. Siendo altamente probable que se dieran antes o después de los flancos de los pulsos asociados a variaciones bruscas de la señal al momento de cruzar los umbrales de detección. En el caso de la figura, si se hubiera sacado el dato sin verificar si estaba o no seguido por espureo, se habría detectado un punto en vez de la raya que sería lo correcto.

La solución ante un espureo es no enviar el dato para ser sacado en pantalla, o enviado al detector ASCII, hasta no tener la certeza de que el símbolo es el correcto. En caso de que aparezca un espureo, sumamos al paquete anterior el largo del espureo y habilitamos el detector binario para completar la detección.

5.4.4 CAPA 4- Detector ASCII

5.4.4.1 Decodificación en Caracteres

Esta capa se encarga de decodificar el código Morse en código ASCII de forma de poder ser fácilmente leído por operadores sin nociones de Morse.

Debemos recordar que el código Morse es de largo variable, por lo que los espacios entre caracteres son los que marcan el final de una letra. De confundirse un espacio entre letras por un espacio entre punto y raya cometeríamos un error que no va a poder ser solucionado en ésta etapa, ya que el caracter generado será posiblemente válido.

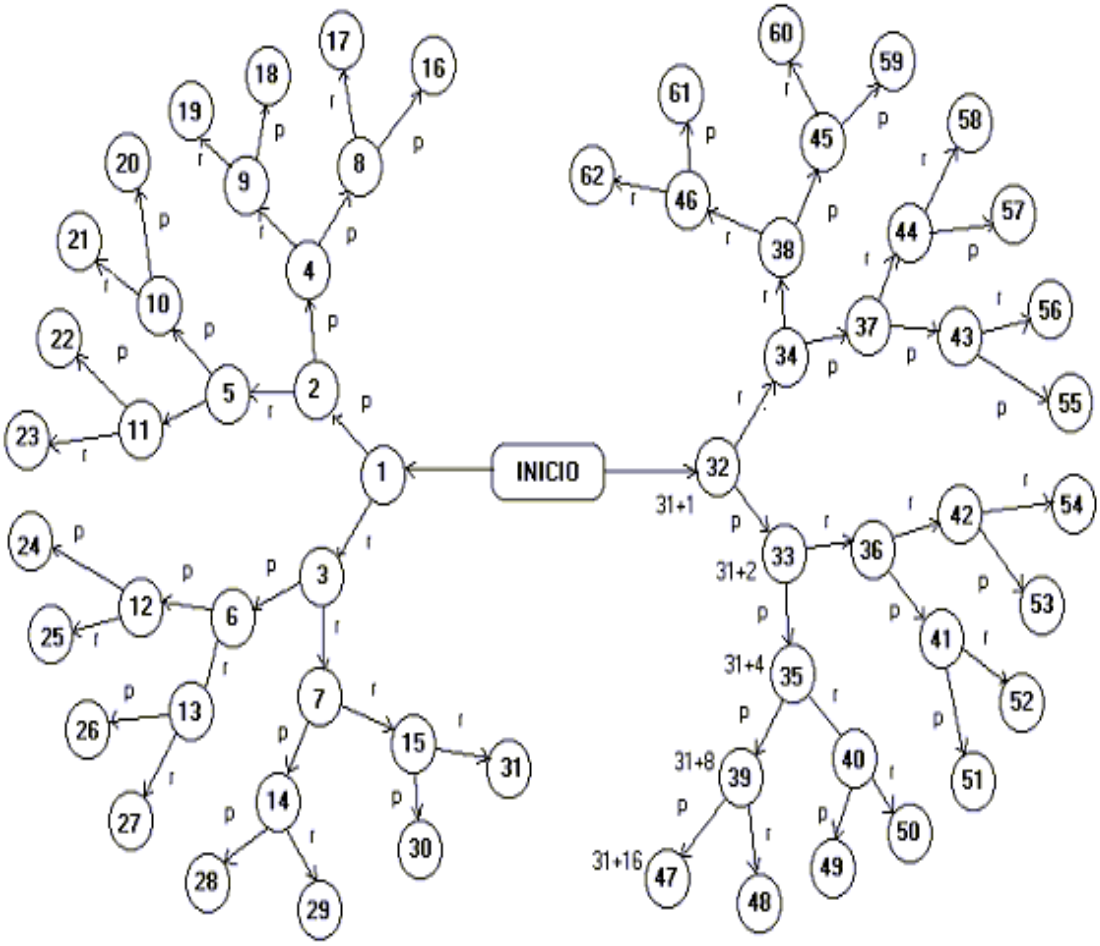
La idea que impulsó el algoritmo utilizado partió de la búsqueda de una forma rápida, que implicara poca programacíon y fuera fácilmente ajustable a otros códigos.

Se puede pensar en el código Morse como binario, en el sentido de que está conformado por puntos y rayas donde los espacios entre símbolos son separadores y los espacios entre letras son delimitadores.

De aquí surge claramente que se puede utilizar un árbol binario para realizar la decodificación.

El árbol será recorrido hasta que aparezca un delimitador o se termine la rama debido a que en algún paso se cometió un error.

El árbol que recorre el algoritmo es el siguiente:



Se observa que si restamos 31 a los valores del lado derecho se obtienen lo mismos valores que el lado izquierdo (esto es importante para formular regla)

Cada hoja se relaciona con un carácter mediante la tabla de asignación dada a continuación:

1.	error	
2.	p	E
3.	pp	I
4.	pr	A
5.	ppp	S
6.	ppr	U
7.	prp	R
8.	prr	W
9.	pppp	H
10.	pppr	V
11.	pprp	F
12.	pprr	
13.	prpp	L
14.	prpr	
15.	prrp	P
16.	prrr	J
17.	ppppp	5
18.	ppppr	4
19.	ppprp	
20.	ppprr	3
21.	pprpp	
22.	pprpr	
23.	pprrp	
24.	pprrr	2
25.	prppp	
26.	prppr	
27.	prprp	
28.	prprr	
29.	prrpp	
30.	prrpr	
31.	prrrp	
32.	prrrr	1
33.	r	T
34.	rp	N
35.	rr	M
36.	rpp	D
37.	rpr	K
38.	rrp	G
39.	rrr	O
40.	rppp	B
41.	rppr	X
42.	rprp	C
43.	rprr	Y

44.	rrpp	Z
45.	rrpr	Q
46.	rrrp	
47.	rrrr	
48.	rpppp	6
49.	rpppr	
50.	rpprp	
51.	rpprr	
52.	rprpp	
53.	rprpr	
54.	rprrp	
55.	rprrr	
56.	rrppp	7
57.	rrppr	
58.	rrprp	
59.	rrprr	
60.	rrrpp	8
61.	rrrpr	
62.	rrrrp	9
63.	Rrrrr	0

En la tabla se colocaron los caracteres básicos como forma de mostrar como se funciona el algoritmo. Al recorrerse el árbol se calcula el índice del carácter para luego buscarlo en una tabla y así tener el carácter correcto. Si se usa otra tabla se tendría un Morse propio o encriptado.

Como forma de optimizar el algoritmo se implementó el mismo en base a una máquina de estados, que cuenta con tres estados:

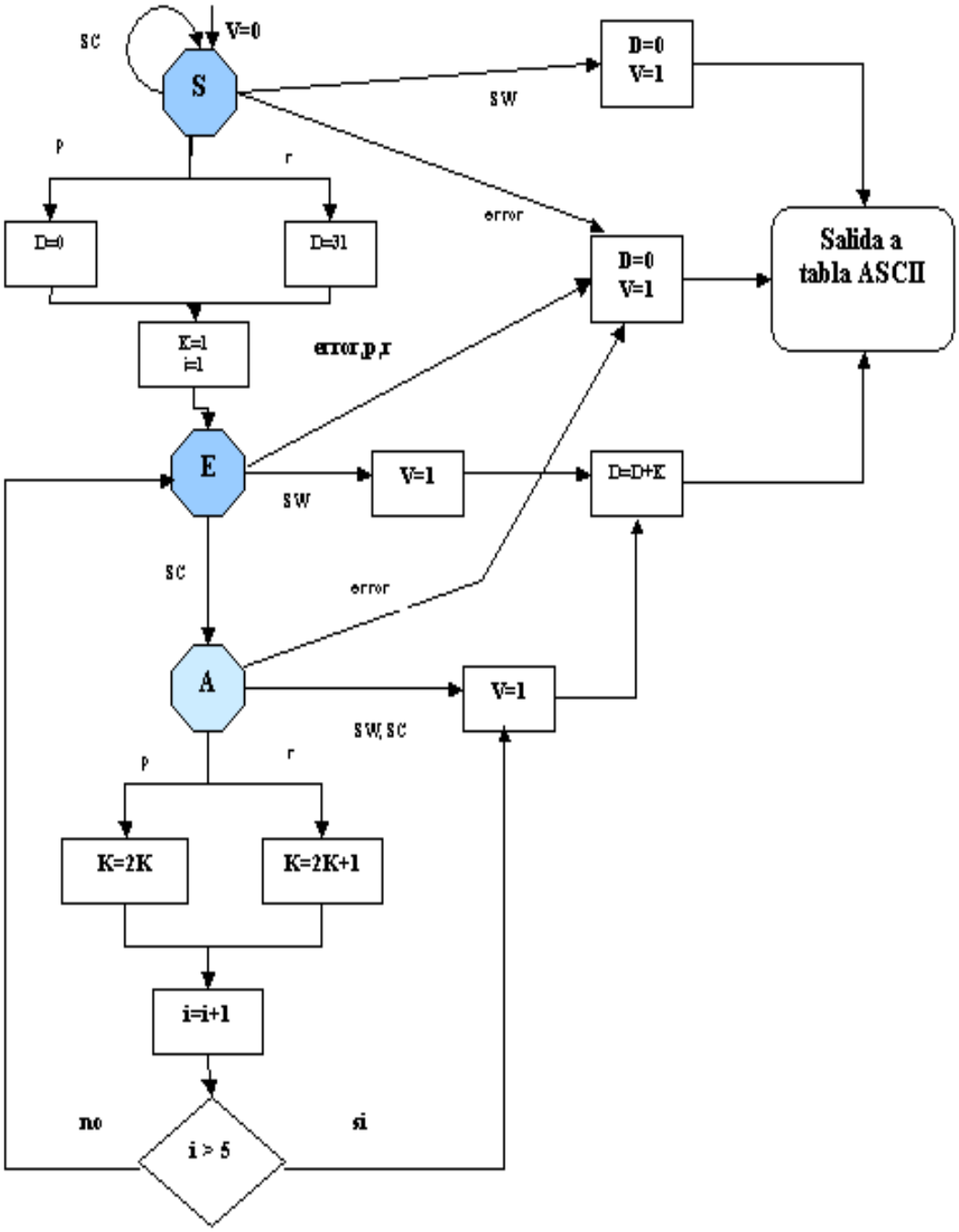
Estado **S**: en este estado se espera por el primer punto o raya de un carácter ASCII.

Estado **E:** este estado espera por silencios o separaciones.

Estado **A:** espera por puntos o rayas del carácter que se está decodificando.

El diagrama que se muestra a continuación permite ver el funcionamiento de la máquina de estado, siendo las variables de la misma:

p	punto
r	raya
SC	separación entre símbolos;
SW	separación entre caracteres;
V	indica dato valido cuando V=1
i	indica cantidad de símbolos del caracter
K	indice del árbol
D	Valor inicial de K



Supongamos que la trama que entra al decodificador ASCII es la siguiente:

[..sw, r, sc, p, sc, p, sc, r, sw..], o lo que es lo mismo 'r p p r'. Si buscamos en la tabla corresponde a la letra X, con índice 40.

A modo de ejemplo si recorremos las ramas del árbol, partiendo del estado inicial S, el primer símbolo 'r' nos lleva al círculo que contiene en su centro el número 32, el siguiente símbolo es 'p' con lo cual vamos al círculo 33 (31+2), luego con 'p' vamos al circulo 35 (31+4) y por último el símbolo 'r' llegamos al circulo 40 (31+9) que corresponde a la dirección en la tabla D=40 donde encontramos el carácter ASCCII 'X'.

