UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA:

DESARROLLO DE UN RECEPTOR DE RADIO PROGRAMADA EN PC MEDIANTE INTERFAZ USB

INGENIERÍA DE TELECOMUNICACIÓN

Málaga, 2005 José Luis Blanco Claraco

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D. _____

D. _____

D. _____

para juzgar el Proyecto Fin de Carrera titulado:

DESARROLLO DE UN RECEPTOR DE RADIO PROGRAMADA EN PC MEDIANTE INTERFAZ USB

del alumno D. José Luis Blanco Claraco

dirigido por D. Mariano Fernández Navarro

ACORDÓ POR: _____ OTORGAR LA CALIFICACIÓN DE

_____

y, para que conste, se extiende firmada por los componentes del Tribunal, la presente diligencia.

Málaga, a ____de_____de 2005

El Presidente: El Secretario: El Vocal:

Fdo. _____ Fdo. _____ Fdo. _____

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UN RECEPTOR DE RADIO PROGRAMADA EN PC MEDIANTE INTERFAZ USB

REALIZADO POR:

José Luis Blanco Claraco

DIRIGIDO POR:

Mariano Fernández Navarro

DEPARTAMENTO DE: Ingeniería de Comunicaciones.

TITULACIÓN: Ingeniería de Telecomunicación

PALABRAS CLAVES: Radiofrecuencia, demodulación en cuadratura, USB, controlador, adquisición.

RESUMEN:

En este proyecto se diseña e implementa un conjunto de hardware y software que permiten la recepción de señales de radiofrecuencia genéricas mediante el uso de un ordenador conectado mediante interfaz USB al radioreceptor hardware. El planteamiento del proyecto es el de diseñar un receptor genérico en cuanto al tipo de modulación de la señal de radio, dando la posibilidad de demodular la señal mediante el uso del software adecuado en el ordenador, permitiendo la recepción tanto de información analógica como digital. El entorno software que se desarrolla ofrece una arquitectura abierta en cuanto a módulos demoduladores, permitiendo fácilmente la extensión futura según las necesidades de procesar un determinado tipo de señal.

Málaga, Julio de 2005

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN 8

2. OBJETIVOS DEL PROYECTO 12

3. DISEÑO CONCEPTUAL 14

3.1. LA DEMODULACIÓN EN CUADRATURA 14

3.2. DISEÑO GLOBAL DEL SISTEMA 17

3.3. ALTERNATIVAS DE DISEÑO 19

4. DISEÑO DEL HARDWARE ANALÓGICO 24

4.1. FILTRO DE PRESELECCIÓN 25

4.2. AMPLIFICADOR RF DE BAJO RUIDO 30

4.3. REDES ADAPTADORAS DE IMPEDANCIAS 33

4.4. AMPLIFICADOR RF DE GANANCIA VARIABLE 43

4.5. DEMODULADOR EN CUADRATURA 47

4.6. FILTROS DE BANDA BASE 53

4.7. AMPLIFICADORES DE BANDA BASE 59

4.8. SINTETIZADOR DE FRECUENCIAS (PLL) 61

4.9. OSCILADOR VARIABLE (VCO) 76

4.10. ALIMENTACIÓN 80

5. DISEÑO DEL HARDWARE DIGITAL 82

5.1. MICROCONTROLADOR 82

5.2. MEMORIA FIFO 87

5.3. LÓGICA DE INTERFAZ FIFO-USB 92

5.4. INTERFAZ USB 98

5.5. CONVERSORES A/D (ADC) 102

5.6. CONVERSORES D/A (DAC) 105

5.7. DISPLAY LCD 106

6. DISEÑO FÍSICO Y MECÁNICO 112

6.1. DISEÑO 112

6.2. HERRAMIENTAS UTILIZADAS 119

7. DISEÑO SOFTWARE 122

7.1. DISEÑO DEL PROTOCOLO DE COMUNICACIONES 122

7.2. DISEÑO DEL FIRMWARE 131

7.3. DISEÑO DE "*SWRADIOSTUDIO*" 150

7.4. DEMODULADORES SOFTWARE 175

8. DISEÑO DEL PROGRAMADOR 182

8.1. INTRODUCCIÓN 182

8.2. DISEÑO HARDWARE 182

8.3. PROTOCOLO DE PROGRAMACIÓN 184

8.4. DISEÑO SOFTWARE 186

9. ESTIMACIÓN TEÓRICA DE PARÁMETROS DEL SISTEMA 190

9.1. GANANCIA TOTAL 190

9.2. PUNTO DE COMPRESIÓN DE 1DB 192

9.3. POTENCIA DE RUIDO A LA SALIDA 194

9.4. SENSIBILIDAD 197

9.5. MARGEN DINÁMICO 197

9.6. RECHAZO A CANAL ADYACENTE 198

10. PLAN DE PRUEBAS 202

10.1. PRUEBA DE LA CIRCUITERÍA DIGITAL 202

10.2. PRUEBAS AL PLL 204

11. MEDICIONES Y RESULTADOS 206

11.1. FLUJO DE TRAMAS 206

11.2. SINTETIZADOR DE FRECUENCIAS: VCO Y PLL 206

11.3. CONVERSORES A/D 219

11.4. FILTRO DE PRESELECCIÓN Y LNA 222

11.5. AMPLIFICADOR VGA 224

11.6. FILTROS DE BANDA BASE 229

11.7. MEDIDAS DEL RECEPTOR COMPLETO 233

11.8. PRUEBAS CON LOS DEMODULADORES SOFTWARE 238

12. CONCLUSIONES Y LINEAS DE FUTURO 246

13. REFERENCIAS 248

ANEXOS 250

A1. PLANOS ELÉCTRICOS Y MECÁNICOS 251

A2. PLACAS PCB 266

A3. HOJAS DE CARACTERÍSTICAS 272

A4. LISTADOS DE CÓDIGO FUENTE 287

ÍNDICE DE FIGURAS

FIGURA 1: TRES FORMAS DE IMPLEMENTAR UN RECEPTOR SUPERHETERODINO (A,B,C) Y EL DEMODULADOR EN CUADRATURA (D). 9
FIGURA 2: EFECTOS DEL ALIASING EN LA DIGITALIZACIÓN DE SEÑALES. 10
FIGURA 3: CONVERSIÓN DE PASO BANDA A BANDA BASE EQUIVALENTE. 15
FIGURA 4: IMPLEMENTACIÓN ZIF REALIZABLE FÍSICAMENTE. 16
FIGURA 5: DIAGRAMA DE BLOQUES GLOBAL DEL RADIORRECEPTOR A DISEÑAR. 17
FIGURA 6: DIAGRAMA DE IMPEDANCIAS PARA EL FILTRO DE PRESELECCIÓN. 25
FIGURA 7: RESPUESTA EN FRECUENCIA E IMPLEMENTACIÓN IDEAL DEL FILTRO. 26
FIGURA 8: IMPLEMENTACIÓN DEL FILTRO DE PRESELECCIÓN. 27
FIGURA 9: RESPUESTA SIMULADA DEL FILTRO DE PRESELECCIÓN. 28
FIGURA 10: COEFICIENTE DE REFLEXIÓN DEL FILTRO. 28
FIGURA 11: DIAGRAMA DE IMPEDANCIAS PARA EL LNA. 30
FIGURA 12: INTEGRADO ELEGIDO COMO LNA. 30
FIGURA 13: GANANCIA EN FUNCIÓN DE LA CORRIENTE. 31
FIGURA 14: IMPLEMENTACIÓN DEL LNA. 31
FIGURA 15: PÉRDIDA DE POTENCIA EN FUNCIÓN DE LA INDUCTANCIA DEL CHOQUE. 32
FIGURA 16: RED ADAPTADORA A LA ENTRADA DEL VGA. 33
FIGURA 17: RED ADAPTADORA A LA SALIDA DEL VGA. 33
FIGURA 18: RED ADAPTADORA A DISEÑAR. 34
FIGURA 19: REDES ADAPTADORAS PASO BAJO Y PASO ALTO. 35
FIGURA 20: RED ADAPTADORA PASO ALTO Y VALORES IDEALES. 36
FIGURA 21: IMPLEMENTACIÓN REAL DE LA RED ADAPTADORA PASO ALTO. 36
FIGURA 22: FUNCIÓN DE TRANSFERENCIA DE POTENCIA. 37
FIGURA 23: COEFICIENTE DE REFLEXIÓN A LA ENTRADA DEL ADAPTADOR. 38
FIGURA 24: FACTOR DE POTENCIA ENTREGADA A LA CARGA. 38
FIGURA 25: RED ADAPTADORA PASO BAJO Y VALORES IDEALES. 39
FIGURA 26: IMPLEMENTACIÓN REAL DE LA RED ADAPTADORA PASO BAJO. 39
FIGURA 27: FUNCIÓN DE TRANSFERENCIA DE POTENCIA. 40
FIGURA 28: COEFICIENTE DE REFLEXIÓN A LA ENTRADA DEL ADAPTADOR. 40
FIGURA 29: FACTOR DE POTENCIA ENTREGADA A LA CARGA. 41
FIGURA 30: FUNCIÓN DE TRANSFERENCIA DE POTENCIA COMBINADA. 41
FIGURA 31: TRANSFERENCIA DE POTENCIA COMBINADA. 42
FIGURA 32: DIAGRAMA INTERNO DEL VGA. 43
FIGURA 33: GANANCIA EN FUNCIÓN DE LA FRECUENCIA Y TENSIÓN DE CONTROL. 43

FIGURA 34: IMPLEMENTACIÓN DEL VGA. 44
FIGURA 35: LOS DOS MODOS DE GANANCIA DEL VGA. 45
FIGURA 36: FRECUENCIA DE CORTE INFERIOR DEL VGA SEGÚN EL CONDENSADOR DE DESACOPLE. 46
FIGURA 37: LAS IMPEDANCIAS DE ENTRADA Y SALIDA PARA EL VGA. 46
FIGURA 38: DIAGRAMA DE IMPEDANCIAS PARA EL VGA. 46
FIGURA 39: INTEGRADO ELEGIDO COMO DEMODULADOR. 47
FIGURA 40: CONEXIONES DEL DEMODULADOR CON OTROS ELEMENTOS DEL RECEPTOR. 47
FIGURA 41: IMPEDANCIA DE ENTRADA AL DEMODULADOR. 48
FIGURA 42: ADAPTADOR A LA ENTRADA DEL LO Y PÉRDIDAS POR DESADAPTACIÓN. 49
FIGURA 43: IMPLEMENTACIÓN DEL DEMODULADOR. 50
FIGURA 44: GANANCIA (SIN INCLUIR EL ÚLTIMO AMPLIFICADOR BANDA BASE) EN FUNCIÓN DEL VOLTAJE PARA EL DEMODULADOR. 51
FIGURA 45: EJEMPLO DE FILTRO DE BANDA BASE (NO ES EL USADO EN ESTE RECEPTOR) 53
FIGURA 46: EFECTO DEL ALIASING SOBRE EL FILTRO DE BANDA BASE. 55
FIGURA 47: RESPUESTA DEL FILTRO DE BANDA BASE IDEAL 56
FIGURA 48: FILTRO DE BANDA BASE IDEAL 57
FIGURA 49: IMPLEMENTACIÓN DEL FILTRO *ANTIALIASING* ELÍPTICO DE QUINTO ORDEN. 57
FIGURA 50: RESPUESTA EN FRECUENCIA DEL FILTRO *ANTIALIASING*. 58
FIGURA 51: AMPLIFICADOR DIFERENCIAL Y LA CONFIGURACIÓN USADA. 59
FIGURA 52: AMPLIFICADOR DE BANDA BASE. 60
FIGURA 53: CARACTERÍSTICAS DEL *VARICAP* ELEGIDO EN EL VCO A BJT. 62
FIGURA 54: IMPLEMENTACIÓN DEL VCO A BJT. 62
FIGURA 55: RESULTADOS DE LAS MEDIDAS AL VCO A BJT. 63
FIGURA 56: ESQUEMA INTERNO DEL SINTETIZADOR DE FRECUENCIAS. 65
FIGURA 57: SELECCIÓN DEL TIPO DE FILTRO DE LAZO. 68
FIGURA 58: PARÁMETROS DE DISEÑO DEL FILTRO DE LAZO Y VALORES COMERCIALES USADOS FINALMENTE. 69
FIGURA 59: PARÁMETROS FINALES DEL FILTRO DE LAZO. 69
FIGURA 60: CRONOGRAMA MOSTRANDO EL FUNCIONAMIENTO DEL BUS SPI DEL PLL 70
FIGURA 61: EL REGISTRO DE CONTROL 71
FIGURA 62: EL REGISTRO DEL DIVISOR B 72
FIGURA 63: EL REGISTRO DEL DIVISOR R 72
FIGURA 64: REGULACIÓN Y FILTRADO DE ALIMENTACIÓN DEL PLL 74
FIGURA 65: ADAPTADOR BIDIRECCIONAL 75
FIGURA 66: ADAPTADOR DE NIVELES LÓGICOS 75
FIGURA 67: RANGO DE FRECUENCIAS SEGÚN EL VALOR DE LA INDUCTANCIA 76

FIGURA 68: FRECUENCIA DE OSCILACIÓN EN FUNCIÓN DEL VOLTAJE PARA EL VCO INTEGRADO 77
FIGURA 69: SENSIBILIDAD KV EN FUNCIÓN DEL VALOR DE LA INDUCTANCIA 78
FIGURA 70: INDUCTANCIA DE ADAPTACIÓN DE LA SALIDA DEL VCO 79
FIGURA 71: CONEXIÓN FINAL DEL PLL. 79
FIGURA 72: CHOQUE COLOCADO A LA ENTRADA DE LA ALIMENTACIÓN 81
FIGURA 73: DIAGRAMA DE BLOQUES INTERNO DEL MICROCONTROLADOR 83
FIGURA 74: EL MICROCONTROLADOR Y SUS CONEXIONES. 84
FIGURA 75: CRONOGRAMA TÍPICO DEL BUS SPI. 85
FIGURA 76: ESQUEMA DEL FLUJO DE DATOS EN EL RECEPTOR 87
FIGURA 77: FLUJO DE DATOS DURANTE LECTURA DEL CHIP USB 88
FIGURA 78: FLUJO DE DATOS DURANTE ESCRITURA DESDE EL MICROCONTROLADOR 89
FIGURA 79: FLUJO DE DATOS DURANTE ESCRITURA DESDE LOS ADC 89
FIGURA 80: MEMORIA FIFO: DIAGRAMA Y *PINOUT*. 90
FIGURA 81: CONEXIÓN DE LA MEMORIA FIFO 90
FIGURA 82: TEMPORIZACIÓN DE LAS LECTURAS Y ESCRITURAS EN LA FIFO 91
FIGURA 83: REINICIALIZACIÓN DE LA FIFO 92
FIGURA 84: CRONOGRAMA MOSTRANDO LA SINCRONIZACIÓN A CONSEGUIR 93
FIGURA 85: GENERACIÓN DE RELOJ PARA LOS *FLIP-FLOPS* 95
FIGURA 86: MÁQUINA DE ESTADOS A IMPLEMENTAR 95
FIGURA 87: IMPLEMENTACIÓN DE LA MÁQUINA DE ESTADOS 97
FIGURA 88: ESQUEMA DE LA INTERFAZ HARDWARE USB. 98
FIGURA 89: CONEXIONES DE LA INTERFAZ USB. 99
FIGURA 90: APLICACIÓN USADA PARA GRABAR LA EEPROM EXTERNA AL CHIP USB. 100
FIGURA 91: DETECCIÓN *PLUG-N-PLAY* DEL RADIORRECEPTOR USB. 101
FIGURA 92: DIAGRAMA INTERNO DEL CONVERSOR ANALÓGICO DIGITAL. 102
FIGURA 93: CONEXIÓN DEL CONVERSOR ANALÓGICO DIGITAL. 103
FIGURA 94: EVOLUCIÓN DE LA SEÑAL EN BANDA BASE EN LAS DIFERENTES ETAPAS. 103
FIGURA 95: ESQUEMA INTERNO DEL DAC. 105
FIGURA 96: CONEXIÓN DEL DAC 106
FIGURA 97: INTERFAZ ESTÁNDAR DE LOS LCD 106
FIGURA 98: CRONOGRAMA MOSTRANDO EL PROTOCOLO DEL DISPLAY LCD 107
FIGURA 99: CONJUNTO DE CARACTERES ESTÁNDAR PARA UN LCD 109
FIGURA 100: BUFFER UTILIZADO PARA LAS SEÑALES DEL LCD 110
FIGURA 101: CONECTOR DEL LCD EN LA PLACA PRINCIPAL 111
FIGURA 102: DIVISIÓN DEL RECEPTOR EN DISTINTAS PLACAS 112

FIGURA 103: (A)CONECTORES HEMBRA PARA CONEXIONES CON LA PLACA PRINCIPAL. (B)EL CONECTOR "F". (C)EL CONECTOR USB TIPO B HEMBRA. 113
FIGURA 104: ESQUEMA DE CONEXIÓN DE LAS CUATRO PLACAS DEL RECEPTOR. 114
FIGURA 105: FOTO DE LA CAJA A-61 EN SU ESTADO ORIGINAL. 115
FIGURA 106: FOTO DE LA PLACA PRINCIPAL ANTES DE CONECTAR LAS OTRAS PLACAS. 115
FIGURA 107: FOTO DE TODAS LAS PARTES DEL RECEPTOR Y SUS CONEXIONES. 117
FIGURA 108: FOTO DEL RECEPTOR TOTALMENTE MONTADO Y CERRADO. 119
FIGURA 109: FOTO DE LA PARTE TRASERA DEL RECEPTOR. 119
FIGURA 110: IMAGEN DE LA APLICACIÓN USADA PARA LA EDICIÓN DE LOS ESQUEMAS. 120
FIGURA 111:ESQUEMA GLOBAL DEL CONJUNTO RECEPTOR-PC 122
FIGURA 112:LOS DISTINTOS NIVELES DE LA COMUNICACIÓN 123
FIGURA 113:FORMATO DE LA TRAMA DE LA CAPA DE ENLACE 124
FIGURA 114:EJEMPLO DE INTERCAMBIO DE TRAMAS. 125
FIGURA 115:PARA EL DESARROLLO DEL FIRMWARE SE HA USADO EL ENTORNO AVRSTUDIO 4. 132
FIGURA 116: DIAGRAMA DE FLUJO SIMPLIFICADO PARA EL *FIRMWARE* DEL RECEPTOR. 133
FIGURA 117: IMÁGENES DEL DISPLAY DURANTE (A) EL TIEMPO DE ESPERA DEL BOOTLOADER, Y (B) LA GRABACIÓN DE LA FLASH DESDE EL PC 135
FIGURA 118:DIAGRAMA DE FLUJO SIMPLIFICADO PARA EL PROGRAMA PRINCIPAL. 136
FIGURA 119:DIAGRAMA DE FLUJO DETALLADO DEL BUCLE DE MUESTREO. 140
FIGURA 120: TRAMAS DEL TIPO 0XC0. 141
FIGURA 121: TEMPORIZACIÓN DEL ADC. 141
FIGURA 122: LOS DOS CAMINOS POSIBLES EN LA EJECUCIÓN DEL BUCLE DE MUESTREO. 143
FIGURA 123:CRONOGRAMA DE LAS PRINCIPALES SEÑALES INVOLUCRADAS EN EL MUESTREO DE LAS SEÑALES ANALÓGICAS. 145
FIGURA 124:ESQUEMA GLOBAL DEL CONJUNTO RECEPTOR-PC 150
FIGURA 125:EL PANEL DE CONTROL DE SWRADIOSTUDIO. 151
FIGURA 126:CAPTURA DE PANTALLA DEL MÓDULO DE CONTROL MANUAL. 152
FIGURA 127:CAPTURA DE PANTALLA DEL MÓDULO DE DEMODULACIÓN. 153
FIGURA 128:EL ENTORNO DE TRABAJO DE C++ BUILDER 5. 154
FIGURA 129:MÓDULO DE CONTROL MANUAL: CONVERSORES ADC 156
FIGURA 130:MÓDULO DE CONTROL MANUAL: CONFIGURACIÓN DEL PLL. 157
FIGURA 131:MÓDULO DE CONTROL MANUAL: CONVERSORES DAC 159
FIGURA 132:MÓDULO DE CONTROL MANUAL: VARIOS. 160
FIGURA 133:MÓDULO DE CONTROL MANUAL: PROGRAMACIÓN. 161
FIGURA 134:MÓDULO DE CONTROL MANUAL: GRÁFICO EN EL DOMINIO DE LAS MUESTRAS. .. 162
FIGURA 135:MÓDULO DE CONTROL MANUAL: GRÁFICO EN EL DOMINIO DE LA FRECUENCIA. . 164

FIGURA 136:MÓDULO DE CONTROL MANUAL: GRÁFICO DE LA SEÑAL MUXOUT 165
FIGURA 137:PARTES DEL MÓDULO DE DEMODULADORES. 166
FIGURA 138:FILTROS DIGITALES PASO BAJO DISPONIBLES. 167
FIGURA 139:VENTANA DE INFORMACIÓN DE DEMODULADOR. 168
FIGURA 140:EJEMPLO DE "SONDAS VIRTUALES" EN EL CAMINO DE LA SEÑAL. 169
FIGURA 141:HILOS INVOLUCRADOS EN EL DEMODULADO DE LA SEÑAL. 173
FIGURA 142:DIEZMADO DE LA SEÑAL A LA SALIDA DEL DEMODULADOR. 177
FIGURA 143:RESPUESTA DEL FILTRO DIEZMADOR. 177
FIGURA 144: SEÑALES USADAS PARA PROGRAMAR EL MICROCONTROLADOR EN EL MODO SPI 182
FIGURA 145: ESQUEMA DEL PROGRAMADOR DEL MICROCONTROLADOR. 183
FIGURA 146: TABLA DE COMANDOS DE PROGRAMACIÓN. 185
FIGURA 147: CAPTURA DE PANTALLA DE LA INTERFAZ GRÁFICA PROGRAMADORA DEL MICROCONTROLADOR. 187
FIGURA 148: CÁLCULO GLOBAL DE GANANCIAS. 190
FIGURA 149: BALANCE DE POTENCIAS PARA PUNTO DE COMPRESIÓN DE 1DB. 192
FIGURA 150: DIAGRAMA DE BLOQUES PARA CÁLCULO DEL FACTOR DE RUIDO GLOBAL 194
FIGURA 151: EJEMPLO DE CANAL NO DESEADO FUERA DE BW 199
FIGURA 152: EJEMPLO DE CANAL NO DESEADO DENTRO DE BW 199
FIGURA 153: EJEMPLO DEL PROBLEMA DEL MARGEN DINÁMICO 200
FIGURA 154: COMPROBACIÓN DEL FUNCIONAMIENTO DE LOS OSCILADORES A CRISTAL. 202
FIGURA 155: COMPROBACIÓN MEDIANTE OSCILOSCOPIO DEL FUNCIONAMIENTO DE LA INTERFAZ USB. 203
FIGURA 156: VISUALIZACIÓN DEL FUNCIONAMIENTO DEL BUS SPI 203
FIGURA 157: LA SEÑAL DE REFERENCIA DE 2.5MHZ CONVERTIDA A NIVELES LÓGICOS DE 3.3V 204
FIGURA 158: LA SALIDA MUXOUT EN MODO "N DIVISOR" 205
FIGURA 159: LA SALIDA MUXOUT EN MODO "R DIVISOR" 205
FIGURA 160: MEDICIÓN CON OSCILOSCOPIO DEL *SLEW RATE* DE LA SEÑAL REFCLK DEL PLL. 208
FIGURA 161: ASPECTO DE LA GRÁFICA MUXOUT PARA EL CASO DE QUE EL PLL SE ENGANCHE (IZQUIERDA) O QUE NO LO HAGA (DERECHA) 209
FIGURA 162: POTENCIA DE SALIDA DEL PLL RESPECTO A LA TEÓRICA. 210
FIGURA 163: LA POTENCIA DE SALIDA DEL PLL SE MANTIENE PRÁCTICAMENTE CONSTANTE EN TODO EL RANGO DE FUNCIONAMIENTO 211
FIGURA 164: IMAGEN DEL ANALIZADOR DE ESPECTROS MIDIENDO EL RUIDO DE FASE DEL OSCILADOR. 212

FIGURA 165: DENSIDAD ESPECTRAL DE RUIDO MEDIDO RESPECTO A LA PORTADORA. 213
FIGURA 166: ASPECTO QUE DEBERÍA TENER TEÓRICAMENTE LA D.E.P. DE RUIDO. 213
FIGURA 167: IMAGEN DEL ANALIZADOR DE ESPECTROS MIDIENDO LOS ESPÚREOS DEL OSCILADOR. 214
FIGURA 168: POTENCIA DE LOS DISTINTOS ARMÓNICOS DEL OSCILADOR. 216
FIGURA 169: RECONSTRUCCIÓN DE LA SALIDA DEL OSCILADOR. 216
FIGURA 170: MEDICIÓN CON EL OSCILOSCOPIO DEL TIEMPO DE ENGANCHE DEL PLL. 217
FIGURA 171: MEDICIÓN CON EL OSCILOSCOPIO DE LA SEÑAL DE CONTROL DEL VCO DURANTE EL ENGANCHE. 218
FIGURA 172: RESPUESTA EN FRECUENCIA DE LOS ADC. 219
FIGURA 173: RELACIÓN SNR EN FUNCIÓN DE LA FRECUENCIA. 220
FIGURA 174: DENSIDADES ESPECTRALES DE POTENCIA PARA LAS DOS SEÑALES I Y Q. 221
FIGURA 175:RESPUESTA EN FRECUENCIA DEL FILTRO Y EL LNA 223
FIGURA 176: EFECTO DE LA RESISTENCIA EN SERIE A LA ENTRADA DEL ANALIZADOR DE ESPECTROS 225
FIGURA 177: GANANCIA HASTA LA SALIDA DEL VGA 227
FIGURA 178: POTENCIA DE SALIDA EN FUNCIÓN DE LA POTENCIA DE ENTRADA 228
FIGURA 179. RESPUESTA EN FRECUENCIA DE LOS FILTROS EN BANDA BASE 230
FIGURA 180: MODELO DE UNA INDUCTANCIA REAL 231
FIGURA 181: MODELADO DEL FILTRO INCLUYENDO LAS RESISTENCIAS DE LAS INDUCTANCIAS. 231
FIGURA 182: SIMULACIÓN DE LA RESPUESTA PARA INDUCTANCIAS IDEALES Y REALES. 232
FIGURA 183: ESTIMACIÓN DE LA SENSIBILIDAD DEL RECEPTOR. 235
FIGURA 184: POTENCIA RECIBIDA EN BANDA BASE EN FUNCIÓN DE POTENCIA ENTRANTE DE RF. 236
FIGURA 185: DIFERENCIA DE LA GANANCIA CON SU LINEALIZACIÓN. 237
FIGURA 186: ASPECTO DEL MÓDULO DE DEMODULACIÓN MIENTRAS SE RECIBE UNA SEÑAL DE FM COMERCIAL. 238
FIGURA 187: EFECTO DEL CONTROL AUTOMÁTICO DE GANANCIA. 239
FIGURA 188: AÚN ATENUADA VARIOS ÓRDENES DE MAGNITUD DEBIDO AL FILTRO DE AUDIO, LA PORTADORA DE ESTÉREO AÚN SE PUEDE APRECIAR A LA SALIDA. 240
FIGURA 189: RESULTADOS PARA DEMODULADOR ASK Y PATRÓN DE BITS 0-1. 241
FIGURA 190: RESULTADOS PARA DEMODULADOR ASK Y PATRÓN PRBS DE 9 BITS. 242
FIGURA 191: RESULTADOS PARA DEMODULADOR FSK Y PATRÓN PRBS DE 9 BITS. 243
FIGURA 192: RESULTADOS PARA DEMODULADOR COHERENTE BPSK Y PATRÓN PRBS DE 9 BITS. 244

1. INTRODUCCIÓN

Un radiorreceptor es un sistema electrónico capaz de extraer la información transportada por una señal de radio, o portadora. La manera en que ésta información es llevada por la portadora depende de la modulación usada, que puede ser de muy diferentes tipos. Para empezar, se distingue entre modulaciones analógicas y digitales, según sea la naturaleza de la información que se transmite. Aunque dentro de cada grupo hay multitud de técnicas distintas, existe con diferencia mucha más variedad en las modulaciones digitales, que son tanto las más usadas actualmente como las que más perspectivas tienen de expansión a todos los campos de radiocomunicación. Sin embargo, no existe una modulación digital que se pueda considerar la óptima para todas las aplicaciones, ya que las restricciones de cada aplicación concreta demandan características distintas de la modulación. Por ejemplo, mientras en comunicaciones con módems se explotan modulaciones digitales en fase y amplitud, la demanda de linealidad que ésta impone hace su uso prohibitivo en comunicaciones por satélite, donde son más usadas las modulaciones sólo de fase. En cuanto a las comunicaciones analógicas, siguen siendo ampliamente usadas técnicas de modulación tanto en fase como en amplitud en la radiodifusión de audio y vídeo. Aunque en España está prevista la sustitución de la difusión analógica de televisión para el 2012, no está previsto que la difusión analógica de audio desaparezca, quizás por el costo que supondría para los consumidores comparado con las relativamente pocas ventajas de la migración a digital en este caso concreto.

Por lo tanto, el diseño de un radiorreceptor debe orientarse obligatoriamente hacia una aplicación concreta adoptando el esquema de modulación definido para dicha aplicación. Éste ha sido el método de diseño usado tradicionalmente, por ejemplo, en los superheterodinos. En estos, todo el procesado de la señal, desde radiofrecuencia (RF) hasta banda base (BB) pasando por la frecuencia intermedia (IF), se realiza con componentes analógicos como se muestra esquemáticamente en la figura 1.a. Las desventajas de un receptor analógico son: (i) la tolerancia limitada de los componentes hace que los márgenes de frecuencias de osciladores, filtros, etc. tengan un amplio margen de indeterminación, que sólo se puede reducir a costa de la sencillez de los diseños. (ii) muchos parámetros del receptor quedan fijados una vez implementado,

siendo necesaria la sustitución de componentes para variarlos. (iii) entre las diferentes partes del circuito se producen acoples o fugas de señales, produciendo diversos efectos indeseados (por ejemplo, la fuga del oscilador local hacia la antena a través de un mezclador). (iv) en general, para cada canal que se quiera recibir es necesario un demodulador independiente, lo que multiplica el coste en aplicaciones como estaciones base de telefonía móvil donde se procesan simultáneamente muchos canales.

Para solventar estos problemas, y aprovechando el gran desarrollo en velocidad y potencia de los sistemas digitales en la última década, principalmente microprocesadores, procesadores digitales de señal y conversores analógico-digitales, ha aparecido la técnica conocida como radio software [1]. Un receptor de radio software realiza las mismas operaciones que un receptor analógico, pero digitalizando la señal RF y realizando todo el proceso en el dominio digital. Así, un superheterodino implementado como radio software sería como el mostrado en la figura 1.b, donde se han omitido los conversores analógico-digitales y digitales-analógicos por claridad.

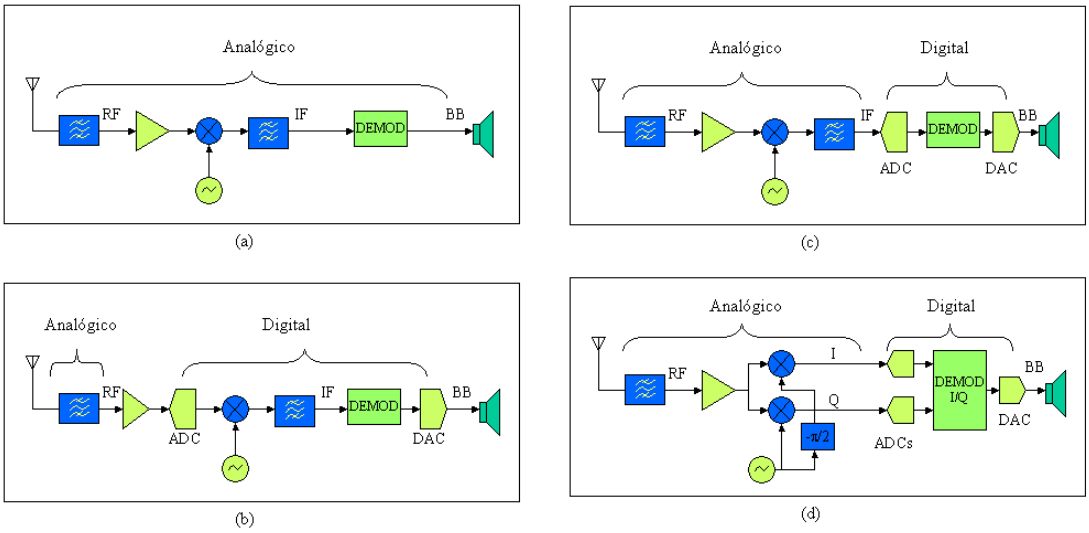


Figura 1: Tres formas de implementar un receptor superheterodino (a,b,c) y el demodulador en cuadratura (d).

Un diseño como éste, donde la señal de radiofrecuencia es digitalizada directamente, es el mejor desde el punto de vista de la generalidad, al estar todo el proceso de demodulación en el dominio digital y ser fácilmente modificable. Sin embargo, la tecnología actual para realizar esto tiene un alto coste o no permite su aplicación a todos los campos, debido sobre todo a tres problemas:

- Según el teorema de Nyquist, la frecuencia de muestreo de una señal debe ser, al menos, el doble que la componente de frecuencia más alta de ésta si se quiere

evitar el *aliasing*, que destruye la información en ciertas bandas de frecuencias, como se muestra en la figura 2. Como la mayoría de radiocomunicaciones actuales se realizan a frecuencias muy elevadas, dentro o cerca del rango de microondas, esto supone que el conversor analógico digital debería funcionar al doble de estas frecuencias. Por ejemplo, un receptor GSM funcionando a 1800MHz, necesitaría un conversor funcionando a 3600MSPS. Actualmente es difícil encontrar conversores por encima de 1GSPS, y sus precios son prohibitivos para productos de consumo. Como ejemplo, el conversor ADC081000 de National Semiconductors, que soporta 1GSPS (siendo capaz de digitalizar señales de hasta 500Mhz) cuesta 275$. Sin embargo este problema se soluciona utilizando técnicas de submuestreo, como se comenta más abajo.

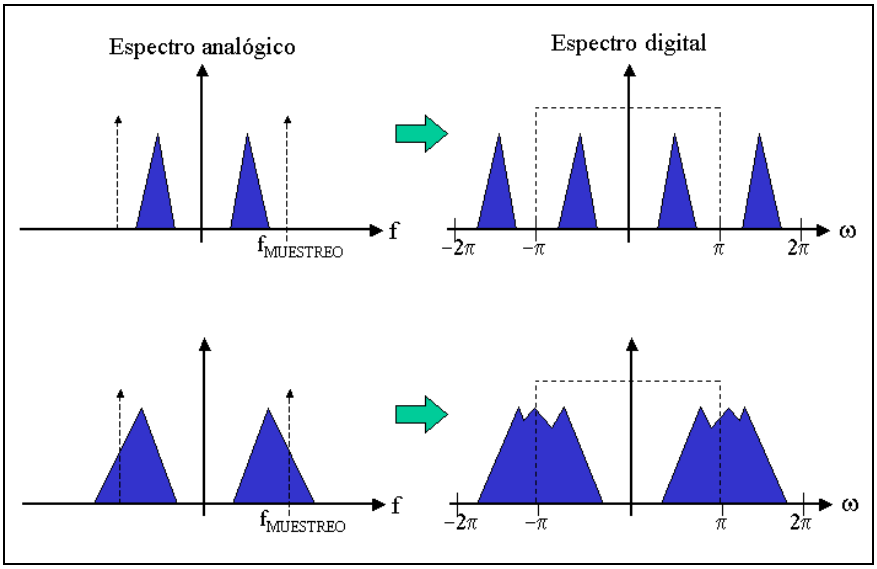


Figura 2: Efectos del aliasing en la digitalización de señales.

- Otro problema es el margen dinámico de los conversores, es decir, su resolución. Si se quiere recibir señales con un amplio margen dinámico, donde unos canales tienen mucha más potencia que otros, se requiere una alta resolución en la digitalización, pero precisamente cuanto más rápido es un conversor, menos resolución suele tener.
- Incluso existiendo un conversor adecuado, existe el problema de encontrar un procesador de señal capaz de trabajar a la frecuencia necesaria. Además, el simple transporte de las señales digitales a frecuencias elevadas representa un problema en sí mismo (niveles lógicos ECL, líneas de transmisión, acople, etc.)

Para aliviar en parte estos problemas, se usan técnicas de submuestreo, consistentes en muestrear a más del doble del ancho de banda del canal y utilizar el aliasing para que el canal deseado caiga dentro del ancho de banda digitalizado. A pesar de esto las radio software totalmente digitales no son viables con la tecnología actual para productos de bajo coste.

Una alternativa más viable es no digitalizar directamente la señal en radio frecuencia, sino en frecuencia intermedia, como se muestra en la figura 1.c. En esta técnica, solamente se requiere un conversor funcionando a poco más del doble de la frecuencia IF. Aunque esto relaja las imposiciones de frecuencia para los conversores, aún puede ser un problema en muchos casos. Una tercera técnica, la ZIF (Zero intermediate frequency) permite el procesado digital de la señal sin requisitos altos para los conversores. Como se muestra en la figura 1.d, en esta técnica se mezcla la señal de radiofrecuencia con dos versiones del oscilador local, desfasadas $\pi/2$ radianes entre ellas. En este caso el oscilador local funciona a la frecuencia exacta del canal que se quiere sintonizar, de ahí el nombre de frecuencia intermedia cero, y se separa la señal en sus componentes en fase y cuadratura, que son las dos señales que realmente se digitalizan. Por lo tanto, en este método se requieren dos conversores, pero a cambio la frecuencia de funcionamiento es del orden de magnitud del ancho de banda de la señal en banda base, lo que supone varios órdenes de magnitud menos que las otras técnicas, y por tanto, mayor oferta de conversores a precios más asequibles y de mucho menor consumo. El uso de una frecuencia intermedia nula evita el problema de la frecuencia imagen, pero a cambio introduce el nuevo problema del *offset* de continua en las salidas demoduladas, generado por fugas del oscilador local hacia la entrada de radiofrecuencia. A pesar de esto, sigue manteniendo la ventaja de una demodulación totalmente configurable por software, lo que da a esta técnica la capacidad de demodular cualquier modulación presente o futura. De hecho, esta técnica (también llamada demodulación en cuadratura) es la utilizada en la mayoría de terminales actuales de telefonía móvil de segunda (GSM) y tercera (UMTS) generación, receptores GPS, etc.

Por todas estas ventajas, y por su relativa fácil implementación, esta es la técnica elegida para el desarrollo de este proyecto, y su idea central: la creación de un radiorreceptor que mediante el software adecuado sea capaz de demodular cualquier tipo de señal de radiofrecuencia.

2. OBJETIVOS DEL PROYECTO

En este proyecto, se propone el estudio, diseño e implementación de un sistema radiorreceptor software, incluyendo para esto el diseño del hardware y software necesario de acuerdo con estos objetivos y criterios:

- Bajo coste.
- Reducido tamaño, de forma que el sistema sea fácilmente portátil.
- Soporte software para sistemas operativos Windows, al menos.
- Arquitectura de software abierta que permita la ampliación con nuevos demoduladores de forma sencilla, mediante un sistema de *plugins*.
- Cobertura de, al menos, el ancho de banda de FM comercial (88MHz a 108MHz)
- Anchura de canal paso-banda máxima de 200KHz al menos, siendo posible reducirlo por software.
- Utilización de la técnica ZIF (*Zero IF*), que elimina el problema de la frecuencia imagen y es independiente del tipo de modulación.
- Uso de la interfaz USB, por ser una interfaz presente en todos los equipos actuales.
- Bajo consumo de potencia, para no requerir fuente de alimentación adicional a la obtenida del cable USB, lo que dará mayor portabilidad al sistema.
- Control digital del receptor, que permita, desde el PC:
    - Selección del canal.
    - Variación de parámetros del receptor, como la ganancia, ajustes de niveles de *offset*, anchos de banda, etc.
- Soporte para estos tipos de modulaciones:
    - Analógicas:
        - Al menos la modulación FM
    - Digitales:
        - Al menos una modulación coherente y una no coherente.

3. DISEÑO CONCEPTUAL

3.1. La demodulación en cuadratura

Debido al uso que en el proyecto se hace de la demodulación en cuadratura, en este apartado se hace un breve repaso de los conceptos básicos de Teoría de la Comunicación necesarios para: (a) Justificar el diseño del sistema radiorreceptor que se presenta a continuación, y (b) definir matemáticamente el problema que se aborda en el apartado 7.3, donde se deducen las ecuaciones necesarias para demodular distintas modulaciones analógicas y digitales a partir de sus componentes. El modelado matemático también es necesario para justificar la elección de ciertos parámetros en el diseño del hardware, como se verá en el capítulo del diseño de hardware analógico.