Examinando con cuidado el árbol llegamos a una regla que nos permite encontrar las direcciones en la tabla ASCII con pocas y sencillas operaciones. La regla que sigue el algoritmo la siguiente:

1 Si el primer símbolo es 'p' ? D=0 , si es 'r' ? D=31

2 Para siguientes símbolos:
   a. si es un 'p' el índice K✍ 2?K
   b. si es un 'r' es K✍ (2?K)+1

3 La dirección del dato valido es D=D+K.

A continuación presentamos el código que refleja el diagrama:

```
;.....
; RUTINA QUE DA CARACTER ASCII A PARTIR DE DATOS DE
        DETECTOR_CW
;.....
  ;Obs: IMPORTANTE
;Este programa funciona como una máquina de estados cada vez
que es llamado se encuentra en
; un estado [inicio S, estado E, estado A] cuando se usa por primer vez
(o se intente resetear)deberá iniciarse ;con x:estado =1 (inicial)
;
; x:estados                -> 1= inicio
;                          -> E = estado E
;                          -> A = estado A
;
; x:dato                   ->  0= punto
;                          ->  1= raya
;                          ->  2 = sw (separacion entre caracteres
```

```
;                       -> 3 = sc (separacion entre punto/raya;
raya/punto; punto/punto o ....
;                       -> 4 = error
;
; x:dir                 ->  0 = error
;                       -> 11 = espacio
;                       -> 1=E, 2=I, 3=A,... segun TABLA        ;

;_____________________________________________________________

;---------------------------------------------
; subrutina main
;---------------------------------------------

mainn

        move  x:estado,x0
        move  y:k3,a
        cmp   x0,a
        jne   top1          ;estado<>E->top1
        jsr   subE
              rts

top1    move  y:k0,a
        cmp   x0,a
        jne   top2          ;estado<>A->top2
        jsr   subA
        rts

top2    move  y:k10,a
        cmp   x0,a
        jeq   start         ;estado=inicio->start

gn3     rts

;-------------------------- -
; subrutina start
;--------------------------

start
                nop
                move  x:dato,x0          ;dato->x0
                move  y:k2,a
                cmp   x0,a
                jne   next1         ;dato<>0 -> next1
                move  y:k12,a
                move  a,x:dir       ;0->D (dir)
                jmp   next2
next1           move  y:k1,a
                cmp   x0,a
                jne   next3         ;dato<>1 -> next3
                move  y:k7,a
                move  a,x:dir       ;31 -> D (dir)
next2           move  y:k10,a
                move  a,x:k         ;1->k
                move  a,x:i         ;1->i
                move  y:k12,a
```

```
                move  a,x:valid           ;0->valid
                move  y:k3,a
                move  a,x:estado    ;E->estado
                jmp   gn3

next3           move  y:k5,a
                cmp   x0,a
                jne   next4         ;dato<>sc -> next4
                move  y:k12,b
                move  b,x:valid           ;0->valid
                move  y:k10,b
                move  b,x:estado    ;1 (inicio) -> estado
                jmp   gn3

next4           move  y:k6,a
                cmp   x0,a
                jeq   next5a        ;dato<>sw->next5
                move  y:k13,a
                cmp   x0,a
                jne   next5

next5a      move  y:k10,b
                move  b,x:valid           ;1->v
                move  b,x:estado    ;1->estado
                move  y:k9,a
                move  a,x:dir       ;11->dir 'espacio'
                jmp   gn3

next5           move  y:k12,a             ;dato=error
                move  a,x:dir
                move  y:k10,a
                move  a,x:valid           ;1->valid
                move  a,x:estado    ;inicio->estado
                jmp   gn3

;---------------------------
; subrutina subE
;---------------------------

subE        move  x:dato,x0           ;dato->x0
                move  y:k6,a
                cmp   x0,a
                jeq   next10a       ;dato<>sw -> next10
                move  y:k13,a
                cmp   x0,a
                jne   next10

next10a     move  y:k10,a
                move  a,x:valid           ;1 -> V
                move  x:dir,b
                move  x:k,a
                add   b,a
                move  a,x:dir       ;D=D+K
                move  y:k10,b
                move  b,x:estado    ;inicio->estado
                rts
                ;jmp   eend
```

```
next10      move  y:k5,a
                  cmp   x0,a
                  jne   next11          ;dato<>sc -> next11
                  move  y:k0,b
                  move  b,x:estado      ;A->estado
                  move  y:k12,b
                  move  b,x:valid               ;0->V
                  rts
                  ;jmp   eend
next11      move  y:k12,b               ;dato=error,0,1
                  move  b,x:dir         ;0->D
                  move  y:k10,a
                  move  a,x:valid               ;1->V
                  move  a,x:estado      ;inicio->estado
                  rts
                  ;jmp   eend

;---------------------------
; subrutina subA
;---------------------------

subA        move  x:dato,x0             ;dato->x0
                  move  y:k2,a
                  cmp   x0,a
                  jne   next12                  ;dato<>0 -> next12
                  move  x:k,b
                  asl   b
                  move  b,x:k                   ;k=2k
                  jmp   next13
next12      move  y:k1,a
                  cmp   x0,a
                  jne   next14                  ;dato<>1 -> next14
                  move  x:k,b
                  asl   b
                  move  y:k10,y0
                  add   y0,b
                  move  b,x:k                   ;k=2k+1
next13      move  x:i,a
                  move  y:k10,b
                  add   b,a
                  move  a,x:i                   ;i=i+1
                  move  y:k8,b
                  cmp   b,a
                  jle   next15                  ;i<=5 -> next15
                  move  y:k10,a
                  move  a,x:valid                       ;1->V
                  move  a,x:estado              ;inicio -> estado
                  move  x:dir,a
                  move  x:k,b
                  add   b,a
                  move  a,x:dir                 ;D=D+K
                  rts
next15      move  y:k12,a
                  move  a,x:valid                       ;0 -> V
                  move  y:k3,a
                  move  a,x:estado              ;E->estado
                  rts
```

```
next14      move  y:k5,a
                  cmp   x0,a
                  jne   next16              ;dato<>sc -> next16
next17      move  y:k10,a
                  move  a,x:estado          ;inicio->estado
                  move  a,x:valid                   ;1->V
                  move  x:k,a
                  move  x:dir,b
                  add   b,a
                  move  a,x:dir             ;D=D+K
                  rts
next16      move  y:k6,a
                  cmp   x0,a
                  jeq   next17              ;dato=sw->next17
                  move  y:k13,a
                  cmp   x0,a
                  jeq   next17
                  move  y:k4,a
                  cmp   x0,a
                  jne   next18              ;dato<>error->next18
                  move  y:k12,a
                  move  a,x:dir             ;0->dir
                  move  y:k10,a             ;1->valid
                  move  a,x:valid
                  move  a,x:estado    ;inicio->estado
next18      rts
```

5.4.4.2 Corrección de Errores

La corrección de errores no la realizamos en ésta capa; aquí solamente se realiza la decodificación.

Dejamos para una capa superior no implementada todo lo relacionado con la corrección, la que pensamos se deberá construir en bases a diccionarios y controles gramaticales que no creímos correspondiera incluirlos en el código que corre el DSP.

5.4.5 CAPA 5- Interfase Humana

5.4.5.1 Consola entrada-salida

Como fuera mencionado, la interfase gráfica (GUI) que utilizamosk está basada en la consola VT100.

Nuestro objetivo fue lograr la comunicación con elusuario a través de una interfase simple, estándar y que requiriera de poco hardware adicional.

Es también fácil de implementar el intercambio de datos con aplicaciones más complejas.

El punto de partida para estudiar la GUI es el puerto C.

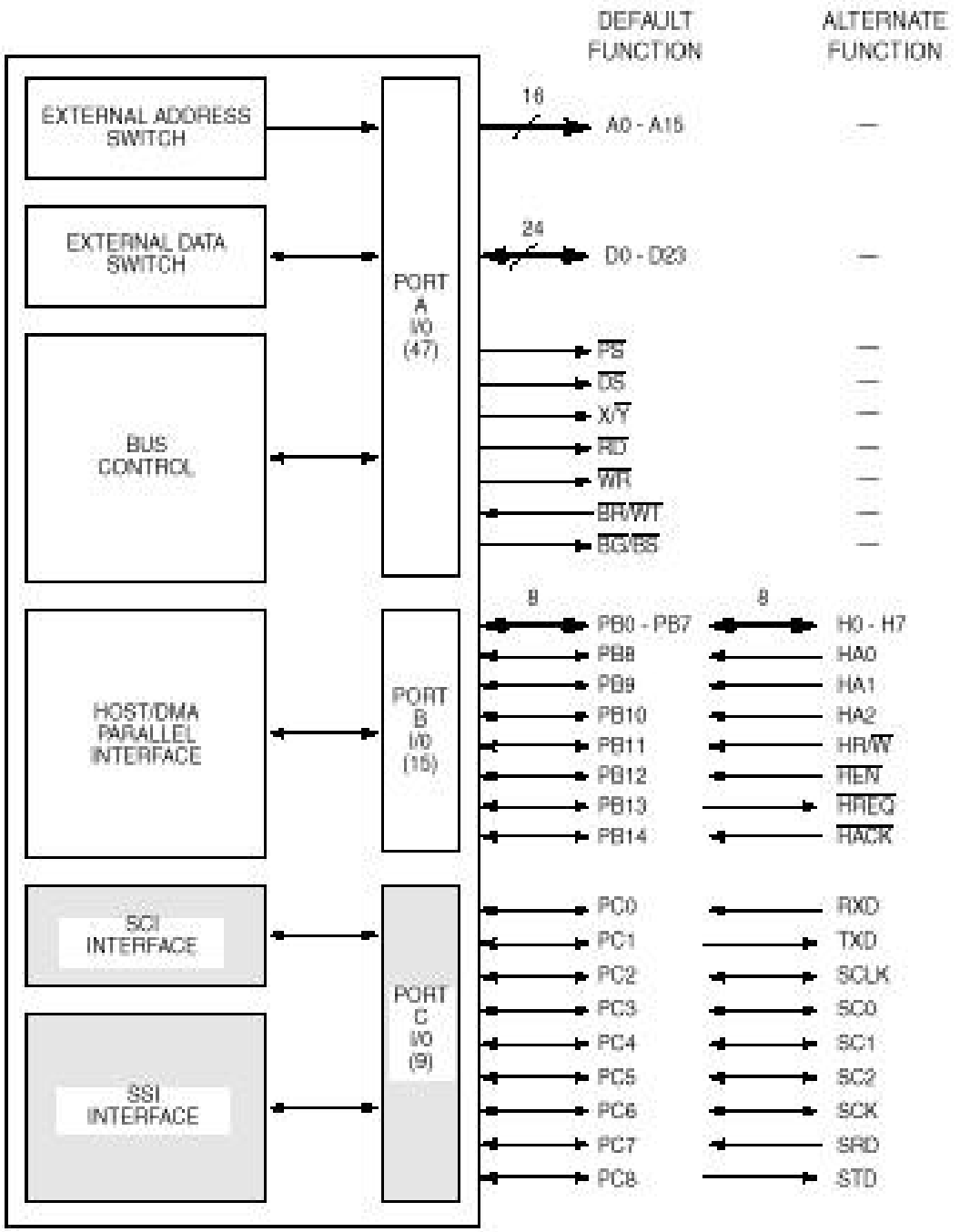


Figure 11-1 Port C Interface

Este puerto I/O puede cumplir tres funciones diferentes con sus nueve pines:

1. PC0-PC2 forman la interfase serial asíncrona de comunicaciones (SCI).
2. PC3-PC8 forman la interfase serial síncrona de comunicaciones (SSI).
3. PC3-PC8 pueden configurarse como puertos generales programables I/O (GPIO).

Lo interesante del puerto C es su flexibilidad y la posibilidad de utilizar las 3 formas de operación al mismo tiempo. Por ejemplo, puedo utilizar PC0-PC1 para realizar la interfase SCI, usar PC3 como GPIO para activar un dispositivo externo y PC3-PC8 como interfase SSI para comunicación con el CODEC. Esto se debe a que cada PIN es configurado individualmente.

En nuestro caso utilizamos el puerto C :

1. Como interfase de usuario asíncrona en forma de una consola VT100.

2. Para intercambiar información con el CODEC a través de la interfase SSI.

De ahora en más pasaremos a detallar la implementación de las distintas interfases

5.4.5.2 Interfase SCI.

Para implementar la interfase SCI a partir de la placa EVM, es necesario remitirse a los diagramas de circuito, ya que la interfase física viene como opcional (el carácter de opcional se refiere a que simplemente hace falta soldar un conector DB9 - en el lugar marcado en la placa - para de esta forma completar la interfase).

Luego se debe colocar el Jumper J8, en DCE para que así se pueda conectar el DSP al puerto serie de un PC con un cable derecho. Y por último, se deben colocar los Jumpers en J10 de forma que las señales del procesador lleguen a las patas del integrado MAX232CSE, que proveerá la

adaptación de las tensiones de las señales de TTL del procesador a RS232.

Una vez culminadas las modificaciones del hardware pasemos a describir el software que hace posible el envío y recepción de información a través de este puerto.

Los registros utilizados por el puerto SCI para su configuración y funcionamiento son los siguientes:

SCR	Sci Control Register
SCCR	Sci Clock Control Register
SSR	Sci Status Register
SRX	Sci Receive register
STX	Sci Trasmit register

Lo primero es definir el modo de operación de puerto SCI, como buscamos emular una consola VT100 elegimos:

Velocidad de transferencia 115 kbit/s
1 bit de arranque
8 bits de datos
1 bit de parada
Sin control de flujo

La elección de la velocidad no es arbitraria, ya que nuestra meta era poder independizar ésta interfase de las interrupciones causadas por el CODEC. Por eso, era imprescindible poder trasmitir un carácter (8 bits) más rápido que la velocidad de muestreo que está fijada en 8 Khz; es decir, deberíamos poder trasmitir más rápido que 64 kbit/s. Este detalle que parece menor no lo es.

El DSP56002 genera su clock como múltiplo de un cristal de 4 Mhz. Este clock es ajustado mediante multiplicadores y divisores enteros para generar el clock del puerto SCI.

Operando se llega a conclusión de que no es posible alcanzar el valor de 115 kbit/s sin cometer un deslizamiento tal que hace que se pierdan bits. Como es un puerto asíncrono, los relojes del receptor y del transmisor son independientes y sólo se sincronizan con el bit de arranque. La pérdida de bits se refleja en que el carácter enviado no es el mismo que el carácter recibido.

Una opción es sustituir el cristal por otro tal que se pueda alcanzar los 115 kbit/s con el menor error posible (ésta modificación está realizada en DSPs posteriores al 56002). Sin embargo, en el 56002 esto no es posible debido a que el microcontrolador del puerto OnCE toma su Clk del cristal de 4Mhz y lo reduce a 9600 Hz. mediante retardos, por lo que cambiar el cristal implica cambiar la velocidad del puerto OnCE, generando problemas con el debugger.

La solución que encontramos, luego de buscar en la documentación del DSP, fue subir la velocidad de operación del procesador de 40Mhz. a 44Mhz. Esta variación del 10% - que es soportada por el procesador sin inconvenientes - nos permite lograr los 115 kbits (y más) con el mínimo error.

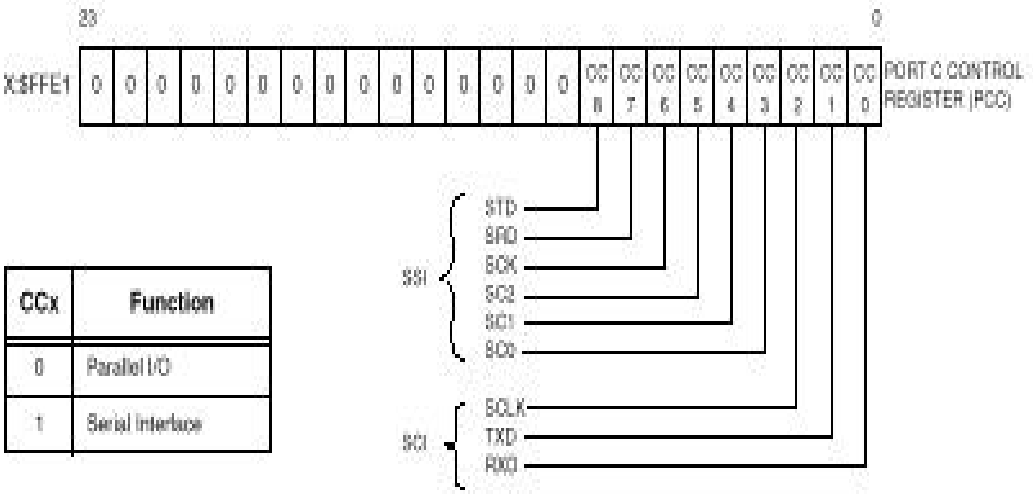
Ahora pasemos a detallar la inicialización del puerto realizada en la rutina cw_main.asm :

Los valores de las variables que se utilizan para la operativa de los puertos está definida en los archivos ioequ.asm y code_equ.asm.

1) asignamos el modo de operación de los pines del puerto C

```
movep   #$14,x:PCD     ; D/C~ pin = 1  ==> data mode
movep #$01EB,x:PCC     ;turn on ssi port (enable SSI and SCI now.).
```

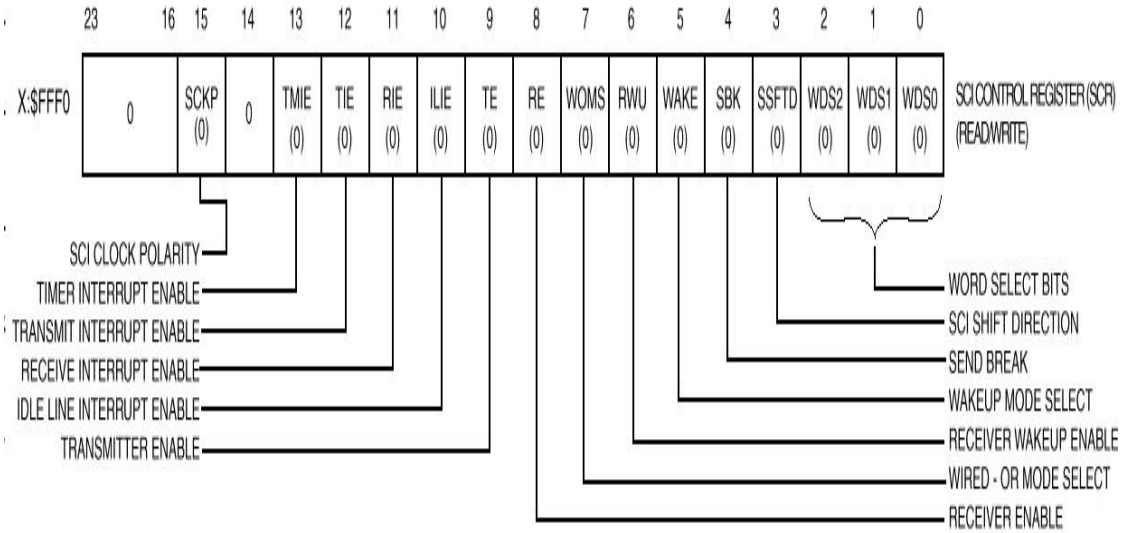
Observación: el proceso de asignación se realiza al mismo tiempo para ambos puertos durante la inicialización del puerto SSI (codec_init.asm).



CCx	Function
0	Parallel I/O
1	Serial Interface

2) Configuramos el modo de operación de la interfase RS232 y habilitamos la recepción y transmisión.

```
movep   #0,x:M_SCR          ; se inicializa SCR (Sci Control Register)
bset    #M_WDS1,x:M_SCR     ; largo de palabra 10 bits (1start + 8data + 1stop)
;bset   #3,x:M_SCR          ; shift direction. Define cuando se envia el MSBit
bset    #M_TE,x:M_SCR       ;enable the transmitter (WDS=0 10 ones )
bset    #M_RE,x:M_SCR       ;enable the receiver
```



3) Habilitamos las interrupciones. El uso de las interrupciones en la interfase SCI fue pensado con detenimiento ya que no queríamos tener interferencias con otros procesos. Las interrupciones son utilizadas para recibir entradas desde el teclado mientras que la escritura se realiza por medio de Pulling . Le asignamos, además, menor prioridad que a las interrupciones generadas por el CODEC.

```
bset    #M_SCL1,x:M_IPR     ;sci Interrupt level=1
bset    #M_RIE,x:M_SCR      ;enable the reciever interupt
```

4) Definimos la velocidad del puerto en el registro SCCR (Sci Clock Control Register). La misma se realiza (como indica la figura) mediante un divisor CD11-CD0 y un multiplicador de escala SCP. La velocidad del puerto está dada por la formula que aparece en la figura,

siendo: fo el clk del procesador, CD el divisor y SCP el valor del Prescaler.

TCM	RCM	TX Clock	RX Clock	SCLK Pin	Mode
0	0	Internal	Internal	Output	Synchronous/Asynchronous
0	1	Internal	External	Input	Asynchronous Only
1	0	External	Internal	Input	Asynchronous Only
1	1	External	External	Input	Synchronous/Asynchronous

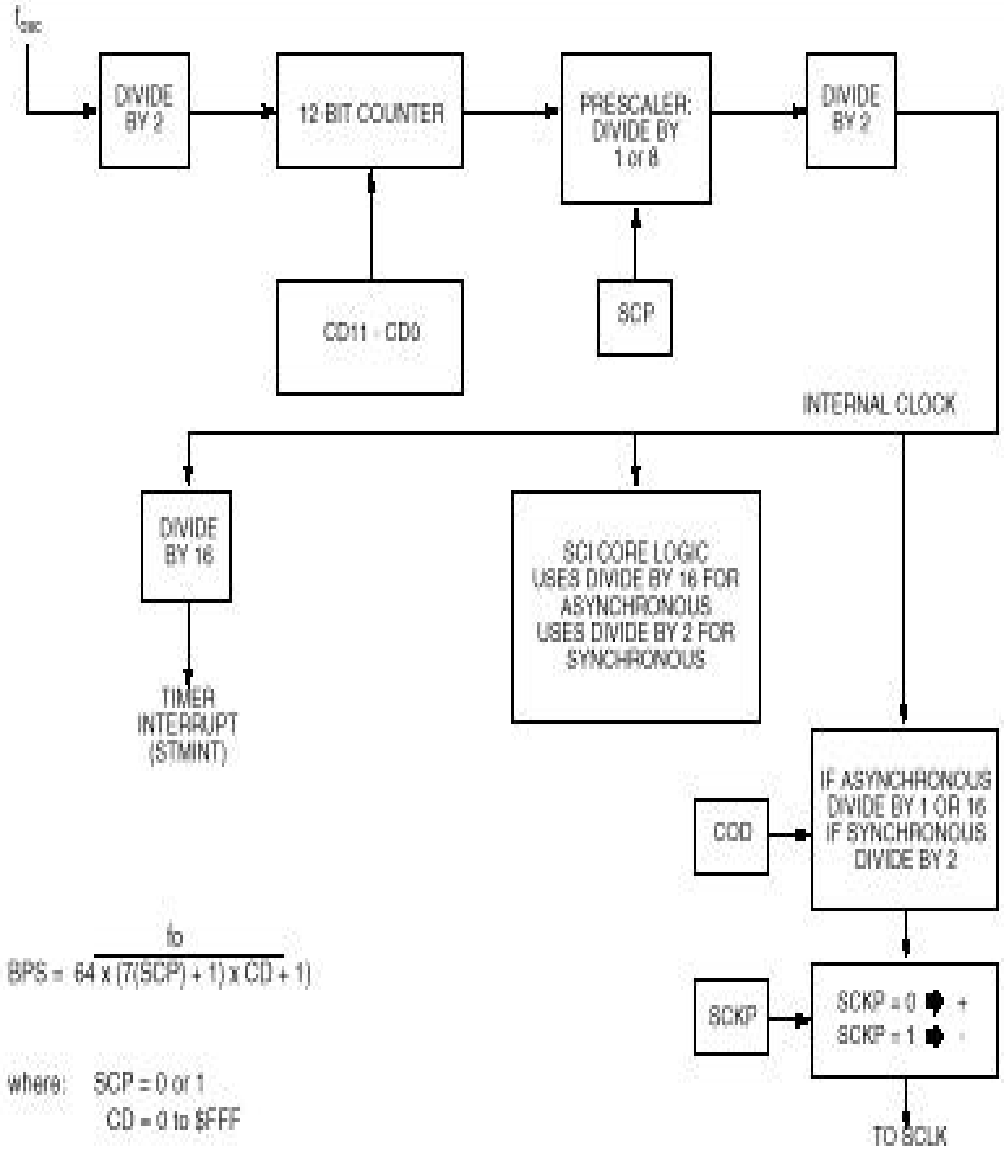
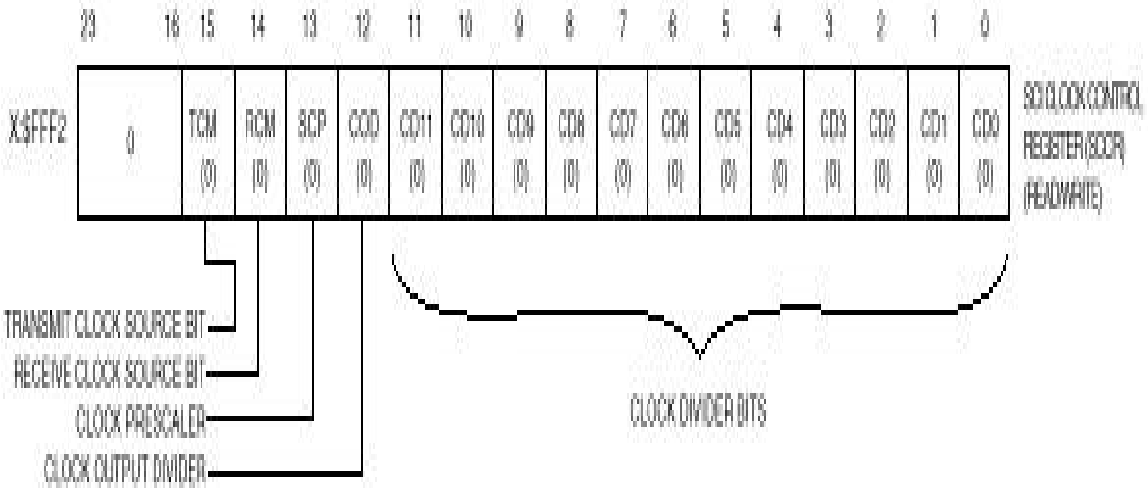


Figure 11-13 SCI Baud Rate Generator

movep #5,x:M_SCCR ; clk=40Mhz : 300bps=#2082 19.2 kbps=#32
; clk=44 MHz : 115.2kbps=#5 19.2 kbps=#35



De esta forma hemos completado la inicialización del puerto SCI. Ahora pasemos a estudiar la implementación de las rutinas que hacen posible la interfase gráfica.

5.4.5.3 Recepción

Como ya hemos mencionamos, la recepción se realiza mediante interrupciones. Al recibirse un dato nuevo en el puerto este genera una interrupción, por lo que el procesador guarda los registros que están en uso y apunta el Program Counter (PC) a la dirección correspondiente a la rutina que atiende esta interrupción .

En cw_main.asm asignamos durante la inicialización la rutina que atiende a la interrupción.

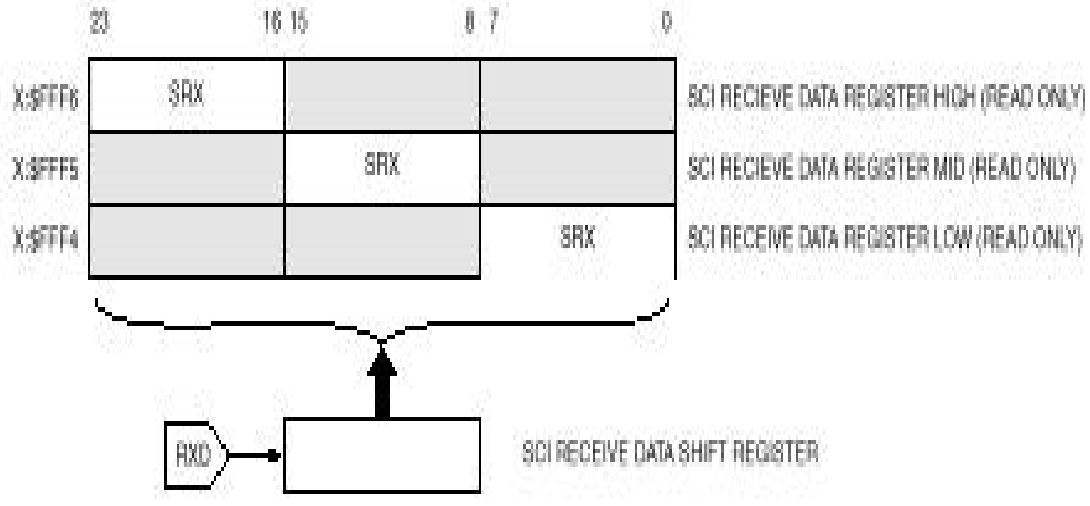
```
DEFINICIÓN DE VECTORES DE INTERRUPCIÓN:

org     p:$000c
jsr     ssi_rx_isr      ;SSI receive data
jsr     ssi_rx_isr      ;SSI receive data with exception
```

```
jsr     ssi_tx_isr      ;SSI transmit data
jsr     ssi_tx_isr      ;SSI transmit with exception

org     p:$0014     ;SCI receive interupt vector location
jsr     rx_sci

org     p:$0016     ;SCI receive with exception vector
jsr     rx_sci
```

En cw_librería.asm se encuentra la subrutina de atención a la interrupción.

```
rx_sci                  ;receive data del sci port by interruption

bclr #M_RIE,x:M_SCR     ; deshabilito la recepcion de
interrupciones
move        x:M_SRXL,a0         ; muevo el contenido buffer de
recepcion                   move    a0,x:rx_simb

bset        #act_menu,y:flags           ;prendo el flag de datos
recibidos

bset #M_RIE,x:M_SCR     ; habilito las interrupciones en rx rti
```

El buffer de entrada se organiza de la siguiente manera:



NOTE: SRX is the same register decoded at three different addresses.