Las señales de información que se quieren transmitir en cualquier sistema de comunicaciones son señales en banda base, es decir, sus espectros de energía están alrededor o muy cerca del origen (componente continua). Sin embargo, debido a que los medios de transmisión son normalmente del tipo paso banda, y también para permitir la división en canales y la multiplexación en frecuencia, es necesario modular la información sobre portadoras paso banda. Una señal paso banda, real y modulada se puede escribir como:

$$M(t) = a(t) \cdot \cos(2\pi f_c t + \theta(t))$$

Dónde a(t) se llama envolvente real de la señal, θ(t) la variación de fase y f_c es la frecuencia central de la portadora en hertzios. Si llamamos x(t) a la información a transmitir, una señal real en banda base, la relación entre ésta y las funciones a(t) y θ(t), depende del tipo de modulación usada. Esta forma de describir la señal paso banda se llama forma magnitud-fase. Una señal paso banda *real* también puede descomponerse en sus componentes en fase y cuadratura de esta forma:

$$M(t) = I(t) \cdot \cos 2\pi f_c t - Q(t) \cdot \sin 2\pi f_c t$$

Donde se define la señal *compleja* banda base equivalente como:

$$M_{bb}(t) = I(t) + jQ(t)$$

Siendo:

$$I(t) = a(t) \cdot \cos \theta(t)$$

$$Q(t) = a(t) \cdot \sin \theta(t)$$

Donde I(t) y Q(t) se llaman componentes en fase y cuadratura, respectivamente, de la señal banda base compleja. Se demuestra fácilmente ([5]) que esta señal en banda base contiene la misma información que la señal paso banda original, con la ventaja de tener un espectro centrado en el origen. Las ventajas de esta representación se consiguen a costa de convertir una señal real en otra compleja. Esta es la base de la demodulación ZIF (*Zero-Intermediate-Frequency*), donde la señal de radiofrecuencia se lleva directamente a banda base obteniéndose su señal banda base compleja equivalente. La disminución a la mitad del espectro (si se elige OL justo en la mitad del espectro paso banda) se ilustra en la siguiente figura, donde se muestra el proceso matemático asociado a la técnica ZIF.

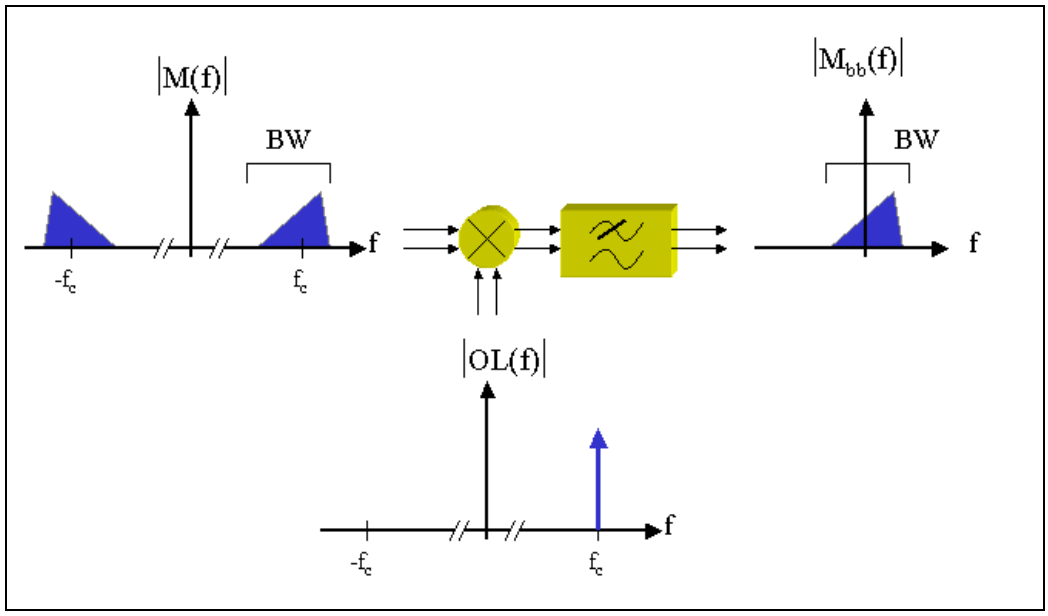


Figura 3: Conversión de paso banda a banda base equivalente.

Como físicamente no se pueden manejar señales complejas, lo que se hace en la práctica es obtener las dos señales reales I(t) y Q(t) por separado. Para explicar esto, hay que fijarse que, si $M_{bb}(f)$ es el espectro de Fourier de la señal banda base equivalente y M(f) el de la señal paso banda original, tenemos que:

$$M_{bb}(f) = M(f+f_c), \quad |f| < BW$$

Esta translación en el espectro puede verse como una multiplicación en el dominio del tiempo por una señal portadora compleja:

$$OL(t) = e^{j 2\pi f_c t}$$

Este proceso se lleva acabo por el mezclador complejo mostrado en la figura anterior. Ahora bien, la portadora compleja se puede expandir como:

$$OL(t)=\cos(2\pi f_c t) - j \cdot \sin(2\pi f_c t)$$

Si la mezcla de M(t) con OL(t) se realiza separando las partes reales e imaginarias, usando dos mezcladores reales tal y como se ve en la figura 4, tenemos:

Re $\{M(t) \cdot OL(t)\}$ = **Re** $\{M(t) \cdot (\cos\phi - j \sin\phi)\}$ =

$M(t) \cos\phi = (I(t) \cos\phi - Q(t) \sin\phi) \cos\phi = I(t) \cos^2\phi - Q(t) \sin\phi \cos\phi =$

$I(t) [\frac{1}{2} + \frac{1}{2} \cos 2\phi] - Q(t) \frac{1}{2} \sin 2\phi$ → Filtro paso bajo → **I(t)**

Im $\{M(t) \cdot OL(t)\}$ = **Im** $\{M(t) \cdot (\cos\phi - j \sin\phi)\}$ =

$- M(t) \sin\phi = (I(t) \cos\phi - Q(t) \sin\phi) (-\sin\phi) = -I(t) \cos\phi \sin\phi + Q(t) \sin^2\phi =$

$- I(t) \frac{1}{2} \sin 2\phi + Q(t) [\frac{1}{2} - \frac{1}{2} \cos 2\phi]$ → Filtro paso bajo → **Q(t)**

Donde por comodidad se ha usado $\phi=2\pi f_c t$. Como se puede ver, como resultado de la parte real e imaginaria de la multiplicación se obtienen I(t) y Q(t), respectivamente. Para la generación de las señales $\cos(\phi)$ y $-\sin(\phi)$ en la práctica se recurre a un solo oscilador y alguna técnica para retrasar la señal $\pi/2$ radianes. Por lo tanto, se ha demostrado que la demodulación ZIF es realizable físicamente usando una estructura como la mostrada en la siguiente figura, y que es la usada en este proyecto.

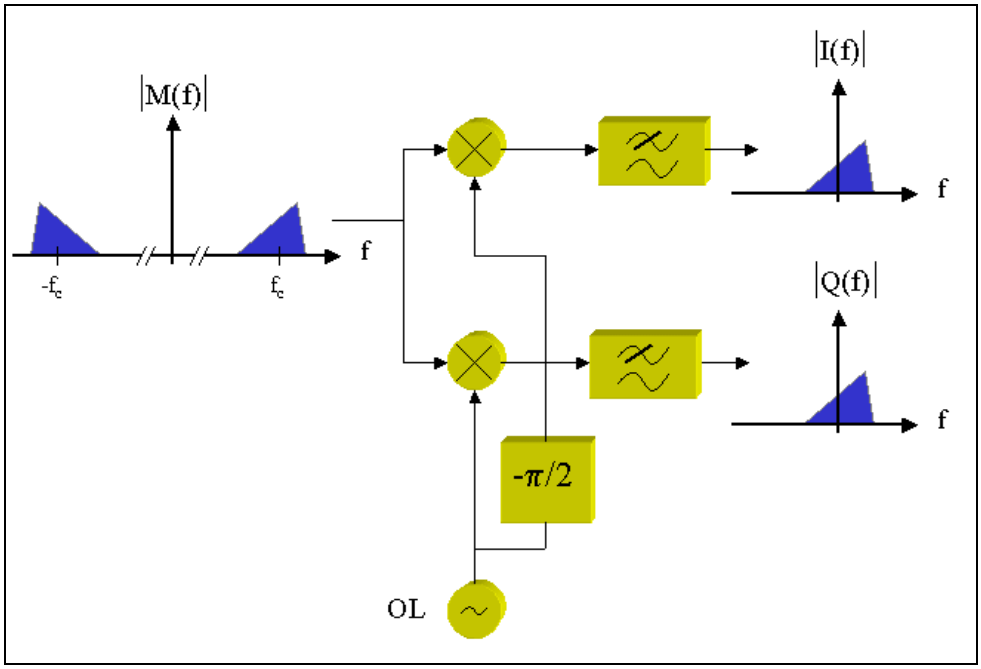


Figura 4: Implementación ZIF realizable físicamente.

3.2. Diseño global del sistema

Teniendo en cuenta los objetivos del proyecto, y tras un largo proceso de búsqueda entre los componentes disponibles comercialmente, las restricciones que unos imponen sobre los otros (por ejemplo, potencias mínimas y máximas permitidas desde las etapas anteriores), se ha llegado al diseño por bloques que se muestra en la figura 5. A continuación se describe cualitativamente el funcionamiento del sistema, empezando por los primeros componentes desde la entrada de radiofrecuencia, hasta llegar a la interfaz con el PC. En siguientes capítulos se describe detalladamente cada uno de estos bloques, su proceso de diseño y características cuantitativas.

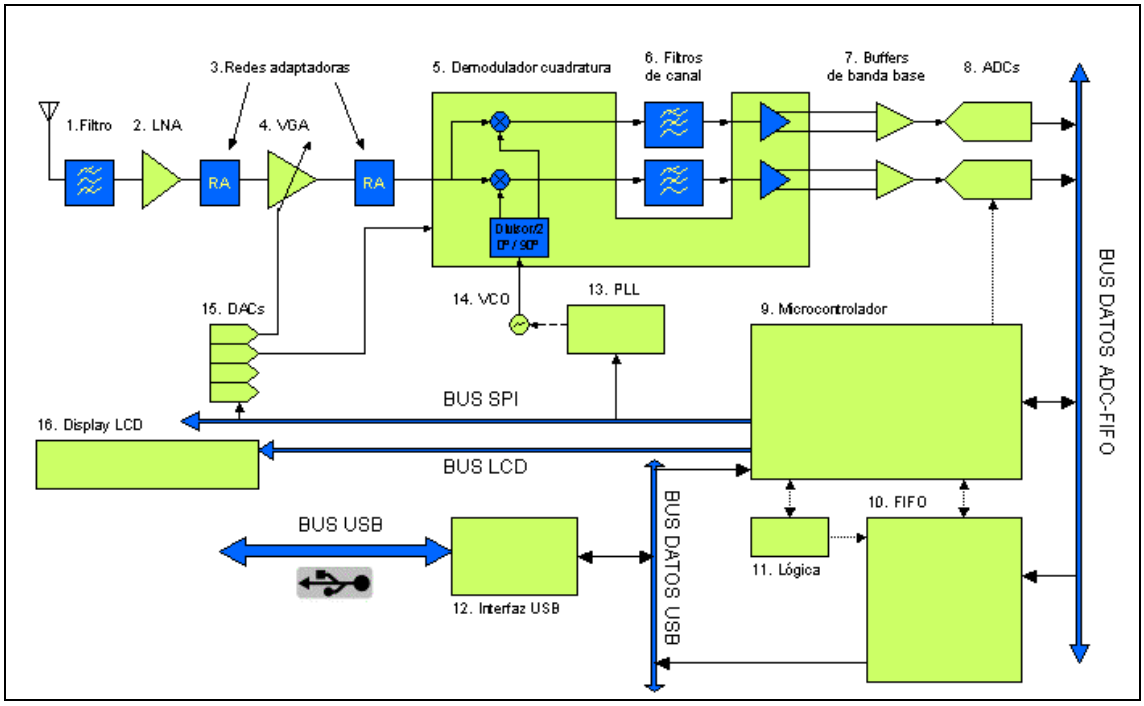


Figura 5: Diagrama de bloques global del radiorreceptor a diseñar.

La señal de radiofrecuencia que entra al sistema se somete primero a un filtrado paso banda (bloque 1, filtro de preselección). Con esto se consigue reducir la potencia de las señales fuera del espectro de interés, con lo que se gana margen dinámico y calidad señal a ruido. Tras esto, la señal es amplificada (bloque 2) con un amplificador de bajo ruido y ganancia fija, y se vuelve a amplificar (bloque 4) con un amplificador de ganancia variable (*Variable Gain Amplifier*, VGA). Tanto a la entrada como a la salida del VGA son necesarias redes adaptadoras de impedancia (bloques número 3). Tras esto la señal se introduce ya en el demodulador por cuadratura (bloque 5), donde la señal en

banda base es filtrada (bloque 6, filtros banda base) antes de la última etapa de amplificación diferencial dentro del mismo demodulador. Tras esto, las señales I y Q pasan de ser diferenciales a ser unipolares (bloque 7), para ser digitalizadas por los conversores analógico digitales (bloque 8). La frecuencia de las conversiones está dirigida por el microcontrolador (bloque 9), y las muestras son almacenadas en una memoria FIFO (bloque 10), controlada en parte de forma externa al microcontrolador por una lógica discreta (bloque 11), que también controla la entrada de estos datos en la interfaz USB (bloque 12), que es la que envía ya los datos al PC, donde el *driver* los recogerá y los pasará a la arquitectura software, que se expone en el capítulo 7.3. Los datos enviados desde el PC al receptor, siguiendo el protocolo definido en el capítulo 7.1, son recibidos directamente por el microcontrolador, que se encargará de realizar las tareas oportunas. La frecuencia a sintonizar se controla desde el microcontrolador a través de un bus SPI (*Serial Peripheral Inteface*, un bus serie a tres hilos) al que está conectado el sintetizador de frecuencias. Este está formado por un PLL integrado, un filtro de lazo (ambos incluidos en el bloque 14 en el esquema anterior) y un VCO (bloque 13).

Las ganancias del VGA y del demodulador en cuadratura se controlan por medio de unos conversores digital analógico (bloque 15), conectados al mismo bus SPI usado por el PLL. También se ha incluido en el diseño una pantalla LCD donde se informa sobre el estado del sistema.

3.3. Alternativas de diseño

Durante las primeras etapas del diseño, se plantearon varias opciones a diversos niveles de detalle del hardware. A continuación se exponen las opciones que se plantearon, las decisiones adoptadas y sus justificaciones.

- Dispositivo externo / Tarjeta interna al PC

  En el diseño de una tarjeta interna al PC se presentan dos opciones: PCI o ISA. Mientras que el diseño de una tarjeta PCI implica el uso de algún chip de interfaz PCI y el diseño de un *driver* específico, el diseño ISA podría ser bastante más sencillo. Ambas opciones proporcionan un ancho de banda de E/S muy elevado. En el diseño de un dispositivo externo sobresalen dos problemas: La alimentación del circuito y la facilidad y velocidad del enlace con el PC. Mientras que los puertos serie y paralelo tradicionales no resuelven estos problemas, los buses USB y Firewire sí que se plantean como soluciones óptimas, ya que extraen la alimentación del mismo PC y tienen altas tasas de transferencia de datos. Además, el uso de una tarjeta interna al PC elimina a los ordenadores portátiles de los potenciales usuarios del sistema. Por todo esto, se decidió diseñar el sistema en forma de dispositivo externo. Como interfaz se escogió la USB por ser la más extendida, de mayor disponibilidad en los ordenadores actuales, tener capacidad *plug-n-play* y poseer varios modos de velocidad que se adaptan a mayores o menores tasas de transferencia.

- USB 2.0 / USB 1.1

  En la especificación USB [3] se definen tres modos de velocidad a los que puede funcionar un dispositivo, que son:

  - *Low-speed* (1.5 Mbps)
  - *Full-speed* (12 Mbps)
  - *High-speed* (480 Mbps)

  Donde la versión *high-speed* sólo está disponible para dispositivos USB 2.0, mientras que las otras dos versiones están disponibles tanto en USB 1.1 como en 2.0. Debido a la mayor disponibilidad de circuitos integrados USB 1.1, menor

precio y ancho de banda suficiente para la aplicación, se eligió usar esta interfaz para el receptor.

- Microcontrolador USB / Interfaz USB dedicada

  La inmensa mayoría de dispositivos de consumo USB disponibles en el mercado funcionan con un microcontrolador con funcionalidad USB integrada en el mismo. La ventaja de esto es un menor coste y un menor espacio ocupado, frente a la opción de chips dedicados a la interfaz USB, que es la opción que se ha escogido para el sistema. Esto es así porque para estas interfaces los fabricantes suelen disponer de un *driver* gratuito, mientras que para los dispositivos basados en microcontrolador USB, el desarrollo de este *driver* USB es responsabilidad del diseñador. Debido a la complejidad de este tipo de *drivers*, se optó por usar los ya diseñados y comprobados por los fabricantes para interfaces USB hardware dedicadas.

- Oscilador analógico (VCO) / Digital (NVCO)

  A la hora de generar el oscilador local necesario en el receptor, se planteó la posibilidad de realizarlo mediante un oscilador tipo VCO o mediante *Direct Digital Synthesis* (DDS), técnica en que la señal sinusoidal es generada mediante hardware digital y un conversor digital-analógico. Tras comprobar las opciones de integrados DDS en el mercado, se comprobó que para generar señales de hasta unos 160Mhz hay bastantes integrados DDS con relativamente poco consumo, pero como en este proyecto se requiere un oscilador de más de 200Mhz, el único integrado que se ha encontrado capaz de esto es el AD9858. Sin embargo este integrado sobrepasa los objetivos de este proyecto, ya que es un sistema DDS a 1GSPS, con los problemas de la generación del reloj a esta frecuencia y de consumo, ya que consumía 2W de potencia a 3.3V, más que el resto del radioreceptor en conjunto y más corriente de la permitida por el estándar USB.

- Procesado en DSP / PC

  La demodulación digital de la señal en cuadratura se podría realizar en el mismo dispositivo, enviando al PC directamente la señal demodulada, o por el contrario, enviar la señal en cuadratura al PC y que sea allí donde sea

demodulada mediante software. Debido a la mayor versatilidad y sencillez de programación que ofrecen los PC frente a los DSP, se decidió que la demodulación no se realizase en el mismo receptor, sino en el PC.

- Usar FIFO / no usarla.

  Habiendo decidido que la señal en cuadratura (señales I y Q) debería ser transmitida al PC, se estudiaron dos opciones para controlar el alto flujo de datos que, saliendo de dos conversores analógico digitales distintos, debería ser multiplexada y secuenciada para enviarse al PC. Las dos opciones estudiadas fueron:

  - Que el microcontrolador se encargara de recoger los datos de los conversores y enviarlos a la interfaz USB, usando su memoria interna a forma de buffer temporal.
  - Usar una memoria FIFO dedicada exclusivamente a hacer de buffer intermedio entre los conversores y la interfaz USB, controlada por lógica discreta y librando así al microcontrolador de esta tarea.

  Tras simulaciones en un entorno de desarrollo del microcontrolador, se vio que la primera opción demandaba mucha más capacidad de proceso que la segunda, llegándose incluso en ocasiones a la pérdida de datos. Esto es así debido al funcionamiento de la interfaz USB elegida, que no siempre es capaz de recibir más datos, debido a su limitada memoria buffer interna y a la naturaleza de comunicación por ráfagas de USB. Durante estos intervalos, el microprocesador debería ser capaz de almacenar los datos a la velocidad de muestreo y posteriormente enviarlos a la interfaz USB a una velocidad mucho mayor. Por estos problemas, se optó por el uso de una memoria FIFO dedicada y lógica externa para librar la carga del microcontrolador.

- Diseño separado / una sola placa (PCB)

  Esta última decisión proviene de una voluntad de diseño modular del sistema hardware. Las ventajas por las que se decidió realizar el receptor como un conjunto de placas separadas, fueron:

  - Modularidad: Si una parte del sistema fallase, o necesitara ser rediseñada, no afectaría al resto del circuito.

- Espacio: Una sola placa ocuparía más espacio, ya que una de las placas diseñadas hace de placa principal, estando las otras colocadas físicamente encima de ésta, y ocupando por tanto menos superficie en total.

La división se ha realizado en cuatro placas PCB distintas:

- Placa 1: La placa principal, donde van el microcontrolador, la interfaz USB, la memoria FIFO con su lógica asociada, los conversores analógico-digitales y los digital-analógicos.
- Placa 2: Aquí se incluye el oscilador VCO y el PLL.
- Placa 3: Esta placa contiene el demodulador, los filtros en banda base y los conversores de diferencial a unipolar de las señales I y Q.
- Placa 4: Es la placa que contiene el conector de entrada de radiofrecuencia, el filtro de preselección, el LNA y el VGA.

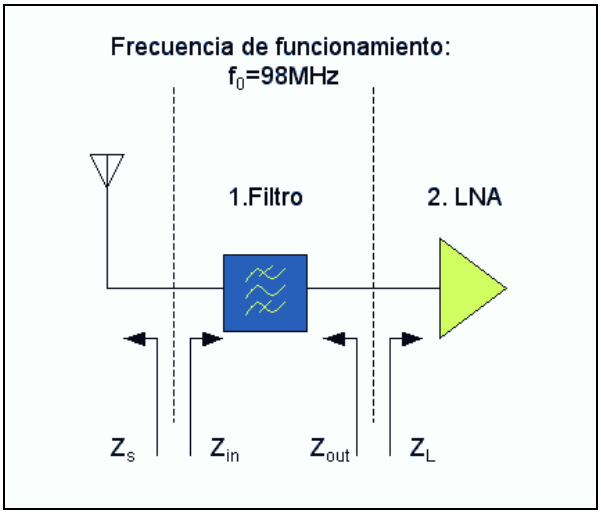
Para aclarar los fragmentos de esquemas que aparecen en el siguiente apartado, hay que decir que esta numeración de las placas se ha usado para estructurar la numeración de los componentes. Por ejemplo, si un integrado se llama “U1-15” es porque está en la placa 1, la principal, mientras que si una resistencia se llama “R2-26” es porque está situada en la placa del oscilador. En el capítulo sobre diseño mecánico se explica más detalladamente la división en placas y las conexiones entre ellas.

4. DISEÑO DEL HARDWARE ANALÓGICO

Entre este capítulo y el siguiente se detalla el proceso de diseño y de selección de componentes que se ha seguido para cada uno de los bloques en que se ha dividido conceptualmente el sistema y que se mostraba en la figura 5. Dado el elevado número de estos subsistemas, se presentarán primero en este capítulo los subsistemas analógicos y en el siguiente los digitales. Algunos subsistemas mixtos, como el PLL, se consideran analógicos ya que su funcionalidad principal está en el dominio analógico. Este diseño se complementará con el capítulo 6, dedicado al diseño físico y mecánico del radioreceptor. Aunque a lo largo de estos capítulos se muestran esquemas parciales, las catorce páginas con los esquemas completos se pueden encontrar en el anexo 1.

Hay que señalar que en estos capítulos dedicados al diseño todos los parámetros que aparecen, como impedancias, ganancias, factores de ruido, etc. son valores teóricos dados por los fabricantes, y que las mediciones sobre el circuito real se muestran en un apartado posterior dedicado a las mediciones.

4.1. Filtro de preselección



$Z_S = 50\Omega$ (Entrada de antena)

$Z_{in} = 63 - j3\ \Omega$ (Entrada al filtro)

$Z_{out} = 50\Omega$ (Salida del filtro)

$Z_L = 63 - j3\ \Omega$ (Entrada del LNA)

$L_{desadaptación} = 0.06dB$

Figura 6: Diagrama de impedancias para el filtro de preselección.

Los objetivos del filtro de preselección en los radiorreceptores son:

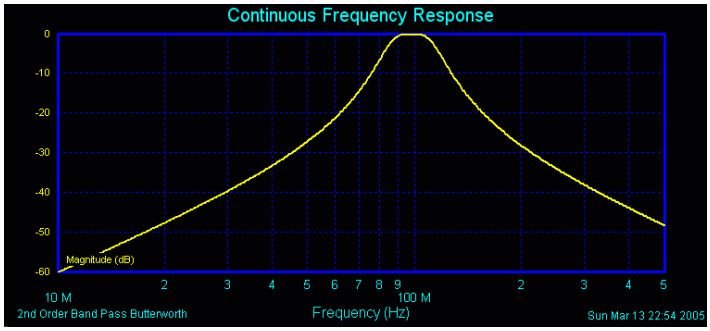
1. Limitar la potencia de ruido que entra al receptor. En la señal que llega de una antena, se encuentra no sólo la señal del canal deseado, sino toda la multitud de emisiones en las diferentes bandas de frecuencias y con distintas potencias cada una, además de ruido a cualquier frecuencia. Por lo tanto, sería deseable que se filtrara todo el espectro menos el canal deseado. Sin embargo, si se quiere evitar el diseño de un filtro de preselección variable, este debe abarcar al menos todos los canales del ancho de banda de funcionamiento del receptor.
2. Eliminar la frecuencia imagen del canal sintonizado. En receptores superheterodinos, es necesario filtrar las frecuencias imágenes, que se encuentran a dos veces la frecuencia intermedia del canal sintonizado. Si no se filtra, éste canal imagen contribuiría a la señal demodulada de igual forma que el canal deseado, siendo imposible separarlos después.

El radiorreceptor que se está diseñando en este proyecto utiliza la técnica de demodulación en cuadratura, por lo que el problema de la frecuencia imagen no es aplicable, y por tanto solo hay que tener en cuenta el problema del ruido fuera de banda. Según las especificaciones del proyecto, queremos sintonizar el rango de frecuencias de FM comercial, es decir desde 88Mhz hasta 108Mhz, un rango de 20Mhz centrado en 98Mhz. Ahora bien, como no existe el problema de la frecuencia imagen, no es un inconveniente el aumentar el ancho de banda de paso del filtro de preselección, lo que

se ha hecho por dos motivos: (i) con la intención de dejar un margen de seguridad para que no se atenúen en exceso algunos canales en los extremos de la banda de paso debido a la limitada tolerancia en los valores de los componentes, y (ii) para que el diseño del filtro produzca valores de componentes cercanos a valores disponibles comercialmente. Concretamente, se ha usado un ancho de banda de paso de 30Mhz. Para implementar el filtro, se ha elegido un filtro de tipo Butterworth de segundo orden. Un segundo orden es suficiente para un filtro de preselección, ya que no hay necesidad de una zona de transición abrupta y mantiene bajo el número de componentes discretos necesarios para implementar el filtro. Además se usa un tipo Butterworth debido a que es el tipo de filtro con la respuesta en frecuencia más plana en la banda de paso. Como se ve en el esquema de la figura anterior, este filtro tiene una impedancia de fuente de 50Ω y una impedancia de carga de 63-3jΩ, que es la impedancia de entrada al LNA. Con todos estos datos y usando la herramienta de ayuda al diseño de filtros *Filter Lite* de *Nuhertz Technologies*, se obtiene la siguiente función de transferencia que cumple con los parámetros de diseño:

$$H(s) = \frac{3.553 \cdot 10^{16} s^2}{s^4 + 2.666 \cdot 10^8 s^3 + 7.938 \cdot 10^{17} s^2 + 1.01 \cdot 10^{26} s + 1.438 \cdot 10^{35}}$$

La respuesta en frecuencia ideal y la implementación con elementos discretos (condensadores e inductancias) se muestran respectivamente en las siguientes figuras.



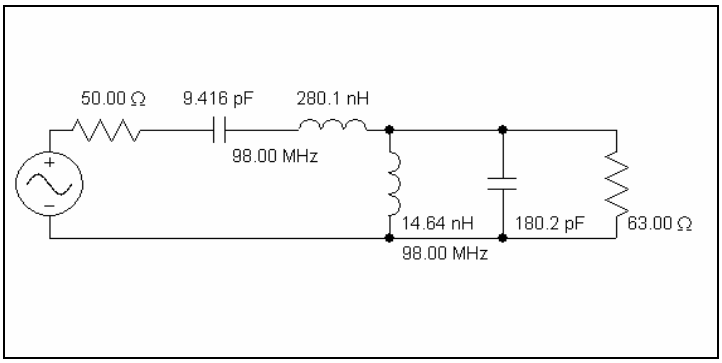


Figura 7: Respuesta en frecuencia e implementación ideal del filtro.

Existen dos posibles soluciones para este filtro, ambas con los elementos en serie primero y valores razonables para los componentes. Arbitrariamente se ha elegido la que se muestra en la figura. Como se ve, el primero elemento es un condensador en serie por lo que no es necesario un condensador de desacople de continua a la entrada, aunque sí en la salida. Para este caso, se usará un condensador de 2.2nF, ya que su valor es suficientemente alto para no afectar a la respuesta del filtro, ya que su impedancia es despreciable:

$$Z_{cap}\Big|_{max} = \frac{1}{j\omega_{min}C} = \frac{1}{j\cdot 2\pi\cdot 88\cdot 10^{6}\cdot 2.2\cdot 10^{-9}} = -j\,0.82\,\Omega \quad << \quad 50\Omega$$

En cuanto a los valores ideales calculados para los componentes, estos no están disponibles comercialmente, por lo que en la implementación real se usan los valores cercanos que menos impacto tienen en la respuesta global del filtro, quedando el filtro definitivamente como se ve en el siguiente esquema eléctrico.

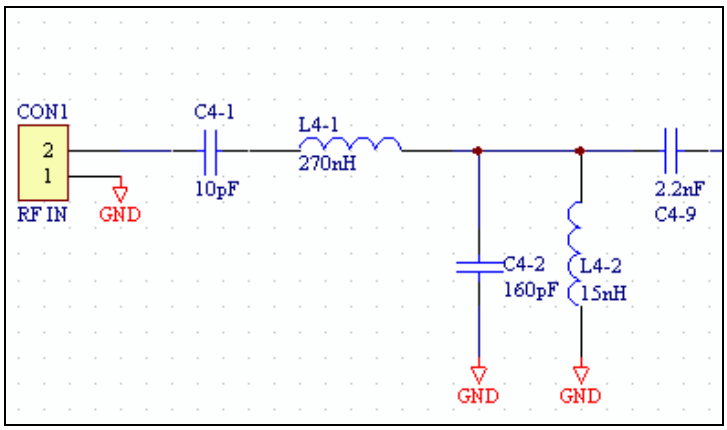


Figura 8: Implementación del filtro de preselección.

Hay que indicar que aunque el diseño de filtros utilizando tecnología *microstrip* permite eliminar el problema de encontrar valores comerciales cercanos a los necesarios, no resulta una técnica práctica a las frecuencias de trabajo de este proyecto: Para hacerse una idea, la longitud de onda a 100MHz, es de tres metros. En la siguiente figura se muestran dos respuestas en frecuencia superpuestas: la ideal (en rojo) y la resultante de la pequeña desviación en los valores de los componentes para adaptarse a valores existentes comercialmente (en azul discontinuo). Como se ve, la variación es pequeña, resultando finalmente que la banda de paso del filtro a –3dB esta centrada en 102.0Mhz con un ancho de banda de 32.4MHz. Estas gráficas se han obtenido mediante unos

scripts de MATLAB que simulan el comportamiento del filtro a partir de los valores de los componentes, incluyendo el condensador de desacoplo de continua. Los *scripts* se muestran en el anexo A5.1.

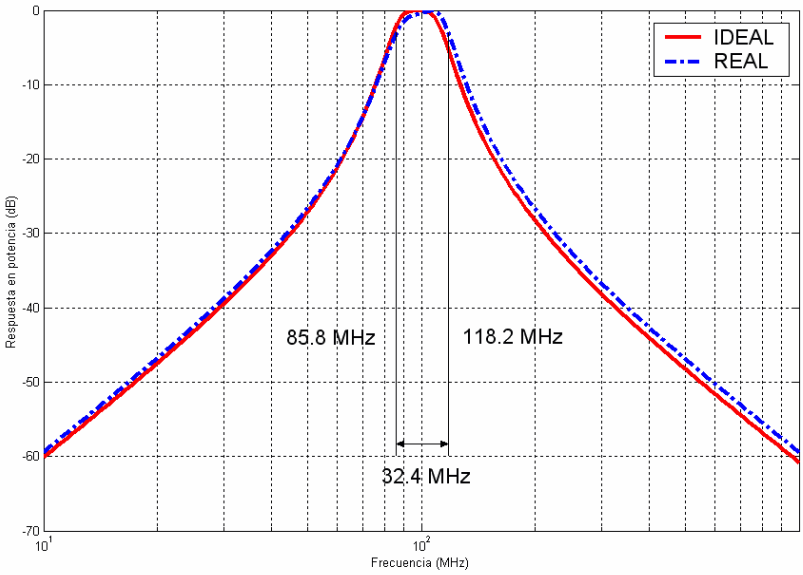


Figura 9: Respuesta simulada del filtro de preselección.

A estos cálculos hay que añadir la estimación de las pérdidas por desacoplo de impedancias, producto de que la impedancia de entrada al LNA no sea exactamente 50Ω. En la siguiente gráfica se muestra el coeficiente de reflexión del filtro en función de la frecuencia de la señal:

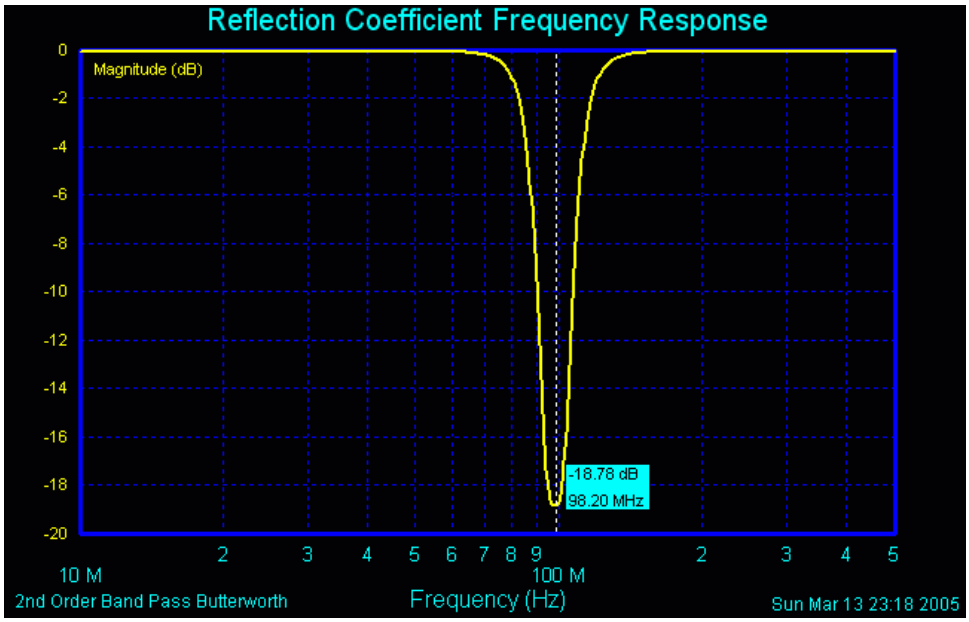


Figura 10: Coeficiente de reflexión del filtro.

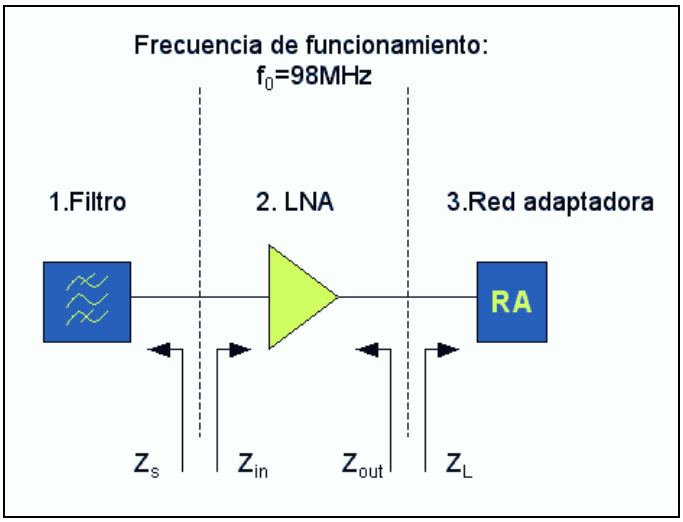
Como se trata de un filtro paso banda, el coeficiente de reflexión (S_{11}) de este es igual a la unidad (cero decibelios) en las frecuencias rechazadas, y tiene un valor bajo en la banda de paso. Idealmente el valor de S_{11} debería ser de cero si la carga estuviese adaptada, pero en este caso el mínimo valor del coeficiente de reflexión es de –18.78dB lo que equivale a $S_{11} = 0.12$, el valor indicado en las hojas de características del LNA para esta frecuencia. Esta desadaptación de impedancias produce una pérdida de potencia por reflexión de:

$$L_{\text{desadaptación}} = -10 \log(1 - |S_{11}|^2) = 0.06\text{dB}$$

Debido al bajo valor de esta pérdida, no se ha diseñado una red de adaptación de impedancias en este caso.

Solo queda señalar que este filtro va conectado directamente a la entrada de RF, a través de un conector de tipo “F” soldado directamente a la placa.

4.2. Amplificador RF de bajo ruido



Z_S = 50Ω (Del filtro de preselección)

Z_{in} = 63-j3 Ω (Entrada al LNA)

Z_{out} = 53+j4 Ω (Salida del LNA)

Z_L = 53-j4 Ω (Red adaptadora)

$L_{desadaptación}$ = 0.06dB

Figura 11: Diagrama de impedancias para el LNA.

En cualquier cadena de recepción, el primer amplificador es el más importante en cuanto al factor de ruido global resultante, como se deriva de la ecuación de Friis (ver apartado 9.3). Por eso, este primer amplificador debe ser un amplificador de bajo ruido (LNA) para degradar lo menos posible la calidad de la señal. Para este amplificador se consideró la opción de implementarlo mediante transistores discretos o mediante un integrado. Sobre todo por la ventaja en cuanto a superficie ocupada, y por la sencillez del diseño, se optó realizarlo mediante un integrado, el MAX2611 de la compañía *Maxim Integrated Circuits*.

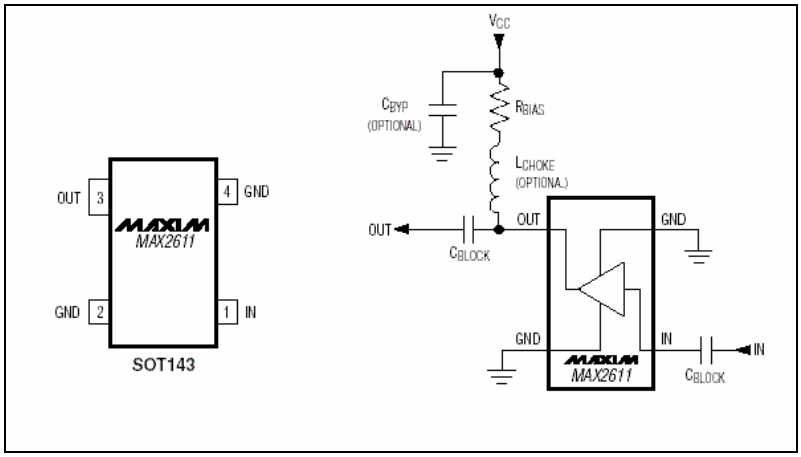


Figura 12: Integrado elegido como LNA.

Es un amplificador que funciona hasta 2GHz, y para la banda de frecuencias de trabajo de este proyecto, en torno a 100Mhz, su factor de ruido es F_{LNA}=3dB. Su ganancia es

controlable mediante la corriente de polarización (regulable a través de una resistencia), según esta gráfica:

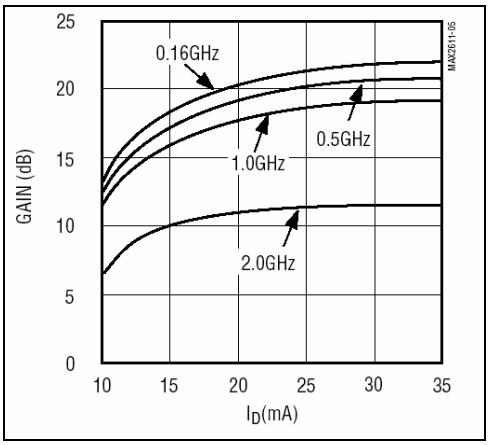


Figura 13: Ganancia en función de la corriente.

Lógicamente interesa una ganancia lo más alta posible, pero cuanto mayor sea ésta, mayor también el factor de ruido. Cómo en esta etapa es el factor de ruido lo que queremos minimizar, aún a costa de no usar la máxima ganancia del amplificador se escoge como corriente de funcionamiento I_D = 16mA, la recomendada por el fabricante. Según la gráfica anterior, con esto se consigue una ganancia de G_{LNA}=18dB. Con el valor fijado para la corriente y con un voltaje de polarización fijo a 3.8V, resulta una resistencia de polarización de 75Ω. El circuito queda así:

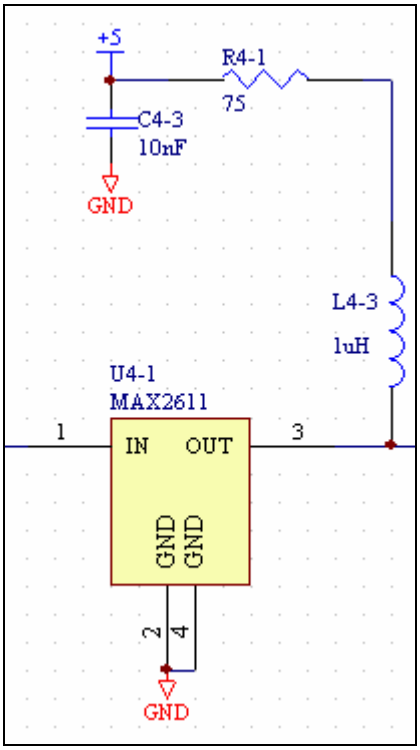


Figura 14: Implementación del LNA.