(a) Receive Data Register

Los datos de entrada son decodificados en 8 bits, esto posibilita la reconstrucción de datos de hasta 24 bits. A nosotros nos basta con el registro más bajo SRXL para contener la información de un caracter. Observamos que los datos son movidos a A0 y no directamente al acumulador A. De ésta forma evitamos el shift hacia la derecha que introduciría de otra forma el procesador.

El dato recibido es guardado en memoria y se enciende el flag que avisa que hay una entrada para procesar. Así, el proceso de lectura se acelera y el dato es utilizado solamente cuando hay recursos para hacerlo

En cw_main.asm se chequea éste flag y se llama a la rutina menu contenida en el archivo cw_librería.asm.

menu

```
clr a
clr b
move x:rx_simb,b1        ;chequeo que opcion se presiono
move #>'1',a1
cmp b,a
jeq menuop1
move #>'2',a1
cmp b,a
jeq menuop2
move #>'3',a1
cmp b,a
jeq menuop3
move #>'m',a1
cmp b,a
jeq menuop
jmp menu_end

menuop1
bclr #act_menu,y:flags
bclr #act_sh_cw,y:flags
jmp menu_end


menuop2
bclr #act_menu,y:flags
bset #act_sh_cw,y:flags
jmp menu_end
menuop3
bset #act_reboot,y:flags
jmp menu_end
menuop
move #men_menu,r0     ;Mensaje menu,menu1,menu2,menu3
```

```
move #53,n0
jsr tx_sci
bclr #act_menu,y:flags

menustp
jclr #act_menu,y:flags,menustp      ;loop      desplegando
mensaje
rts

menu_end
rts
```

El menú que se despliega en pantalla, contiene las siguientes opciones :

1- MORSE (Decodifica en puntos y rayas)
2- ASCII (Decodifica en código ASCII)
3- REBOOT (Reinicia el sistema)

Para llamar al menú durante la decodificación se debe presionar la letra "m". El flujo del programa se detiene ya que entra en Loop, hasta que, al elegir una de las opciones, se sale de éste. El cambio de una forma de decodificación a otra es posible de realizar sin llamar al menú, por lo que, presionando la tecla 1 o 2, se pasa de ver la decodificación en ASCII a ver Morse puro sin interrupciones. El flag act_sh_cw es el encargado de inhibir o no al decodificador ASCII.

Al llamar a la opción reboot el flujo del programa hace un salto a p:sys_reboot.

5.4.5.4 Transmisión

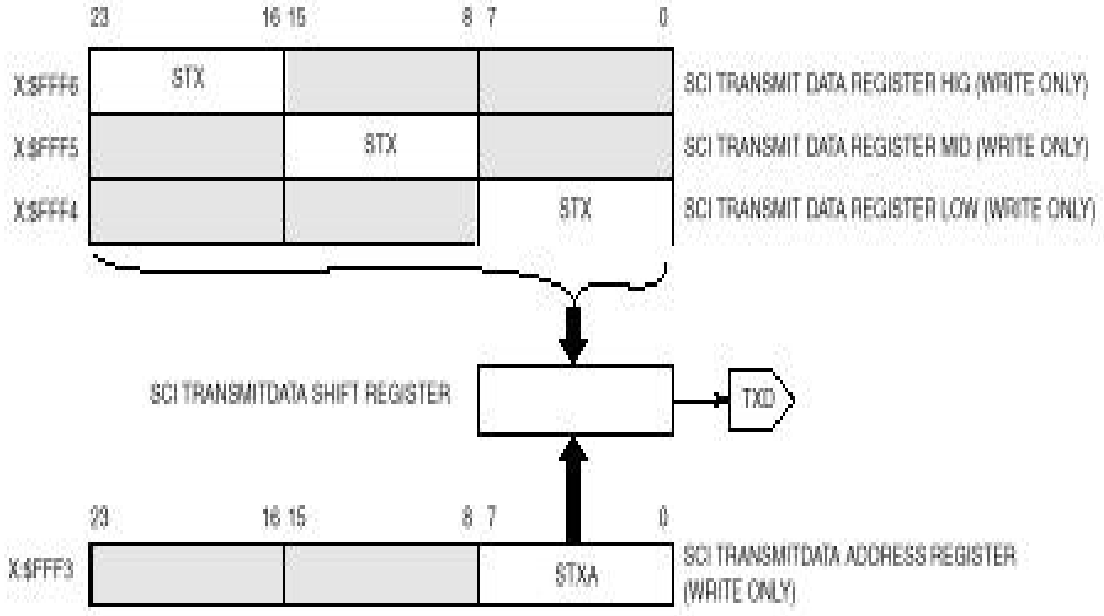
La trasmisión es realizada por Polling, es decir, mediante una rutina los datos son enviados al puerto SCI y procesada su salida. No se hace uso de interrupciones en este caso.

Se tiene la rutina tx_simbolo definida en librería.asm que se encarga de procesar la salida de un símbolo y pasarlo a la rutina que controla el puerto tx_sci:

```
tx_simbolo              ;rutina para trasmitir 1 simbolo
            jclr #act_sh_cw,y:flags,tx_morse
tx_asci
move      x:tx_simb,r0
```

```
jmp tx_out
tx_morse
move        x:cw_simbout,r0
tx_out
move        #1,n0
jsr    tx_sci
rts
```

Para comprender el andamiento de la rutina tx_sci debemos profundizar un poco en el mecanismo que tiene el DSP para procesar una salida. Al igual que la recepción la transmisión cuenta con un registro dividido en tres.

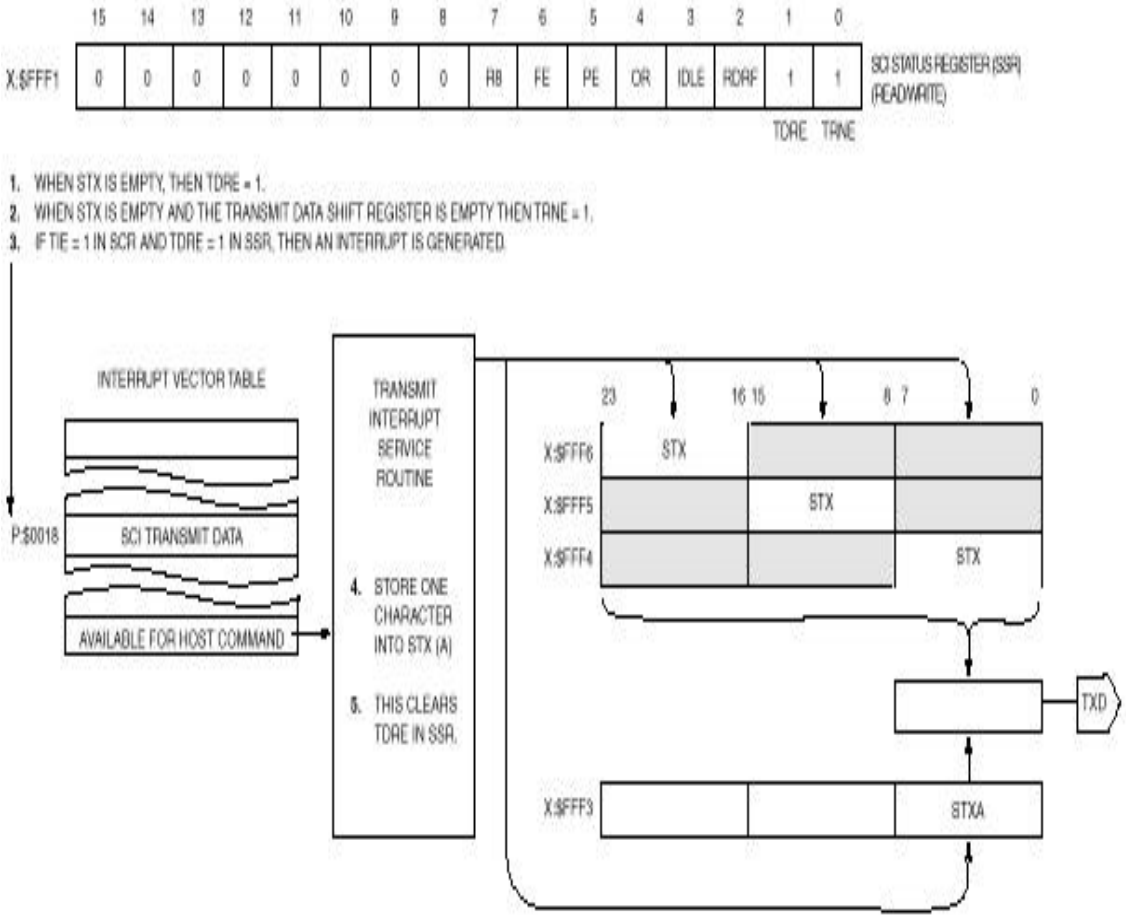


NOTES:
1. Bytes are masked on the fly.
2. STX is the same register decoded at three different addresses.

(b) Transmit Data Register

```
tx_sci

; r0 -> posicion de memoria del primer carácter del mensaje
; n0 -> cantidad de caracteres del mensaje
do      n0,finword
move  x:(r0)+,x1
move x1,x:M_STXL;escribe sci_ tx low byte
att   j
clr#M_TDRE,x:M_SSR,att       espera que Transmit Data Register se
vacie
```

```
att2 jset #M_TRNE,x:M_SSR,att2    espera que serial shift register
se vacie
finword     nop
  rts
```



Los Loops que aparecen en la rutina están pensados para que no se sobrescriban los datos. Primero se espera que los datos escritos en STXL pasen al Shift Register, lo que se avisa con el flag M_TDRE, y luego se espera hasta que se transmitan todos los bit alojados en el Shift Register. Cuando esto culmina se está en condiciones de enviar otro caracter. Observar que esos Loops fuerzan a esperar por el tiempo que tarde el mensaje en ser transmitido. De ahí la necesidad de transmitir lo más rápido posible para no interferir con el puerto SSI

5.4.5.5 Salida de Audio

El audio que el operador recibirá como respuesta del DSP será la salida del filtro Notch, por lo que escuchará la señal filtrada y optimizada del Morse que se está recibiendo en ese momento y que el Decodificador estará desplegando en pantalla.

Con esto buscamos dar una ayuda extra al operador para juzgar la precisión de la decodificación, como así también la calidad de la señal que se está procesando.

El audio es manejado por el CODEC que, mediante los conversores A/D y D/A, representa el nexo entre el mundo analógico exterior y el digital que compone el DSP.

El canal de comunicación del CODEC y el DSP se da a través de la interfase sincrónica SSI que forma parte del puerto C.

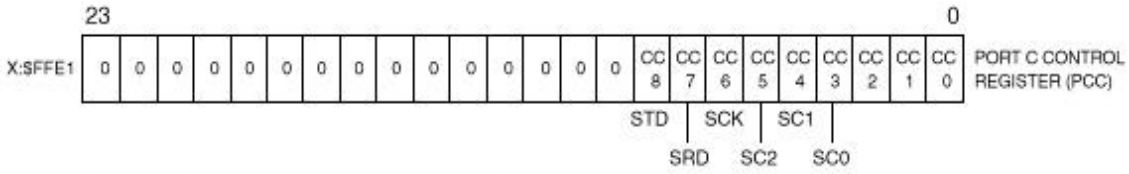
Esta comunicación involucra, como en el caso de la interfase SCI, un driver que pasaremos a detallar a continuación.

5.4.5.6 Interfase SSI.

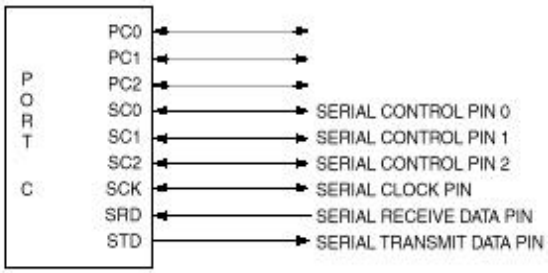
Esta interfase nos permite obtener los datos del CODEC mediante una interfase sincrónica full-duplex.

La inicialización de la comunicación es un proceso algo complejo que fue implementado a partir de la rutina ada_init.asm, que forma parte del software que acompaña a la placa de desarrollo. De esta surgieron los archivos codec_equ.asm (donde se encuentran definidas las variables) y codec_init (donde se inicializa el CODEC).

Vamos usar esta última para describir esta interfase.