Dónde la bobina L4-3 actúa como choque de radiofrecuencia. Es necesaria para que la resistencia de polarización de 75Ω no disminuya la impedancia de carga y produzca la desadaptación de la etapa de salida, con la consiguiente pérdida de ganancia. Para obtener un indicador de esta pérdida de potencia, se han realizado unos scripts en MATLAB, que se pueden encontrar en el anexo A5.1, y que muestran la pérdida de potencia en función del valor de esta inductancia:

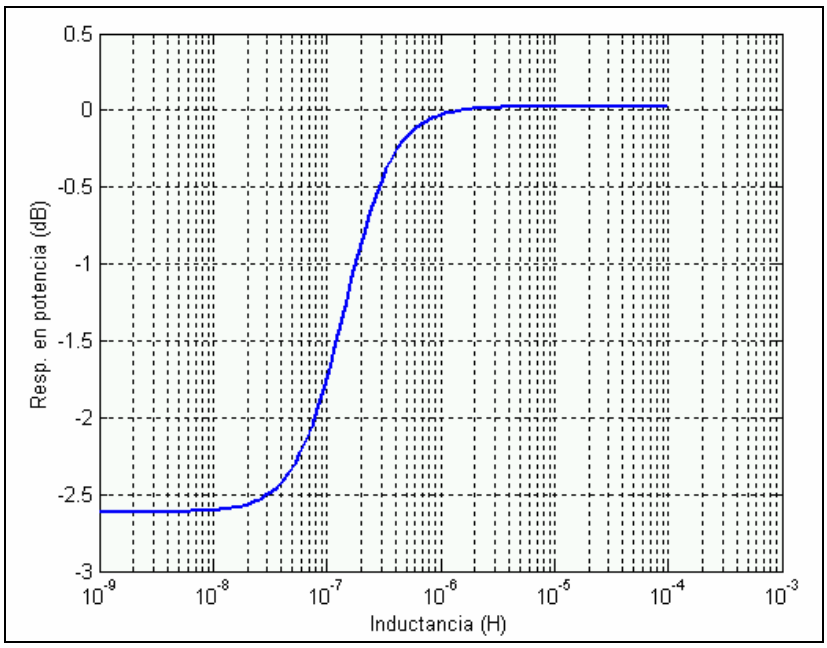


Figura 15: Pérdida de potencia en función de la inductancia del choque.

Como se puede ver claramente, a partir de cerca de 1µH, la pérdida de potencia se hace despreciable, siendo esta la justificación de haber escogido este valor para el choque. En cuanto a las impedancias de entrada y salida ambas son cercanas a 50Ω para la frecuencia de trabajo:

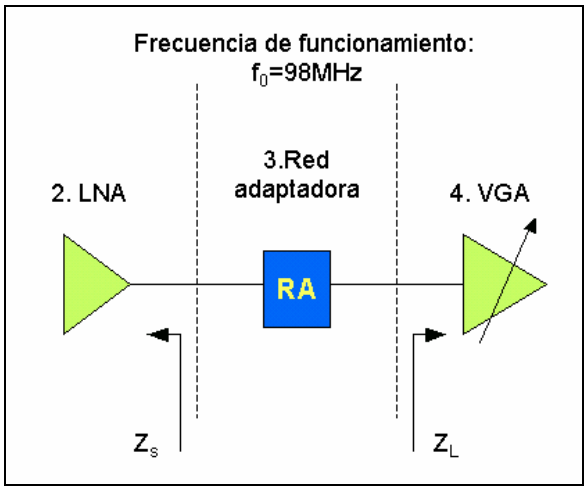
$$S_{11} = 0.12 \angle -11º \ @ \ 100 \text{ MHz} \rightarrow Z_{in} = 63 - j3 \ \Omega$$

$$S_{22} = 0.05 \angle 53º \ @ \ 100 \text{ MHz} \rightarrow Z_{out} = 53 + j4 \ \Omega$$

Los condensadores de desacople de continua no se han añadido, ya que tanto a la entrada como a la salida se encuentran condensadores en serie y por lo tanto bloquean la señal continua.

4.3. Redes adaptadoras de impedancias

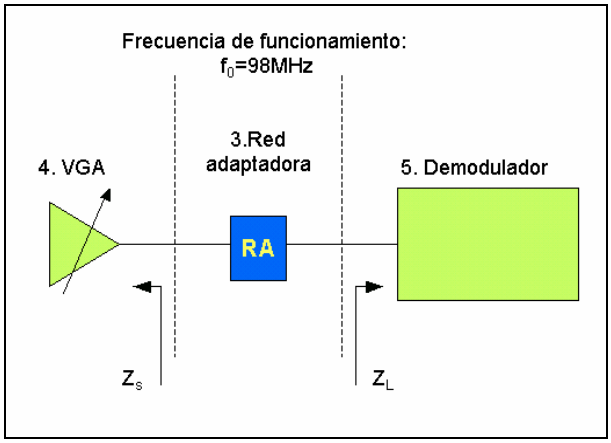
Entre los valores de impedancias que son un estándar de facto en el mercado de los circuitos de radio y microondas, están los de 50Ω, 75Ω y 200Ω. Mientras que los valores de 50Ω y 75Ω suelen usarse para señales de RF, el valor de 200Ω es usado habitualmente en circuitos de frecuencia intermedia (IF). En el diseño del receptor objeto de este proyecto, se ha usado un amplificador de RF de ganancia variable, que se presenta en el siguiente apartado, pero que está pensado para circuitos de IF, y por lo tanto para impedancias de entrada de 200Ω. Lo mismo ocurre con el integrado usado como demodulador en cuadratura. Por lo tanto se han de diseñar sendas redes adaptadoras de impedancias para los dos casos. En las siguientes figuras se muestran las posiciones de las redes en el camino de la señal.



$Z_S = 53 + j4\ \Omega$ (Salida del LNA)

$Z_L = 190 - j60\ \Omega$ (Entrada del VGA)

Figura 16: Red adaptadora a la entrada del VGA.

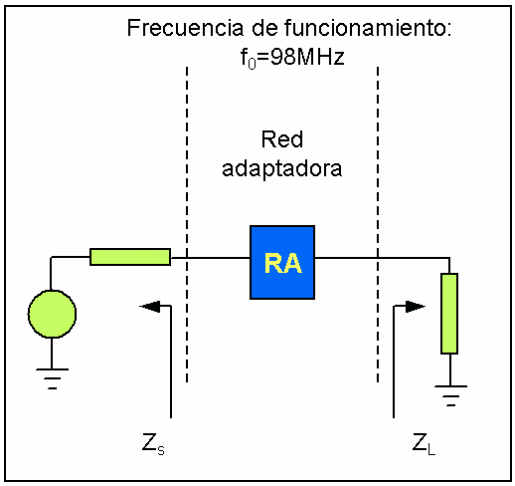


$Z_S = 53 + j2\ \Omega$ (Salida del VGA)

$Z_L = 195.6 - j29.5\ \Omega$ (Entrada al demodulador)

Figura 17: Red adaptadora a la salida del VGA.

Los valores de impedancias de entrada y salida tanto del LNA, del VGA y del demodulador, se presentan en sus respectivos apartados, que son el 4.2, el 4.4 y el 4.5.
Aunque los valores de impedancias de fuente y carga son distintos en cada una de las dos redes adaptadoras, son lo suficientemente parecidos como para justificar la aproximación del diseño de una sola red adaptadora, con impedancias de fuente y carga que se muestran a continuación:



$Z_S = 53\ \Omega$

$Z_L = 193 - j\ 45\ \Omega$

Figura 18: Red adaptadora a diseñar.

Donde como Z_S y Z_L se han usado la media de las impedancias indicadas más arriba, y la parte imaginaria de Z_S se ha despreciado por su bajo valor.
A la hora de elegir un tipo de red adaptadora, se consideraron dos opciones:

- Red resistiva: Por ejemplo, un adaptador en "L". Su ventaja consiste en una respuesta constante en frecuencia, pero se desechó esta opción por representar una perdida de potencia entregada a la carga. Esta atenuación es inaceptable en las primeras etapas de un receptor, ya que además de disminuir la potencia de señal recibida, afectaría al factor de ruido global del receptor.
- Red reactiva: Conteniendo solamente condensadores e inductancias. Su desventaja consiste en conseguir la plena adaptación solamente para una frecuencia determinada (la frecuencia de diseño) y para una estrecha banda de frecuencias alrededor de esta. Su ventaja es que no disipa potencia, al menos con componentes ideales.

Se ha optado por el diseño de redes adaptadoras reactivas, principalmente por estos motivos:

- No representan una disipación de potencia de señal, lo que sí ocurre en redes resistivas.
- Se ha aprovechado la variabilidad de la adaptación en función de la frecuencia para introducir otra etapa de filtrado de preselección. La forma en que se ha conseguido este filtrado se detalla a continuación.

Cuando se va a diseñar una red adaptadora reactiva, se dispone de muchos tipos de redes que realizan la adaptación, y que se diferencian fundamentalmente en el tipo de respuesta global que tienen en frecuencia. Aunque todas ellas adapten las impedancias perfectamente para la frecuencia de diseño, el qué ocurre en otras frecuencias sirve para distinguir estas redes como de filtro paso bajo, paso alto o paso banda. Además, dentro de cada tipo hay varias posibilidades, por ejemplo, según el orden de las distintas etapas. En este proyecto, se ha utilizado la herramienta de diseño "*Impedance Matching Network Designer*" [7], que es capaz de sintetizar hasta dieciséis tipos distintos de redes dadas una frecuencia de trabajo y las impedancias complejas de entrada y salida.

Ahora bien, aunque las dos redes adaptadoras que se necesitan (a la entrada del VGA y a la entrada del demodulador) tienen especificaciones idénticas, su implementación se ha realizado con distintos circuitos: en el caso de la primera se ha usado un tipo paso alto y en segundo un tipo paso bajo, en ambos casos de segundo orden. Los prototipos correspondientes se ven en la siguiente figura.

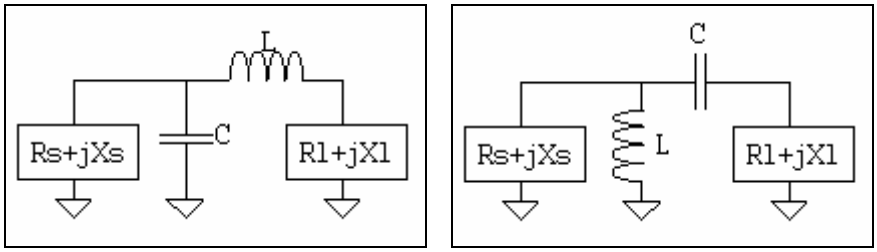


Figura 19: Redes adaptadoras paso bajo y paso alto.

La idea de diseñar cada red adaptadora con un tipo distinto es que el efecto combinado, tomando todas las etapas de radiofrecuencia del receptor en cascada, produzca un filtrado paso banda, eliminando potencia de las señales fuera de la banda de interés. Se han escogido las dos redes más simples de todas las existentes, pensando en el área ocupada por los componentes en las placas, y su uso queda justificado por cumplirse los requisitos de ancho de banda deseados, como se verá a continuación.

4.3.1. Red adaptadora paso alto

Usando los valores de impedancias complejas para la fuente y la carga que se especificaron arriba, se obtienen estos valores para esta primera red adaptadora:

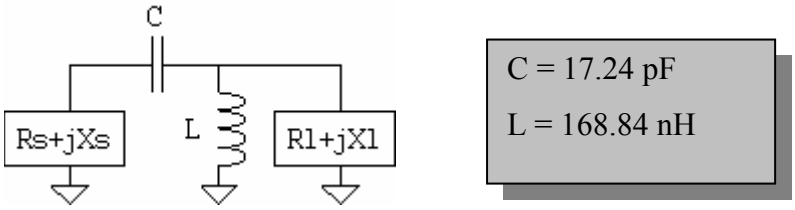


Figura 20: Red adaptadora paso alto y valores ideales.

En la implementación real que se ha hecho de este adaptador se han usado los valores más cercanos disponibles comercialmente. Hay que señalar además que el condensador actúa de desacoplo de continua a la salida del LNA, y a que en la entrada del VGA (la siguiente etapa) se ha añadido otro condensador de desacoplo ya que la salida del adaptador es una inductancia hacia masa:

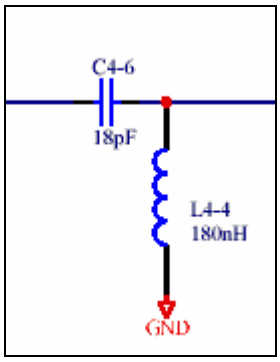


Figura 21: Implementación real de la red adaptadora paso alto.

Mediante unos *scripts* realizados en MATLAB (cuyo código fuente se muestra en el anexo A5.1) se ha verificado este diseño, calculando y representado gráficamente tres factores de esta red adaptadora:

- Función de transferencia de potencia: Representa la relación, en decibelios, entre la potencia entregada a la parte real de la impedancia de carga con respecto al máximo, que es la potencia disponible del generador.

$$G(f) = 10 \cdot \log\left(\frac{P_L}{P_{disp}}\right)$$

En la gráfica se pueden ver las respuestas tanto para los valores ideales de los componentes (en rojo y discontinuo) como para los valores reales utilizados en el receptor (en azul y continuo). Se puede apreciar que la respuesta prácticamente no varía, y que la frecuencia de corte a 3dB cae lejos (sobre los 70Mhz) del rango de trabajo:

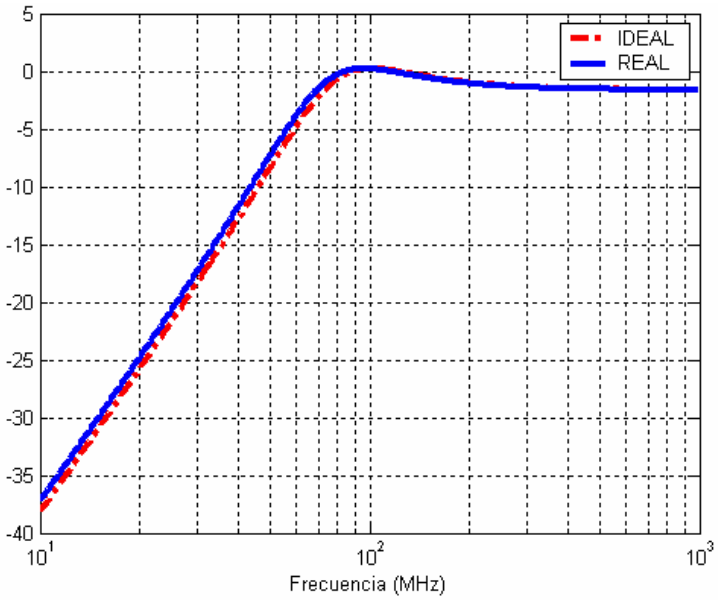


Figura 22: Función de transferencia de potencia.

- Coeficiente de reflexión a la entrada: Representa lo bien adaptada que está la impedancia de carga que el generador ve a la entrada de la red, y se mide mediante el coeficiente de reflexión:

$$\Gamma = \frac{Z_{in} - Z_S}{Z_{in} + Z_S}$$

En la gráfica se muestra el módulo del coeficiente de reflexión, tanto para los valores ideales (en rojo y discontinuo) como para los valores realmente usados (en azul y continuo). Se puede ver como para el rango de frecuencias de interés el factor es prácticamente cero, lo que indica que la señal no es reflejada, sino que pasa hasta la carga.

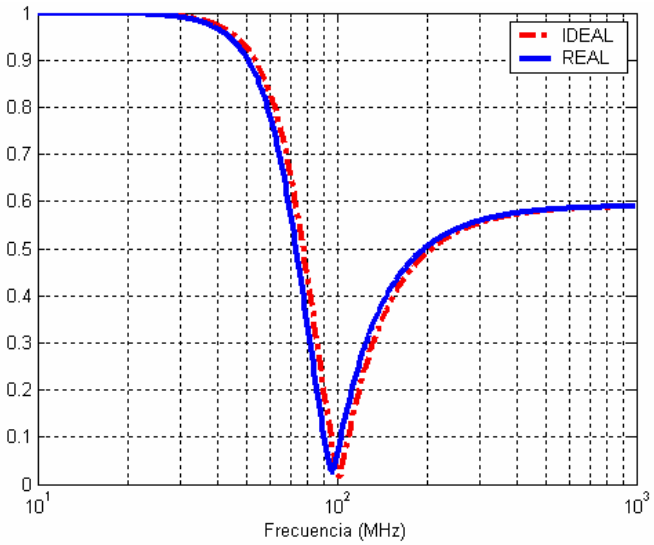


Figura 23: Coeficiente de reflexión a la entrada del adaptador.

- Factor de potencia entregada: Aunque el coeficiente de reflexión ya es un indicador de qué parte de la potencia incidente es reflejada y cuál pasa, este factor no se puede utilizar para encadenar varios elementos en cascada, que es lo que se hará posteriormente. Para eso se utiliza este factor, que indica en tanto por uno de potencia incidente que realmente se entrega a la carga, asumiendo que los componentes reactivos sean ideales y no presenten pérdidas:

$$F = \left(1 - |\Gamma|^2\right)$$

Se puede ver como para las frecuencias de interés, este factor está cerca del 99.9%

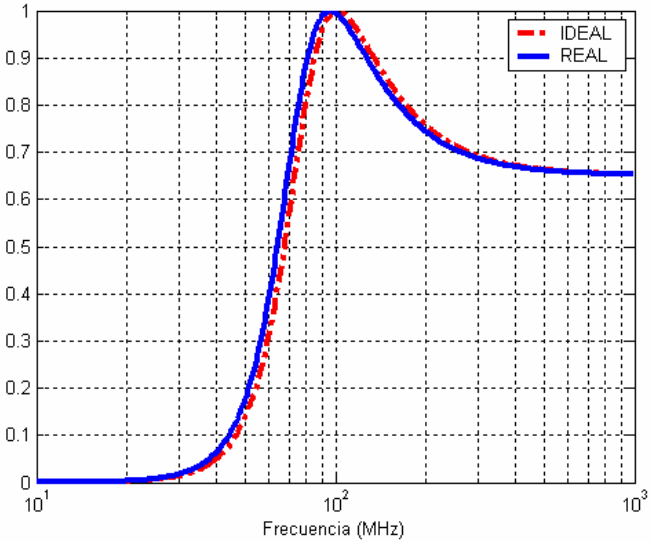


Figura 24: Factor de potencia entregada a la carga.

4.3.2. Red adaptadora paso bajo

En este caso se ha obtenido este diseño, con estos valores ideales para los componentes:

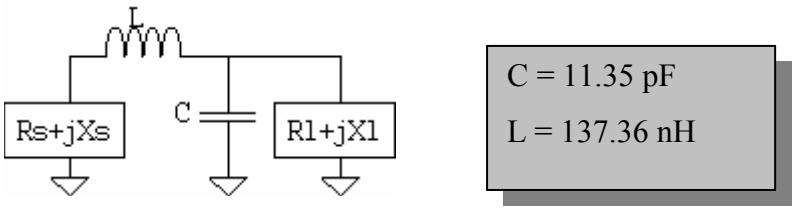


Figura 25: Red adaptadora paso bajo y valores ideales.

Y esta es la implementación real que se ha hecho de este adaptador:

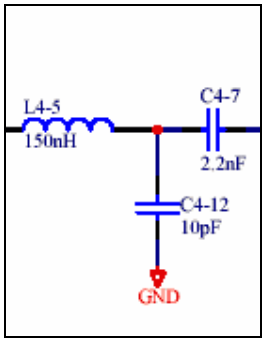


Figura 26: Implementación real de la red adaptadora paso bajo.

En este caso, la entrada del adaptador es la salida del VGA, y aunque el primer elemento es una inductancia en serie, no es necesario un condensador de desacoplo de continua porque a continuación sólo hay condensadores, siendo C4-7 un condensador de desacoplo, ya que esa salida irá a la entrada del demodulador en cuadratura.

Igual que con el adaptador paso alto, se ha realizado la simulación en MATLAB para asegurar el correcto diseño del adaptador:

- Función de transferencia de potencia: Se puede ver como, al igual que en el caso anterior, la frecuencia de corte de 3dB cae bastante alejada (sobre los 160Mhz) del rango de frecuencias de trabajo.

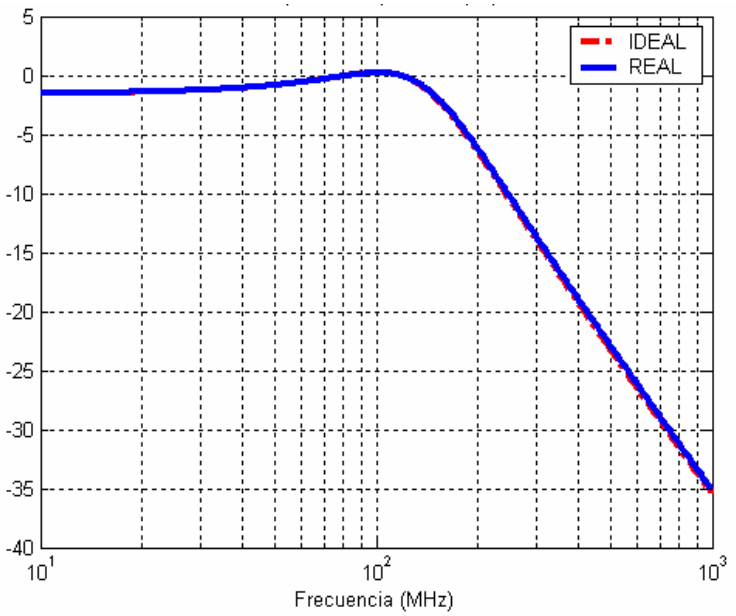


Figura 27: Función de transferencia de potencia.

- Coeficiente de reflexión a la entrada: Para la banda de trabajo, el coeficiente de reflexión está en torno a 0.07, valor suficientemente bajo:

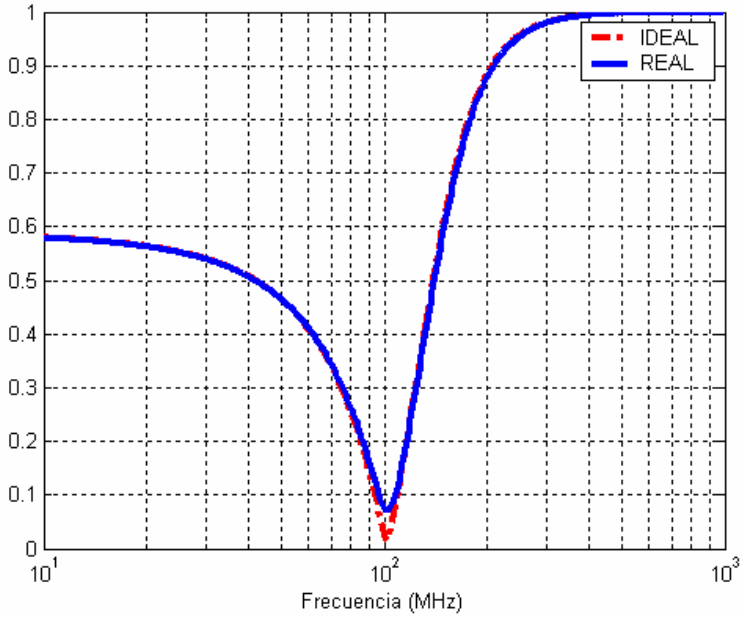


Figura 28: Coeficiente de reflexión a la entrada del adaptador.

- Factor de potencia entregada: Este factor se mantiene también en este caso cercano a la unidad, concretamente alrededor del 99.5%

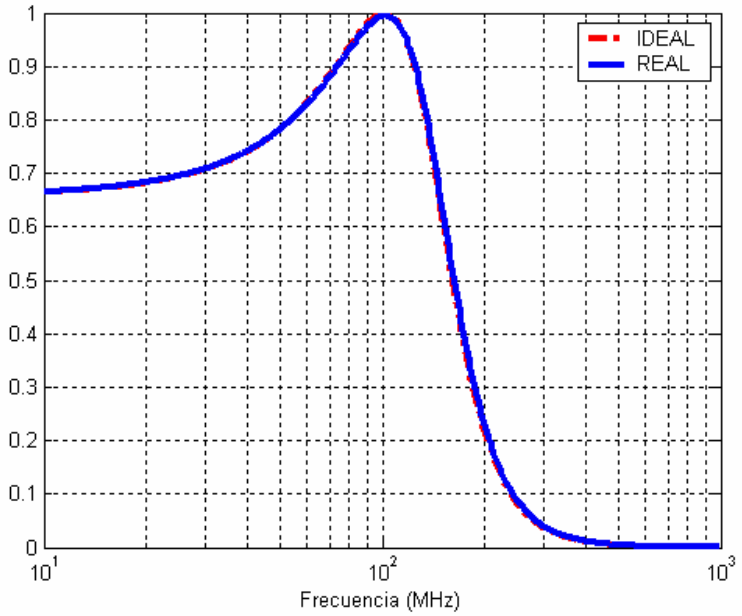


Figura 29: Factor de potencia entregada a la carga.

4.3.3. Análisis del efecto combinado

Aunque entre las dos redes adaptadoras se encuentra el VGA, su efecto en la ganancia global de la cadena de la señal en RF se puede aproximar mediante la superposición de ambas funciones de transferencia de potencia (mediante suma, ya que están en decibelios) y mediante los factores de potencia entregada a la carga (multiplicándolos, ya que están en tanto por uno). En el primer caso se obtiene la esperada respuesta paso banda, centrada en 98.8MHz y con un ancho de 3dB de 92Mhz.

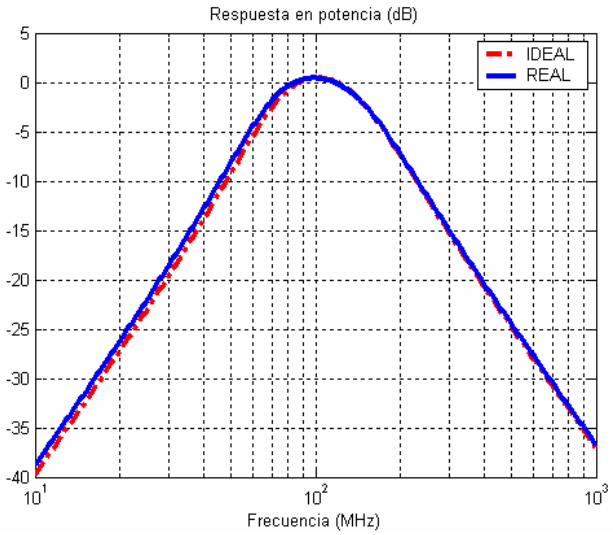


Figura 30: Función de transferencia de potencia combinada.

En cuanto a los factores de transferencia de potencia, si se combinan de esta forma:

$$F = \left(1 - \left|\Gamma_{paso\_alto}\right|^2\right) \cdot \left(1 - \left|\Gamma_{paso\_bajo}\right|^2\right)$$

Se obtiene la respuesta que se ve en la siguiente gráfica, donde se puede comprobar que para la banda de trabajo se transmite el 99.1% de la potencia:

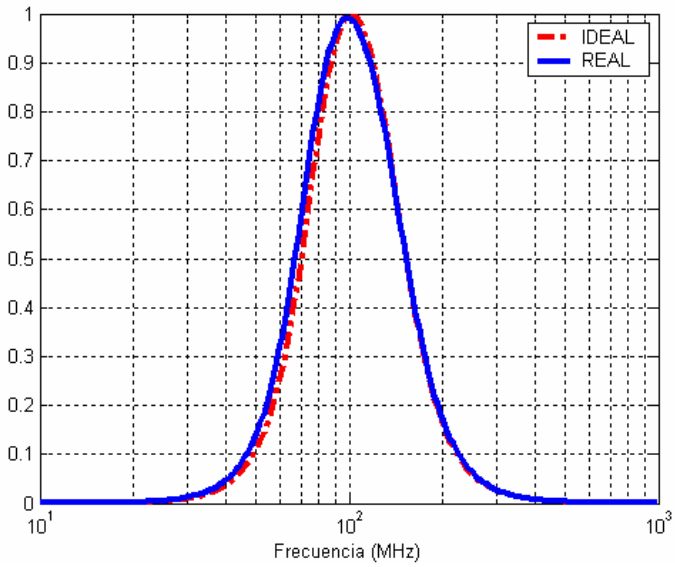


Figura 31: Transferencia de potencia combinada.

Realmente las dos gráficas representan lo mismo, aunque obtenidas de dos formas independientes, y representando la segunda en decibelios se ha comprobado que efectivamente coinciden, demostrando la validez de todos los cálculos.

En resumen, se han diseñado los dos adaptadores de impedancias necesarios para que no se pierda potencia entre etapas por desadaptación, y se ha usado la desadaptación para añadir un filtrado extra a las señales (interferencias o ruido) fuera de la banda de trabajo del receptor.

4.4. Amplificador RF de ganancia variable

En este segundo amplificador ya no es tan importante el factor de ruido, sino una alta ganancia. Sin embargo, es deseable un control sobre la ganancia total del sistema para aumentar el margen dinámico. Por ello para este segundo amplificador se ha escogido el integrado AD8367, que es regulable en ganancia de forma lineal en decibelios. Aunque incluye soporte para crear un bucle de control automático de ganancia (AGC), no se hará uso de él, ya que será el software desde el PC el que regule esta ganancia.

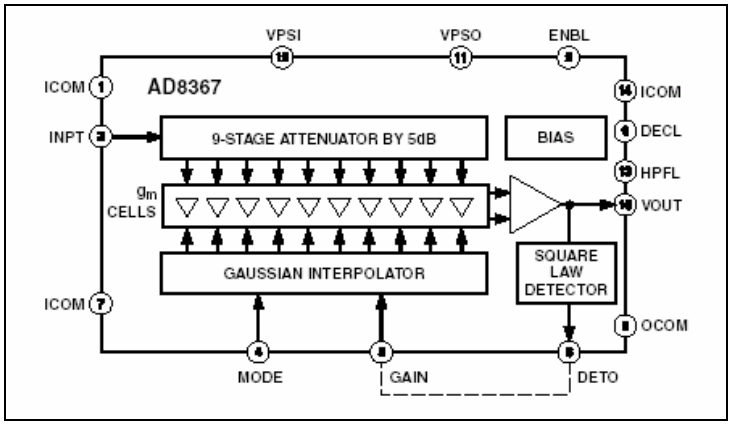


Figura 32: Diagrama interno del VGA.

En la siguiente gráfica se muestra la variación de la ganancia de potencia de este amplificador en función de la frecuencia y para distintos valores de la tensión de control de la ganancia.

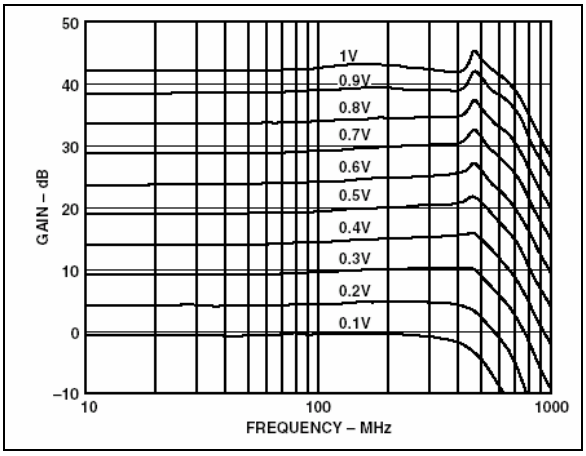


Figura 33: Ganancia en función de la frecuencia y tensión de control.

Se puede observar que para el rango de frecuencias de funcionamiento, en torno a los 100MHz, la ganancia se presenta constante, lo que es deseable. Además, el margen de ganancia oscila entre 42.5dB y –2.5dB, lo que da 45dB de margen de control. Su

impedancia de entrada es de 200Ω, dato utilizado en el diseño del segundo filtro de preselección.

Como la entrada de control de ganancia es analógica, se usará uno de los conversores digital-analógicos para este fin (apartado 5.6). Dada la resolución de estos, de ocho bits, y que la sensibilidad a la tensión de control es 20mV/dB, se puede calcular el mínimo escalón de ganancia que se puede añadir o quitar desde software:

$$\Delta G = \frac{\text{dB}}{20\,\text{mV}} \frac{5\,\text{V}}{2^8} = 0.977\,\text{dB}$$

Es decir, la ganancia se puede controlar con una precisión ligeramente mejor que un decibelio. El chip tiene también un modo de ahorro de energía (*powerdown*), señal que es controlada desde el microcontrolador para poner todos los integrados del sistema simultáneamente en modo de bajo consumo cuando no estén en uso. Como esta señal tiene el mismo uso en todos los integrados del receptor, no se volverá a comentar en los siguientes apartados. Sólo son necesarios algunos condensadores de desacoplo, y dos resistencias en serie para la alimentación del circuito (también para evitar acoples en la línea de alimentación) y el circuito queda como se ve en el siguiente esquema.

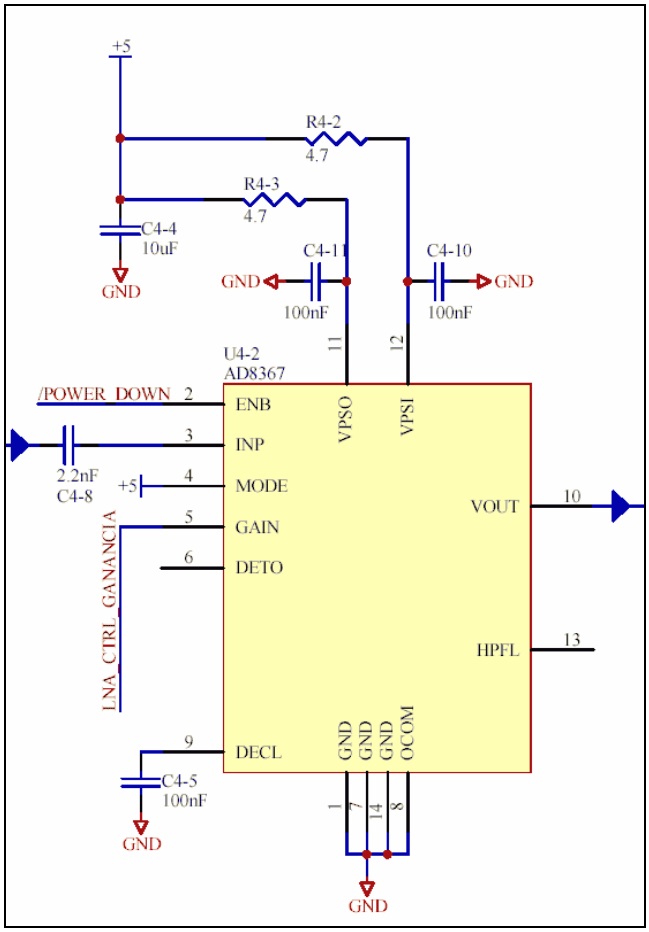


Figura 34: Implementación del VGA.

Hay que hacer las siguientes aclaraciones sobre el esquema anterior:

- La señal `LNA_CTRL_GANANCIA` es el valor analógico que controla la ganancia del amplificador, como se indicaba anteriormente.
- La entrada `MODE` permite elegir el signo de la sensibilidad de la ganancia del amplificador. Es una entrada digital que permite seleccionar si la ganancia debe aumentar o disminuir con el voltaje de control. Arbitrariamente se ha tomado la decisión de usar el modo de ganancia creciente con el voltaje y por lo tanto la entrada `MODE` se ha puesto a valor lógico alto.

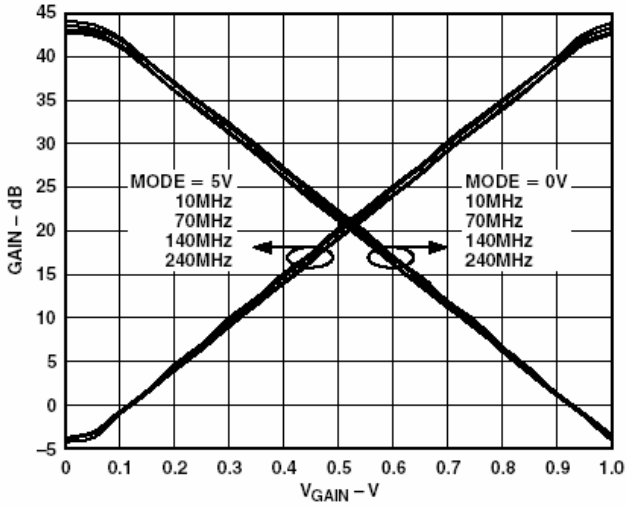


Figura 35: Los dos modos de ganancia del VGA.

- La salida `DETO` proviene del detector de potencia RMS interno a este integrado. Como se comentó antes, permite crear lazos de control automático de ganancia, pero en este proyecto no se utiliza.
- La patilla HPFL está pensada para colocar un condensador de desacoplo, que permite fijar la mínima frecuencia para la que el amplificador funcionará. Como se ve en la siguiente gráfica, sino se coloca ningún condensador la capacidad interna establece esta frecuencia en torno a 1MHz. Como el rango de frecuencias de trabajo para este proyecto está en torno a 100MHz, esto no supone ninguna restricción y esta es la razón de no haber colocado ningún condensador en este pin.

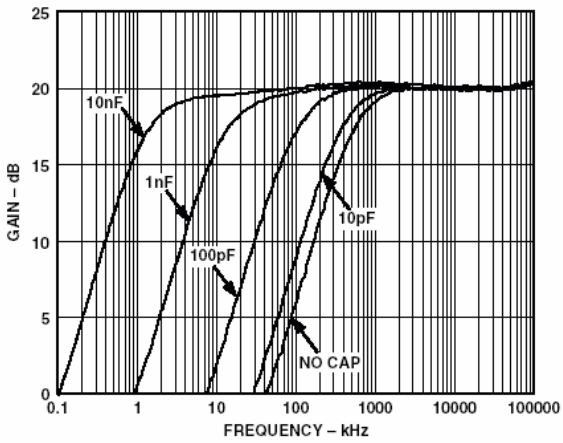


Figura 36: Frecuencia de corte inferior del VGA según el condensador de desacople.

En cuanto a las impedancias de entrada y salida de este amplificador, se han utilizado las siguientes gráficas para estimarlas en torno a 100MHz:

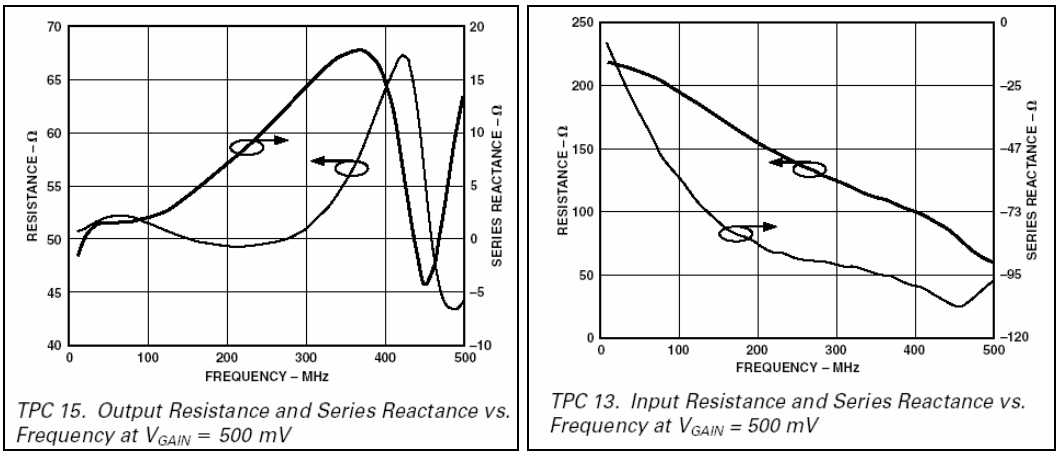
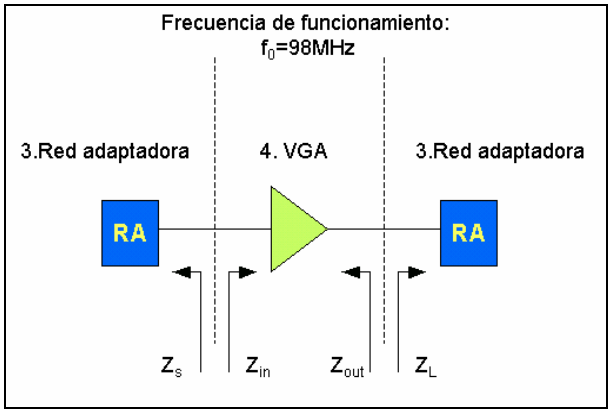


Figura 37: Las impedancias de entrada y salida para el VGA.

Con lo que se obtienen los siguientes valores de impedancias, que ya han sido utilizados en el diseño de las redes adaptadoras de impedancias en el apartado anterior.



$Z_{in} = 190 - j\,60\ \Omega$

$Z_{out} = 53 + j\,2\ \Omega$

Figura 38: Diagrama de impedancias para el VGA.

4.5. Demodulador en cuadratura

4.5.1. Introducción

Como demodulador en cuadratura se ha elegido el integrado AD8348, un demodulador diseñado para señales de radiofrecuencia, o frecuencia intermedia, entre 50MHz y 1GHz, cuyo esquema interno se muestra en la siguiente figura.

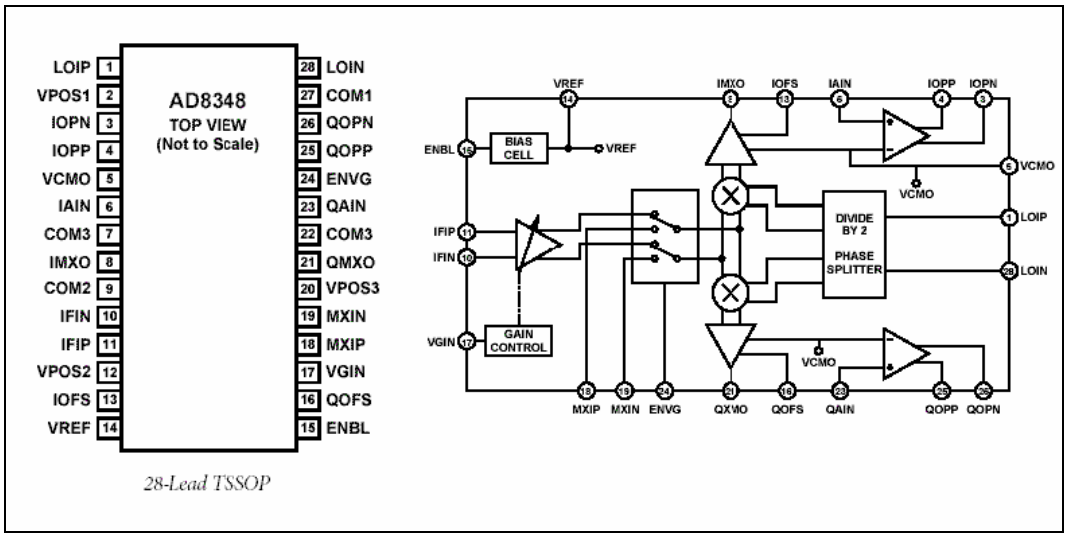


Figura 39: Integrado elegido como demodulador.