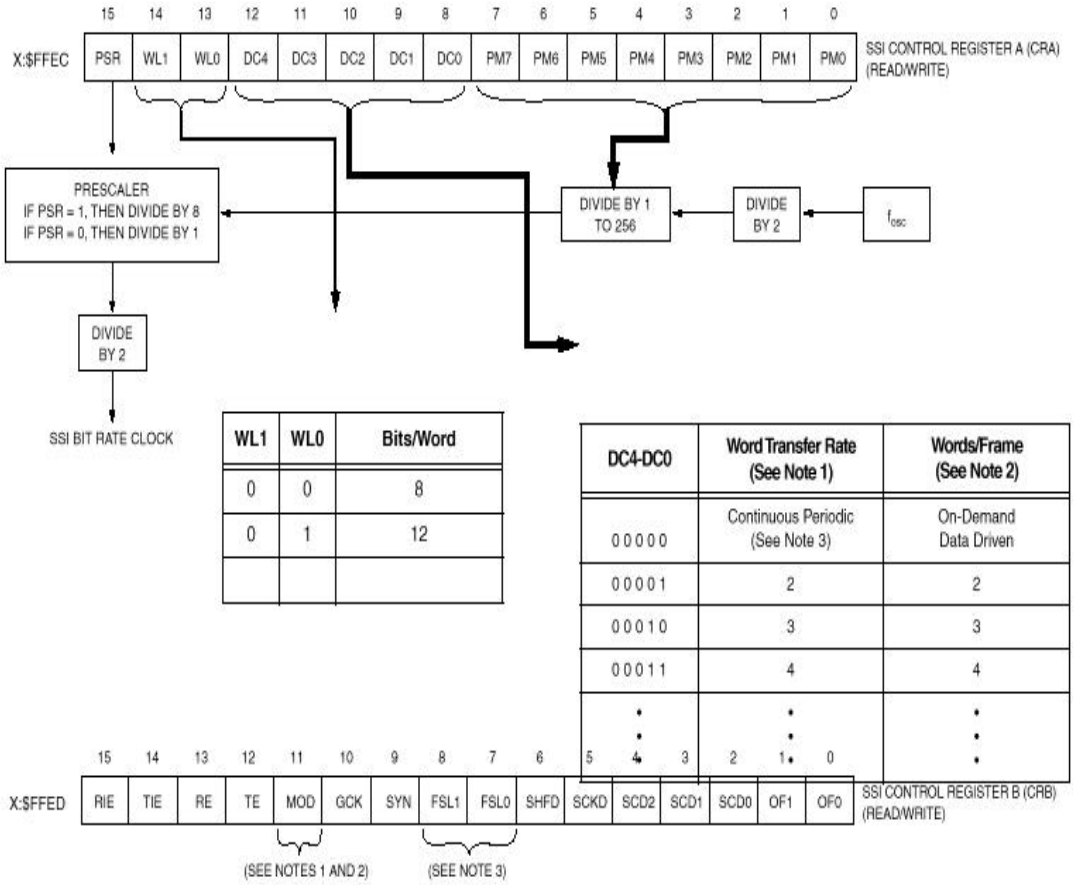


CCx	Function
0	Parallel I/O
1	Serial Interface




```
;     PortC usage: PCC REGISTER
;           bit8: SSI TX (from DSP to Codec)
;       bit7: SSI RX (from Codec to DSP)
;       bit6: SSI Clock
;       bit5: SSI Frame Sync
;       bit4: codec reset (from DSP to Codec)
;  bit3:
;       bit2: data/control bar
;           0=control
;           1=data

;*****************************************************************************
;*****                    initialize the CS4215 codec                  *****
;*****************************************************************************
;   1.  Configure the SSI port and the GPIO lines used (Data/Control  &
Reset)
;   2.  Select Control Mode and reset the codec (50ms @ 40MHz)
;   3.  Initialize the Transmit Data Buffer with Control Words (CLB=0)
;   4.  Enable SSI
;   5.  Wait until CLB in receive buffer = 0
;   6.  Set CLB = 1 in TX buffer
;   7.  Send 4 frames of data
;   8.  Disable SSI port
;   9.  Program SSI to receive Frame Sync and Clock
;  10.  Select Data Mode (Dats /Control = 1)
;  11.  Enable SSI port
;*****************************************************************************

org     p:codec_init

move    #RX_BUFF_BASE,x0
move    x0,x:RX_PTR                         ; Initialize the rx pointer
move    #TX_BUFF_BASE,x0
move    x0,x:TX_PTR                 ; Initialize the tx pointer
```

```
;----- initialize SSI -----
; Assuming the DSP56002 is running at 40 MHz,  this section selects a
;   2.5 MHz SSI clock to be sent from the 56002's SSI clock circuit
;   to the codec.  It also sets up 64-bit frame length (4 words per
;   frame x 16-bits per word), enables SSI TX & RX, enables TX & RX
;   interrupts and synchronous operation with bit-length frame sync.
;  crb ($fb30) -> bit 9  syn on -> mod 6 norm with internal gated clk
;                                bit 10 gck on
;                                bit 11 mod off
;             bit 5  sckd 1
;-------------------------------------------------------------------------

movep   #$0000,x:PCC    ;  turn off ssi port

        movep   #$4303,x:CRA    ;
;  PBR=1 ✍ prescaler X 8 fijo
; WL1-WL0 =10 ✍ largo de palabra 16 bits
; DC4 -DC0 =11 ✍ 4 palabras por frame
; PM7-PM0  =11 ✍ 3 x 8 =24 ✍ ;  44MHz/24 = 1.8MHz  slck

movep   #$FB30,x:CRB

; RIE =1 ✍ el DSP puede ser interrumpido cuando RDF=1 (receive data
register full) en  SSI status
```



NOTES:
1. NORMAL — MOD = 0
2. NETWORK — MOD = 1
3. FSL1 = 1, FSL0 = 0

Figure 11-49 SSI CRA Initialization Procedure

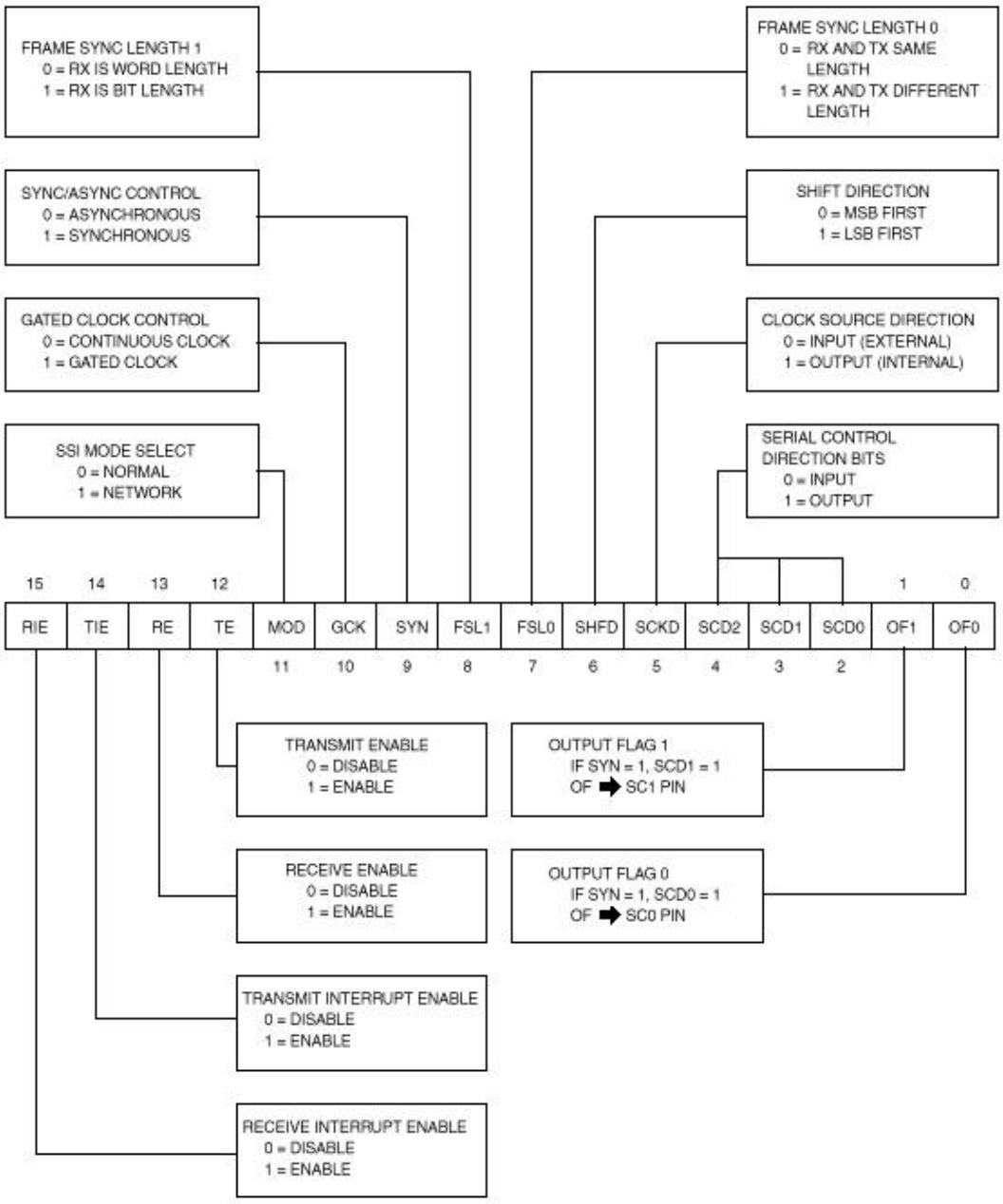


Figure 11-50 SSI CRB Initialization Procedure

; TIE =1 ✍ el DSP puede ser interrumpido cuando TDE=1 (trasmit data register empty)

; RE =1 ✍ habilita la recepción

; TE =1✍ habilita la transmisión

; MOD=0 ✍ modo normal , el frame rate divider impone la velocidad de transferencia

; SYN=1 ✍ transmisión y recepción se realizan sincrónicamente

; GCK=1 ✍ el reloj se pone solo cuando hay datos para transmitir

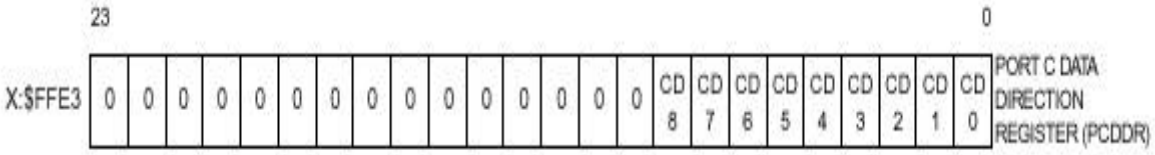
;FSL1-FSL0 =10 ✍ se utiliza para RX y TX el largo de un bit para sincronizar

;SHFD=0 ✍ se envía el bit mas significativo primero

;SCD3-SCD0=100 ✍ define sentido de los canales de control input

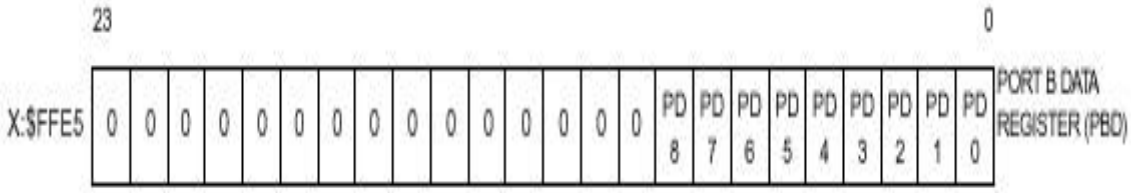
;OF1-OF0=00 ✍ output flags

movep #$14,x:PCDDR ; define direccion de pines de PCDR, PCD2 and PCD4 as outputs



CDx	Data Direction
0	Input
1	Output

movep #$0,x:PCD ; Data/Control and RESET = 0 ==> entra modo control



NOTE: Hardware and software reset clears PCC and

;----espera que se establezca el modo de contrl en el codec ----
do #500,_delay_loop
rep #2000 ; 100 us delay
nop
_delay_loop

bset #4,x:PCD ; RESET = 1 ✍ resetea el codec desde el DSP
movep #$3000,x:IPR ; Habilita interrupciones de nivel 2
movep #$01E8,x:PCC ; habilita puerto SSI

;--- Establece TX buffer con datos de control, define modo de operación valores en codec_equ.asm
move #CTRL_WD_12,x0
move x0,x:TX_BUFF_BASE
move #CTRL_WD_34,x0
move x0,x:TX_BUFF_BASE+1
move #CTRL_WD_56,x0
move x0,x:TX_BUFF_BASE+2
move #CTRL_WD_78,x0

```
move                x0,x:TX_BUFF_BASE+3

andi    #$FC,mr       ; Habilita interrupciones desde nivel 1 al 3

;
; CLB == 0 in TX Buffer,  wait for CLB == 1 in RX Buffer ✍ proceso para
sincronizar el canal
;
jclr    #3,x:SSISR,*          ; wait until rx frame bit==1
jset    #3,x:SSISR,*          ; wait until rx frame bit==0
jclr    #3,x:SSISR,*          ; wait until rx frame bit==1
jset    #18,x:RX_BUFF_BASE,*  ; loop until CLB set

;
; CLB == 1 in RX Buffer, send 4 frames and then disable SSI
;
bset    #18,x:TX_BUFF_BASE          ;set CLB
do      #4,_init_loopB              ; Delay as 4 full frames to pass
jclr    #2,x:SSISR,*                ; wait until tx frame bit==1
jset    #2,x:SSISR,*                ; wait until tx frame bit==0
_init_loopB
movep   #0,x:PCC                    ;reset SSI port (disable SSI...)

; fin del proceso de sincronizado ----------------------------------------------
;
;    Ahora CLB debe ser 1 -✍ se reprograma  fsync and sclk como
entradas
;
;     movep   #$4303,x:CRA        ; 16bits,4 word/frame, /2/4/2=2.5
MHz
movep   #$FB00,x:CRB  ; rcv,xmt & int ena,netwk,syn,sclk==inp,msb 1st
movep   #$14,x:PCD      ; Data/Control pin = 1  ==> data mode
movep   #$01EB,x:PCC    ;  Habilita puerto SSI y SCI
ori #$03,mr           ;Permite interrupciones de nivel 3 solamente hasta
completar inicialización del DSP
```

Una vez inicializada la comunicación con el CODEC y liberadas las interrupciones, el DSP comenzará a recibir las muestras como se definieron: velocidad de muestreo 8KHz, estereo, muestras de 16 bit con signo.