Por claridad se repite aquí parte del esquema general del receptor que se mostraba en el apartado 3.2, donde se ve la conexión del demodulador con otros elementos del sistema:

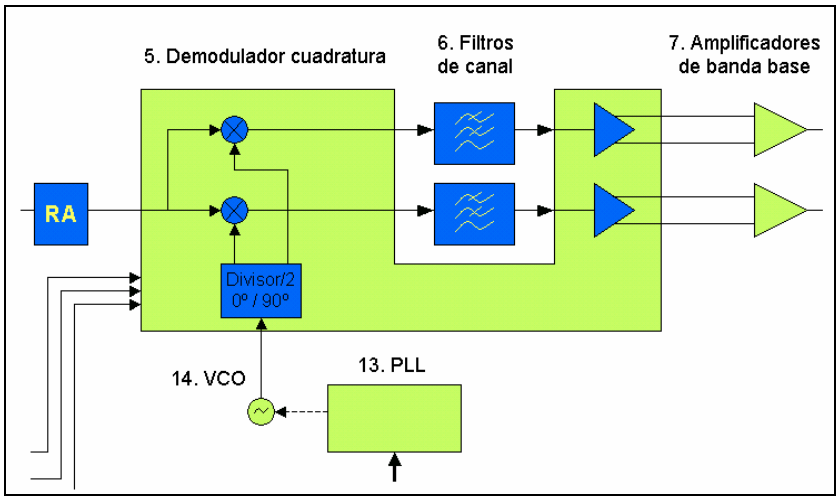


Figura 40: Conexiones del demodulador con otros elementos del receptor.

El integrado se compone de:

- Una entapa de entrada de RF (aunque en el integrado se le llama entrada de IF, esto no importa) con otro amplificador VGA. Esta etapa proporciona otro margen de 44dB de control de ganancia, lo que aumenta el rango dinámico total del sistema. La impedancia de entrada nominal es de 200Ω, aunque más adelante se obtendrá un valor más exacto.
- Un multiplexor para permitir una entrada alternativa de RF, que en este proyecto no se usa.
- Dos mezcladores de celdas Gilbert, para extraer las componentes en fase y cuadratura.
- Un divisor en cuadratura del oscilador local. Estos divisores se usan para obtener dos osciladores desfasados 90º con mucha precisión a cualquier frecuencia, a coste de requerir una entrada exactamente al doble de frecuencia de la deseada. Este dato es básico para el diseño del VCO, ya que varía el rango de funcionamiento de éste, como se recordará en los apartados 4.8 y 4.9.
- Un amplificador en banda base, con filtrado externo al integrado anterior al amplificar opcional, que sí se ha añadido en este proyecto.

4.5.2. Entradas y salidas del demodulador

Tanto las entradas del oscilador local, la entrada de RF y las dos salidas de banda base son diferenciales. En el caso de la entrada de RF, la impedancia diferencial de entrada nominal es de 200Ω, aunque se proporciona la siguiente gráfica con el valor exacto dependiendo de la frecuencia.

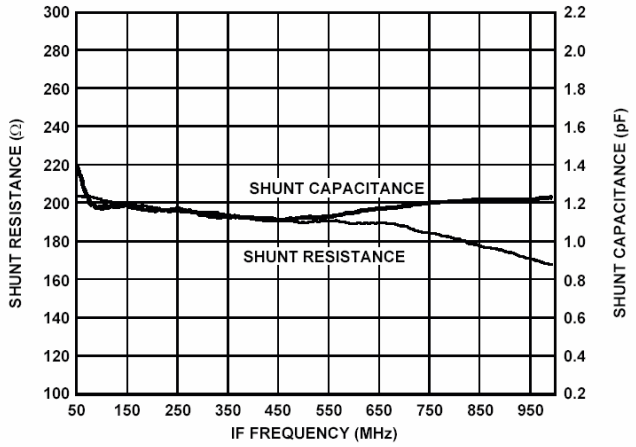


Figura 41: Impedancia de entrada al demodulador.

En este caso no se da directamente las partes reales e imaginarias de la impedancia de entrada (como en el integrado VGA), sino un modelo de resistencia y capacitancia en paralelo. Mirando sus valores para la frecuencia de trabajo, en torno a 100MHz, se obtiene R=200Ω y C=1.2pF. Siguiendo unos sencillos cálculos se obtiene que la impedancia compleja de entrada es de Z_{IN} = 195.6 – 29.5jΩ, siempre para 100MHz. Este valor es el que se usó en el apartado 4.3 para el diseño de los adaptadores de impedancias.

En cuanto a la entrada del oscilador local, se recomienda usar un transformador (*balun*) para convertir la señal en balanceada, aunque se permite su entrada directamente en forma unipolar acoplando la entrada negativa a masa a través de un condensador, que es lo que se ha hecho. Además, hay que añadir una resistencia de 60.4Ω en paralelo a esta entrada para conseguir una impedancia de entrada próxima a 50Ω. En la siguiente gráfica se muestra esta disposición, que es la adoptada en este proyecto, y las pérdidas por desadaptación que supone el no usar un *balun*.

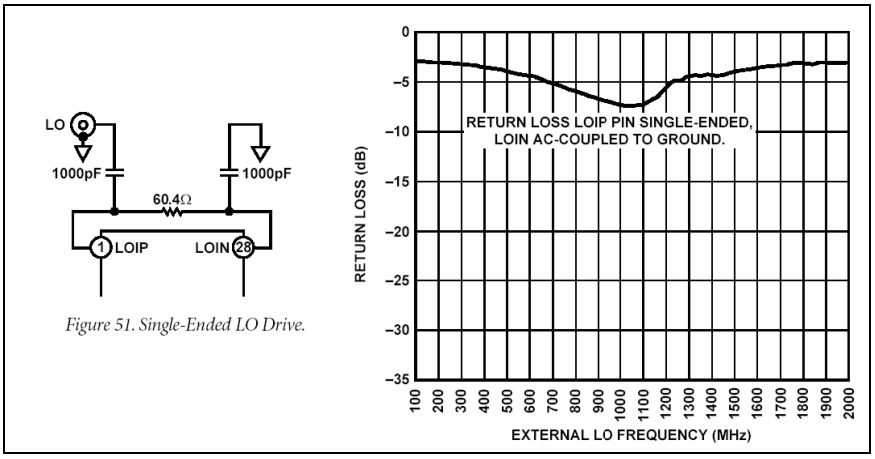


Figura 42: Adaptador a la entrada del LO y pérdidas por desadaptación.

Aparte de esto, el integrado necesita un voltaje de referencia para su entrada `VCMO`, que será el voltaje medio sobre el que se superpondrán las señales diferenciales de salida I y Q (la señal demodulada en fase y cuadratura, respectivamente). Para esta referencia de voltaje se ha utilizado el generador incorporado en el mismo integrado, con una salida nominal de 1.0V, disponible externamente a través de una salida del chip. Este voltaje de referencia se ha desacoplado a masa mediante varios condensadores de distintos valores, como es necesario para evitar ruidos en esta referencia que se reflejarían directamente en la salida.

El demodulador permite intercalar filtros de banda base en el camino de las señales I y Q justo antes del último amplificador dentro del integrado. Para ello, la señal sale por los terminales IMXO y QMXO en forma unipolar, pero referenciadas a VCMO, no a masa, y tras su filtrado (detallado en el siguiente apartado) entran por IAIN y QAIN, respectivamente, para volver a salir amplificadas en forma diferencial hacia los amplificadores diferenciales. En el siguiente esquema se muestra como queda finalmente el circuito en torno al demodulador.

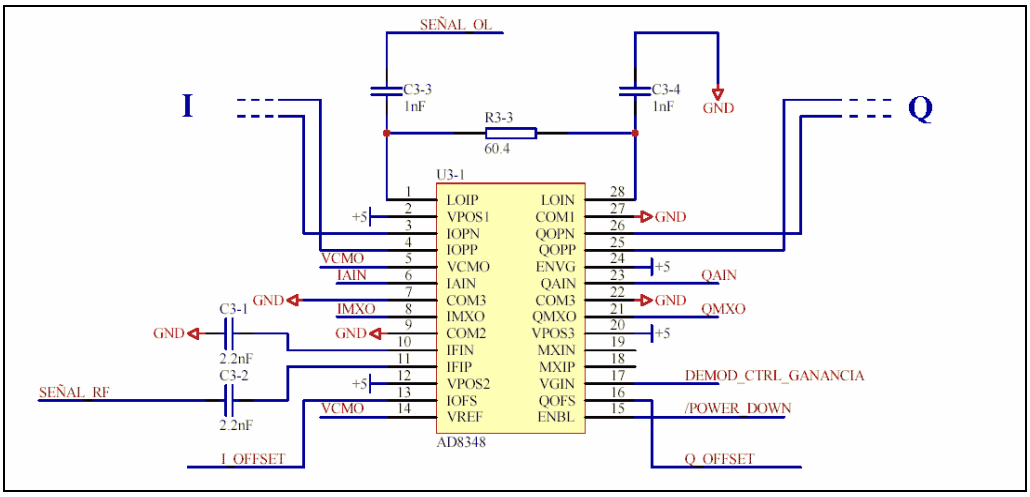


Figura 43: Implementación del demodulador.

Las salidas diferenciales correspondientes a las señales I y Q son las que van a los amplificadores de banda base (apartado 4.7), que se encargan de convertirlas en señales unipolares. El amplificador variable interno del demodulador es controlado a través de la señal `DEMOD_CTRL_GANANCIA` desde un conversor digital-analógico.

4.5.3. Control de la ganancia

En cuanto a la resolución del control de esta ganancia, en este caso la sensibilidad es de 50dB/mV, con lo que el mínimo escalón de ganancia modificable con el conversor digital-analógico de ocho bits es:

$$\Delta G = \frac{50\,\text{dB}}{\text{V}} \frac{5\,\text{V}}{2^8} = 0.976\,\text{dB}$$

Por lo que en este caso la ganancia también puede ser controlada con una resolución mejor que un decibelio. En este integrado la ganancia disminuye al aumentar el voltaje de control, siguiendo esta relación:

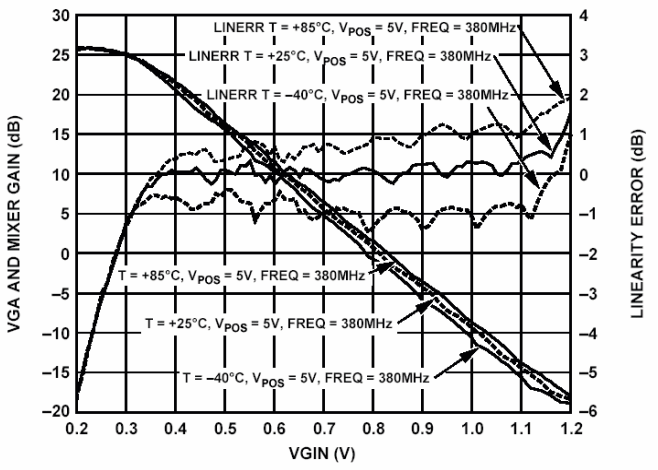


Figura 44: Ganancia (sin incluir el último amplificador banda base) en función del voltaje para el demodulador.

4.5.4. Compensación del *offset* a las salidas

Como ya se comentó en la introducción a la demodulación ZIF (primer capítulo), un problema de la demodulación en cuadratura es el ajuste del *offset* de voltaje de las señales en fase y cuadratura. Estos offsets son desconocidos en principio y, como se ha comprobado, dependen de multitud de variables, como las potencias de entrada o la ganancia, y estan provocadas por el pequeño, pero no nulo, acople del oscilador local a las entradas de los demoduladores. El integrado dispone para solucionar esto de dos señales que se pueden utilizar de dos formas distintas:

- Como filtros paso alto. En este modo se colocan condensadores entre estos terminales y masa, resultando en un filtrado paso alto de las señales en fase y cuadratura, y por lo tanto eliminando cualquier *offset* de continua.
- Como entradas de voltaje desde un coversor DAC externo. La idea es generar el voltaje necesario en cada momento para compensar el *offset*.

En este proyecto se optó inicialmente por la segunda opción, ya que permite un ancho de banda hasta 0Hz en la señal de banda base compleja. Sin embargo se tuvieron problemas para eliminar el offset en todos los casos, ya que este bucle de control hay que cerrarlo desde el PC, donde solamente se dispone de las señales muestreadas, generadas como se verá posteriormente a partir de las señales diferenciales de salida del demodulador. El problema es que estas salidas diferenciales están limitadas a un rango de voltaje que, de superarse, provoca una saturación que distorsiona la señal diferencial hasta el punto de no poder distinguir entre multitud de posibles valores de offset

causantes de esta situación. Además se daba el problema de un retardo de varios milisegundos en este bucle de control, lo que hacía el control más difícil aún.

Por estas razones, se optó finalmente por usar el segundo método de filtrado paso alto. La fórmula que se da para la frecuencia de corte generada es:

$$f_{paso} = \frac{1}{2\pi \cdot 2650\Omega \cdot C}$$

Como se ha utilizado una capacidad de 0.1μF, se tiene una frecuencia de corte de:

$$f_{paso} = \frac{1}{2\pi \cdot 2650\Omega \cdot 0.1\mu F} = 600Hz$$

Valor suficientemente bajo como para no ser un problema en la recepción de cualquier señal con modulaciones típicas.

4.6. Filtros de banda base

4.6.1. Introducción

Como se ha indicado en el apartado anterior, el demodulador permite intercalar un filtrado de banda base externo al integrado para mejorar la calidad señal a ruido y limitar el ancho de banda de la señal demodulada. En las hojas de características se muestra este ejemplo de cómo implementar uno de estos filtros:

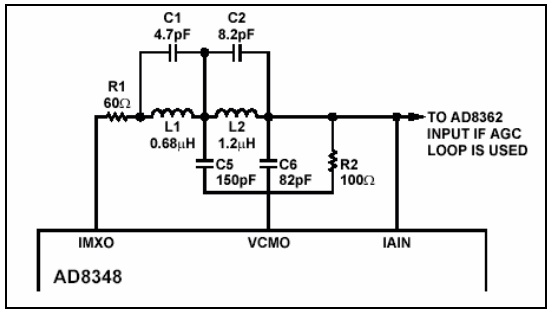


Figura 45: Ejemplo de filtro de banda base (No es el usado en este receptor)

En los filtros diseñados para este receptor se ha usado este filtro de ejemplo en cuanto a:

- Usar una resistencia de carga hacia `VCMO`, que es necesaria porque las entradas `IAIN` y `QAIN` necesitan un nivel de continua igual a `VCMO`.
- Añadir una resistencia de 60Ω a la salida del demodulador. Se ha comprobado que la salida del mezclador se hace inestable fácilmente si no se añade esta resistencia. Sin embargo la inclusión de la resistencia de 60Ω en serie con la salida de los mezcladores producirá una pérdida adicional en la potencia entregada.

Por otro lado, existe la limitación de que la máxima corriente de salida de los mezcladores es de 2.5mA, por lo que a menor resistencia de carga, menor amplitud de voltaje de salida se podrá obtener. Aunque la mínima impedancia de carga recomendada es de 200Ω, esta solamente podría generar una señal de 0.5V de amplitud. Por esto se ha escogido un valor de resistencia de carga en el rango indicado como típico, de 1.2KΩ, que permite señales de hasta 3V de amplitud.

4.6.2. Proceso de diseño

Las características de estos filtros son críticas, ya que marcan un parámetro importante del receptor: el ancho de banda de canal que este es capaz de recibir. Este parámetro está también íntimamente ligado a la frecuencia de muestreo de las señales I y Q, ya que actúa de filtro *antialiasing* para los conversores analógico-digitales. Por un lado, las especificaciones del proyecto indican un ancho de banda de canal de, al menos, 200KHz. Este ancho de banda se refiere al ancho de banda de la señal RF paso banda. Ahora bien, mediante la técnica ZIF y escogiendo la frecuencia del oscilador local justo en mitad del canal, las señales I y Q quedan con un ancho de banda máximo igual a la mitad del ancho de banda paso banda. Por esto, este filtro debe tener una frecuencia de corte superior a 100KHz. Por el lado del *aliasing*, y dado que la frecuencia de muestreo del sistema es 250KHz (apartado 5.4), la frecuencia de corte del filtro debe ser inferior a 125KHz. Más concretamente, hay que elegir un ancho de banda en la parte alta del espectro muestreado en el que la señal no se considere válida. Esto es así porque debido al efecto del aliasing, las componentes espectrales que sobrepasen la frecuencia de Nyquist (125KHz en este caso) aparecerán reflejadas en frecuencias dentro del ancho de banda muestreado (por ejemplo, 130KHz aparecerá en 120KHz). Por lo tanto hay que atenuar estas componentes suficientemente. Una guía que se puede tomar para saber cuanto atenuar la señal es el margen dinámico del conversor ADC. Si la atenuación es superior a este margen, se puede asegurar que aunque la señal fuera de banda tuviera una amplitud máxima, se atenuará hasta ser inferior a un LSB del conversor, siendo su contribución similar al ruido de cuantificación, que es el mínimo posible. En el caso de los conversores de ocho bits, se tiene un margen dinámico de:

$$\mathrm{MD}_{\mathrm{ADC}} = 20 \cdot \log \frac{2^7}{2^0} = 41.1\mathrm{dB}$$

En cuanto a la frecuencia de corte del filtro, hay que tener en cuenta que la atenuación no aparece de inmediato, sino que hay un período de transición. Para maximizar el ancho de banda útil, hay que diseñar de forma que la mitad de esta banda de transición caiga en la frecuencia de Nyquist. Todo esto se ejemplifica en la siguiente figura:

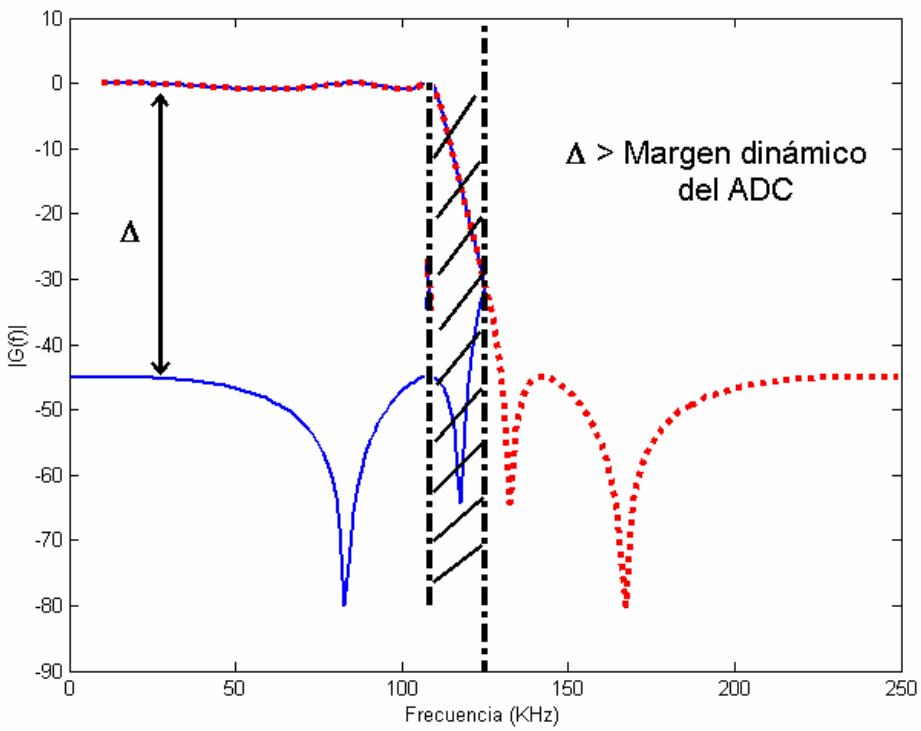


Figura 46: Efecto del aliasing sobre el filtro de banda base.

En la gráfica se representa con línea punteada la respuesta de potencia en frecuencia del filtro de banda base. Sin embargo el efecto del aliasing hace que las frecuencias por encima de 125KHz aparezcan por debajo de esta frecuencia, tal y como se ha representado en línea continua. La zona tachada representa las frecuencias que contendrán componentes espectrales producto de *aliasing* no suficientemente atenuado por lo que serán filtradas posteriormente mediante filtros software. En la gráfica también se puede observar el requisito de la atenuación mínima correspondiente al margen dinámico del conversor ADC.

Teniendo todo esto en cuenta, se ha elegido diseñar un filtro tipo elíptico de quinto orden. El tipo elíptico se ha elegido por ser el filtro que, para un orden dado, resultaba en una zona de transición más abrupta, efecto deseable en un filtro *antialiasing* como éste. El orden se ha escogido lo suficientemente alto como para producir una atenuación importante en la banda eliminada pero que manteniendo el número de componentes dentro de un número razonable pensando en el espacio físico que ocuparán en la placa. Hay que tener en cuenta que los parámetros del filtro elíptico no son independientes, ya que existen estas restricciones:

- A mayor orden del filtro, todos los parámetros se pueden mejorar, pero a costa de:
    - Un mayor número de componentes físicos.

  - Una mayor sensibilidad de la respuesta del filtro ante variaciones en los valores de los componentes. Despues de realizar numerosas pruebas se ha observado que a partir de un séptimo orden, las pequeñas variaciones necesarias para ajustar los valores de los componentes a los disponibles comercialmente provocan grandes cambios de hasta 10dB en algunas zonas de la respuesta del filtro.
- Si se quiere una zona de transición más abrupta hay que aumentar el orden del filtro.
- A menor rizado en la banda de paso, menor atenuación en la banda eliminada.

Teniendo en cuenta todo esto, la frecuencia de corte se ha establecido en 110KHz, el rizado permitido en la banda de paso en 1dB y la frecuencia de corte en 140Khz, que es la simétrica de 110Khz con respecto a la frecuencia de Nyquist de 125KHz. Con estos parámetros se obtiene una atenuación de unos 43dB en la zona rechazada.

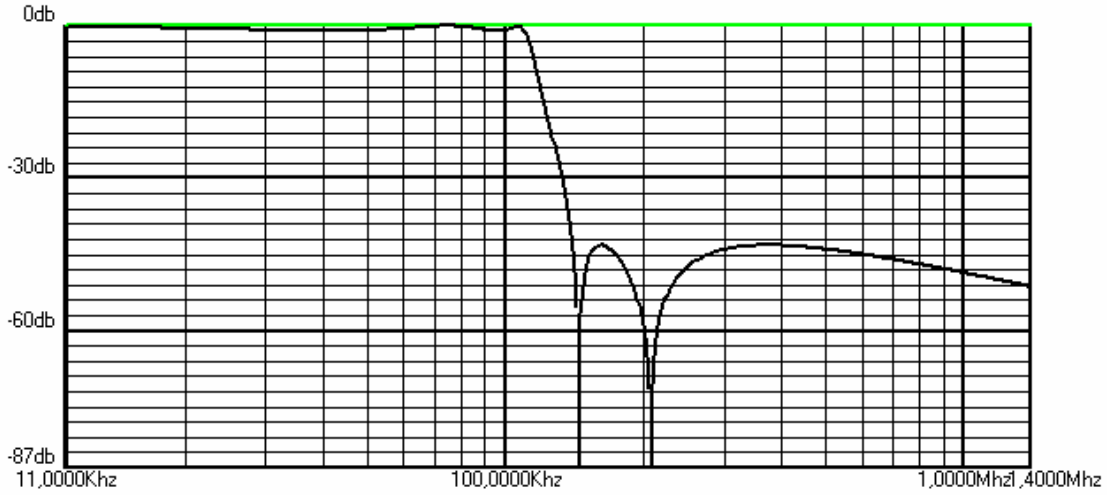


Figura 47: Respuesta del filtro de banda base ideal

Estableciendo las impedancias de fuente y carga en 100Ω y 1.5KΩ, respectivamente se obtiene el circuito que implementa el filtro. De las dos posibles implementaciones se ha escogido la que presenta un menor número de inductancias, por ser los componentes más caros, siendo éste el circuito resultante:

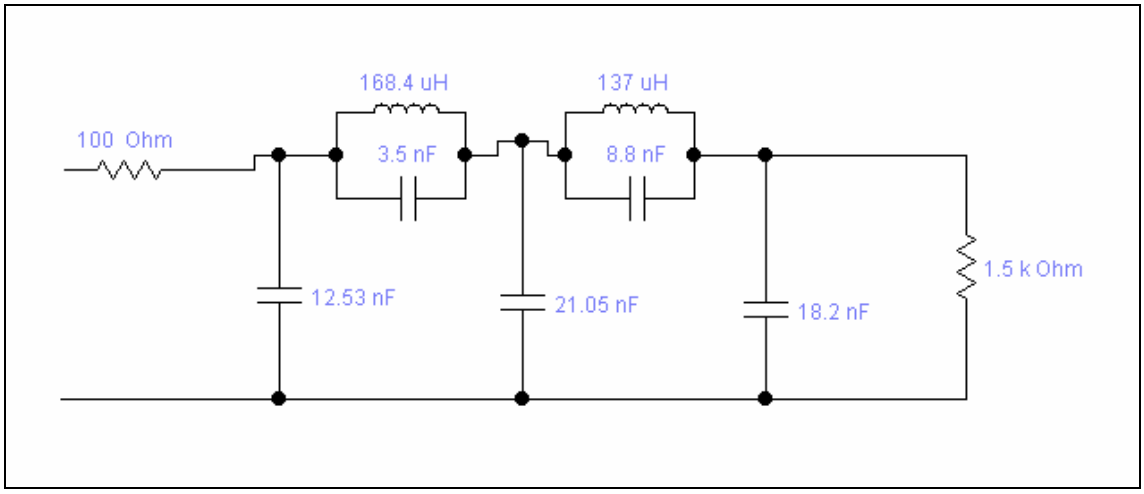


Figura 48: Filtro de banda base ideal

De nuevo se escogen los valores reales de condensadores e inductancias que menos modifiquen la respuesta global del filtro, quedando así el circuito finalmente implementado:

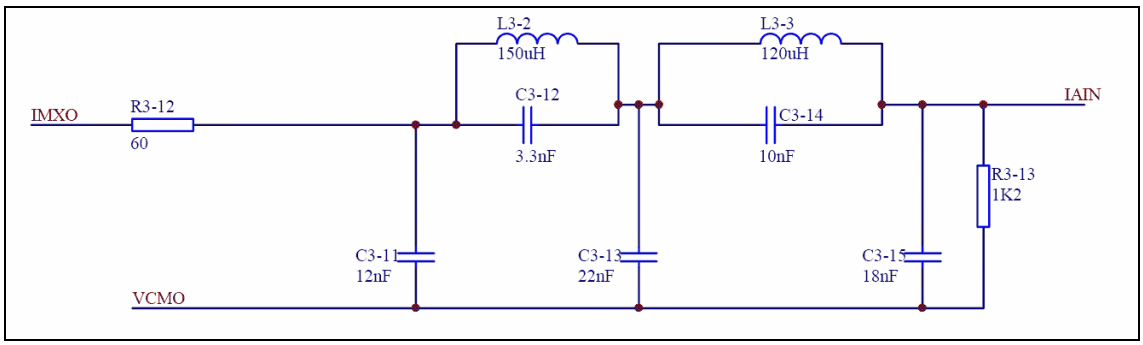


Figura 49: Implementación del filtro *antialiasing* elíptico de quinto orden.

Este es el filtro del canal I, mientras que el del canal Q es idéntico pero conectado entre `QMXO` y `QAIN` en lugar de entre `IMXO` e `IAIN`. En el esquema se puede ver cómo la referencia de la señal no es masa, sino la fuente de tensión de baja impedancia VCMO, que por esto está convenientemente desacoplada a masa mediante condensadores de diferentes valores.

La respuesta en frecuencia del filtro con los valores ideales de los componentes se puede ver en la siguiente gráfica (en línea discontinua), junto con la respuesta con los valores comerciales de los componentes (en línea continua).

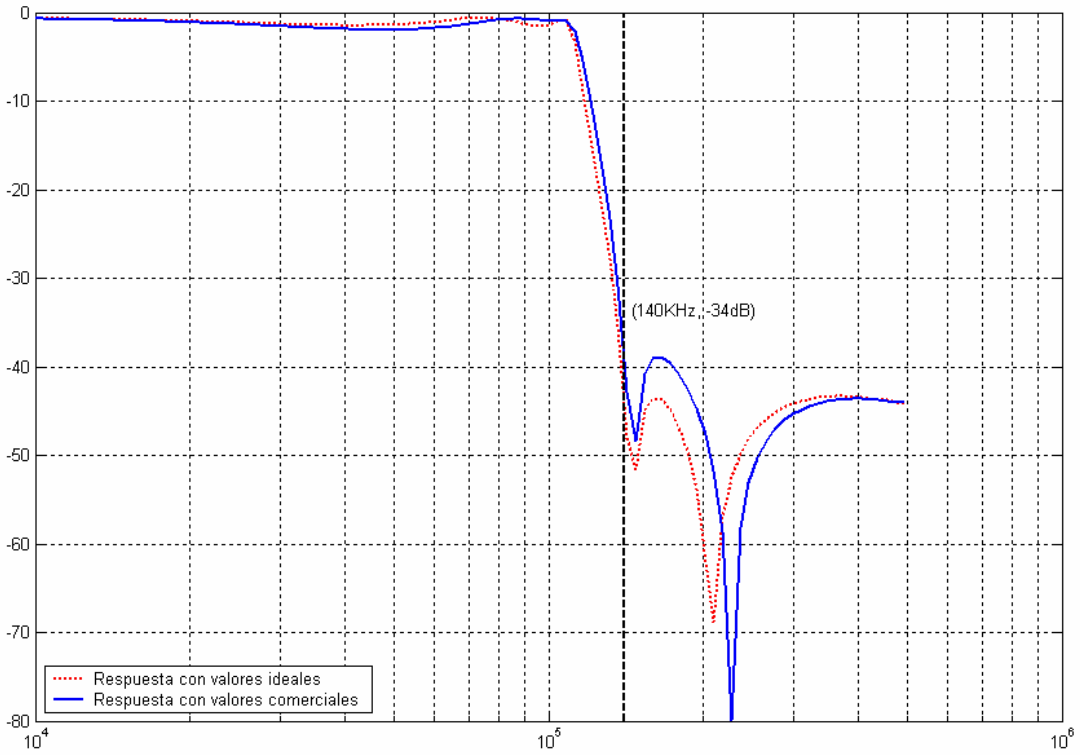


Figura 50: Respuesta en frecuencia del filtro *antialiasing*.

En estas simulaciones las inductancias se han considerado ideales. Como se verá en el apartado 11.6 dedicado a las medidas sobre el filtro real, la no idealidad de las inductancias afectará bastante a la respuesta de los filtros.

4.7. Amplificadores de banda base

Los amplificadores de banda base para las señales I y Q tienen una doble finalidad: (i) Convertir las señales diferenciales en señales unipolares, ya que este es el tipo de señal que manejan los conversores analógico-digitales, y (ii) proporcionar una etapa de ganancia adicional en la cadena de la señal. Esta ganancia fija se ha establecido en 20.8dB y se usa para adaptar el rango de la señal al del conversor, como se mostrará gráficamente en el apartado 5.5.

Para estos amplificadores se ha elegido el integrado AD8129. Este amplificador, cuyo esquema interno se muestra en la siguiente figura, usa una realimentación activa, incluyendo dos amplificadores operacionales, proporcionando dos pares de entradas diferenciales en lugar de un solo par, lo que según las hojas de características, mejoran el CRM y el factor de ruido, entre otros factores. Entre las distintas configuraciones en que se puede usar (amplificador de ganancia unidad, de ganancia negativa, con y sin niveles de *offset*, etc.) se ha usado la mostrada en la siguiente figura:

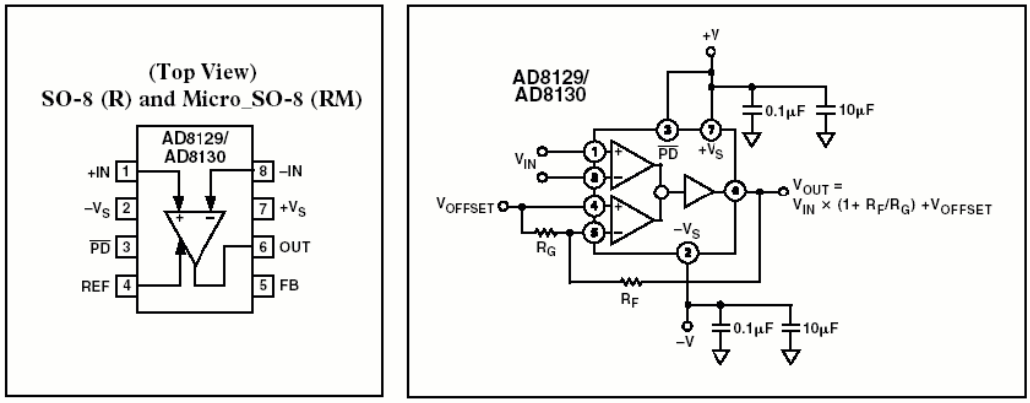


Figura 51: Amplificador diferencial y la configuración usada.

Esta configuración proporciona una ganancia a la señal diferencial de entrada, ajustable mediante dos resistencias, mientras que el voltaje de *offset* deseado a la salida se puede establecer desde una terminal de entrada apareciendo a la salida con ganancia unidad, como es lo deseado. Este voltaje medio de salida se toma del mismo integrado demodulador, que como se vio tenía un valor de 1.0V. Ahora, como la ganancia de potencia deseada es de 20.8dB, ganancia lineal de voltaje de 11, los valores de las resistencias deben cumplir:

$$G = 1 + \frac{R_F}{R_G} = 20.8dB = 11 \rightarrow R_F = 10\,R_G$$

Dos valores que cumplen esto son 10KΩ y 1KΩ, con lo que el amplificador para el canal Q queda como se ve en la siguiente figura:

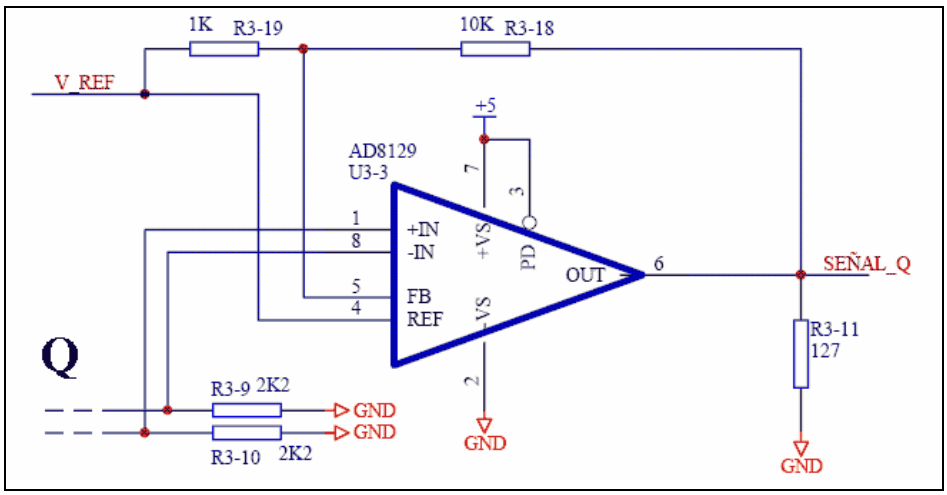


Figura 52: Amplificador de banda base.

El amplificador correspondiente al canal I es idéntico a este. Hay que señalar que las salidas I y Q del demodulador requieren una impedancia de carga típica de 2KΩ, por lo que se han añadido las resistencias de carga en cada línea diferencial de 2.2KΩ. Las impedancias de entrada del amplificador no se tienen en cuenta ya que son muy altas (1MΩ || 3pF) y en paralelo no afectan a estas resistencias. Como se puede ver se ha utilizado la señal V_REF, generada por el integrado demodulador, para centrar la salida alrededor de 1.0V.

4.8. Sintetizador de frecuencias (PLL)

Para generar un oscilador local estable, se ha usado un sintetizador de frecuencias basado en un bucle PLL (*Phase Locked Loop*), compuesto por:

- Un oscilador VCO, cuyo diseño se describe en el apartado 6.14.
- Un circuito integrado sintetizador de frecuencias, que incluye toda la lógica digital necesaria y el control del VCO. La elección y configuración de este integrado se trata en este apartado.
- El filtro del bucle, diseñado también en este apartado.

Las ventajas de usar un PLL para generar el oscilador local son:

- Estabilidad en frecuencia: Permite sintonizar sin error el canal deseado, compensando automáticamente por realimentación los desvíos del oscilador producidos por la temperatura, transitorios en la alimentación o cualquier otra causa.
- Estabilidad en fase. Un PLL reduce el ruido de fase de un VCO. Este ruido se traduciría en una mayor incertidumbre de la fase de la señal demodulada, y por lo tanto, en mayor ruido (SNR) para modulaciones de fase analógicas y mayor probabilidad de error (BER) en modulaciones digitales.

El proceso de elección del integrado PLL y del tipo de VCO ha sido uno de los más complejos de este proyecto, llegándose a diseñar, construir y probar hasta tres versiones de tipos muy distintos. Además se consideró y descartó previamente el uso de un sistema DDS (*Direct Digital Synthesis*) como se explicó en el apartado 5. Las tres alternativas que se llegaron a construir son las siguientes:

- El uso de un chip PLL dedicado y un VCO integrado, el TLC2934 de *Texas Instruments*. Este integrado dispone de un oscilador VCO basado en un oscilador basado en un circuito RC en lugar de uno LC. La frecuencia central de funcionamiento se establece mediante una resistencia externa. Se diseñó y construyó una placa para comprobar el rango de frecuencias de que es capaz este integrado, y se vio que para variaciones de la tensión de control entre 0.5V y 4.5V, la frecuencia variaba 18MHz. Debido a este pequeño margen de variación, se desechó usar este integrado como oscilador.

- El uso de un chip PLL dedicado y un VCO basado en un oscilador a transistor BJT. Para el VCO se optó por un diodo *varicap* hiperabrupto, el ZC933 y el transistor de microondas BFR92. El diseño era un oscilador de base común en configuración *Colpitts*, como se ve en el esquema de abajo.

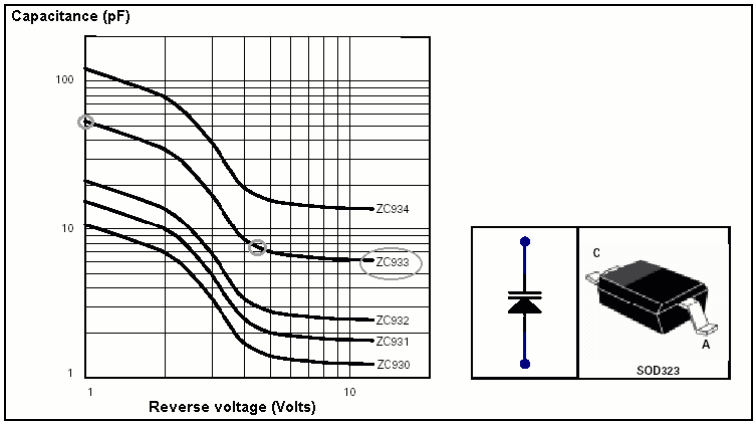


Figura 53: Características del *varicap* elegido en el VCO a BJT.

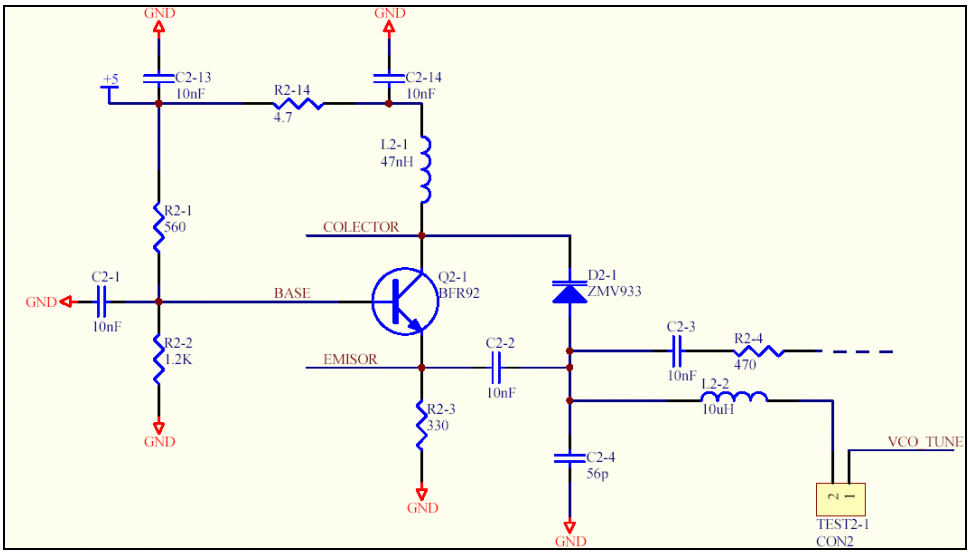


Figura 54: Implementación del VCO a BJT.

Aunque este diseño funcionó correctamente e incluso el oscilador abarcaba el rango requerido de frecuencias, se encontró la dificultad de conseguir una potencia de salida suficiente y aproximadamente estable en todo el rango de salida. Como muestra del funcionamiento que se consiguió con este oscilador, las siguientes gráficas muestran, respectivamente, la frecuencia del oscilador según la tensión de control y la potencia de salida según esta frecuencia. Las gráficas se muestran para dos valores distintos de la resistencia de emisor, y por tanto, de corriente de polarización. Estas medidas se tomaron usando el analizador de espectros a través de una sonda resistiva en serie de 1KΩ, que posteriormente se tuvo en cuenta para

corregir los datos de potencia. En el siguiente punto se expone la opción que finalmente se ha escogido y las razones de rechazar este diseño basado en un BJT discreto.

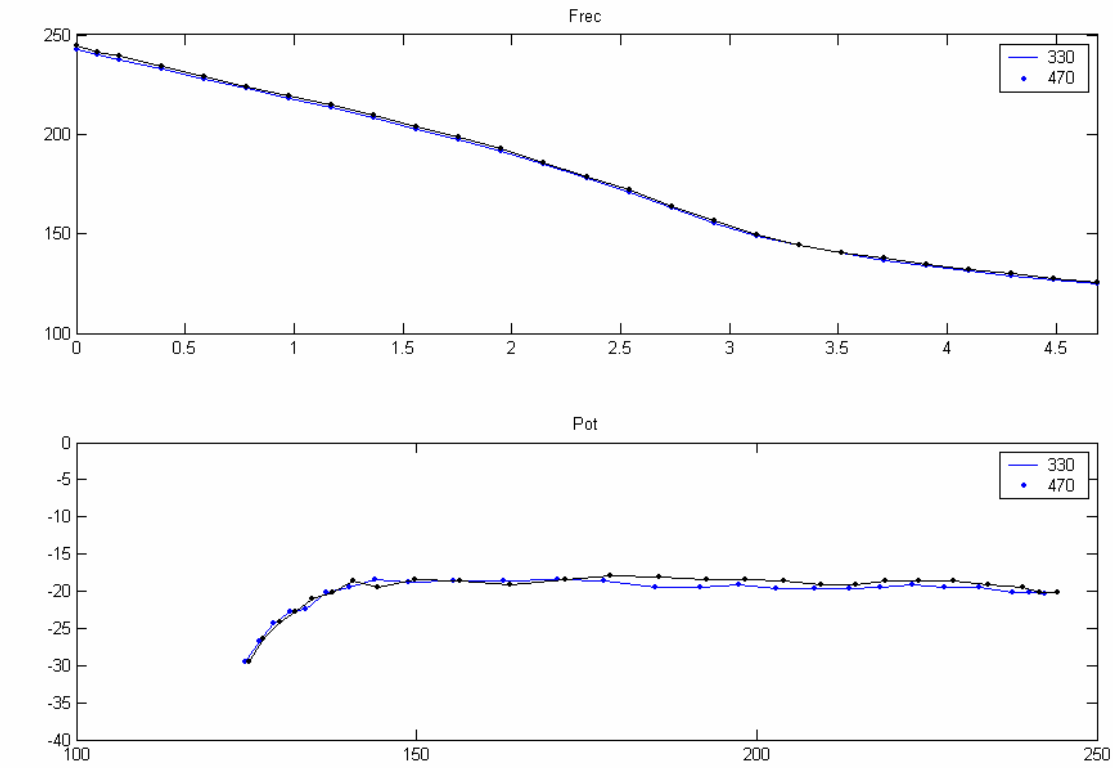


Figura 55: Resultados de las medidas al VCO a BJT.

- Como tercera opción se planteó el uso de un integrado que contenga los dos, la lógica asociada al PLL y el oscilador VCO analógico basado en BJTs. Una familia de este tipo de integrados fue introducida en el mercado por *Analog Devices Incorporate* en octubre de 2004, durante la realización de este proyecto. Las razones que hicieron de esta opción mucho más ventajosa frente al VCO basado en un BJT discreto y que han llevado a tomar finalmente esta opción son:
  - El diseño del VCO se reduce al cálculo de unos pocos componentes externos.
  - El espacio ocupado por el integrado para el PLL y el VCO es de tan solo 4 x 4 mm. La placa diseñada para el oscilador en este proyecto incluye también los componentes pasivos necesarios, generación de la alimentación a 3.3V para este chip y filtrado de alimentación, y ocupa tan sólo 14cm^2, quedando una placa ordenada y poco ocupada.

- Otra ventaja crítica es el control digital que se tiene de la potencia de salida del VCO que se mantiene bastante constante en todo el rango de frecuencias.
- La última ventaja viene de la sensibilidad del VCO. Mediante una técnica que se expone en el apartado 4.9, el VCO integrado consigue una sensibilidad (factor K_V) mucho más bajo que en el caso del BJT discreto, lo que se traduce en un menor ruido de fase final.

La familia de integrados que constan de VCO y PLL en el mismo chip monolítico (mezclando procesos CMOS y BJT en el mismo chip) consta de varias versiones, dando lugar a la familia ADF4360-X, donde X varía entre 1 y 8. Todos ellos, menos el último, llevan el VCO completamente integrado, incluyendo las inductancias y el *varicap*. Los rangos de frecuencias de funcionamiento de estos coinciden con estándares de las diferentes generaciones de telefonía móvil en distintas partes del mundo. Es el último modelo, el ADF4360–8 el que se ha usado en este proyecto ya que la frecuencia del VCO se puede configurar mediante inductancias externas.

Una vez decidido el integrado que se usará para implementar el VCO y el PLL, los primeros parámetros que hay que decidir son: (i) el rango de frecuencias de funcionamiento, y (ii) el espaciamiento entre canales. Los objetivos del proyecto señalan un rango de recepción de, al menos, el intervalo 88MHz a 108MHz. Sin embargo, como se vio en al apartado 6.5, el demodulador en cuadratura requiere un oscilador local al doble de la frecuencia deseada, con lo que el rango de diseño se convierte en el intervalo 176-216MHz. Pero este margen debería ser el mínimo de funcionamiento asegurado para el receptor, por lo que el intervalo objetivo del diseño se ha fijado ligeramente más amplio, ya que en el diseño de osciladores VCO las frecuencias obtenidas realmente pueden tener cierto margen de inexactitud, debido a capacidades parásitas, tolerancias de componentes, etc. Por todo esto, se establecen estos límites para el diseño tanto del PLL como del VCO:

$$f_{MIN} = 170\text{MHz}$$

$$f_{MAX} = 222\text{MHz}$$

Hay que resaltar que estos márgenes permiten recibir señales entre 85Mhz y 111Mhz, margen algo más amplio del mínimo fijado en los objetivos del proyecto. En cuanto al espaciamiento entre canales, se ha estimado que una resolución de 50KHz en la

selección del canal es razonable para los objetivos del proyecto. De hecho es el espaciamiento que se ha observado tienen algunos receptores de FM comercial basados en PLL en equipos Hi-Fi. Como la señal sintetizada tiene una frecuencia doble del canal seleccionado, se tiene que el espaciamiento queda:

$$\Delta f = 100KHz$$

Para configurar el PLL integrado hay que analizar su funcionamiento interno:

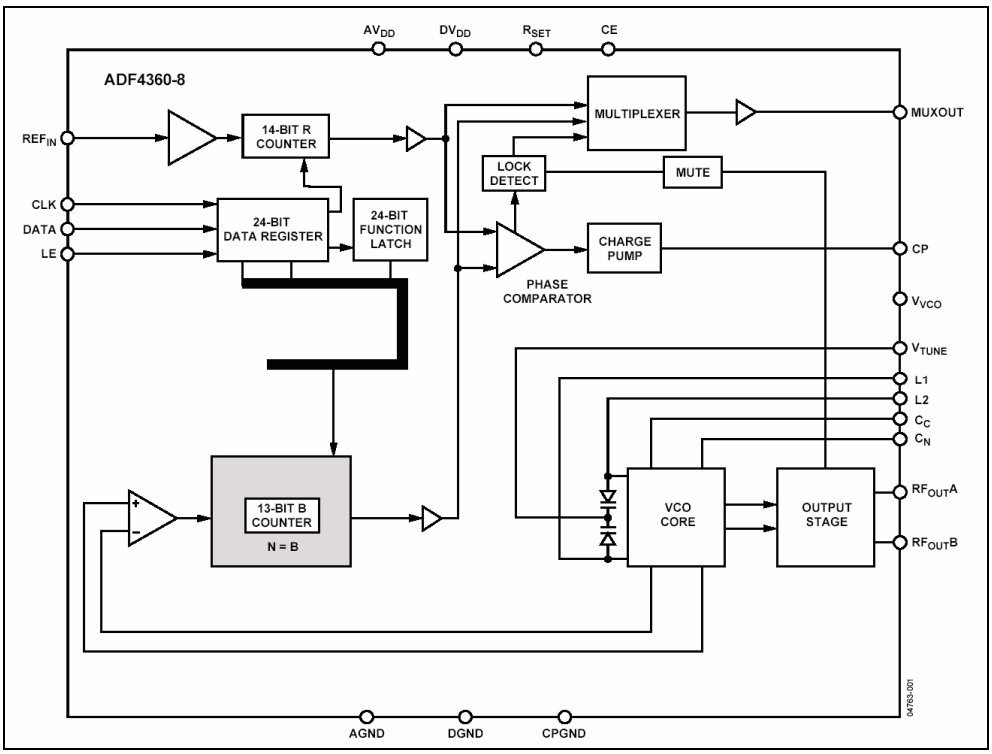


Figura 56: Esquema interno del sintetizador de frecuencias.

Es un sintetizador que no dispone de *prescaler*. La señal de referencia es dividida directamente según el contenido del registro de 14 bits "R". Además, mientras que en otros integrados con *prescaler* de doble módulo el valor "N" por el que se divide la señal del VCO se compone por "N=B·P+A", en este caso solamente es necesario un registro de 13 bits, el "B" y directamente es "N=B". La salida analógica de control del VCO es del tipo *charge-pump*, por lo que todos los problemas asociados al margen de enganche y de seguimiento en PLLs más antiguos no se dan, ya que ambos márgenes son virtualmente infinitos, limitados solo por el VCO (Ver [6]). La configuración de esta salida, los registros divisores y de otras opciones más se realizan mediante un bus digital SPI, que se detalla más abajo. Este integrado esta pensado para funcionar únicamente a 3.3V, mientras que en el receptor el voltaje de alimentación global usado es de 5V, por lo que hay que afrontar el problema de generar esta tensión de

alimentación y adaptar los niveles de las señales lógicas. Como el diseño completo del sintetizador incluye varios aspectos, estos se exponen de forma separada por claridad.

4.8.1. Valores para los registros `R` y `B`

La configuración de la frecuencia de salida deseada se consigue mediante la selección del contenido de dos registros (B y R, de 13 y 14 bits respectivamente) y la frecuencia de referencia (f_{REF}). Así, la frecuencia de salida viene dada simplemente por:

$$f_{VCO} = B \cdot \Delta f$$

$$\Delta f = \frac{f_{REF}}{R}$$

A partir de las especificaciones de frecuencia máxima y mínima, junto al espacio entre canales deseado, se puede calcular el intervalo de valores para el divisor programable B:

$$B \rightarrow \begin{cases} B_{MAX} = \dfrac{f_{MAX}}{\Delta f} = \dfrac{222\,MHz}{100\,KHz} = 2220 \\ \\ B_{MIN} = \dfrac{f_{MIN}}{\Delta f} = \dfrac{170\,MHz}{100\,KHz} = 1700 \end{cases}$$

El mínimo escalón de variación de frecuencia, y por tanto el espaciamiento entre canales, viene fijado por Δf. Para evitar incluir otro cristal en el diseño, y como es un valor apropiado, se aprovechará la generación del reloj a 2.5MHz (apartado 6.11) como frecuencia de referencia. Como se quiere obtener un espacio entre canales de 100KHz, se tiene:

$$R = \frac{f_{REF}}{\Delta f} = \frac{2.5MHz}{100KHz} = 25$$

4.8.2. Corriente del *charge-pump*

A diferencia de los antiguos PLLs en que el detector de fase consistía en una salida de voltaje, desde hace unos años la técnica de *charge-pump* domina en todos los PLL comerciales de RF. Esta salida consiste en pulsos de corriente (no de voltaje), que al pasar por el filtro del lazo se convertirán en el voltaje de control del VCO. Estos pulsos pueden ser positivos o negativos y tienen una amplitud fija. Por lo tanto, cada pulso consigue aumentar o disminuir el voltaje en un cierto escalón. El valor de esta corriente es configurable en este integrado (y en la mayoría de PLL integrados) por dos medios:

- Una resistencia externa en un pin, en este caso "R_{set}", hacia masa. Se tiene que la corriente máxima del *charge-pump* es:

$$I_{CPmax} = \frac{11.75}{R_{set}}$$

El valor recomendado por el fabricante, que ha sido el usado, es el de R_{set}=4.7KΩ, dando una I_{CPmax} de 2.5mA

- Mediante unos bits en el registro de configuración, que ofrecen un rango de ocho valores entre cero y el valor máximo. Como se recomienda usar la mayor corriente posible para limitar el ruido de fase ([6]), se ha usado el valor máximo de 2.5mA

4.8.3. Filtro del lazo

Este filtro paso bajo es un elemento crítico en cuanto a que representa un compromiso entre varios parámetros que caracterizaran al PLL y a la señal sintetizada, como:

- Tiempo de establecimiento: Cuando se cambia el valor del divisor "B", se generará una nueva frecuencia, pero esta tarda un tiempo en estabilizarse. Cuanto menos ancho de banda tenga el filtro de lazo, más tardará en estabilizarse la nueva frecuencia.
- Ruido de fase: Según [6] el ruido de fase final del oscilador al cerrar el bucle, se divide aproximadamente en dos partes. La primera, centrada alrededor de la frecuencia de oscilación, donde el ruido de fase depende del ruido del VCO. La otra zona son las frecuencias a más distancia de la frecuencia de oscilación que el ancho de banda del filtro de lazo. En esta zona el ruido de fase depende del tipo de integrado PLL. Como este suele ser menor que el de los VCO, conviene un ancho de banda pequeño para minimizar el ruido de fase.
- Filtrado de espúreos. De este filtro depende la atenuación de los espúreos, al menos a los que dependen de la señal de referencia. Para atenuar suficientemente los espúreos, el filtro debe atenuar lo máximo posible en la frecuencia de referencia, en este caso 100KHz. Otros espúreos dependen del grado de *matching* conseguido en la fuente y el sumidero de corriente del *charge-pump*.

Para el diseño de este filtro se ha utilizado la herramienta automatizada "*ADI SimPLL v2.5*", que el fabricante *Analog Devices* distribuye gratuitamente en su web. En el diseño del sistema PLL, una de las primeras decisiones que hay que hacer es el tipo de

filtro pasivo a usar. En este caso se usa un filtro de tercer orden, como el que se ve en la figura:

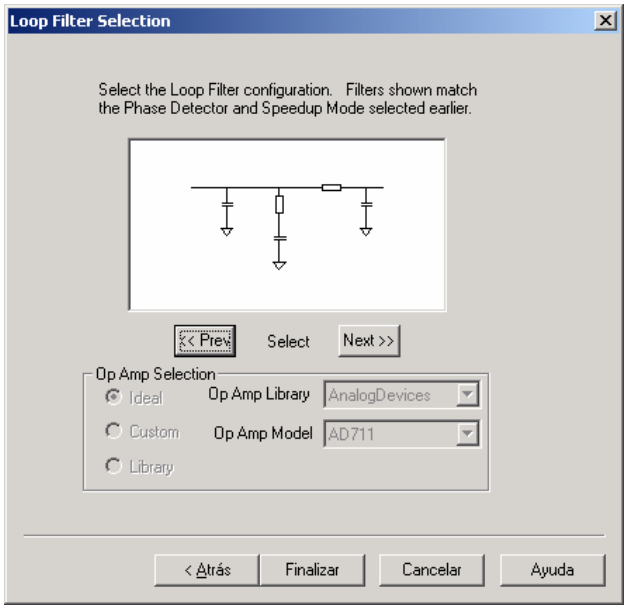


Figura 57: Selección del tipo de filtro de lazo.

Este tipo de filtro ofrece cuatro grados de libertad, que se corresponden con estos cuatro parámetros de diseño:

- Ancho de banda del filtro: El valor recomendado es una década menos que la frecuencia de referencia, que es 100KHz, así que el ancho de banda queda 10KHz.
- Margen de fase del filtro. Se ha usado el valor típico y recomendado de 45º
- La posición del último polo: Se ha aceptado el valor recomendado de la frecuencia de referencia, esto es, de 100KHz.
- Escalado de los valores de los componentes: En los valores de los componentes queda un grado de libertad, que en esta aplicación se traduce en la libertad de dar el valor de una resistencia. Ajustándola para que los demás componentes tengan valores cercanos a los disponibles comercialmente, se establece el valor de R2 en 100KΩ.

En la siguiente figura se ven todos estos parámetros y los valores de los componentes calculados por el programa, junto al esquema para identificar la posición de cada uno de estos componentes:

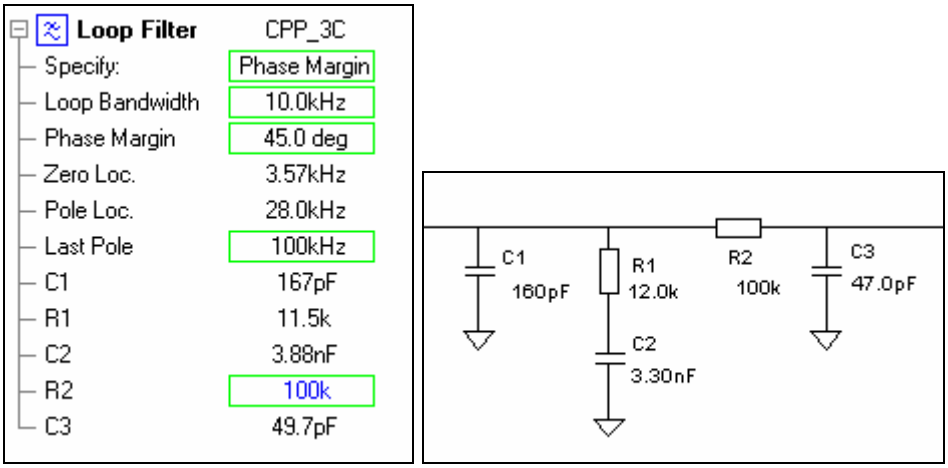


Figura 58: Parámetros de diseño del filtro de lazo y valores comerciales usados finalmente.

Al modificar ligeramente los valores de los componentes para ajustarlos a valores reales, se obtienen los parámetros finales del filtro del lazo:

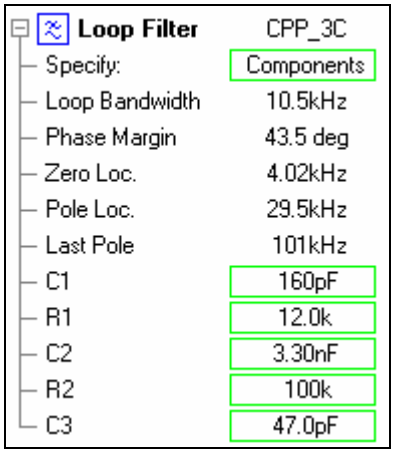


Figura 59: Parámetros finales del filtro de lazo.

Se puede ver como ahora el ancho de banda del lazo es de 10.5KHz y el margen de fase de 43.5º, lo que sólo representa una pequeña variación con los valores de diseño.

4.8.4. Impedancia y potencias de entrada

Al estar el VCO dentro del mismo integrado, en este caso no hay que preocuparse ni por adaptar las impedancias del VCO y del PLL ni de sensibilidad de este, ya que está asegurado que internamente los dos bloques están correctamente conectados.

4.8.5. Comunicación con el microcontrolador

Toda la configuración del PLL y del VCO se realiza a través de la interfaz SPI (*Serial Peripheral Interface*) del integrado. Este bus serie síncrono dispone de una línea de datos, una de reloj y una de habilitación, como se ve en el siguiente cronograma:

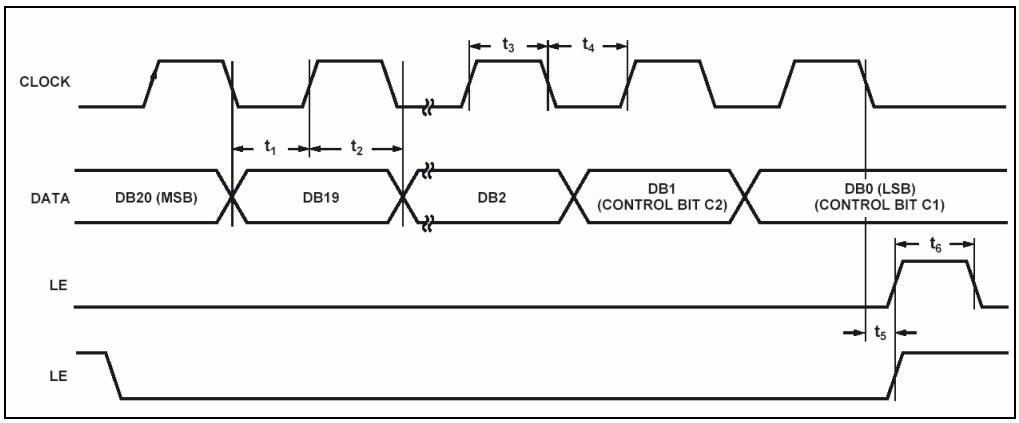


Figura 60: Cronograma mostrando el funcionamiento del bus SPI del PLL

En cada flanco de subida del reloj, el *bit* presente en la línea de datos entra a un registro de desplazamiento de 24 bits dentro del PLL. Para indicar que se ha completado el envío y para indicar al PLL que los datos iban dirigidos a él, el microcontrolador mandará un pulso a la entrada "`LE`" (*Latch enable*).

En este momento, el contenido del registro de desplazamiento se copia a uno de tres posibles registros de configuración del integrado, según indiquen los dos bits más bajos:

Bits	Registro	Contenido
00	*Control latch*	Configuraciones varias
01	*R Counter latch*	Valor del registro R y otras opciones
10	*N Counter latch*	Valor del registro N y otras opciones

A continuación se detallan brevemente las partes más importantes de estos registros y que se han usado en este proyecto, ya que la descripción detallada se hace en el *datasheet*. Debido a lo engorroso de codificar las opciones en este conjunto de bits, en el software del PC se provee un método para configurar el PLL a partir de una estructura de datos de forma más cómoda.

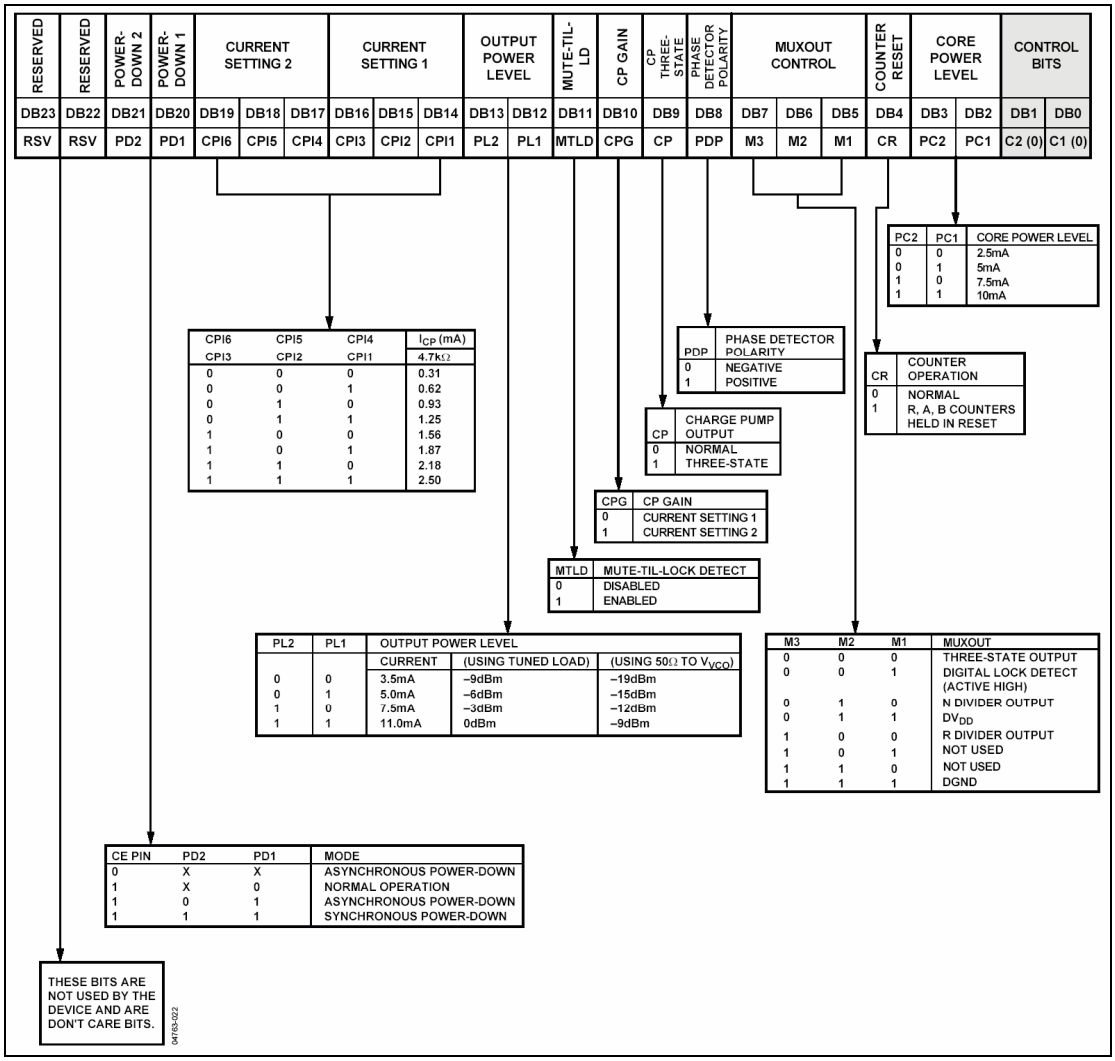


Figura 61: El registro de control

- El registro de control: En este registro se encuentran entre otras estas opciones:
    - Conexión del pin MUXOUT: Se puede elegir entre un valor lógico alto o bajo, el detector de enganche digital y el divisor de la señal de referencia. Normalmente se conectará la salida del detector de enganche, aunque los otros modos tienen su utilidad en pruebas como se ve en el apartado 13.
    - La configuración de corriente del *charge-pump*: Como ya se ha dicho, se seleccionará de forma permanente el máximo valor de 2.5mA
    - La potencia de salida del VCO: Se escogerá el segundo valor, correspondiente teóricamente a –3dBm con la impedancia de salida adaptada, como es el caso.

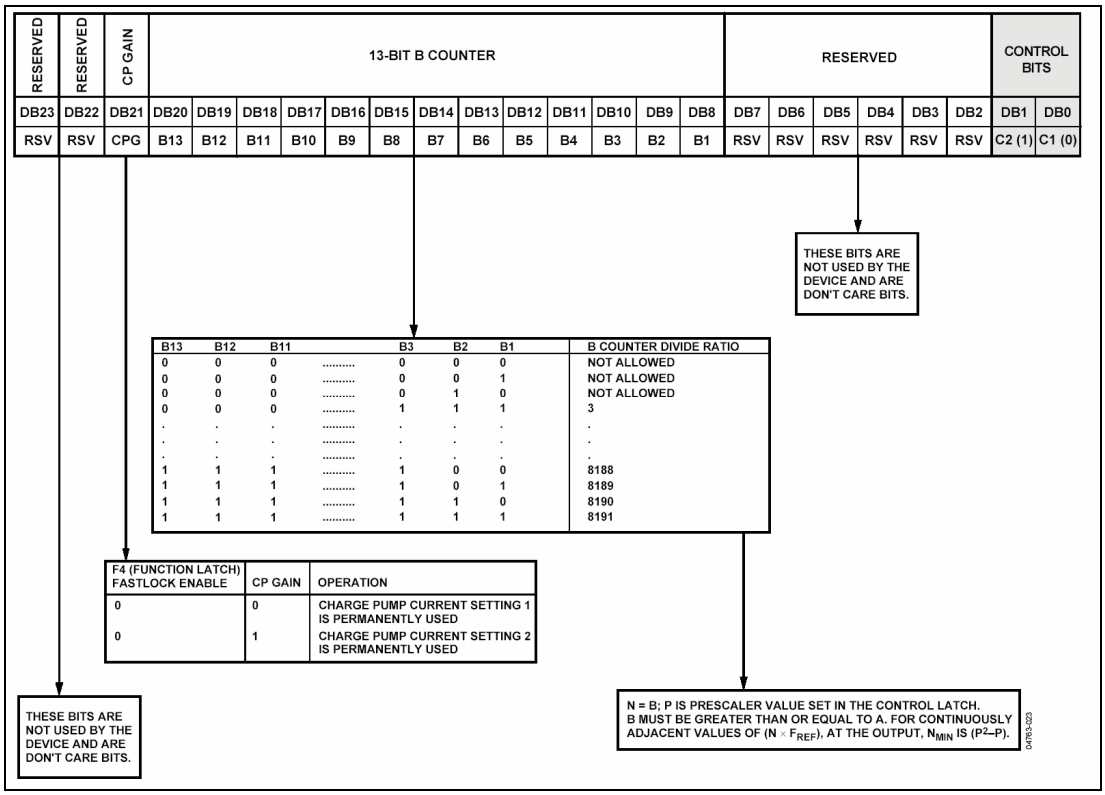


Figura 62: El registro del divisor B

- El registro del divisor "B": En este registro se encuentran los 14 bits de este divisor y un *bit* para seleccionar entre una de las dos configuraciones de corriente posibles. En este proyecto siempre se utilizará la primera, y por lo tanto este *bit* estará a cero.

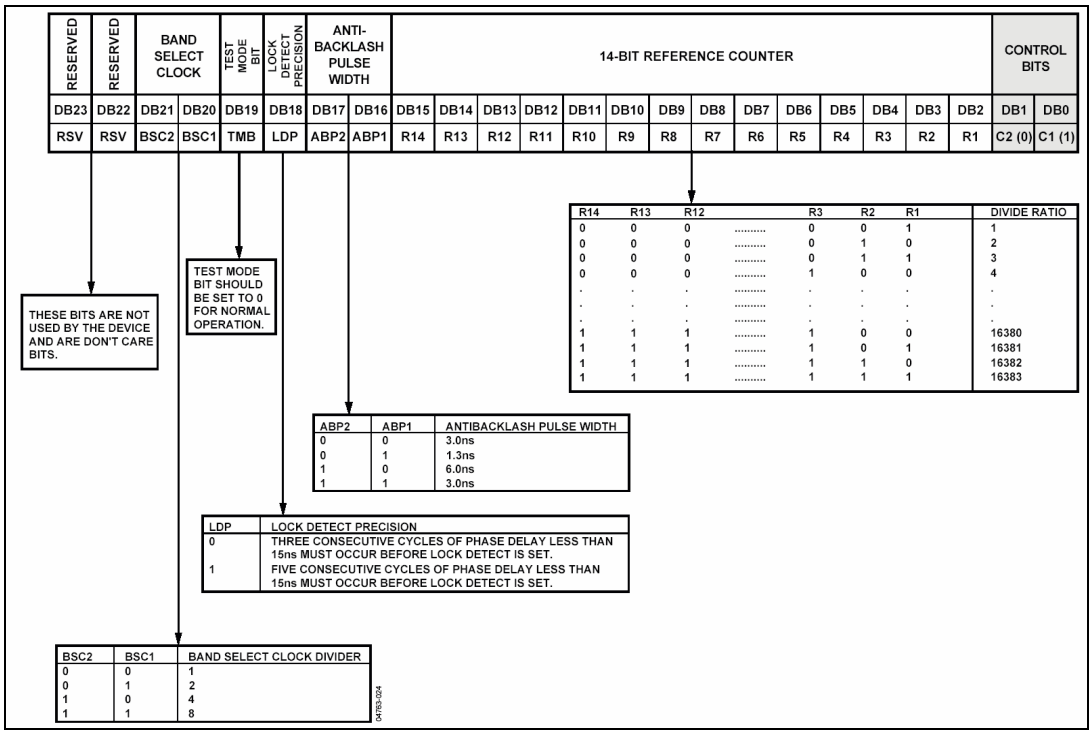


Figura 63: El registro del divisor R

- El registro del divisor "R": Además del valor de 14 bits de este divisor, en este registro están los bits de configuración del detector de enganche digital (cuya precisión se ha establecido en tres pulsos) y un divisor que hace las veces de *prescaler* en este chip, aunque en este proyecto no se ha usado y se ha dejado en "divisor por uno".

4.8.6. Manejo de la salida multiplexada

La salida MUXOUT del integrado puede tener distintas señales según se programe el chip desde el microcontrolador. Las distintas salidas entre la que se pueden elegir son: Un nivel lógico alto, un nivel bajo, la salida del divisor R, la salida del divisor N y un detector digital de enganche del PLL. Para que el microcontrolador y el software del PC puedan tener acceso a esta señal, ésta se ha conectado a una entrada digital del microcontrolador. Aunque los niveles lógicos de salida en `MUXOUT` son CMOS a 3.3V, estos son compatibles con los niveles lógicos de entrada del microcontrolador que son CMOS a 5V. Esto es así porque un nivel bajo de salida es inferior a 0.4V y el nivel alto es superior a 2.9V. Como las entradas CMOS del microcontrolador tienen un umbral de 2.5V, los niveles lógicos del PLL son recibidos sin problema. En el caso contrario no ocurre así, como se ve a continuación.

4.8.7. Alimentación y adaptación de niveles lógicos

El integrado ADF4360-8 está diseñado para funcionar solamente a un voltaje en el rango de 3.0V a 3.6V. Como la alimentación del receptor se toma del PC a través del conector USB y ésta es de 5.0V, en la placa del oscilador se ha añadido un regulador lineal, el `ADP3338`. Este integrado es un regulador lineal que proporciona hasta 1A de corriente, aunque en el caso del PLL sólo se consumirán unos 30mA. En el siguiente esquema se muestra la conexión de este integrado, dónde también se reflejan los condensadores de desacoplo de alimentación. Estos condensadores realmente se han colocado físicamente cada uno junto a una entrada de alimentación del PLL, de forma que los ruidos en la línea de alimentación se eliminen lo antes posible y se presente un camino de AC de la mínima inductancia posible hacia el plano de masa, lo que es esencial en circuitos analógicos como el oscilador integrado VCO.

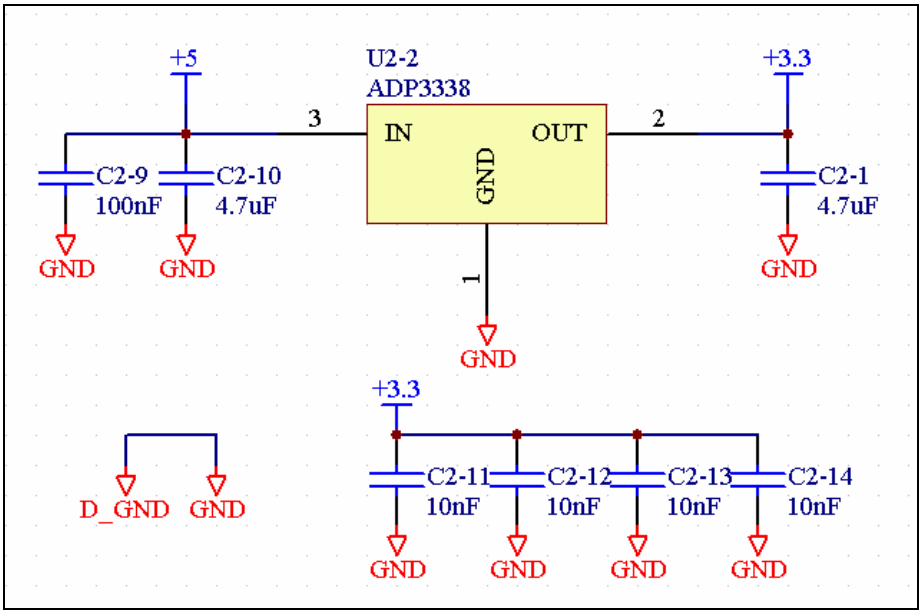


Figura 64: Regulación y filtrado de alimentación del PLL

También es necesario adaptar las señales digitales que vienen del microcontrolador hacia el PLL. Como señales lógicas CMOS, sus niveles serán de cerca de 0V y cerca de 5V para los niveles bajo y alto, respectivamente. Consultando las especificaciones del PLL en el apartado de "*Absolute Maximum Ratings*" se ve que las entradas lógicas tienen un límite de 3.6V, valor que de superarse podría inutilizar el dispositivo. Por lo tanto el nivel lógico alto debe escalarse para adaptarlo a los niveles de este integrado.

Buscando entre los principales fabricantes electrónicos se ha encontrado que existe una amplia variedad de adaptadores de niveles digitales, los que de entrada se pueden dividir en unidireccionales o bidireccionales. Los primeros solo permiten convertir voltajes de un nivel de voltaje a otro, mientras que los segundos permiten la conversión en ambos sentidos, sin necesidad de un pin de control de dirección. Además de la direccionabilidad de los conversores, otro factor en la elección es la cantidad de conversores por integrado. Se han encontrado conversores de una sola señal, de dos de ocho y de otras cantidades. Como hay que adaptar cinco señales (`/POWER_DOWN`, `PLL_SYNC`, `3W_DOUT`, `3W_SCLK` y `PLL_REF`) entrantes al PLL desde el microcontrolador, se ha optado por un conversor de ocho señales, el MAX3002. Este es un conversor bidireccional, aunque esta habilidad no se utiliza en este proyecto. Su esquema interno se muestra en la siguiente figura, donde queda claro cómo se consigue la bidireccionalidad:

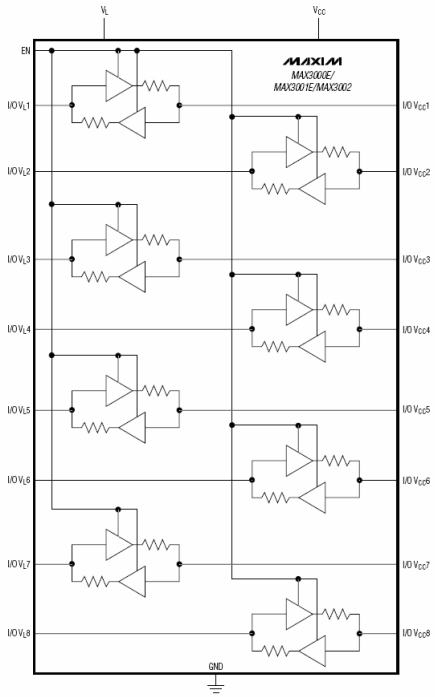


Figura 65: Adaptador bidireccional

Su conexión con las señales del PLL y las del microcontrolador se muestran abajo. Las señales del microcontrolador entran a esta placa por el conector de conexión vertical, como en las demás placas.

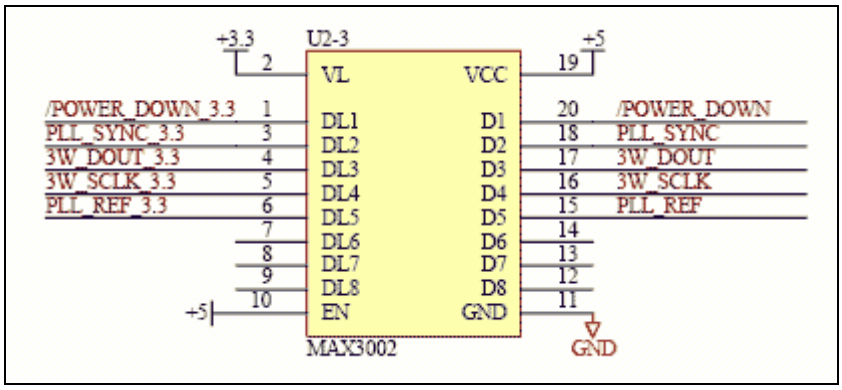


Figura 66: Adaptador de niveles lógicos

En cuanto a las especificaciones temporales del conversor de tensión, la única señal que podría provocar problemas es la señal de referencia del PLL, que es una señal cuadrada de 2.5MHz, y que debe llegar al PLL con un *slew rate* mínimo de 21V/us según las especificaciones del integrado. En cuanto al retardo no existe problema pues el `MAX3002` funciona hasta 20Mbps. En un diseño anterior se encontró un problema con el *slew rate*, pero como se muestra en el apartado de pruebas en este diseño final se cumple sobradamente permitiendo un funcionamiento totalmente correcto.

4.9. Oscilador variable (VCO)

En cuanto al diseño del VCO integrado en el mismo chip del PLL, hay que considerar estos aspectos:

4.9.1. Rango de frecuencias de funcionamiento

El VCO integrado dispone internamente del diodo *varicap*, siendo los únicos elementos externos necesarios dos inductancias, cuyos valores deben ser idénticos, y que fijan la frecuencia central de funcionamiento. En la siguiente gráfica se muestra una estimación del rango de frecuencias del VCO según el valor de las inductancias.

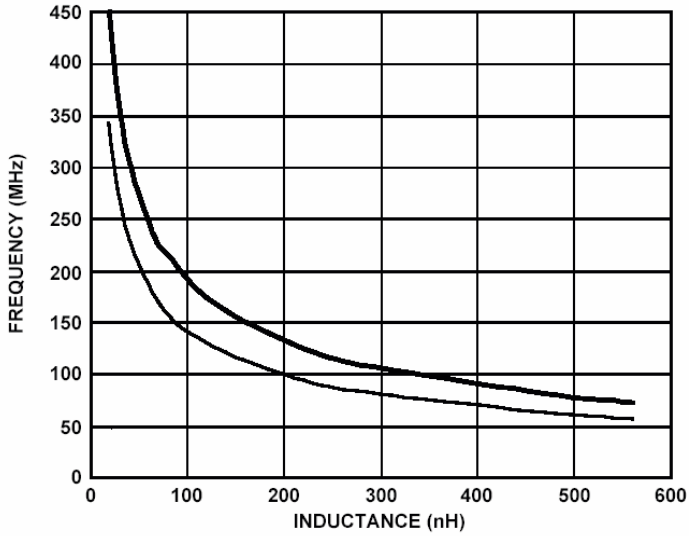


Figura 67: Rango de frecuencias según el valor de la inductancia

El fabricante ofrece la siguiente ecuación para calcular el valor de inductancia externa según la frecuencia central deseada, donde ya va incluido el valor de la inductancia parásita interna:

$$F_0 = \frac{1}{2\pi\sqrt{9.3\text{pF}\left(0.9\text{nH} + L_{EXT}\right)}}$$

Dado que el rango de diseño, que se estableció en el apartado anterior, es de 170-222MHz, el valor central es 196MHz. Despejando de la ecuación el valor de la inductancia, y sustituyendo el valor de la frecuencia central se tiene:

$$L_{EXT} = \frac{1}{\left(2\pi \cdot f_0\right)^2 \cdot 9.3\text{pF}} - 0.9\text{nH} = 69.99\text{nH}$$

El valor comercialmente disponible usado en el circuito es el de 68nH. Estas dos inductancias se conectan entre masa y las entradas del VCO, y se han colocado de forma perpendicular una con la otra, para minimizar el acoplamiento mutuo. Además, hay que colocar una resistencia de 470Ω en paralelo con cada inductancia, como se muestra en el esquema, al final de este apartado.