Es importante saber que al momento de la configuración:

1. Hay que tener sumo cuidado con el parámetro Monitor que presenta el CODEC. Este parámetro atenúa la entrada y la suma directamente a la salida de la decodificación actual del CODEC. Si este parámetro disminuye demasiado el CODEC se desestabiliza y se distorsiona la señal de salida.

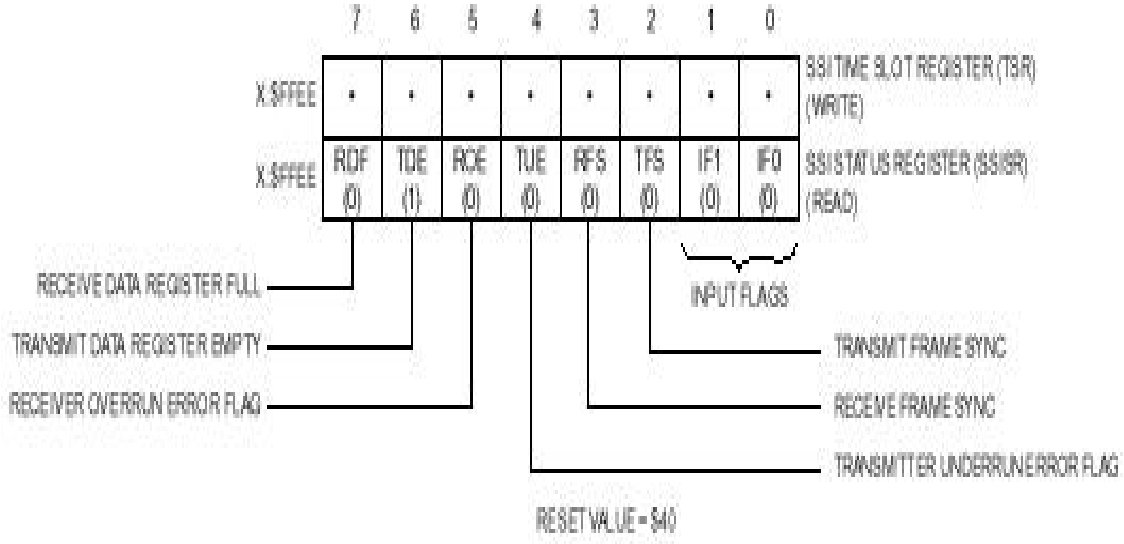
2. Se debe prevenir el problema de las tierras comunes cuando se utiliza como generador de señales de

prueba y monitor de la interfase gráfica el mismo PC. La Tierra de señal, la tierra del SCI y del OnCE, son las mismas en el DSP. Sin embargo en los PC hemos encontrado diferencias entre la tierra de la tarjeta de sonido y el puerto COM. La solución es colocar una resistencia en serie con la tierra del puerto COM para que absorba las diferencia de tensiones.

3. Tener en cuenta el uso de los pre-amplificadores de entrada como fuente de ganancia extra (observar que la pre-amplificación se realiza antes de la conversión A/D, por lo que no se pierden cifras significativas).

Una vez inicializado el CODEC, el puerto SSI y habilitadas las interrupciones, el sistema está pronto para procesar las muestras.

Nos valemos del siguiente código para saber cuándo hay un dato válido para ser leído del puerto:



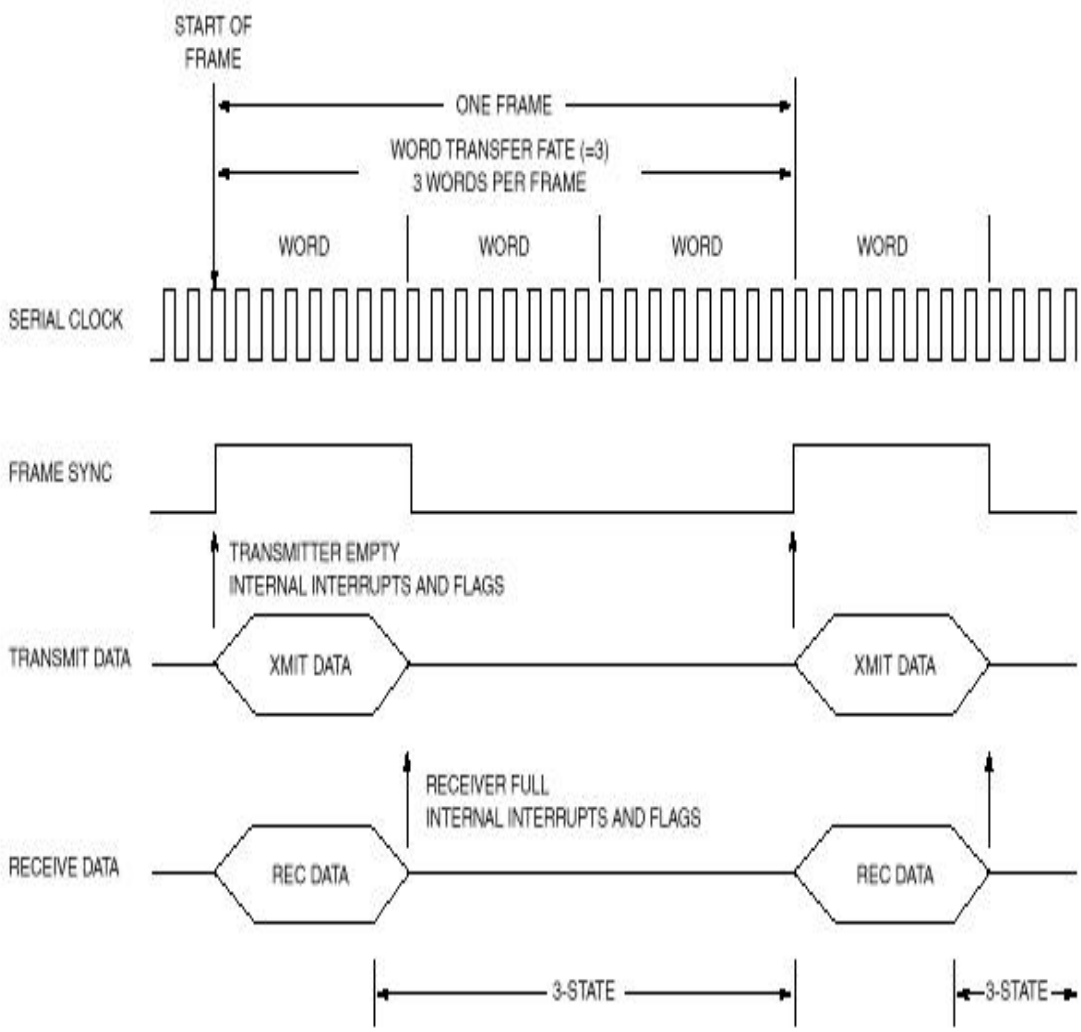


Figure 11-61 Synchronous Communication

```
in_data  (librería.asm)
                                                                    ;
        jset  #2,x:SSISR,*    ;wait for frame sync to pass
jclr    #2,x:SSISR,*    ;wait for frame sync


move    x:RX_BUFF_BASE,a        ;canal izq
move    x:RX_BUFF_BASE+1,b        ;canal derecho
```

Como elegimos el modo sincrónico de transmisión y recepción el intercambio de datos desde y hacia el CODEC se hace al mismo tiempo, por lo que usamos el flag de transmisión para sincronizar (bit 2 del SSISR).

Como se puede observar en la figura, hay que esperar el pulso de sync para comenzar el intercambio de datos.

Al utilizar el modo normal una palabra es intercambiada por frame y es el frame rate divider que fija la tasa de transferencia.

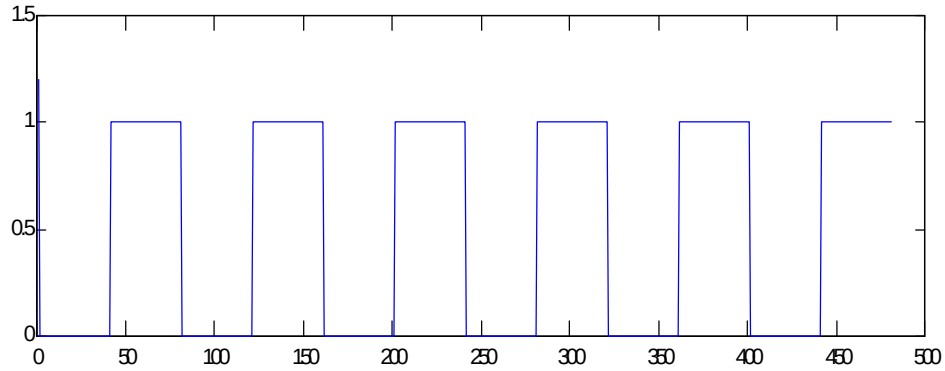
Al pasar el pulso de sincronía están en el buffer los datos recibidos.

6 SIMULACIÓN Y ENSAYO DE DETECTOR

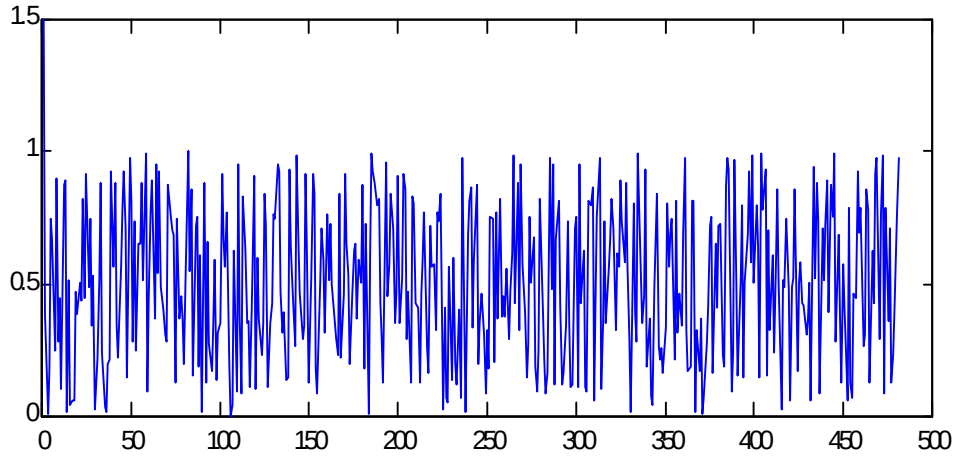
6.1 Pruebas de Perfomance de Decodificador Binario

Tratamos de simular la performance del detector binario implementado, realizando pruebas que buscaban forzar errores de comportamiento del mismo. El ruido que está presente aquí es aquel que paso la etapa tanto del filtro notch como la del filtro de media movil.

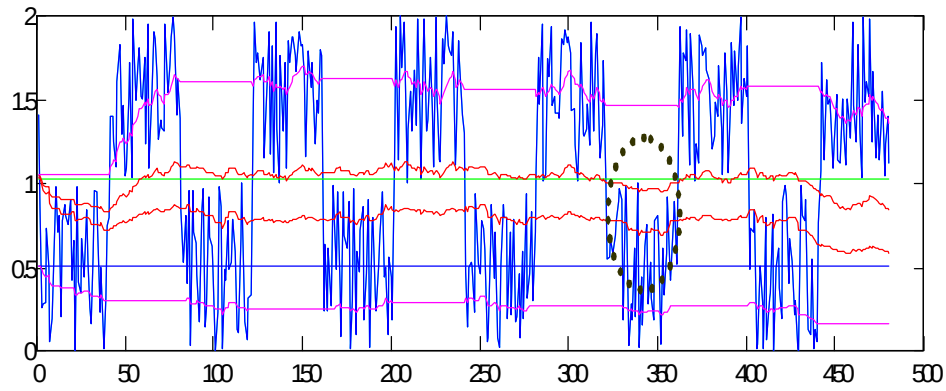
1 Generamos ruido cuya amplitud variara aleatoriamente y en forma acentuada. Las simulaciones se realizaron en Matlab con algoritmos implementados en forma idéntica al que ejecuta el DSP.



El mensaje de prueba es un tren de pulsos de amplitud unidad.

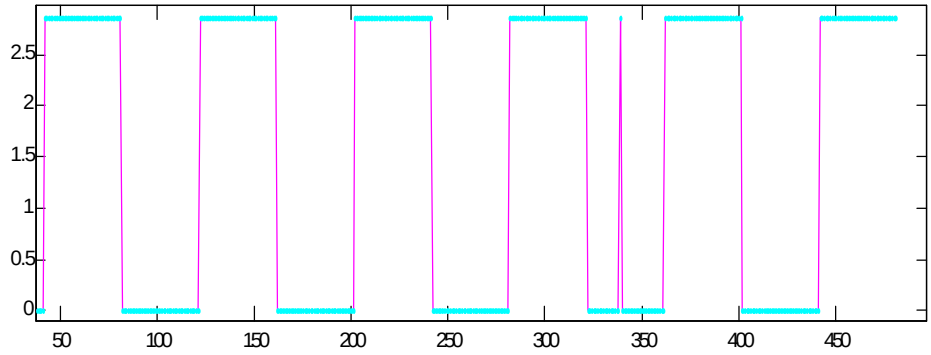


Ruido que distorsionará la señal es aleatorio de distribución uniforme entre 0 y 1.