4.9.2. Sensibilidad

La sensibilidad (K_V) de un VCO indica la relación entre la frecuencia de oscilación y el voltaje de control. Su valor es importante porque normalmente representa un compromiso entre:

- Rango de frecuencias de funcionamiento: A mayor sensibilidad, mayor rango de frecuencias abarca el VCO.
- Factor de ruido del oscilador: Cuanto menor sea la sensibilidad, menos ruido de fase contendrá la señal sintetizada.

Sin embargo, en este integrado en concreto se utiliza una técnica que permite un amplio margen de frecuencias con una baja sensibilidad. Esta técnica consiste en cambiar mediante multiplexores internos las capacidades del oscilador, conmutando entre ocho bandas de frecuencias distintas, como se ve en esta gráfica:

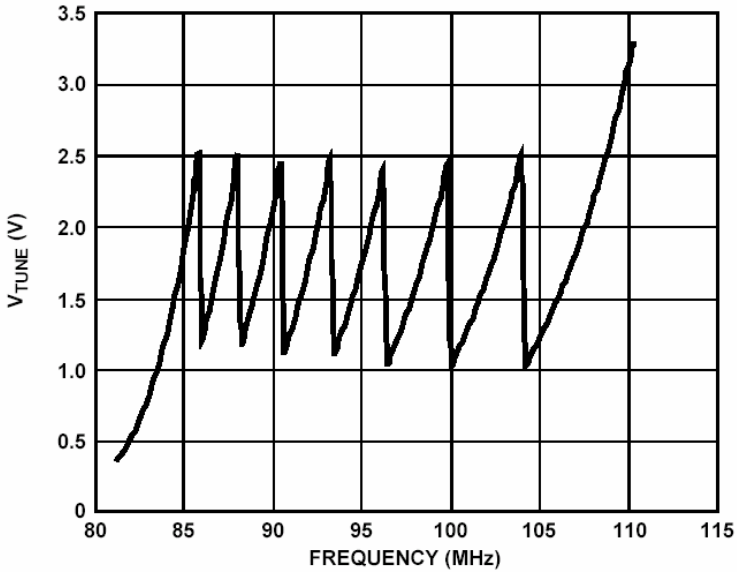


Figura 68: Frecuencia de oscilación en función del voltaje para el VCO integrado

Hay que avisar que esta gráfica está realizada para un valor de inductancia distinto del que se usa en este proyecto, y los valores de frecuencia no se corresponden con los del receptor. Lo que sí se puede observar es que usando esta técnica se reduce notablemente

la sensibilidad K_V del VCO. Usando la siguiente gráfica se estima que la sensibilidad K_V está en torno a 4MHz/V.

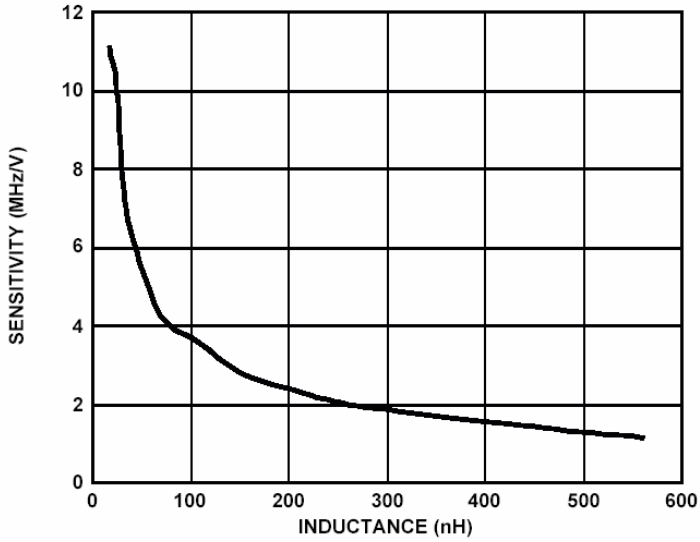


Figura 69: Sensibilidad Kv en función del valor de la inductancia

Como comparación, se puede estimar la sensibilidad del VCO basado en BJT discreto que se diseñó y cuyo esquema se mostraba en el apartado 6.13, usando la gráfica que allí se muestra de frecuencia de oscilación frente a voltaje de control. En este caso la sensibilidad está en torno a 26.7MHz/V. Por lo tanto el VCO integrado presenta una ventaja muy importante con respecto al oscilador discreto: su salida contendrá mucho menos ruido de fase.

4.9.3. Adaptación de impedancias a la salida

El VCO integrado dispone internamente de una etapa de salida que proporciona por duplicado la señal del oscilador. Estas etapas de salida son de colector abierto, por lo que es necesario polarizar hacia alimentación estos pines (`RFOUTA` y `RFOUTB`). Sin embargo una polarización resistiva, que sería lo más sencillo, produce tal desadaptación de impedancia a la salida que se reduce la potencia entregada a la carga en unos 10dB. Por eso se usará la otra opción que se da en el *datasheet*, que es polarizar a través de unas inductancias. Como la impedancia de salida varía con la frecuencia, el valor de esta inductancia óptima también varía. El fabricante no proporciona la impedancia de salida ni los parámetros S, sino directamente el valor de la impedancia que maximiza la entrega de potencia a una carga de 50Ω. En la gráfica, que se muestra abajo, se puede ver que para la frecuencia de diseño esta inductancia está en torno a los 60nH.

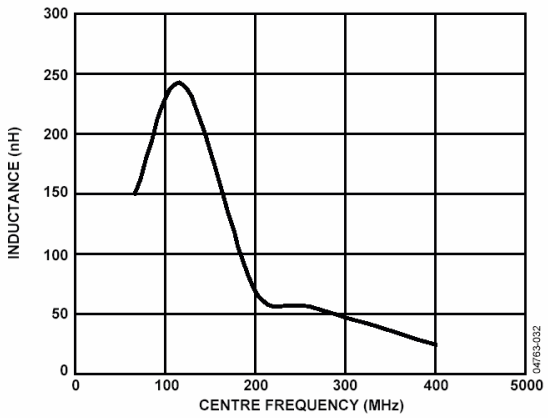


Figura 70: Inductancia de adaptación de la salida del VCO

Como valor disponible comercialmente se ha tomado el de 56nH. Abajo se muestra ya el esquema final del PLL y el VCO con todos los elementos discutidos en este apartado y el anterior. Sobre las dos salidas del VCO, ambas se han conectado a una carga de 50Ω a través de un conector de tres pines. En realidad este conector solo es un punto de prueba sobre la placa: En uno de ellos irá soldado el cable de 50Ω que va hacia el demodulador, mientras que el otro estará normalmente puenteado hacia la carga adaptada, pero para las medidas se abrirá el circuito y ahí se soldará la sonda del analizador de espectros. Para asegurar que el acoplamiento mútuo entre las dos inductancias del oscilador se minimiza, en la placa se han colocado perpendicularmente la una con la otra para evitar que interfieran sus flujos electromagnéticos.

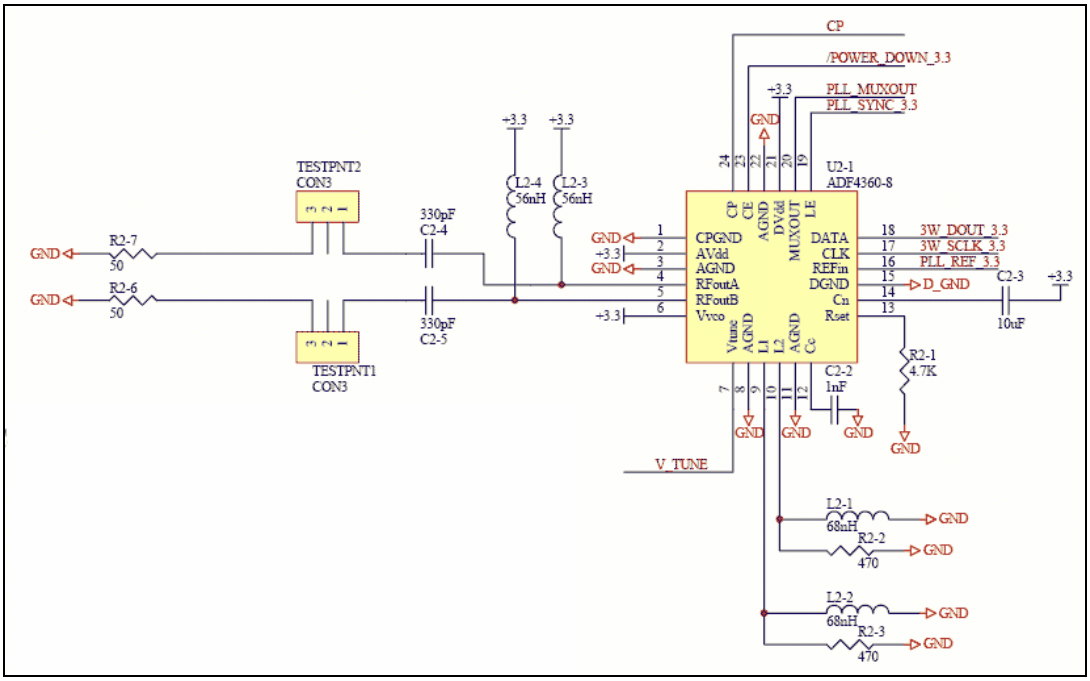


Figura 71: Conexión final del PLL.

4.10. Alimentación

La alimentación de todo el receptor se obtiene directamente del cable USB. El estándar USB [3] define unos cables de cuatro hilos, que son: Alimentación, masa, datos (+) y datos (-), donde las líneas de datos son diferenciales y semiduplex. La tensión de alimentación es de 5.0V, y se establece un límite de corriente por dispositivo USB de 500mA. Para comprobar este límite, en la tabla siguiente se muestran los consumos de cada uno de los integrados del receptor, tanto en modo de bajo consumo (cuando la señal `/POWER_DOWN` esta a nivel bajo) como en estado normal de ejecución. Cuando el receptor se conecta inicialmente al PC, el microcontrolador pondrá a todos los integrados en el modo de bajo consumo hasta que el software del PC indique otra cosa. Como se ve, el límite de 500mA se cumple ya que en total se consume unos 218mA resultando un consumo de potencia de 1.1W en funcionamiento normal, y de 87mA o 0.44W en estado de bajo consumo. El estándar también especifica que un dispositivo debe consumir menos 100mA justo al conectarse al PC, lo que se cumple ya que el receptor se inicia en modo de bajo consumo.

Circuito integrado	**Consumo en *power-down* *(mA)***	**Consumo normal (mA)**
Microcontrolador, ATMEGA8515 a 10Mhz	11	11
PLL y VCO integrados, ADF4360-8	0.007	29
Memoria EEPROM para chip USB, 93C46	0.001	0.001
Interfaz USB, FT8U245BM	25	25
Memoria FIFO, CY7C425	10	20
Amplificador LNA, MAX2611	16	16
Amplificador VGA, AD8367	1.3	26
Demodulador, AD8348	0.075	48
Conversores DAC, AD7304	0.040	6
Conversores ADC, AD7822 (x2)	0.010	16
Lógica externa, 74HC00, 74HC109 (x2)	3	3
Amplificadores diferenciales, AD8129 (x2)	18	18
TOTAL	**87.1**	**218**

También se especifica en el estándar USB que se debe colocar una inductancia a modo de choque de alta frecuencia justo en la entrada del dispositivo en la línea de alimentación que viene del PC. Con esto se consigue reducir la emisión de interferencias de alta frecuencia desde el cable USB.

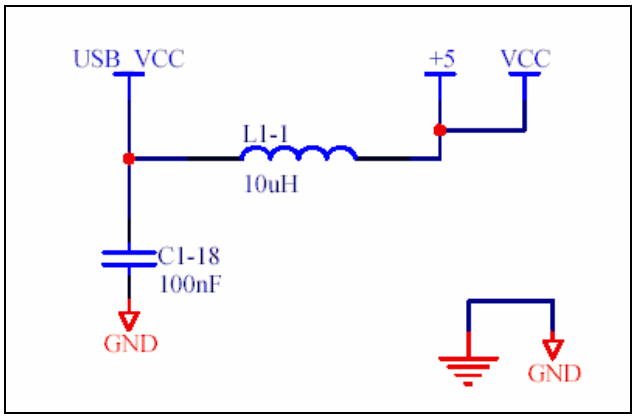


Figura 72: Choque colocado a la entrada de la alimentación

Además de este choque, es necesario eliminar los ruidos y transitorios en las líneas de alimentación del receptor. Estos ruidos se originan sobre todo en circuitos digitales, ya que la conmutación de puertas lógicas dentro de ellos produce aumentos temporales de la corriente de alimentación del integrado. Para eliminar estas fluctuaciones se han colocado varios condensadores de gran capacidad (10µF) en cada extremo de la placa principal, además de un condensador de desacoplo de oscilaciones a alta frecuencia (de 100nF) junto a cada pin de alimentación de cada integrado en cada placa del receptor.

5. DISEÑO DEL HARDWARE DIGITAL

5.1. Microcontrolador

El microcontrolador es el encargado de sincronizar, controlar y configurar el resto de dispositivos del receptor, además de actuar de interprete para los comandos que se envíen desde el PC. Se ha utilizado el microcontrolador ATMEGA8515, de la compañía *Atmel*. Esta decisión se ha tomado principalmente por la extensa experiencia previa en la programación y uso de esta familia de microcontroladores, además de su potencia que permite ser usado para la tarea más crítica en tiempo de las que se tratan en este proyecto, que es el control de la tarea de muestreo de las señales I y Q, proceso que se detalla en el apartado 5.3.

La familia ATMEGA de *Atmel* contiene microcontroladores con arquitectura RISC, que permiten la ejecución de una instrucción por ciclo de reloj. Otras características que han propiciado esta elección son:

- Sistema microcontrolador totalmente auto contenido y fácilmente reprogramable: El integrado contiene 512 bytes de memoria RAM, 512 bytes de EEPROM para datos y 8Kb de memoria flash para el programa, tamaños más que suficientes para esta aplicación. Además, la forma de reprogramar el chip se explica claramente en el *datasheet*, por lo que se ha construido un sencillo programador, descrito en el capítulo 8, con el que se puede programador el chip en pocos segundos de forma sencilla.
- Diseño de memoria lineal. No utiliza bancos de memoria ni segmentos, facilitando enormemente su programación.
- Unos 35 puertos de entrada-salida digitales, organizados en los puertos PA, PB, PC, PD y PE, todos de ocho bits menos el último, que solamente es de tres bits.
- Dispone de varios *timers*, fuentes de interrupción externas y otros periféricos.
- Un conjunto de 32 registros de 8 bits, todos accesibles en todo momento y mapeados directamente en memoria.
- Tiene capacidad de *autoprogramación*: mediante software se puede reprogramar el chip sin necesidad de sacarlo del sistema e introducirlo en el programador. Se ha

hecho uso de esta capacidad para permitir la reprogramación desde la interfaz gráfica del PC, lo que ha sido de enorme utilidad durante el desarrollo y depuración.

A continuación se reproduce el esquema interno del microcontrolador, donde se muestran esquemáticamente todos los periféricos de que dispone:

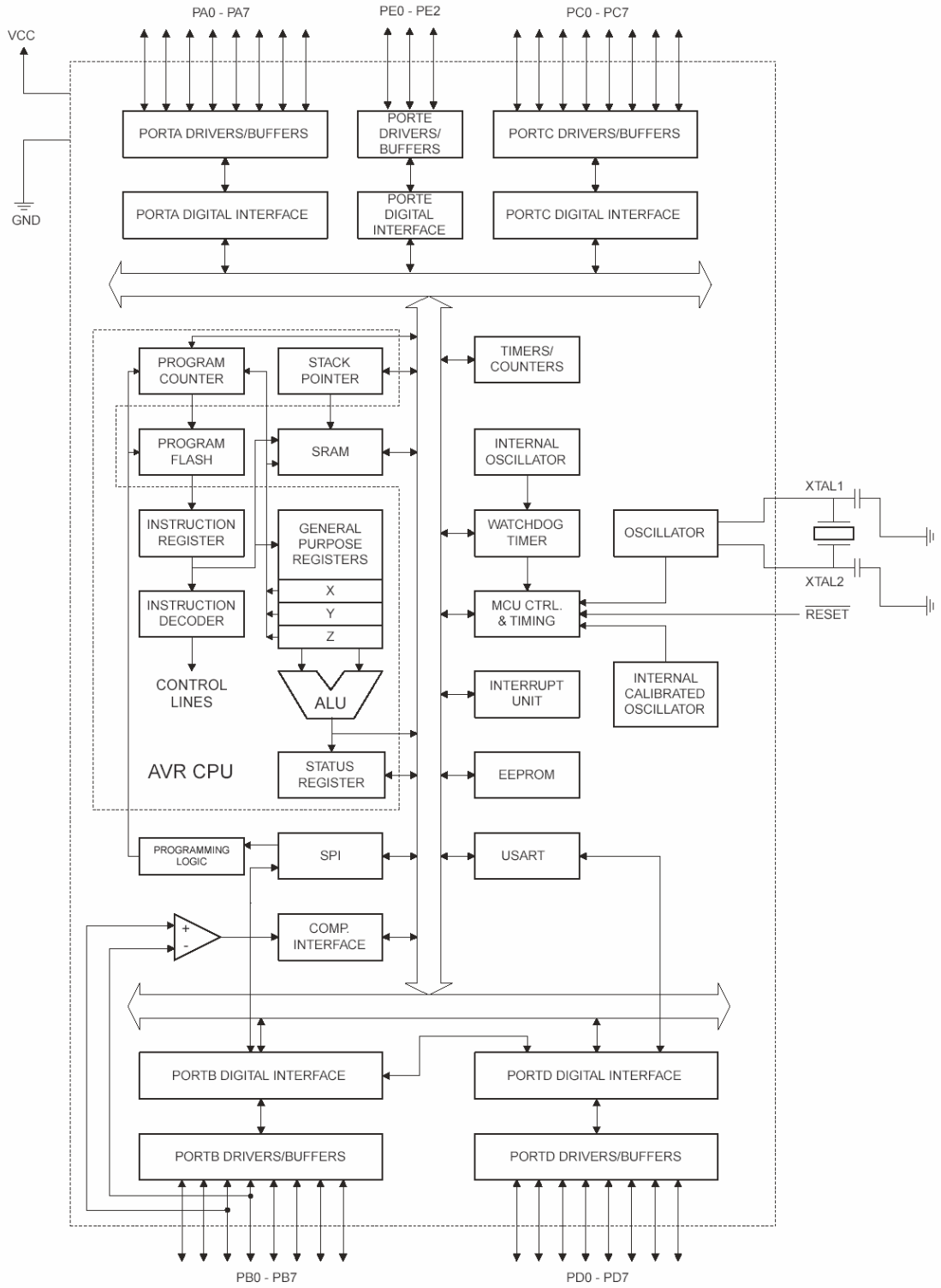


Figura 73: Diagrama de bloques interno del microcontrolador

En cuanto al encapsulado, y aunque la totalidad del receptor está construido con componentes SMD, el caso del microcontrolador es el único chip que se adquirió en formato DIP (en concreto, DIP40). Esto es así para, mediante un zócalo, permitir la extracción del chip para introducirlo en el programador.

En cuanto a la frecuencia de funcionamiento, este microcontrolador llega hasta 16Mhz, pero se usa un cristal de 10Mhz. El circuito oscilador a cristal está incluido en el chip. La elección de esta frecuencia está limitada por un límite en la frecuencia de funcionamiento que se explica en el apartado 5.4 dedicado a la interfaz USB.

A continuación se detallan brevemente los grupos de señales que entran y salen del microcontrolador y que se pueden ver en el siguiente fragmento de esquema (los esquemas completos están en el anexo A1):

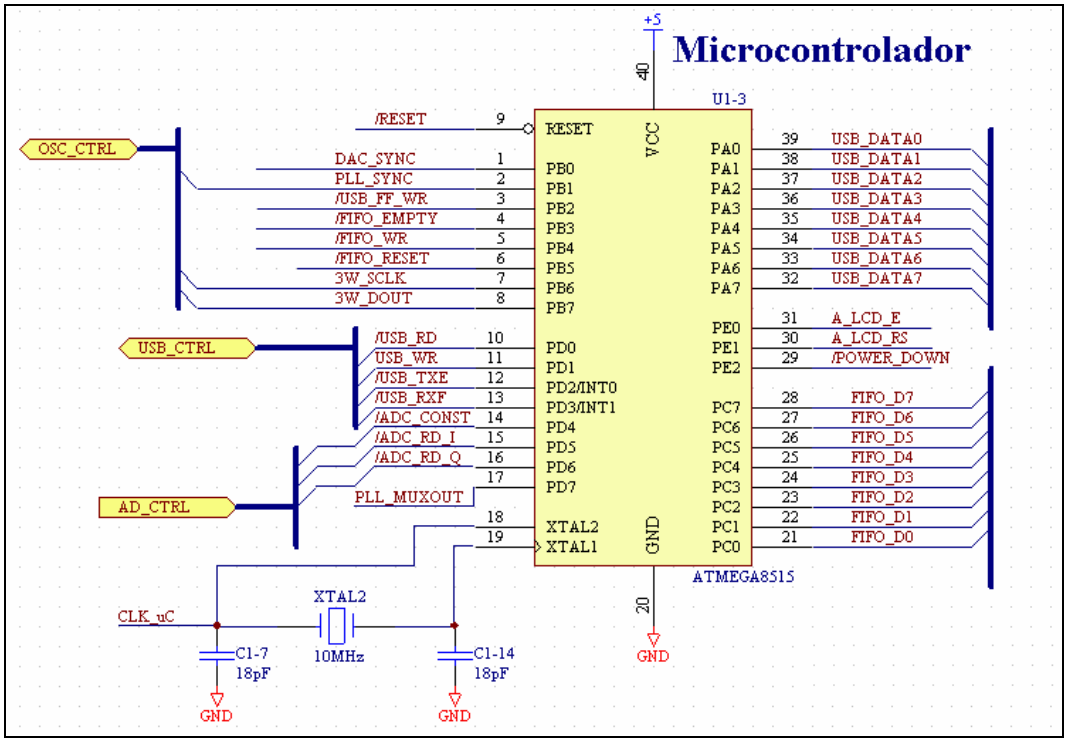


Figura 74: El microcontrolador y sus conexiones.

- Línea de reset (“/RESET”): Esta señal activa a nivel bajo reinicia el microcontrolador. Se genera con un sencillo circuito RC que mantiene esta línea a nivel lógico bajo al conectar la alimentación del receptor (conectarlo al PC) durante unos milisegundos. Esta misma señal está conectada a otros integrados del receptor para asegurar que inicialmente todos los dispositivos se encuentren en un estado inicial conocido.

- Bus de datos del chip interfaz USB (Puerto A, "`USB_DATA[0..7]`"). Este bus es tanto de entrada como de salida. Las transacciones a través de este bus se controlan mediante las señales "`/USB_RD`" y "`USB_WR`", según se explica en el apartado 6.12.
- Indicadores del chip interfaz USB ("`/USB_TXE`" y "`/USB_RXF`"). Estas señales de entrada al microcontrolador indican cuando el chip USB acepta más datos para enviar al PC o hay datos nuevos desde el PC, respectivamente. Aunque esta última sí se procesa directamente desde el microcontrolador para recibir datos desde el PC, la primera se utiliza directamente desde la lógica discreta descrita en el apartado 5.3 y no afecta al microcontrolador. La señal "/USB_FF_WR" no va directamente al chip USB, sino a la lógica externa, y aunque forma parte del diseño del sistema, posteriormente se decidió no hacer uso de esta señal, como se explicará en ese apartado.
- Bus SPI (*Serial Peripheral Interface*). Este bus, un estándar de la industria electrónica, consta de un dispositivo *master* (en este caso el microcontrolador) y uno o varios dispositivos *slave* (en este caso el PLL y los DAC). El bus consta de dos señales comunes a todos los *slaves*: una señal de reloj ("`3W_SCLK`") y otra de datos ("`3W_DATA`"), y debe existir otra señal de sincronización o *latch* para indicar individualmente a qué dispositivo se dirigía la última secuencia enviada. En el receptor, estas señales son "`DAC_SYNC`" y "`PLL_SYNC`". En la siguiente figura se muestra un ejemplo, sacado del *datasheet* del PLL:

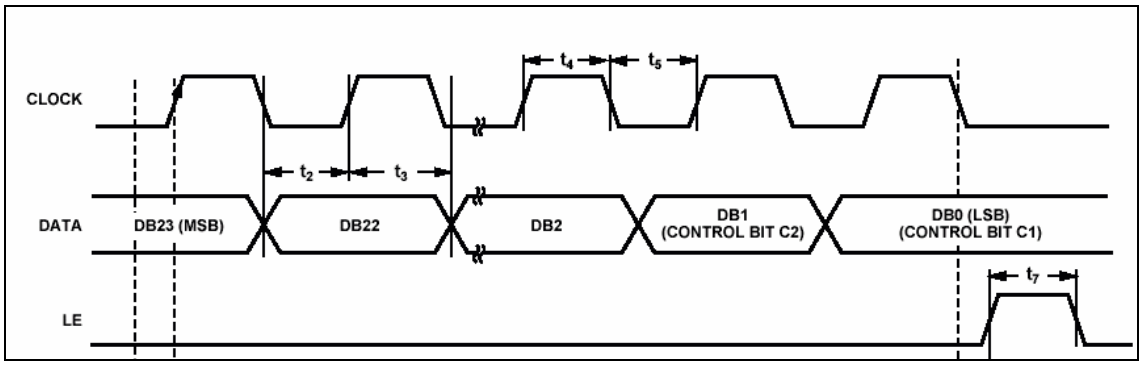


Figura 75: Cronograma típico del bus SPI.

- Señales relacionadas con la FIFO: La memoria FIFO se presenta en el siguiente apartado. La línea "`/FIFO_RESET`" es usada por el microcontrolador para reiniciar la FIFO, la "`/FIFO_EMPTY`" para comprobar si quedan datos en la memoria y la línea "`/FIFO_WR`" se usa para añadir los datos del bus "`FIFO_D[0..7]`" a la cola de la memoria FIFO.

- La salida "/POWER_DOWN" permite poner en modo de bajo consumo (desactivados) a los siguientes dispositivos del receptor: el PLL, el demodulador, el amplificador VGA y los conversores ADC.
- Señales relacionadas con la pantalla LCD: Además de cuatro líneas de datos, cuyo uso se comparte con el bus de datos del chip USB, se usan las salidas "A_LCD_E" y "A_LCD_RS". En el apartado 5.7 se explican las comunicaciones con el LCD.
- La señal de entrada "PLL_MUXOUT" se corresponde con una salida del PLL que tiene multiples funcionalidades ("MUX" es por "multiplexada") y que se utiliza, entre otras cosas, para comprobar la correcta comunicación con el PLL. Se dan más detalles en el apartado 4.8.6.
- Las señales relacionadas con la coordinación de los conversores ADC, cuya temporización se detalla en el apartado 5.3, son "/ADC_CONVST", "/ADC_RD_I" y "/ADC_RD_Q". La conexión de estas señales a los conversores se muestra en el apartado 5.5 correspondiente a estos chips.
- El oscilador a cristal se encuentra entre los pins XTAL1 y XTAL2, siendo este último la salida, que por tanto se usará como entrada a otras puertas lógicas para generar otros relojes en el sistema (apartado 5.3).

El software que se ejecuta sobre este microcontrolador se llama en este proyecto el *firmware* del receptor, y a su diseño y contenido se dedica el apartado 7.2. Para ajustar el comportamiento del microcontrolador existe un conjunto de bits de configuración (*fuse bits*), accesibles en modo de programación (apartado 8) donde las opciones más importantes que se permiten modificar se listan en la siguiente tabla. Las opciones seleccionadas para este proyecto se indican en negrita.

Opción	Descripción
S8515C	Permite habilitar o **deshabilitar** la compatibilidad con una versión anterior del microcontrolador, el AT90S8515
WDTON	Habilitar o **deshabilitar** el *watchdog timer* de forma permanente
CKSEL[0..3]	Permite elegir la fuente del reloj, entre osciladores RC internos o externos o **un oscilador a cristal externo**.
BOOTSZ[0..1]	Tamaño de la zona de código llamada "*boot*". En este proyecto sí se usa y es **de 1024 palabras**.
BOOTRST	Indica si al reiniciar el microcontrolador se debe empezar a ejecutar en la dirección inicial o el **sector "boot"**
SUT[0..1]	Permite elegir el retardo tras la alimentación del chip hasta iniciar la ejecución del código, entre 0ms, 4.1ms y **65ms**

5.2. Memoria FIFO

Antes de entrar en los detalles del integrado con que se ha implementado la memoria FIFO, es conveniente representar mediante un esquema la situación de esta memoria en el flujo de los datos de las señales muestreadas para explicar su necesidad y su uso:

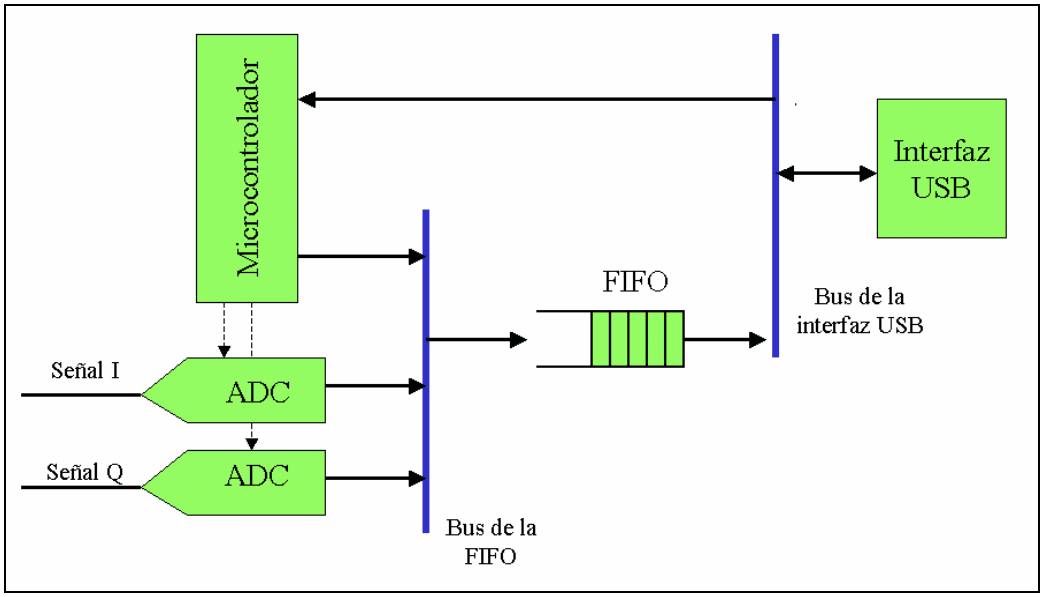


Figura 76: Esquema del flujo de datos en el receptor

En el esquema se puede ver como el microcontrolador solamente lee del bus de la interfaz USB, mientras que sólo escribe en el bus de la memoria FIFO, donde también escriben ambos conversores ADC, bajo control del microcontrolador (concretamente usando las señales "`/ADC_RD_I`" y "`/ADC_RD_Q`"). La única vía de entrada de datos a la interfaz USB (datos con destino al PC) es a través de la FIFO, y el control de estas salidas es la que realiza la lógica discreta descrita en el siguiente apartado.

Aunque el microcontrolador podría escribir directamente a la interfaz USB, ya que sus puertos son de entrada o salida de forma programable, se tomó la decisión de que solamente envíe datos al PC a través de la FIFO. Si no fuera así, se tendría un problema de acceso a un recurso compartido por más de un elemento, y sería difícil asegurar que se respete la secuencia de los *bytes* escritos en la FIFO. En cuanto a la lectura de datos desde el PC, como el microcontrolador es el único elemento que lee de la interfaz USB, no hay ningún conflicto posible y siempre se recibirán todos los datos y en el orden correcto.

Realizando una análisis de las necesidades de comunicación entre el receptor y el PC, se llega a tres casos diferenciados, que son:

1. Lectura de datos provenientes del PC. En este caso, como ya se ha dicho, el microcontrolador puede leer directamente del chip de interfaz USB, ya que está conectado a su bus de datos y a las señales de control de este chip. Sin embargo, el bus de entrada-salida de la interfaz USB se comparte por el microcontrolador y la memoria FIFO. Las escrituras de esta última están gobernadas por una lógica externa, que se encarga de que todos los datos sean escritos, hasta que la FIFO quede vacía. Por eso, el microcontrolador debe siempre verificar que la FIFO esté vacía antes de leer para evitar que dos dispositivos escriban a la vez en el bus. En la siguiente figura se ilustra el caso de lectura y las señales que están activas durante esta (marcadas con un grosor mayor que el resto):

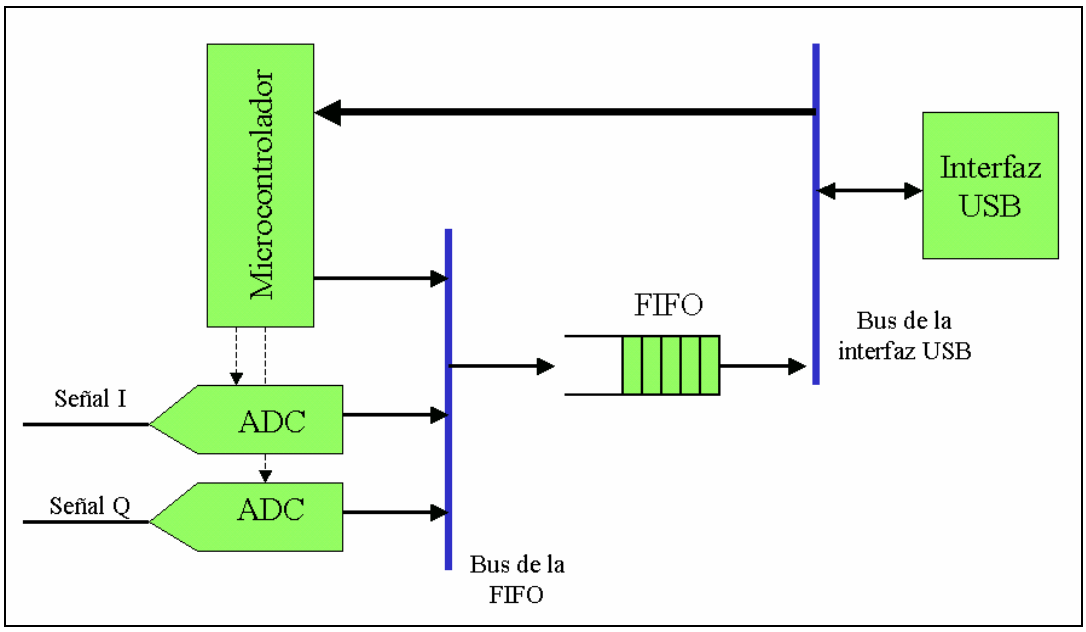


Figura 77: Flujo de datos durante lectura del chip USB

2. Escritura de datos desde el microcontrolador hacia el PC: En este caso, el microcontrolador escribe a la FIFO, y esta volcará los datos hacia el chip de interfaz USB, como se ilustra a continuación:

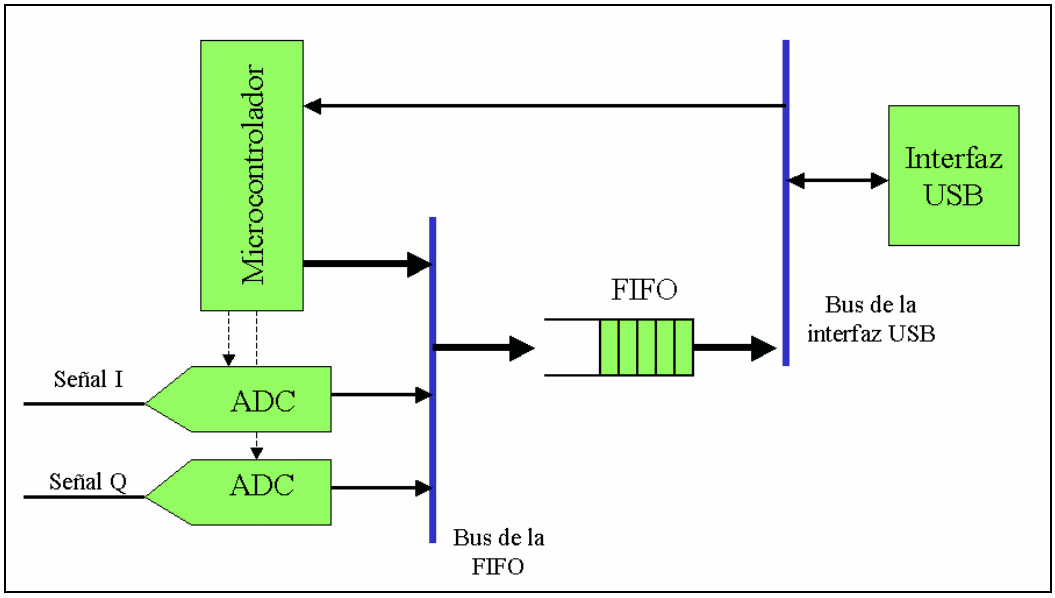


Figura 78: Flujo de datos durante escritura desde el microcontrolador

3. Escritura de datos desde los conversores hacia el PC: En este caso los conversores, de forma sincronizada gracias al microcontrolador, vuelcan las muestras a la FIFO, que las mandará al chip de interfaz USB:

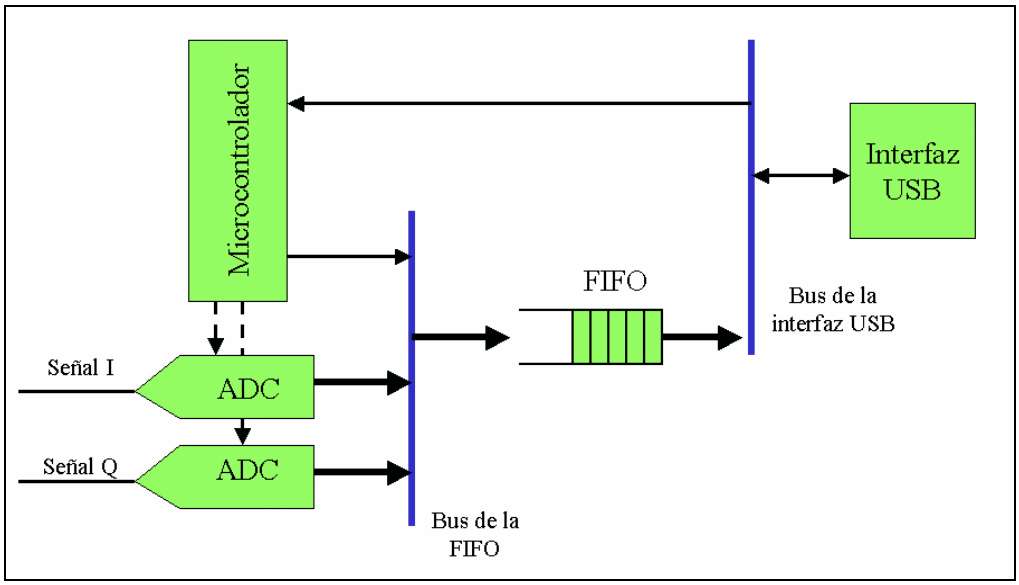


Figura 79: Flujo de datos durante escritura desde los ADC

Estos tres casos incluyen todos los posibles escenarios que se dan en el camino del flujo de datos. En cuanto a la elección del chip para implementar la FIFO, se optó por el integrado CY7C425-10 de CYPRESS, una memoria FIFO de nueve bits (uno de ellos

no se usa en este proyecto) de ancho de palabra, y una profundidad de 1024 palabras. Como se ve en el siguiente esquema, sus elementos más importantes son un bus de entrada ("`D[0..8]`"), un bus de salida ("`Q[0..8]`") y unas señales de control para escribir ("`/W`") y para leer ("`/R`"):

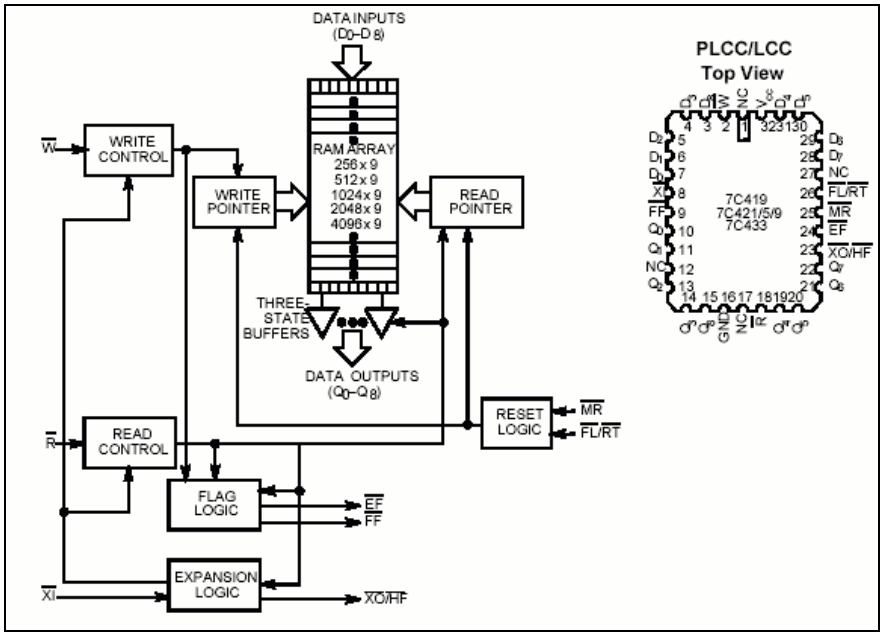


Figura 80: Memoria FIFO: diagrama y *pinout*.

Otras señales que se usan son una entrada para reiniciar la memoria ("`/MR`") y un indicador de que la memoria está vacía ("`/FIFO_EMPTY`"). Además, la FIFO permite el encadenamiento de varias memorias en profundidad, que como no se usa en este proyecto obliga a poner la entrada `/XI` a nivel bajo y `/FL` a nivel alto, quedando el circuito como sigue:

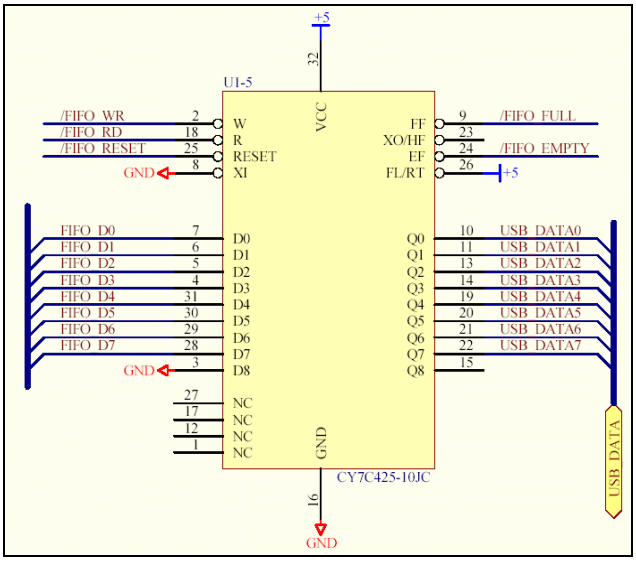


Figura 81: Conexión de la memoria FIFO.

A continuación se describen las secuencias de señales que son necesarias tanto para escribir como para leer de la FIFO, protocolos que serán usados, respectivamente, en el diseño del *firmware* y en el diseño de la lógica externa del siguiente apartado:

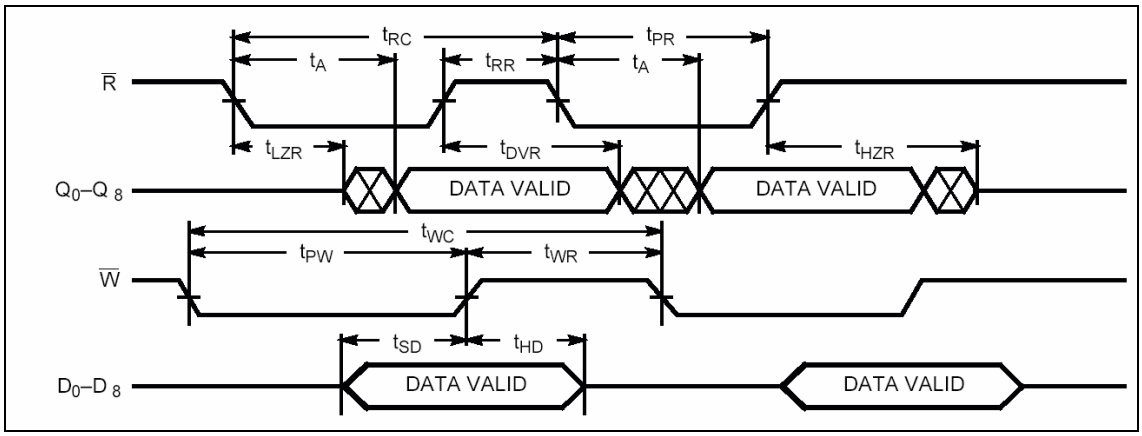


Figura 82: Temporización de las lecturas y escrituras en la FIFO

El anterior cronograma muestra dos operaciones, lectura y escritura, ejecutadas de forma asíncrona ya que son operaciones independientes en este tipo de memorias. Sabiendo que el integrado que se ha usado es la versión CY7C425-10, los tiempos que se ven en el cronograma que son importantes para el diseño del *firmware* tienen estos valores:

Símbolo	Valor	Significado
t_A	10ns max.	Tiempo de acceso
t_{PW}	10ns min.	Ancho del pulso de escritura
t_{PR}	10ns min.	Ancho del pulso de lectura
t_{SD}	6ns min.	Tiempo de *setup* para escritura

Sin embargo, como la frecuencia de funcionamiento del microcontrolador es de 10Mhz, el período de ejecución de instrucciones es de 100ns, siendo este el retardo mínimo entre dos cambios de una señal que venga desde el microcontrolador. Es decir, la memoria FIFO es suficientemente rápida como para no preocuparse de añadir ningún tipo de retardos ni tiempos de guarda cuando se acceda a ella desde el microcontrolador.

Otra secuencia de señales a tener en cuenta es la de reinicio de la memoria, que se muestra en el siguiente cronograma:

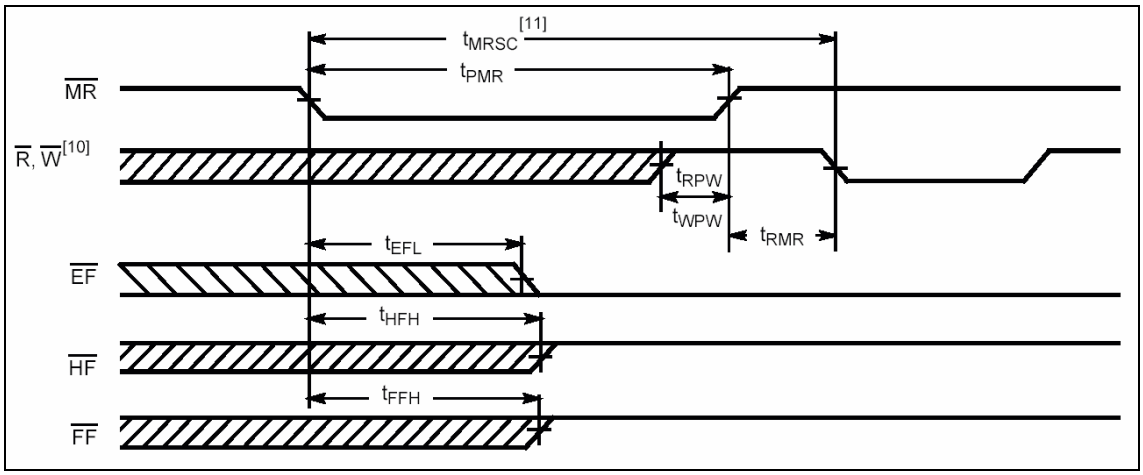


Figura 83: Reinicialización de la FIFO

Donde t_{PMR} es el mínimo pulso de *reset* y que es de 10ns, que también se cumple sobradamente ya que el período del microcontrolador es de 100ns. Lo que el firmware sí debe asegurar es que cuando la señal `/MR` vuelva a nivel alto, las señales `/R` y `/W` estén también a nivel alto, manteniendo la memoria vacía hasta que realmente sea usada.

5.3. Lógica de interfaz FIFO-USB

El objetivo de esta lógica es descargar al microcontrolador del volcado de la memoria FIFO a la interfaz USB para que los datos sean enviados al PC. Este proceso involucra a estas señales:

- "`/USB_TXE`": Salida del chip interfaz USB, que indica con un nivel bajo que está listo para aceptar más datos.
- "`/FIFO_EMPTY`": Salida de la FIFO que indica con un nivel lógico alto que hay datos en la cola.
- "`USB_WR`": Entrada al chip interfaz USB que señala con un nivel alto que el chip debe leer de su bus de entrada-salida el nuevo *byte* a transmitir al PC.
- "`/FIFO_RD`": Entrada a la FIFO que le indica que debe sacar a su bus de salida el primer elemento en su cola.

A continuación se muestra un cronograma como ejemplo para explicar la sincronización que se quiere conseguir entre estas señales, usando una máquina de estados. Las flechas bajo los nombres de las señales indican si las señales son de entrada a la máquina de estados (apuntando a la derecha), o de salida (apuntando a la izquierda):

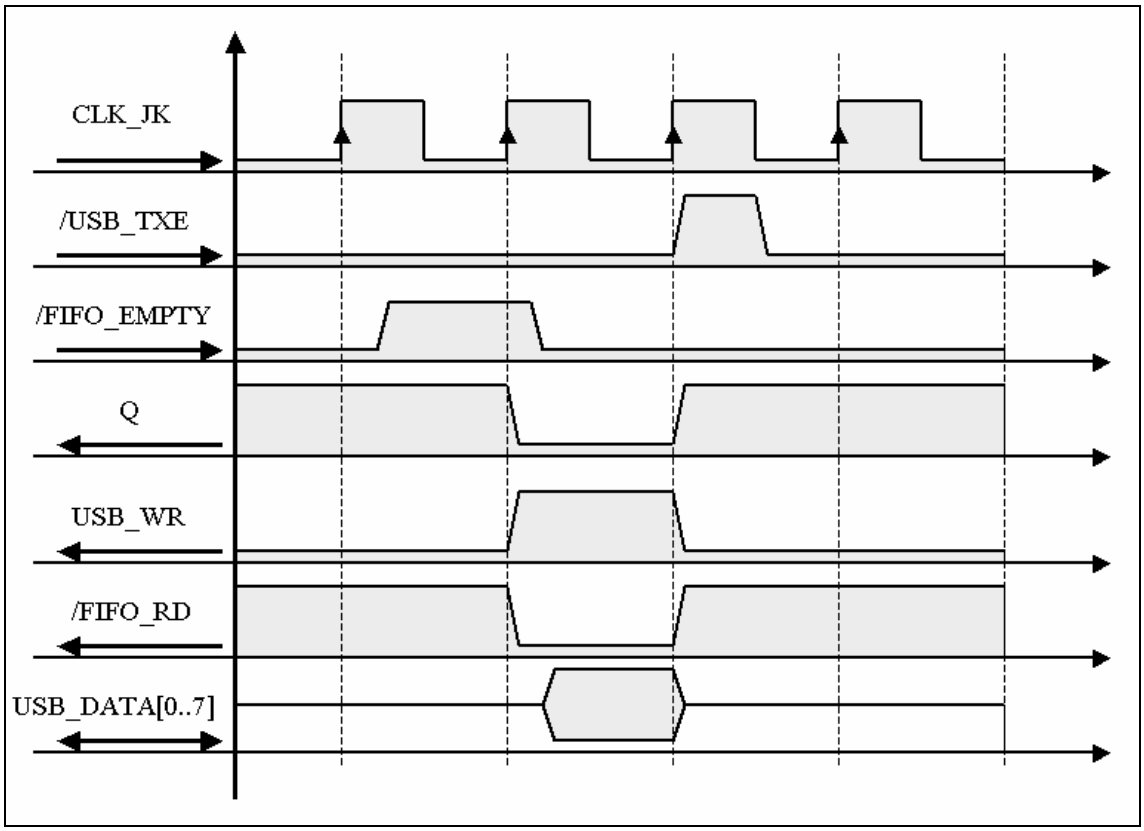


Figura 84: Cronograma mostrando la sincronización a conseguir

La máquina de estados se implementará usando un solo *bit* de estado (un solo *flip-flop*), ya que como se ve a continuación es suficiente. El valor de este *bit* es el mostrado como Q en el cronograma. Las entradas a la máquina de estados son el reloj ("`CLK_JK`") y las dos señales "`/USB_TXE`" y "`/FIFO_EMPTY`" que ya se han descrito. Las dos salidas de la máquina de estados ("`USB_WR`" y "`/FIFO_RD`") también se representan en la gráfica.

La idea es que en estado de reposo, cuando no haya datos en la FIFO ("`/FIFO_EMPTY`" a nivel bajo), la señal de escritura en el chip USB se mantenga a nivel bajo y la de lectura de la FIFO a nivel alto, es decir, ambas inactivas. Cuando en la FIFO haya datos, y simultáneamente la interfaz USB este lista para recibirlos, se debe dar un pulso a las señales `USB_WR` y `/FIFO_RD`, volviendo al estado inactivo

obligatoriamente aunque haya más datos, ya que son necesarios los flancos tanto de subida como de bajada de estas señales para completar la transmisión entre los dos dispositivos. Como se ve en la última fila del cronograma, la FIFO vuelca los datos al bus tras la activación de `/FIFO_RD`, mientras que el chip USB muestrea el bus en el flanco de bajada de `USB_WR`, poniendo `/USB_TXE` a nivel alto indicando que esta procesando el nuevo dato y no acepta más. Según el *datasheet*, y según se ha verificado con un osciloscopio, este pulso suele durar unos 80ns, a menos que se llene el buffer interno del chip USB (el estándar USB en el modo *full-speed* permite volcar datos solamente una vez cada milisegundo, ver [3]), tiempo durante el cual la señal `/USB_TXE` se mantiene a nivel alto. En estos casos, la máquina de estados debe asegurar que los datos no se escriban al chip USB porque se perderían, tiempo durante el cual los datos se irán acumulando en la FIFO.

En cuanto a la frecuencia a la que este proceso debe ejecutarse (la frecuencia de reloj de la señal "`CLK_JK`"), debe ser al menos, de la tasa de transmisión media de datos. Esta tasa, según se explica en el apartado 5.4, es de 500Kb/s. Sin embargo, debido a los períodos en que el chip USB no acepta datos (por lo que se acaba de explicar), los datos acumulados en la FIFO deben ser capaces de volcarse al chip USB a una velocidad lo suficientemente rápida como para que los retrasados se transmitan en rápidas ráfagas y no se acumulen datos indefinidamente en la FIFO. Haciendo un cálculo simplista, se podría pensar en el peor caso en que durante un milisegundo completo el chip USB no acepte datos. Durante este período, se acumularán en la FIFO:

$$500\frac{\text{Kb}}{\text{s}}\cdot 1\,\text{ms} = 500\,\text{bytes}$$

Hay que señalar que este sencillo cálculo se ha usado para dimensionar la memoria FIFO, cuyo tamaño es de 1024 palabras, valor suficientemente superior al calculado, asegurando que nunca se desbordará. Siguiendo con el razonamiento anterior, tras este milisegundo el chip USB debería volver a aceptar datos, y deberían ser volcados tanto los 500 *bytes* acumulados como los 500 nuevos de este milisegundo: es decir, la velocidad de transmisión debe ser de, al menos 1000 *bytes* por milisegundo, o 1000Kb/s. Para evitar añadir otro oscilador a cristal al diseño, y aprovechando que el microcontrolador usa un oscilador 10MHz, se ha diseñado un sencillo divisor de frecuencias entre cuatro usando dos *flip-flops*, que se usará como el reloj de la máquina de estados, como se ve en el siguiente esquema.

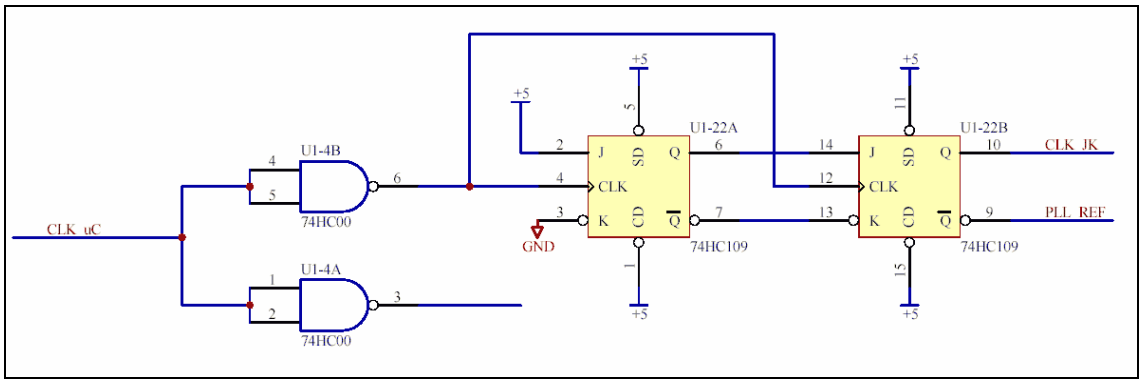


Figura 85: Generación de reloj para los *flip-flops*

La puerta inversora `U1-4B` se usa como buffer, para no cargar la salida del oscilador interno al microcontrolador. Los *flip-flops* usados son del tipo JK (74HC109) y el primero de ellos está conectado con J=1, K=1, de forma que su salida conmute en cada ciclo de reloj. Con esto se consigue un divisor por dos. El segundo de ellos está conectado de tal forma que cuando la salida del primero es un nivel bajo, no cambia (J=0, K=0), pero cuando la salida del primero sea un nivel lógico alto, sí conmute, obteniendo otro divisor por dos, y en total, un divisor por cuatro. Así, se tiene que las salidas "`Q`" y "`/Q`" de `U1-22B` son señales cuadradas de 2.5Mhz de frecuencia. Una de ellas se utiliza como reloj de la máquina de estados, y la otra como referencia para el PLL.

Retomando el diseño de una máquina de estados que implemente el comportamiento mostrado en el último cronograma, se llega a este sencillo diagrama de estados:

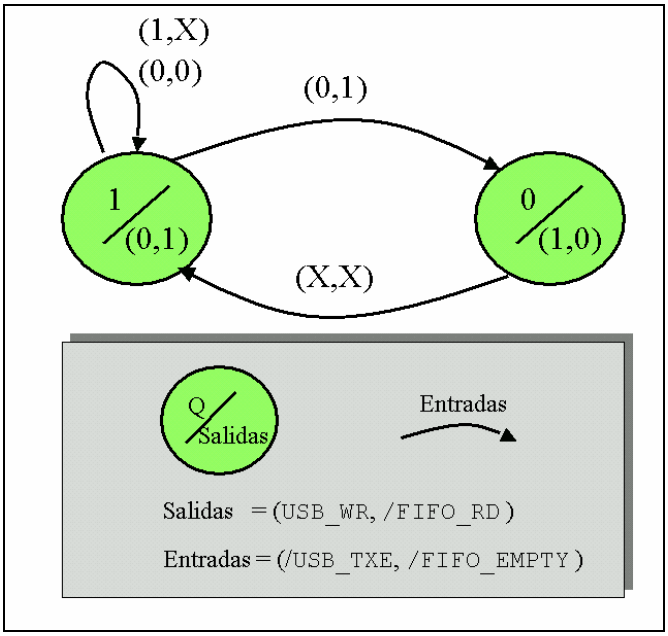


Figura 86: Máquina de estados a implementar

Donde la leyenda explica la notación usada para los estados (el valor del bit "Q") y las entradas y las salidas, que solamente dependen del estado. Por simple observación es inmediato obtener las ecuaciones lógicas de las señales de salida:

```
USB_WR = /Q
/FIFO_RD = Q
```

A continuación hay que extraer la tabla de transición de estados, que indica el siguiente estado para cada combinación de estado anterior y señales de entrada:

Q_n	/USB_TXE	/FIFO_EMPTY	Q_{n+1}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Y sabiendo que la tabla de verdad que controla el funcionamiento de un *flip-flop* JK es la siguiente:

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Se obtienen los valores deseados para las señales J y K del *flip-flop* de la máquina de estados:

Q_n	/USB_TXE	/FIFO_EMPTY	Q_{n+1}	J	K
0	0	0	1	1	X
0	0	1	1	1	X
0	1	0	1	1	X
0	1	1	1	1	X
1	0	0	1	X	0
1	0	1	0	X	1
1	1	0	1	X	0
1	1	1	1	X	0

A partir de esta última tabla es directo obtener las ecuaciones:

$$J = 1$$

$$\overline{K} = \overline{USB\_TXE} \cdot \overline{FIFO\_EMPTY}$$

Que se han implementado usando puertas NAND, como se ve en el esquema definitivo:

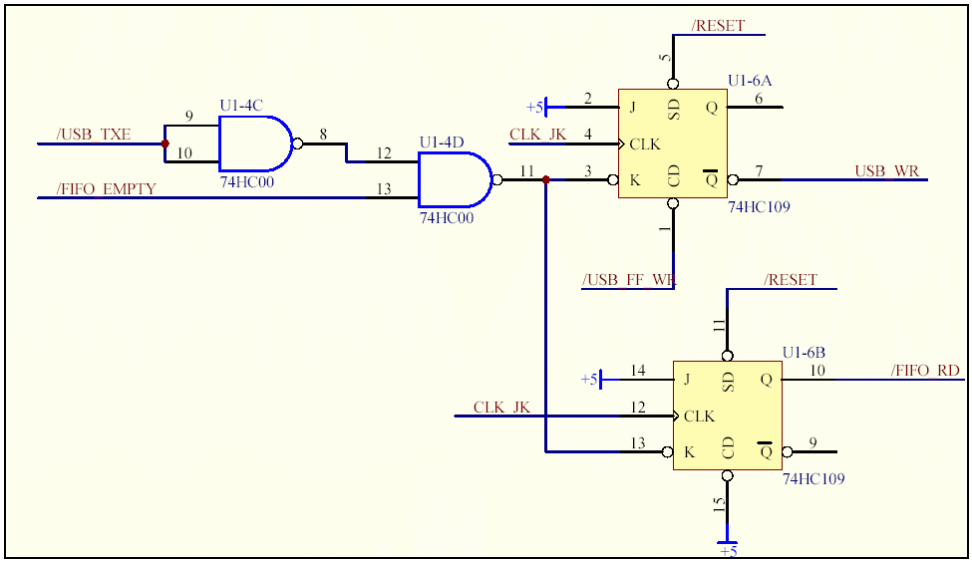


Figura 87: Implementación de la máquina de estados

Aunque la máquina se puede implementar mediante un solo flip-flop, el diseño se hizo originalmente por duplicado: ambos *flip-flops* tienen las mismas entradas, y por tanto se comportan de igual forma. Esto es así porque inicialmente se dejó abierta la posibilidad de controlar manualmente desde el microcontrolador la señal `USB_WR` sin afectar a la señal `/FIFO_RD`, lo que se hizo conectando el terminal `/CLEAR` del flip-flop `U1-6A` al microcontrolador. Como ya se indicó en el apartado 5.1, finalmente este recurso no se usa desde el firmware, ya que si se obliga a que la única forma de escribir datos hacia el PC sea a través de la FIFO se asegura que la secuencia se mantiene íntegra y que no se insertan *bytes* en mitad del flujo de manera desordenada, lo que sí ocurriría si se escribiese directamente desde el microcontrolador hacia el chip USB usando la señal `/USB_FF_WR`. Por lo tanto, esta señal esta permanentemente en un estado lógico alto, por lo que no realiza acción alguna. Solo queda por añadir que la señal `/RESET` asegura que la máquina se inicia en el estado inicial (Q=1).

5.4. Interfaz USB

Para comunicarse con el PC a través del bus USB se ha decidido, por las razones que se explicaron en el apartado 3.3, utilizar un integrado USB dedicado separado del microcontrolador, el FT8U245BM. Este integrado contiene en su interior todo lo necesario para manejar el protocolo USB 2.0, en el modo de funcionamiento *full-speed* (12Mbps). En el lado del microcontrolador del radiorreceptor, esta interfaz permite acceder directamente a la FIFO de recepción y a la de envío, permitiendo abstraer toda la comunicación mediante USB a un flujo de *bytes* de forma transparente.

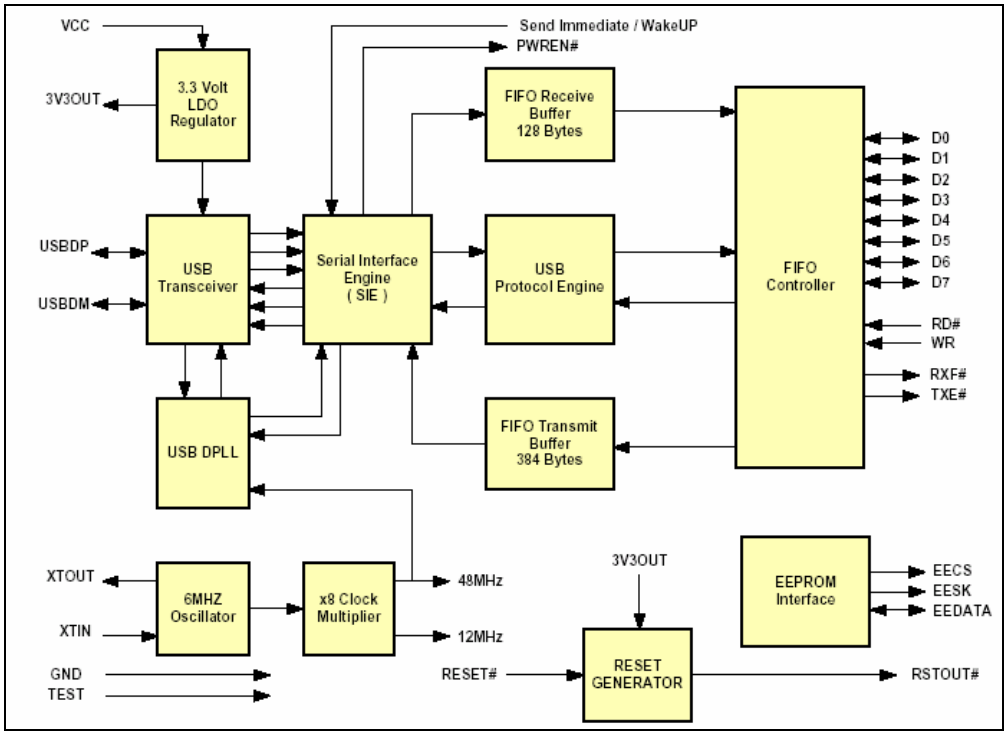


Figura 88: Esquema de la interfaz hardware USB.

Este integrado permite el uso de una memoria EEPROM serie externa para almacenar los datos de identificación del dispositivo, como su nombre, su VID (identificador de vendedor) y su PID (identificador de producto). Esta memoria se ha añadido en el radiorreceptor, y se ha programado a través de la misma interfaz USB usando una aplicación que el mismo fabricante facilita. Con esto se ha conseguido personalizar el nombre que el sistema operativo muestra cuando se conecta el sistema por primera vez.

Aparte de la conexión de esta memoria, sólo hay que conectar un cristal de 6 MHz y algunos componentes más como condensadores de desacoplo, quedando la conexión del integrado como se ve en este esquema.

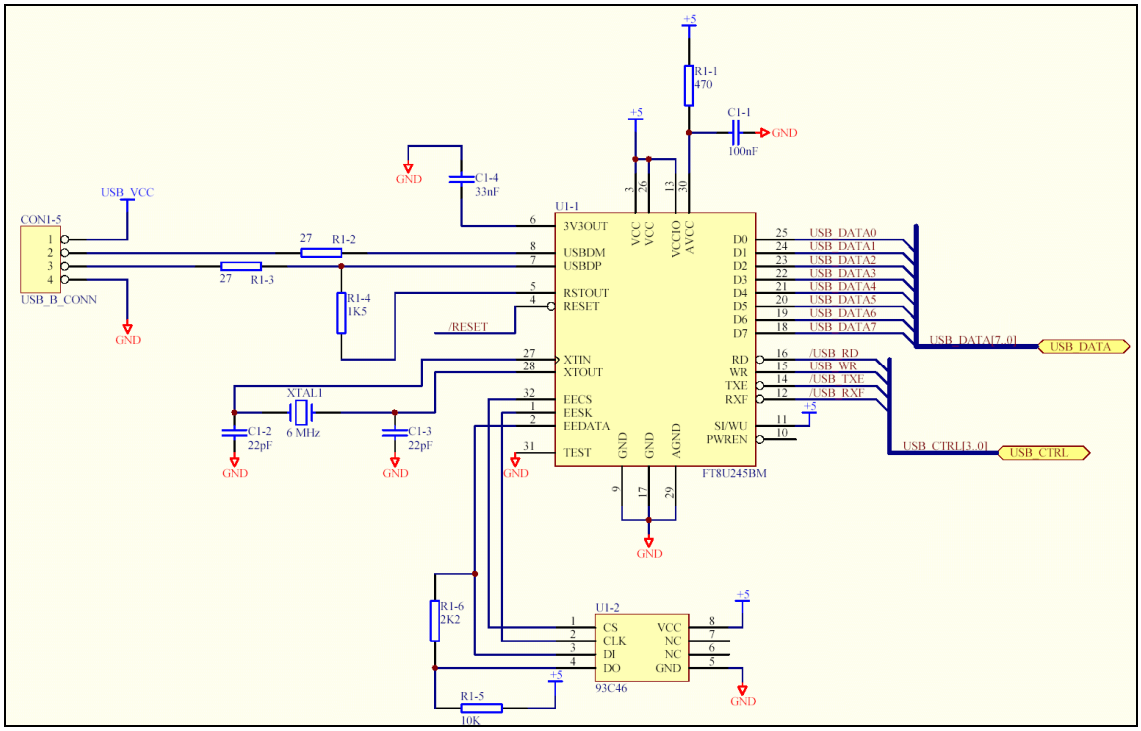


Figura 89: Conexiones de la interfaz USB.

La interfaz a las FIFO de envío y recepción de datos se realiza a través de un bus de ocho líneas de datos (`USB_DATA[0..7]`), controlada por las dos señales `/USB_RD` (activa a nivel bajo) y `USB_WR` (activa a nivel alto). Además, se dispone de dos salidas, `/USB_TXE` y `/USB_RXF`, que indican cuándo se pueden enviar datos al PC o leer desde éste, respectivamente.

En cuanto a la conexión de la memoria EEPROM externa, se ha seguido el esquema recomendado en el *datasheet*, que mediante divisores resistivos combina las señales de entrada y salida de la memoria en una sola señal de entrada-salida.

Esta memoria se programa desde el PC, usando la aplicación "*FTD2XXST*" que el fabricante distribuye de forma gratuita. El proceso de programación de esta EEPROM es como sigue:

- Con el receptor ya montado (o al menos toda la parte digital, incluyendo el chip USB, el microcontrolador, los osciladores a cristal, etc...) se conecta mediante el cable USB normal al PC. El chip USB se iniciará por primera vez, detectando la memoria EEPROM externa. Al verificar que esta no contiene datos validos (está vacía inicialmente), usará los valores por defecto de VID y PID, que son 0x0403 y 0x6001, respectivamente.
- En el PC aparecerá una ventana informando de un nuevo dispositivo, del que se necesitan los *drivers*. Hay que indicar que se usen los *drivers* de que el fabricante

dispone gratuitamente en su web, específicamente, los drivers *Direct Mode* para *FT8U245BM*. Estos *drivers* contienen ficheros *inf*, *dll* y *sys*. Es en los ficheros *inf* donde se especifican los VID y PID del fabricante que se han escrito arriba.

- Una vez instalados los *drivers*, se puede ejecutar la aplicación *FTD2XXST* para modificar la información contenida en la EEPROM externa. Es aquí donde se introduce el nombre del dispositivo "Radio Software..." que aparecerá a partir de ahora en el proceso de detección *plug-n-play* del receptor. Además, hay que introducir un par VID-PID único. Aunque oficialmente habría que solicitar a una autoridad encargada de esta tarea un VID propio, dado que se trata del desarrollo de un prototipo se ha usado el mismo VID que usa la compañía FTDI que es el 0x0403 y por PID se ha usado uno distinto, el 0x6099.

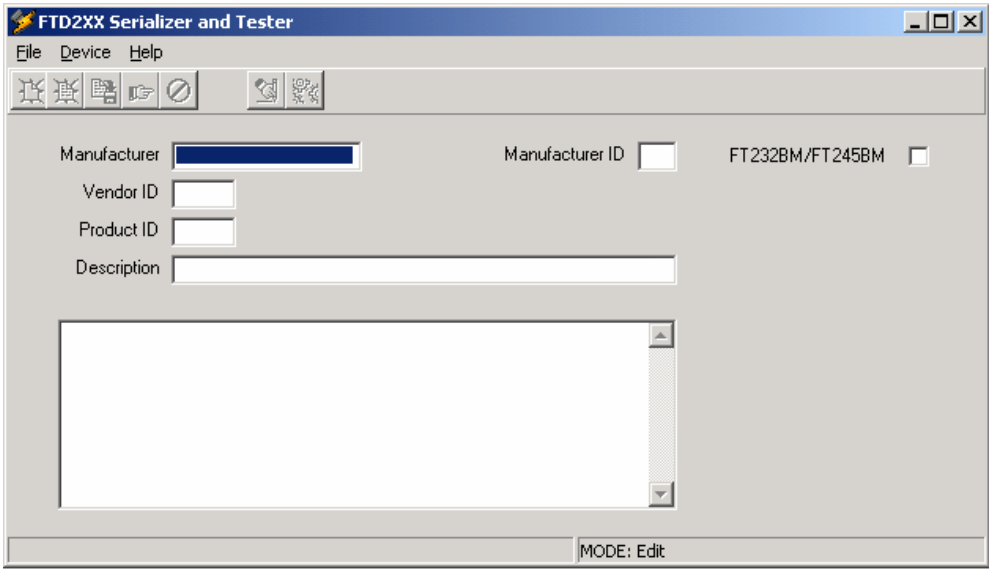


Figura 90: Aplicación usada para grabar la EEPROM externa al chip USB.

- Ahora hay que desconectar el cable USB del receptor del PC.

- Al volver a conectarlo aparecerá como un nuevo dispositivo, ya con el nombre personalizado, como se ve en la primera captura de abajo. Ahora los *drivers* usados anteriormente no son válidos (ya que el VID/PID no coincide) por lo que se usan los *drivers* adaptados al receptor, en los que se ha modificado a mano los ficheros *inf* para incluir el nuevo valor del PID. En la segunda captura se puede ver como aparece el receptor en el administrador de dispositivos de Windows:

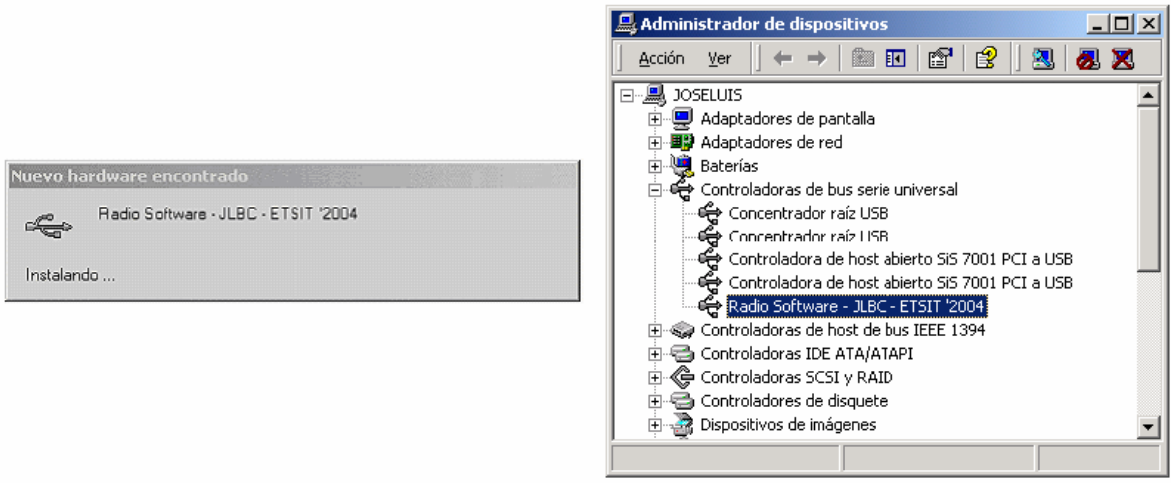


Figura 91: Detección *plug-n-play* del radiorreceptor USB.

Cuando se diseñó inicialmente este proyecto, se pensó en usar una frecuencia de muestreo de los canales I y Q de 500KHz. Esto estaba justificado por la tasa de transferencia máxima que soporta este chip USB que es de 1Mb/s, lo que permite transmitir dos canales de 500.000 muestras por segundo, para muestras de ocho bits como es el caso. Aunque las primeras pruebas con este chip mostraron la validez de este límite, a la hora de probar el funcionamiento de este chip dentro del sistema formado por la FIFO, el microcontrolador y la lógica discreta del volcado de la FIFO, se observó que se producían pérdidas de datos.

Después de varios experimentos, usando frecuencias de muestreo de 500KHz, 400Khz y 300Khz (y por tanto tasas de transmisión de 1Mb/s, 800Kb/s y 600Kb/s respectivamente), y frecuencias de funcionamiento de la máquina de estados de 16Mhz, 10MHz, 5Mhz y 2.5Mhz, se observó que en el mejor de los casos de todas estas combinaciones, siempre quedaba un margen de un 0.5% o menos de *bytes* perdidos que no se recibían en el PC. Este es el motivo por lo que se añadió el divisor por cuatro al reloj de la máquina de estados y se usó una frecuencia de muestreo de 250KHz, ya que con estas frecuencias se ha comprobado que parece no ocurrir nunca la pérdida de un solo *byte*. Se han realizado experimentos transmitiendo cientos de millones de *bytes* sin que se haya detectado una sola pérdida. Aún así, conviene señalar que si ocurriese una pérdida o cualquier otro error, el protocolo diseñado en el apartado 7.1 lo detectaría, por lo que nunca ocurrirá un error sin ser detectado. Hay que tener en cuenta también que los protocolos de las primeras capas USB se encargan de forma transparente de cualquier error en la transmisión física por el cable entre el receptor y el PC.

5.5. Conversores A/D (ADC)

Para la elección del conversor se manejaron más de una docena de opciones, y la decisión se tomó principalmente sobre estas características:

- Salida digital serie o en paralelo: Se decidió usar salida en paralelo para poder obtener una máxima velocidad de digitalización con los mínimos recursos (como registros de desplazamiento, tiempo de proceso en el microcontrolador, etc.)
- Un conversor con múltiples entradas frente a varios conversores. Como en la mayoría de conversores múltiples hay que seleccionar primero el canal y después realizar la conversión, esto no es admisible para el caso de digitalización de señales I y Q, ya que supondría muestrear en instantes distintos. Por esto se han usado dos conversores distintos.

El conversor elegido es el AD7822, un conversor de ocho bits, del tipo *flash* (los más rápidos), con un rango de voltajes de entrada ajustable y salida digital en paralelo:

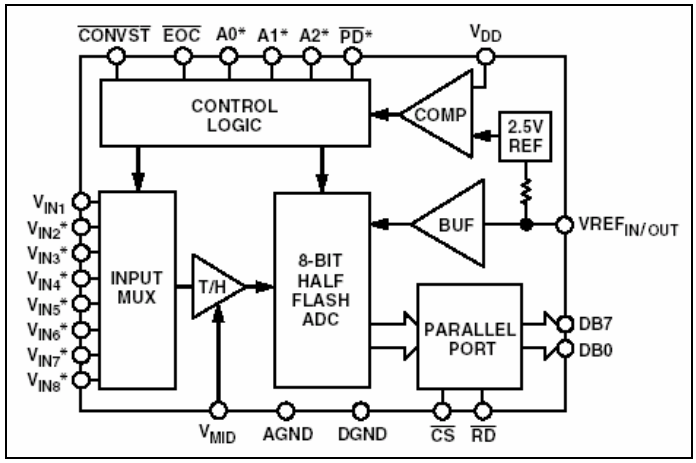


Figura 92: Diagrama interno del conversor analógico digital.

El rango de entrada es de 2.5Vpp, centrado en un valor ajustable externamente. Aprovechando la referencia de voltaje interna de que dispone el integrado, se ha usado esta para centrar este rango en 1.25V, para lo cual el terminal `VMID` debe ser dejado sin conectar. Los terminales `/CONVST`, y `/CS` y `/RD` sirven para iniciar la conversión y leer los resultados de la conversión, respectivamente. Los cronogramas mostrando la temporización de estas señales se muestran en el apartado 5.3. La conexión del conversor correspondiente a la señal I queda por tanto así:

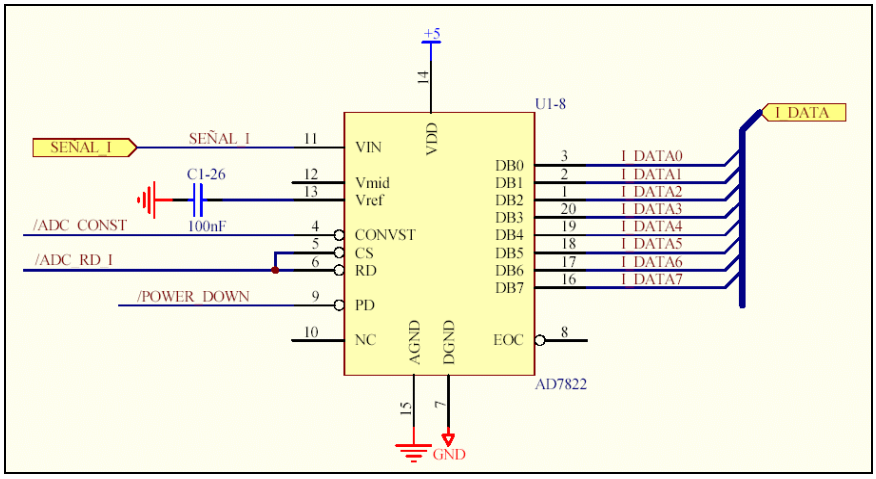


Figura 93: Conexión del conversor analógico digital.

Siendo idéntica la conexión del conversor para la señal Q, excepto que la señal de lectura es `/ADC_RD_Q` en lugar de `/ADC_RD_I`, y que evidentemente, la entrada analógica es la señal Q en lugar de la I.