Al sumar el tren de pulsos con el ruido distorsionante obtenemos la señal a decodificar.

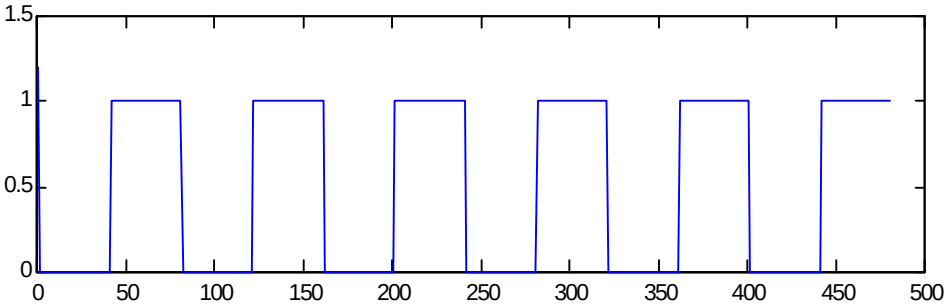
En la figura se pueden apreciar los umbrales calculados mediante el algoritmo adaptativo. En color Lila graficamos los umbrales máxima amplitud y umbral de ruido, y en rojo los umbrales de comparación.



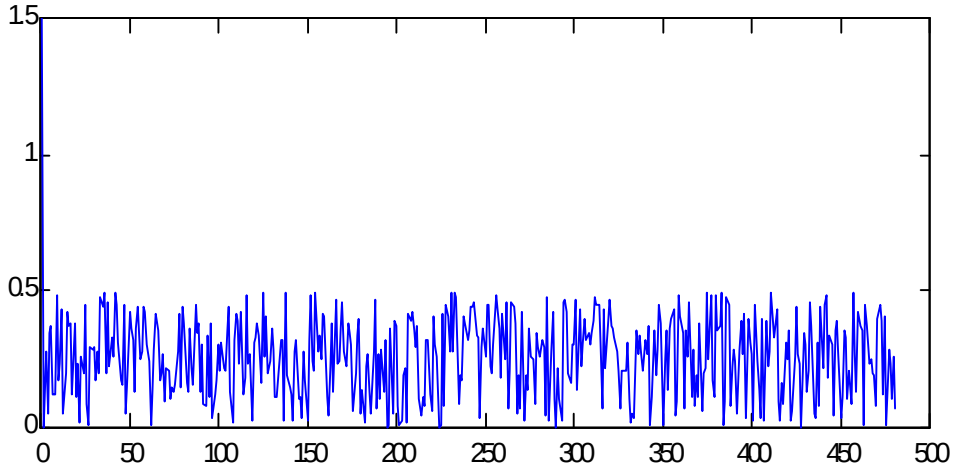
El resultado del decodificador coincide con el mensaje original, por lo que la decodificación fue correcta. Observar que el largo de los pulsos es similar a la original.

Sólo en un punto se generó un pulso espureo, como resultado de una variación brusca que cruzó ambos umbrales. Este error es corregido en capas superiores.

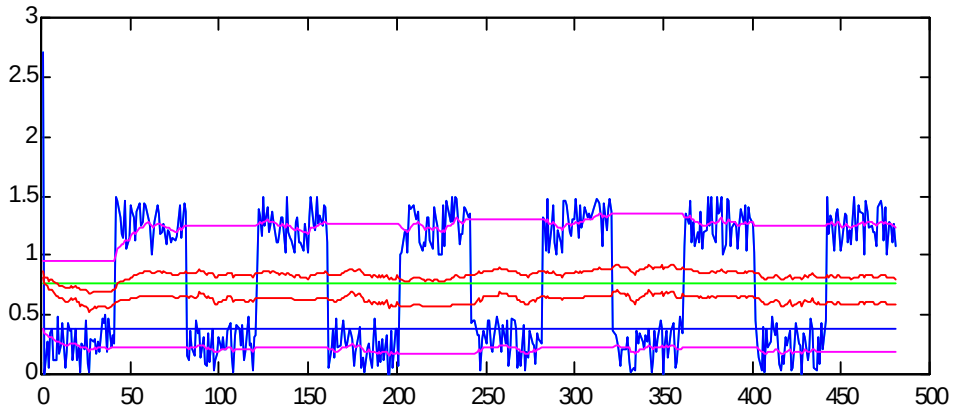
Si la señal de ruido tiene un amplitud máximo de 0.5.



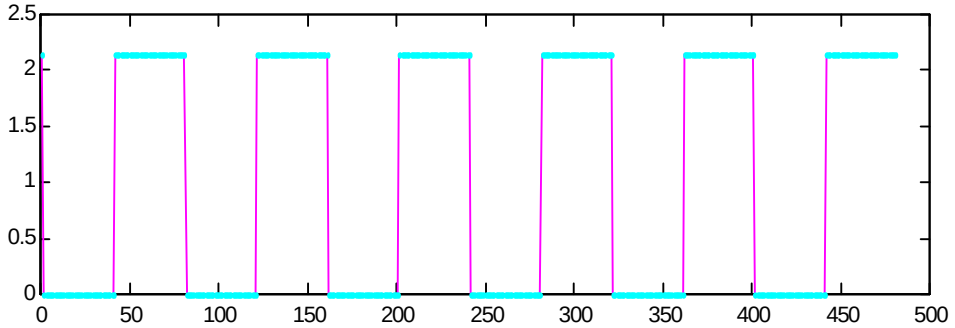
Señal a decodificar.



Ruido que se suma a la señal.

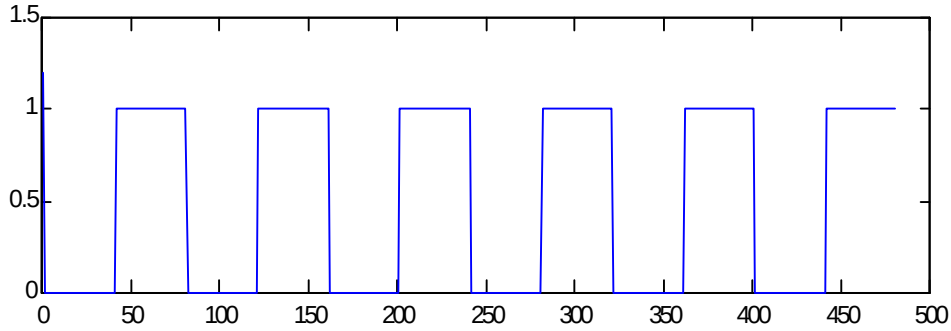


Señal resultante y umbrales de detección.

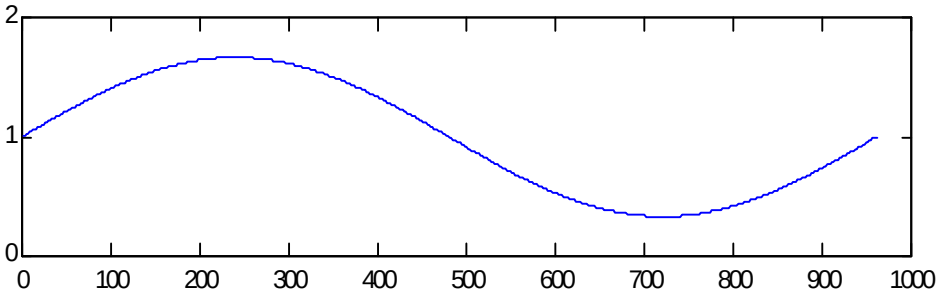


Señal decodificada, en este caso el pulso espureo no se genera.

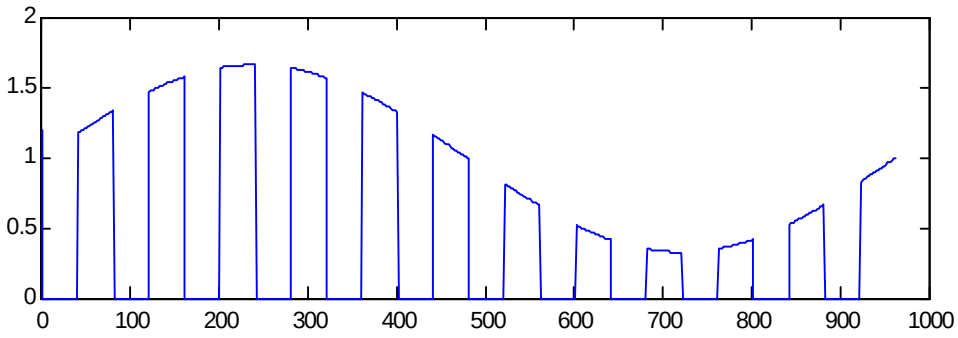
2 Generamos variaciones de amplitud en el mensaje original y verificamos el comportamiento de la adaptación de los umbrales y la decodificación.



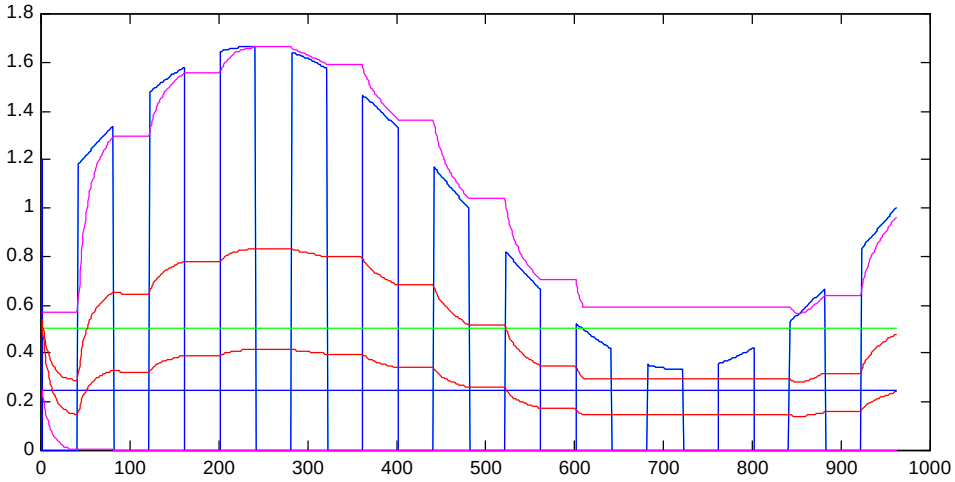
Señal original, tren de pulso de amplitud constante 1.



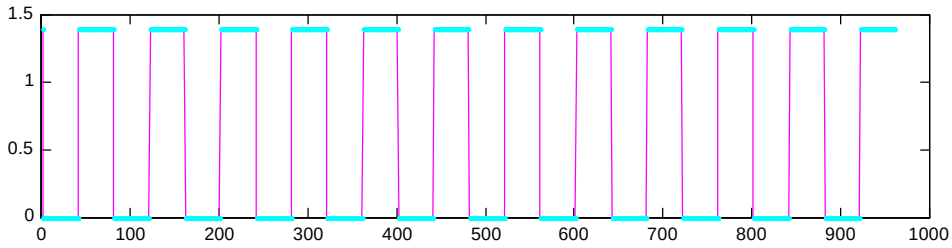
Variación de volumen que modifica la señal.



Señal modificada, se aprecian variaciones importantes en la potencia de cada pulso.

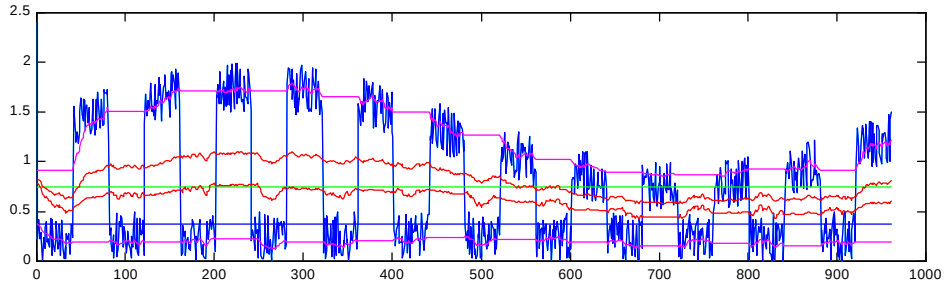


Se observa el andamiento de los umbrales, verificándose su adaptación a variaciones de amplitud de la señal. Por lo observado durante el estudio de las señales este tipo de degradación de amplitud no es un proceso rápido, es decir involucra una degradación paulatina de por lo menos medio pulso durante una transmisión de Morse a velocidad media.

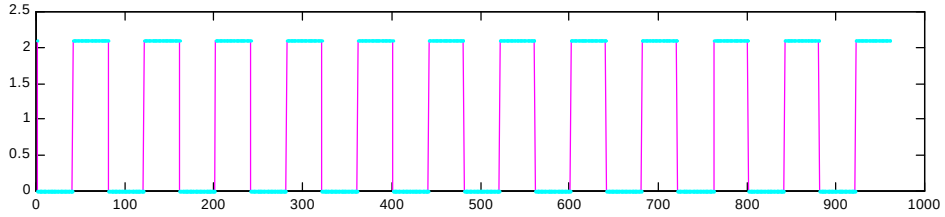


Señal decodificada, sin errores y manteniendo el largo de los pulsos constante.

3 Combinamos ambos efectos con el fin de observar el comportamiento del detector.



La señal de entrada al decodificador presentando ruido y variación de amplitud.



Resultado correcto.

7 APÉNDICE

7.1 Guía Rápida De Uso

En este capítulo vamos a indicar como conectar el DSP y ajustarlo para lograr una optima recepción.

Componentes del KIT de Detección CW:

- 1 gabinete donde se encuentra alojado el DSP y su fuente.
- 1 cable serial A M/H derecho con conectores DB9.
- 1 cable serial B M/M cruzado con conectores DB9.
- 3 cables de audio con 1 jack estéreo de 5mm y 2 RCA (1 por canal).

Conexión de los componentes:

a) Conectar un extremo del cable serial A al conector macho que se encuentra en el gabinete y el otro extremo al COM1 delPC.

b) Conectar un extremo del cable serial B al conector hembra que se encuentra en el gabinete y el otro extremo al COM2 del PC.

c) Conectar a la salida RCA marcada como IN uno de los cables de audio y el otro extremo a la salida de la fuente de señal.

d) Conectar a la salida RCA marcada como OUT el otro de los cables de audio y el otro extremo a un parlante o amplificador.

e) Conectar el gabinete a la alimentación de 220V.

7.1.1 Configuración Inicial del Software

a) Instalar el software de desarrollo Debug-56k que viene con el Kit desarrollo. Requerimiento mínimα PC con sistema operativoWindows 3.11

b) Crear una sesión de Hyperterminal o similar con los siguientes parámetros:

Puerto: COM2
Velocidad: 115k
Bit de Parada: 1

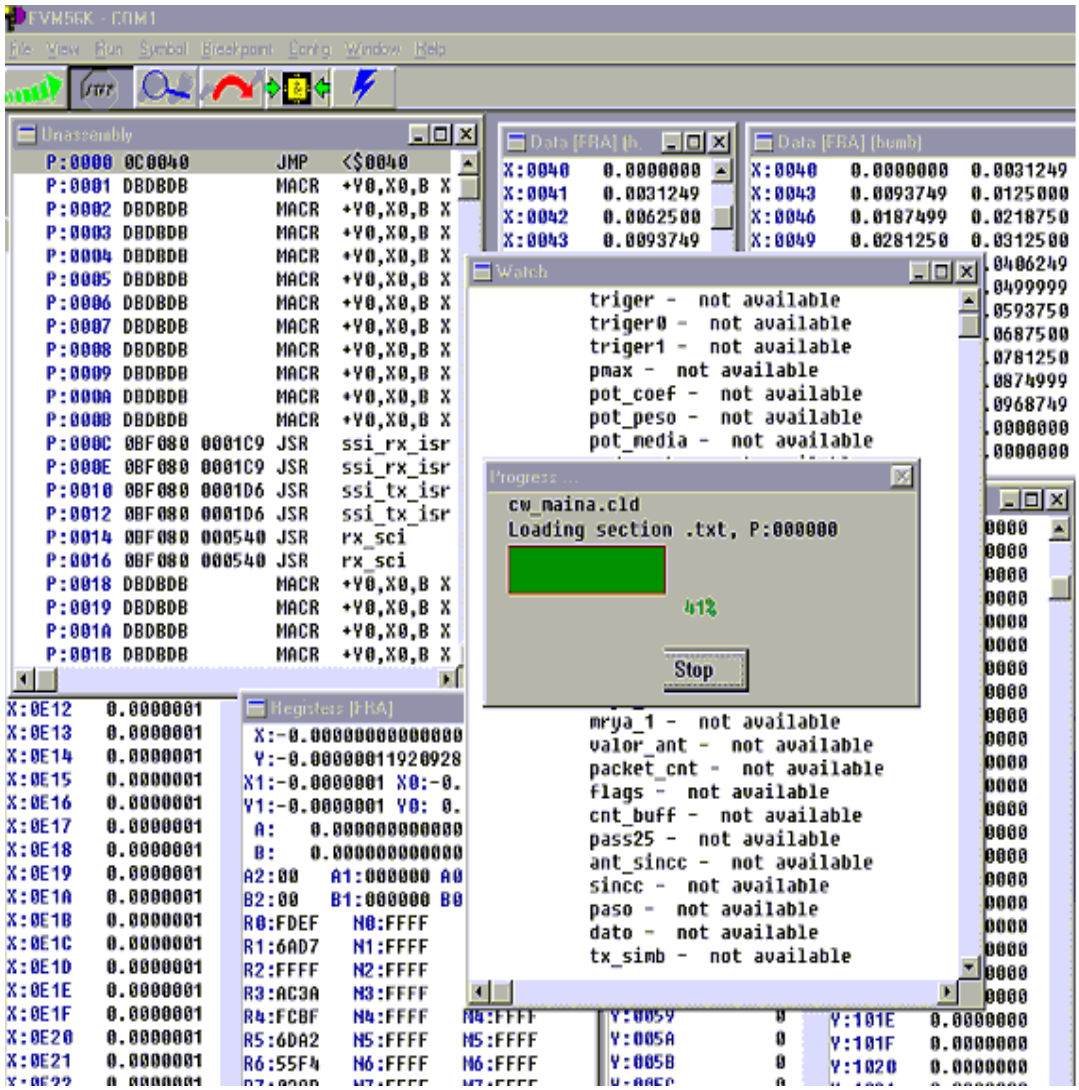
Bit de datos: 8
Paridad: Ninguna
Control de Flujo: Ninguno

7.1.2 Operación del Detector

a) Para cargar el software al DSP utilizamos el Debug-56k. Ejecute el programa, seleccione la opción Load y cargue el archivo maina.cld. Este archivo se alojaáa en la memoria volátil del DSP. Si el programa no se abre correctamente:

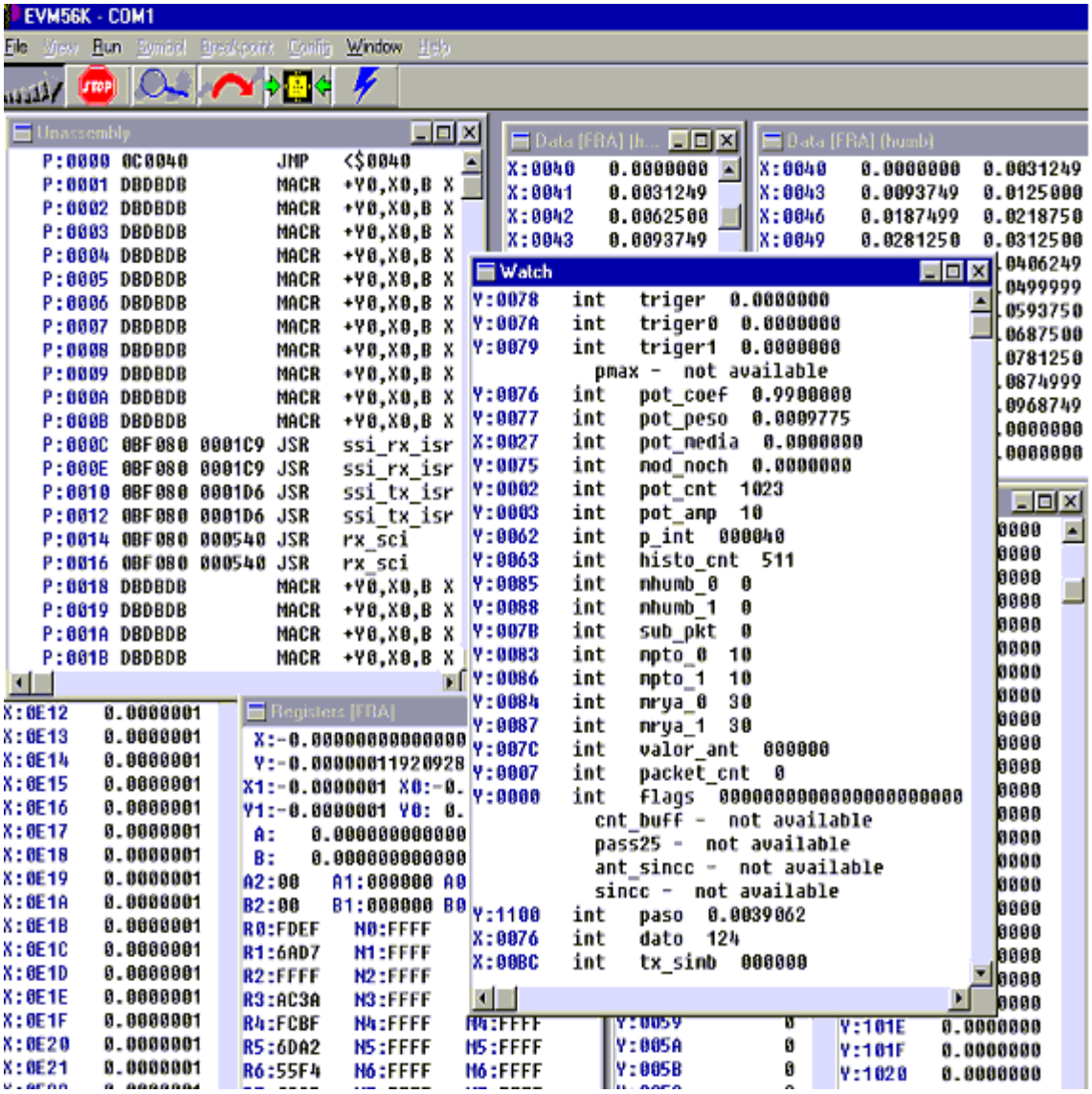
?? revise las conexiones.
?? cierre la aplicación que este usando el COM1.
?? verifique que el DSP está encendido.

El proceso de carga del archivo al DSP lleva un par de minutos.



b) Abrir la sesión de Hyperterminal creada con anterioridad.

c) Seleccionar el Botón RUN en el Debug-56k. En este paso, si todo funciona correctamente, se deberá ver en la pantalla del Hyperterminal el menú del detector.



d) Para comenzar a detectar elegir la opción de detección que se prefiera:

GO MORSE : se decodifica en puntos y rayas
GO ASCII: se decodifica en código ascii

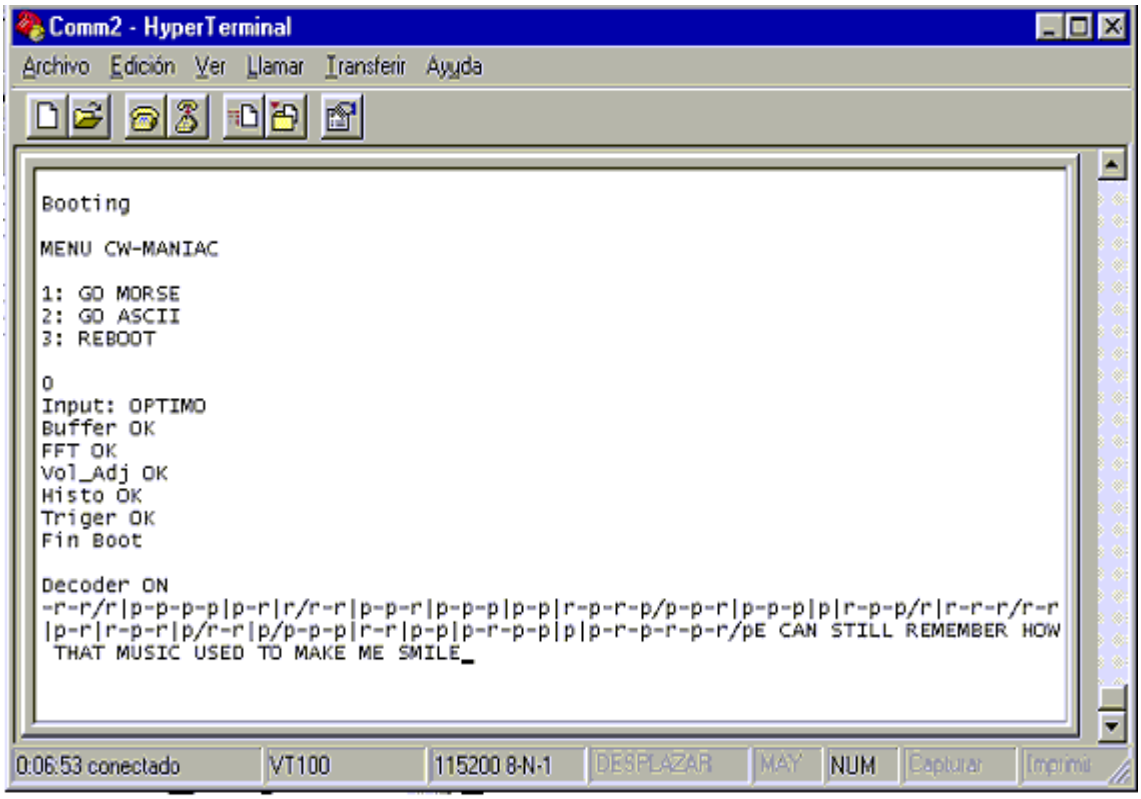
De cualquier forma, una vez iniciada la detección es posible cambiar de MORSE a ASCII sin parar la detección, sólo basta presionar el número de la opción deseada. Una vez elegida la opción de decodificación el DSP comienza el proceso de ajuste inicial.

Si se despliega :

INPUT : SATURA ✍ disminuir el volumen de la fuente de audio
INPUT : RUIDO ✍ aumentar el volumen
INPUT: OPTIMO ✍ el volumen de la entrada es el correcto

Una vez logrado el ajuste del volumen de entrada el DSP comienza el proceso de aprendizaje y caracterización de la señal. De aquí en más no es necesario ningún ajuste por parte del operador.

Una vez que se termina el booteo es activado el decodificador. En este momento se podrán comenzar a leer los mensajes en pantalla y a oír el audio filtrado por el noch al mismo tiempo. La figura muestra el despliegue típico de la aplicación .



Para activar el menú durante la detección presionar la letra m: la decodificación se pausará hasta que se elija una opción.

La opción REBOOT se utiliza para reiniciar el DSP en caso de querer cambiar de señal, esta opción lleva los parámetros al default . Observar que no es necesario volver a cargar el software maina.cld.

8 BIBLIOGRAFÍA

Digital Signal Porcessing

Richard A. Roberts – Clifford t. Mullis
Addison-Wesley Publishing Company
Mayo 1987

Comunications Systems

A. Bruce Carlson
McGraw-Hill
Third Edition 1986

Digital Signal Processor Manual DSP56002
Motorola