El uso de la referencia interna de 1.25V como valor medio para el rango de entrada viene del hecho de que las señales I y Q provenientes de los amplificadores de banda base del demodulador estén centrados en 1.0V. Ahora se puede exponer el proceso de diseño de los rangos de las señales en banda base, desde el conversor hacia atrás hasta el demodulador. Se diseña hacia atrás ya que el objetivo es usar el rango de entrada del ADC lo máximo posible para mejorar la relación señal ruido de la señal digitalizada y no desperdiciar gran parte del rango de posibles valores discretos a la salida del conversor. El proceso se muestra gráficamente a continuación:

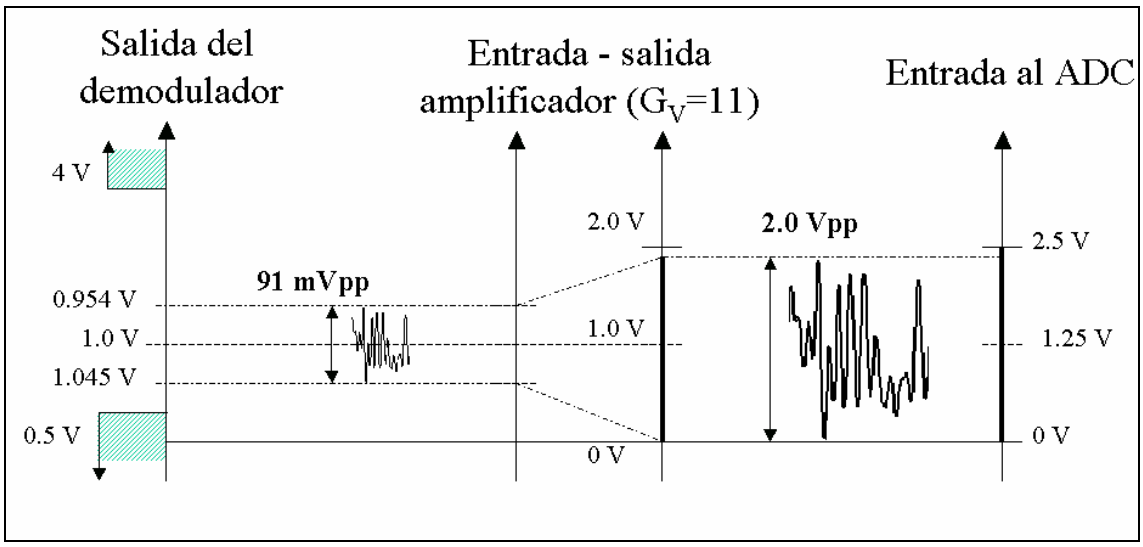


Figura 94: Evolución de la señal en banda base en las diferentes etapas.

Empezando por el final, el rango de entrada al conversor se puede ver que es el que antes se mencionó: 2.5Vpp centrados en 1.25V. El amplificador de banda base fija el nivel medio de la señal según la referencia que se le da, que proviene del regulador lineal descrito antes. Como interesa aprovechar el rango de entrada del ADC, lo ideal sería que la señal estuviese también centrada en 1.25V, pero al aprovechar el regulador ya disponible de 1.0V en el demodulador, el rango útil se reduce a 2.0Vpp centrados en 1.0V (es decir, un rango entre 0V y 2.0V).Al trasladar este rango a la entrada del amplificador solamente hay que tener en cuenta la amplitud del rango, ya que el nivel de continua es modificado (en este caso sin embargo, tanto la entrada como la salida tienen un *offset* de 1.0V). Así, tenemos un rango a la entrada de 2.0Vpp / 11 (donde 11 es la ganancia de voltaje) resultando 182mV para la señal diferencial, lo que se traduce en un rango de 91mVpp para cada una de las dos señales que la forman. Como las salidas diferenciales del demodulador tienen una componente continua de 1.0V, esto quiere decir que la salida del demodulador varía entre 1.045V y 0.954V. Este rango está dentro del rango permitido, que según las hojas de características es desde 0.5V a 4V, y es de tan pequeña amplitud que la linealidad está asegurada.

El evitar la saturación de la señal a la salida del demodulador y asegurar el funcionamiento en la zona lineal es la principal motivación de haber añadido esta etapa extra de ganancia en las señales de banda base.

5.6. Conversores D/A (DAC)

Durante el diseño del receptor se ha encontrado la necesidad de cuatro señales de control analógicas:

- El voltaje de control de ganancia para el VGA de radiofrecuencia.
- El voltaje de control de ganancia del demodulador en cuadratura.
- Las dos señales de corrección de offset para las señales I y Q del demodulador. Como ya se comentó, esta fue la opcion inicial en el desarrollo del proyecto, mientras que finalmente se optó por desconectar estas salidas del conversor DAC al demodulador y la sustitución por el sencillo filtro paso alto que se mostró en el apartado 4.5, por los motivos que también allí se describían.

En todos estos casos, se ha estimado que una resolución de ocho bits es más que suficiente, ya que como se mostró en los apartados correspondientes esta resolución permite controlar la ganancia del VGA y del demodulador en pasos de aproximadamente 1dB, lo que se ha considerado precisión suficiente.
Como integrado para los conversores DAC se ha elegido el AD7304. Este chip se ha elegido por contener en un solo integrado los cuatro conversores, tener ocho bits de resolución, un rango de voltaje de salida entre masa y el voltaje de alimentación (algo que no todos los DAC consiguen) y por controlarse mediante un bus SPI. Este es su diagrama interno:

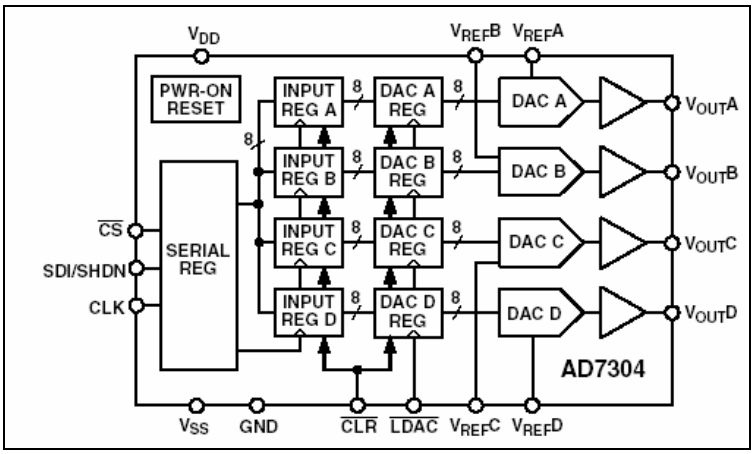


Figura 95: Esquema interno del DAC.

Las referencias de voltaje para todos los conversores se han unido a la tensión de alimentación de cinco voltios. Así se consigue un voltaje de salida en todo el rango 0V-

5V y no es necesario un integrado para generar una referencia de voltaje externa. El chip dispone también de una señal que permite actualizar todos los conversores a la vez, característica que no se ha usado en este proyecto, por lo que la señal `/LDAC` se ha conectado permanentemente a masa. En cuanto al bus SPI, este es el mismo que el usado por el PLL y comparte con este las señales de datos y reloj, mientras que la señal de habilitación es independiente como es lógico. Con todo esto, las conexiones a este chip quedan como se ve en el esquema de abajo.

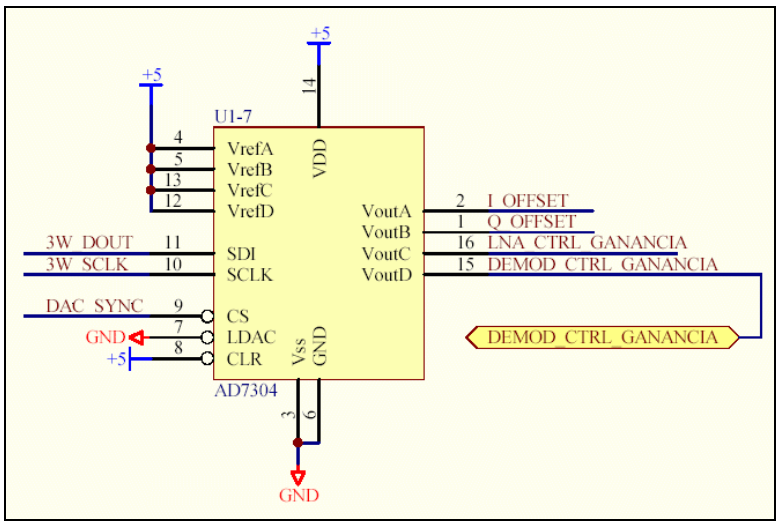


Figura 96: Conexión del DAC.

5.7. Display LCD

La mayoría de los *display* LCD de texto son compatibles con el estándar *de facto* HD44780 de Hitachi. La interfaz de estos LCD esta formado por 14 señales, como se ve en la siguiente figura:

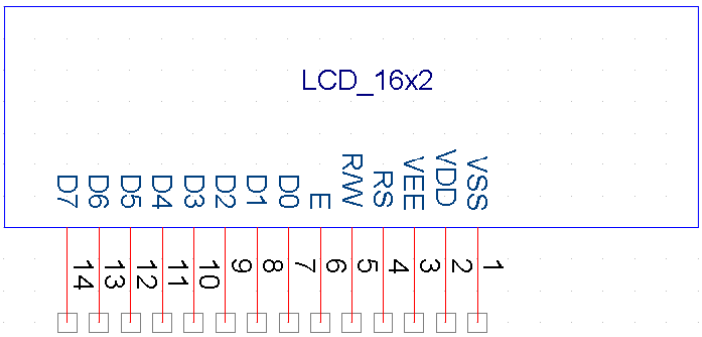


Figura 97: Interfaz estándar de los LCD

Donde las señales tienen este uso:

- V_{DD}, V_{SS}: Alimentación y masa respectivamente.
- V_{EE}: Entrada de ajuste del contraste.
- `D[0..7]`: Es un bus de ocho bits bidireccional.
- `RS` (*Register selection*): Indica si el dato enviado es un carácter(1) o un comando(0).
- `R/W` (*Read/Write*): Indica una operación de lectura (1) o de escritura (0).
- `E` (*Enable*): Señala que en el bus hay datos validos. Señal activa a nivel alto.

Aunque existen LCD de diversos tamaños (1, 2 o 4 líneas, 8, 16, 20 o 40 caracteres por línea) todos aceptan el mismo protocolo y conjunto de comandos. Además, todos soportan los dos modos de comunicación que existen: a bus completo y a medio bus (4 bits). En el caso de este proyecto, se usa un LCD de 16 caracteres y 2 líneas, y el modo de comunicación es el de bus a cuatro bits, por requerir un menor número de conexiones. El protocolo usado para leer y escribir en estos *displays* mediante cuatro líneas de datos se ilustra con ejemplos en la siguiente figura:

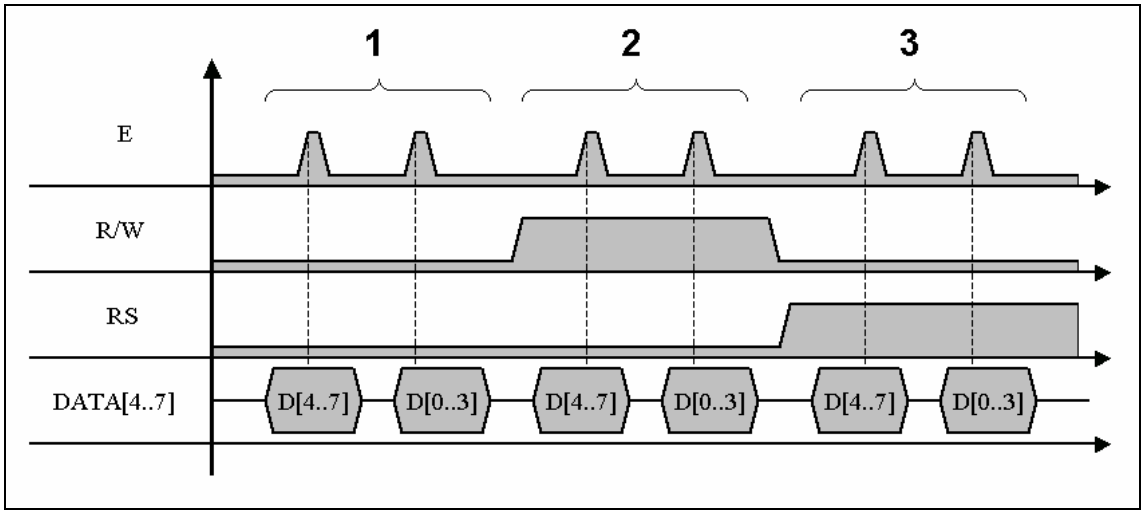


Figura 98: Cronograma mostrando el protocolo del display LCD

En el cronograma se ilustran tres transacciones distintas. La primera de ellas es el envío (`R/W`=0) de un comando (`RS`=0). El *byte* codificando el comando se manda colocando primero en el bus el *nibble* superior, dando un pulso a `E` y repitiendo la operación para el *nibble* bajo. La segunda transacción es una lectura (`R/W`=1) del estado del display (único registro que se puede leer del LCD). Como se muestra en la tabla de comandos más abajo, el bit más significativo de este registro indica si el LCD está ocupado con la ejecución del último comando. Este tiempo de ejecución puede variar entre unos 40μs y algunos milisegundos, según el comando y el fabricante del display. Si se envía algún

otro comando o dato mientras el LCD esté ocupado, éste se pierde. En el tercer caso del cronograma se ve el envío (R/W=0) de un carácter (RS=1). Cuando se manda un carácter al display se escribe en la posición del cursor. A continuación se detallan en forma de tabla todas las operaciones que se pueden realizar sobre el *display* y que señales hay que activar para ejecutarlas:

OPERACIÓN	**SEÑALES**									
	RS	**R/W**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Borrar display.	0	0	0	0	0	0	0	0	0	1
Cursor a posición inicial.	0	0	0	0	0	0	0	0	1	X
I/D:Especifica dirección de entrada de texto: 0=IZQ, 1=DER. S: Realizar scroll automático de display	0	0	0	0	0	0	0	1	I/D	S
D: Display activado (1) o desactivado (0) C: Mostrar (1) o no (0) el cursor B: Cursor parpadeando (1) o fijo (0)	0	0	0	0	0	0	1	D	C	B
S/C: Scroll(1) o mover cursor(0) al introducir texto R/L: Dirección scroll 1=DER, 0=IZQ	0	0	0	0	0	1	S/C	R/L	X	X
DL: 1= Modo 8 bits, 0= Modo 4 bits N: 0= Una línea, 1= Dos líneas	0	0	0	0	1	DL	N	F	X	X
Indica dirección a escribir en la CGRAM	0	0	0	1	A	A	A	A	A	A
Indica dirección a escribir en la DDRAM	0	0	1	A	A	A	A	A	A	A
Leer estado de display: BF=1 ocupado, BF=0, libre. Devuelve también la dirección actual en CG/DDRAM	0	1	BF	A	A	A	A	A	A	A
Escribe carácter en posición actual del cursor (D)	1	0	D	D	D	D	D	D	D	D
Leer carácter en posición del cursor (D)	1	1	D	D	D	D	D	D	D	D

Estos LCD disponen en su interior de un controlador que procesa todos estos comandos además de realizar ciertas tareas automáticamente, como incrementar la posición del cursor al escribir un carácter, parpadeo automático del cursor, etc... Es éste controlador el que realmente controla la gran cantidad de líneas necesarias para mostrar en la pantalla cada uno de los píxeles. Los *displays* contienen dos zonas de memoria:

- DDRAM (*Display Data RAM*): Guarda el carácter a mostrar en cada posición del *display*.
- CGRAM (*Carácter Generator RAM*): Usados para almacenar hasta 8 caracteres definidos por el usuario. La resolución es de 5x7 píxeles por carácter.

Donde esta segunda no se ha usado en este proyecto, ya que los caracteres estándares, que se muestran a continuación, son suficientes para el propósito de esta pantalla en el receptor. La única limitación es que no contiene algunos caracteres especiales del español, pero esto no representa un problema.

Figura 99: Conjunto de caracteres estándar para un LCD

En cuanto a la memoria DDRAM es importante saber qué direcciones se corresponden con el área visible, ya que todos los *displays* tienen la misma memoria independientemente del tamaño real de la pantalla. En el caso de un display de 16x2 caracteres, las direcciones son:

- Línea 1: Posiciones 0x00 a 0x0F
- Línea 2: Posiciones 0x40 a 0x4F

Esto quiere decir que cuando se llena una línea, si se siguen enviando caracteres estos no aparecerán en la siguiente línea, sino que esto se tiene que realizar por software. Estas direcciones de DDRAM se usan en el comando de posicionamiento del cursor.

En el caso concreto de este proyecto, se han realizado estas conexiones:

- La señal `R/W` se ha soldado directamente a masa, señalando siempre una operación de escritura. La señal `RS` indicará si es una escritura de carácter o un comando. Esto elimina una señal de control desde el microcontrolador.
- Para evitar la colocación de un potenciómetro para regular el contraste, se ha conectado la entrada V_{EE} a masa, soldando un puente que a este propósito existe en el módulo. Con esto se consigue el contraste máximo de forma permanente, lo que permite una visualización aceptable en la mayoría de direcciones.
- Las cuatro líneas de datos no van conectadas directamente al microcontrolador. Ante la falta de especificaciones del fabricante sobre el tipo de familia lógica de las entradas del bus de datos, y dado que estas líneas se comparten también por el chip USB, cuyo uso intensivo sin problemas debe estar garantizado, se han aislado estas líneas a través de un buffer 74HC244, como se ve en el siguiente esquema:

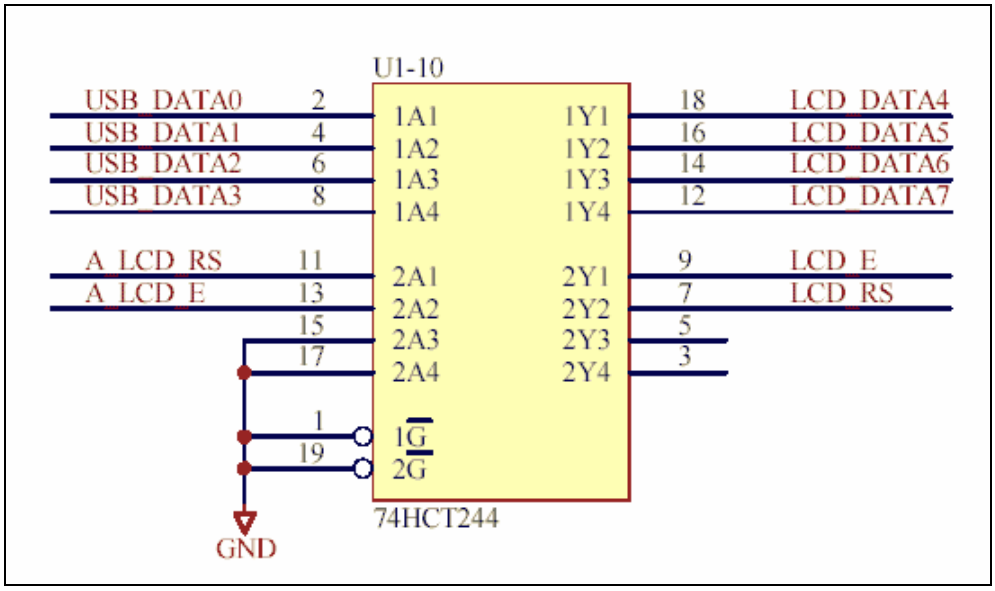


Figura 100: Buffer utilizado para las señales del LCD

Con todo esto se consigue reducir el espacio en placa para el conector del LCD, que de los 16 conectores que tiene, se reduce a un conector de ocho señales, que se conecta a

un cable plano que va soldado al módulo del *display*. A continuación se muestra la conexión de este conector reducido.

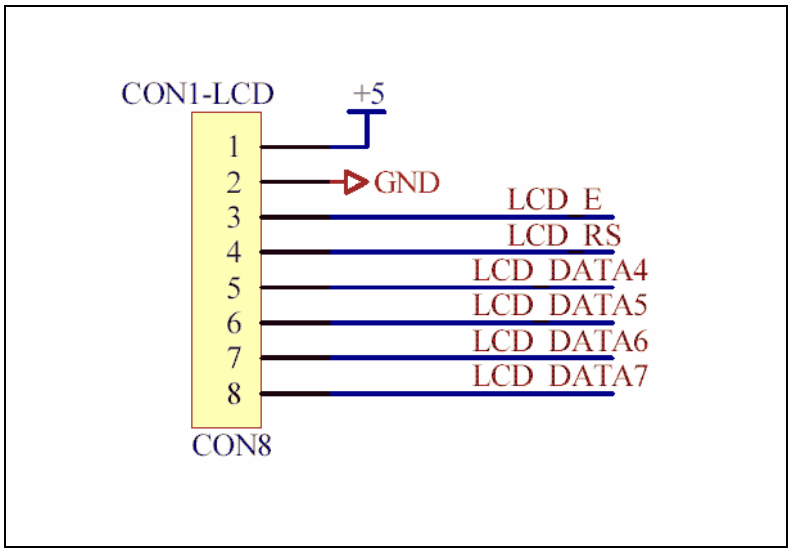


Figura 101: Conector del LCD en la placa principal

6. DISEÑO FÍSICO Y MECÁNICO

6.1. Diseño

Ya se dijo en el capítulo 3.3 que el receptor se ha dividido en distintas placas PCB por motivos de modularidad y facilidad de rediseño durante el desarrollo del proyecto. En la siguiente figura se muestra el esquema conceptual del receptor que ya se mostró en el apartado 3.2, pero ahora señalando qué subsistemas entran en cada una de las placas:

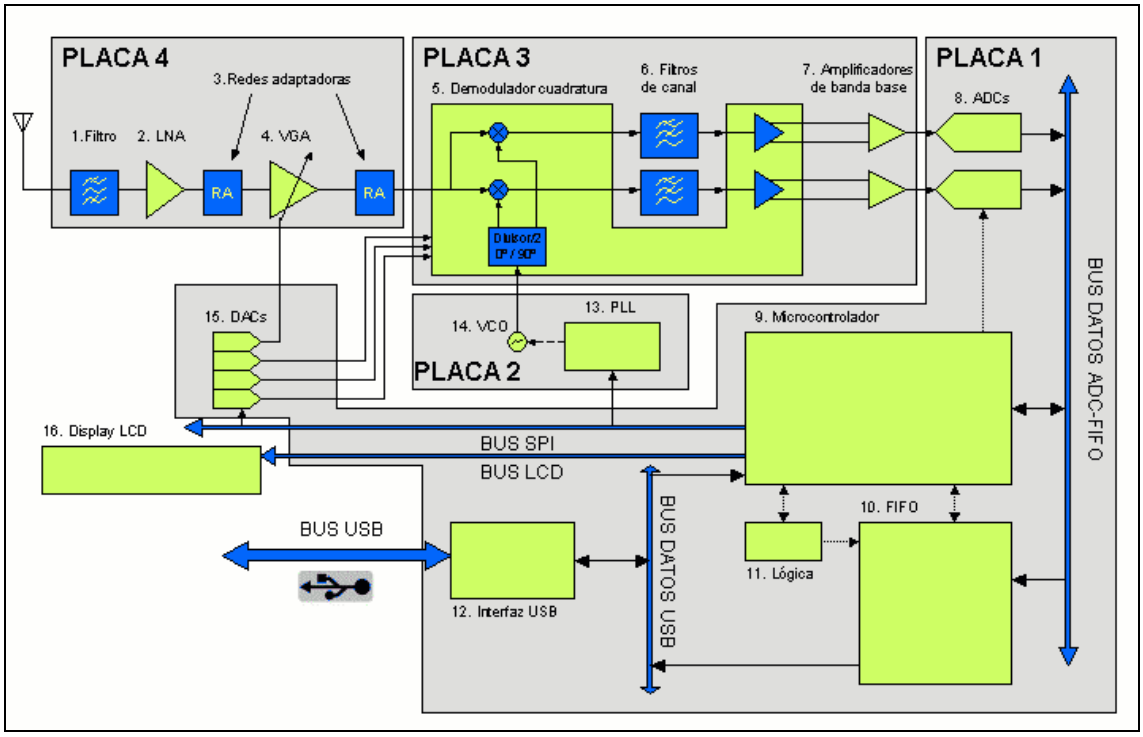


Figura 102: División del receptor en distintas placas

Donde las cuatro placas construidas contienen:

- Placa 1: Es la placa principal, donde van el microcontrolador, la interfaz USB, la memoria FIFO con su lógica asociada, los conversores analógico-digitales y los digital-analógicos.
- Placa 2: Aquí se incluye el oscilador VCO y el PLL.
- Placa 3: Esta placa contiene el demodulador, los filtros en banda base y los conversores de diferencial a unipolar para las señales I y Q.
- Placa 4: Es la placa que contiene el conector de entrada de radiofrecuencia, el filtro de preselección, el LNA y el VGA.

Las conexiones del receptor hacia el exterior son solamente dos:

- El conector de entrada de radiofrecuencia: Este conector está en la placa del LNA, y consiste en un conector hembra tipo F, como se ve en la segunda foto de abajo. Va soldado directamente a la placa.
- El conector USB: Es un conector del tipo USB-B hembra ([3]) para soldar en PCB a través de cuatro patillas. Está en la placa principal, y su esquema se corresponde con la tercera imagen de abajo.

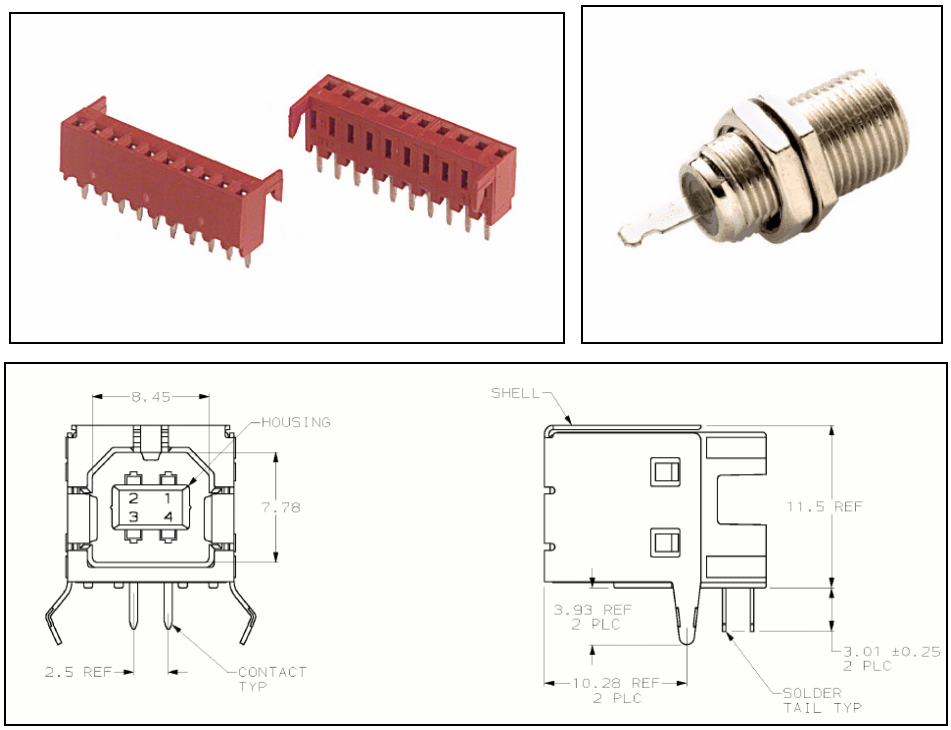


Figura 103: (a)Conectores hembra para conexiones con la placa principal. (b)El conector "F". (c)El conector USB tipo B hembra.

En cuanto a las conexiones entre las tres placas analógicas (la 2, 3 y 4) con la placa principal, estas se han realizado mediante unos pines soldados en la parte inferior de las placas analógicas que entran en los receptáculos hembras como el mostrado en la primera foto de la figura anterior. A continuación se muestra una reconstrucción 3D de las cuatro placas, mostrando dónde van colocadas cada una físicamente sobre la placa principal:

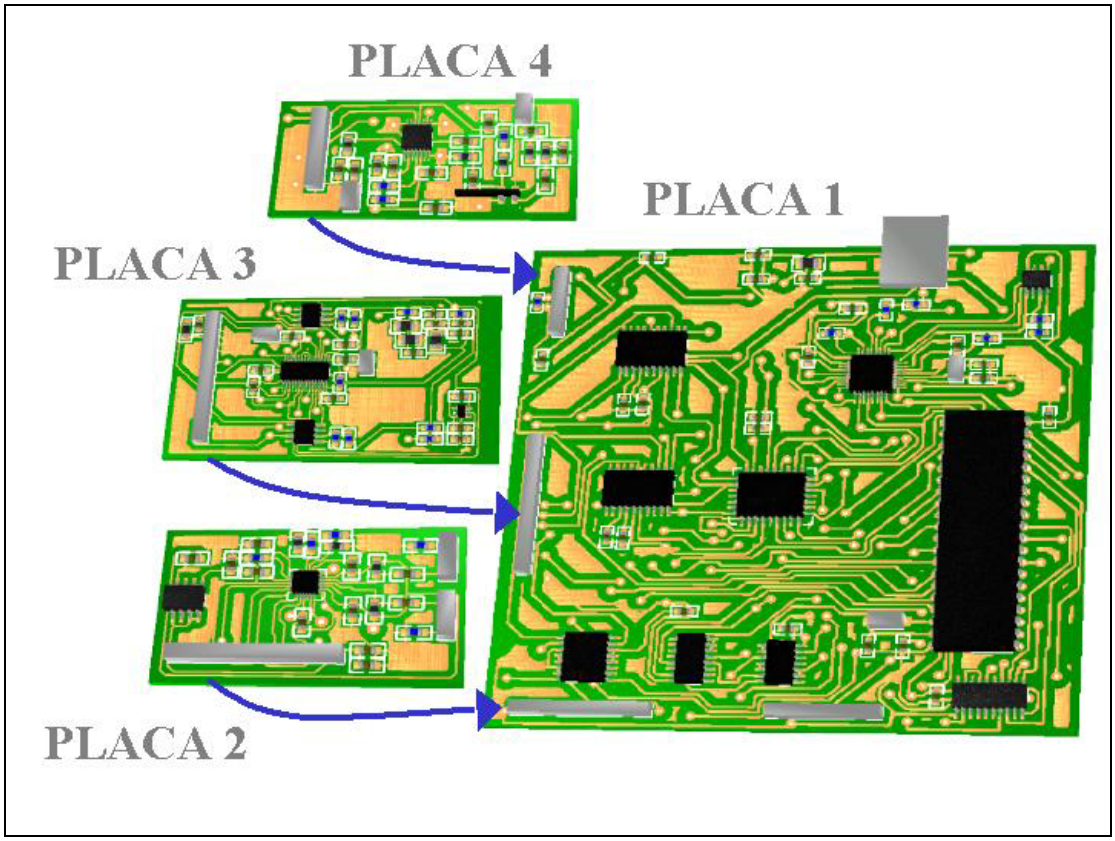


Figura 104: Esquema de conexión de las cuatro placas del receptor.

De los dos conectores que se pueden apreciar en la parte inferior de la placa principal (aunque en la imagen 3D aparecen como un bloque, en realidad son receptáculos hembras para tiras de pines), el de la derecha es el reservado para la conexión del módulo LCD, que no se representa en esta imagen.

En cuanto al empaquetado de todo el receptor en una caja, las placas PCB se dimensionaron teniendo en cuenta la elección de un modelo de caja en concreto. Esta es el modelo A-61 de la empresa SERPAC. Dispone de dos tapas en cada extremo, que se han usado para empotrar el LCD atornillándolo (tapa delantera) y para los conectores de RF y USB (tapa trasera). Estos últimos están soldados a las placas PCB y en las tapas simplemente se han hecho los agujeros del tamaño y forma apropiadas. En la siguiente figura se muestra el estado original de esta caja:

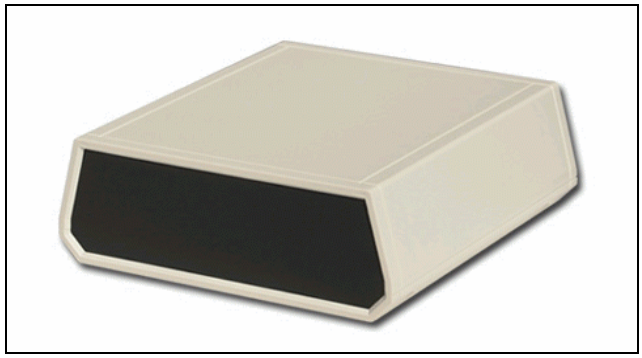


Figura 105: Foto de la caja A-61 en su estado original.

Para fijar correctamente las placas al interior de la caja se tuvo en cuenta la localización de cuatro tornillos de sujeción de la parte inferior de la caja en el diseño del PCB de la placa principal, como se puede ver en el anexo correspondiente a los fotolitos de las placas. A continuación se muestran unas imágenes que representan el estado final en que han quedado las distintas placas y el radioreceptor completo. Para empezar se muestra la placa 1, la placa principal donde está todo el hardware digital, aislada y antes de atornillarla a la caja:

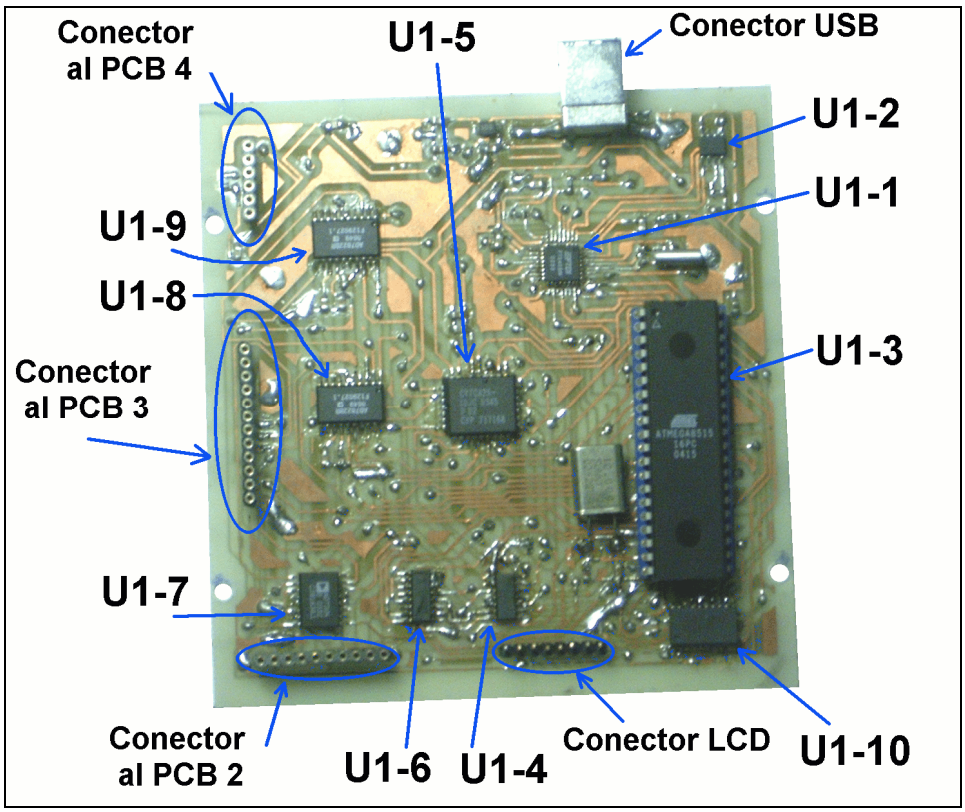


Figura 106: Foto de la placa principal antes de conectar las otras placas.

En la imagen anterior se han señalado cada uno de los integrados que componen el hardware digital del radioreceptor, así como los conectores, que son:

- Los tres conectores (tiras de *pins* hembra) que serán los puntos de inserción para las otras tres placas analógicas del sistema.
- El conector (tira de *pins* macho) para la pantalla LCD. En este conector se introducirá un cabezal hembra de un cable tipo "cables planos", del tipo usado en los discos duros ATA, pero de menor cantidad de señales.
- El conector USB (del tipo USB-B). El tipo de conector se ha escogido para poder conectarse fácilmente a cualquier PC usando los usuales cables tipo A-B, también conocidos como cables para impresoras USB. Este conector es uno de los dos disponibles externamente del receptor hacia el exterior, siendo el otro conector el de la entrada de RF.

En la foto anterior también se puede apreciar los cuatro puntos de anclaje con tornillos que se tuvieron en cuenta en el diseño del PCB. En la siguiente página se muestra esta placa principal ya montada sobre la base de la caja, atornillada a esta y completamente conectada al resto de circuitos. De izquierda a derecha se puede ver:

- La pantalla LCD. Como se vio anteriormente el conector estándar de los displays LCD es una tira de 16 pads, pero se han conectado de forma que solamente son necesarios ocho cables hacia la placa principal. Para estos ocho cables se ha usado un cable plano como se ve en la foto, que va puede conectar y desconectar fácilmente a la tira de pines macho de la placa principal ya que va terminado con un cabezal hembra estándar para este tipo de cables. El LCD va sujeto a la tapa anterior de la caja usando cuatro tornillos usando los agujeros que a este efecto traen los módulos LCD.
- La placa 2 (PLL): Además de la conexión en vertical a través de los pins a la placa principal, se ha usado un pequeño trozo de cable apantallado para llevar la señal del oscilador local hasta la placa del demodulador.
- La placa 3 (Demodulador): Además de la conexión en vertical, hasta esta placa llegan dos cables apantallados de las placas de entrada de RF y del PLL, respectivamente.
- La placa 4 (Entrada RF): Además de la conexión en vertical, dispone de un conector tipo F para la entrada de la señal de antena y un cable coaxial de salida hasta la placa del demodulador.

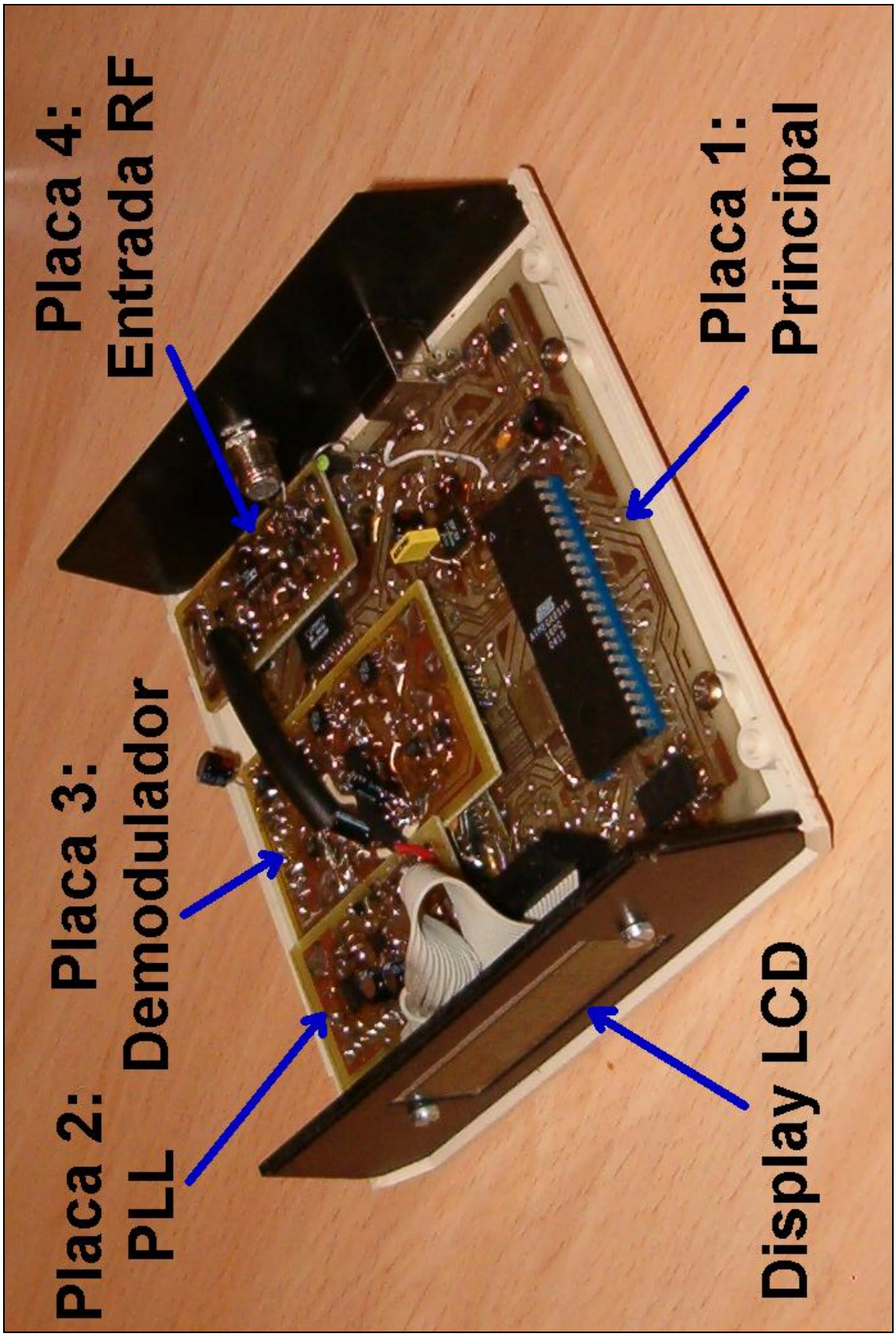


Figura 107: Foto de todas las partes del receptor y sus conexiones.

Hay que justificar el uso del cable apantallado para transportar las señales de alta frecuencia, ya que se podría haber usado un cable coaxial en lugar del apantallado. Se ha escogido dejar un cable apantallado por sus ventajas de flexibilidad y más pequeño grosor que los cables coaxiales.

El uso de cables coaxiales es obligado para el transporte de señales de RF de alta frecuencia a distancias relativamente largas. En cuanto a las frecuencias sí que representan un problema, ya que la señal de RF estará en torno a 100Mhz y la del oscilador local en torno a 200Mhz. A estas frecuencias no se puede permitir una conexión con cables aislados ya que se producirían pérdidas por radiación. En cuanto a las distancias de las conexiones, que son de aproximadamente unos 4cm, estas deben medirse respecto a la longitud de onda de la señal transportada, que en el peor caso de la señal de 200Mhz es de:

$$\lambda = \frac{c}{f} = \frac{3 \cdot 10^8 \, m/s}{200MHz} = 1.5m$$

Es decir, que la conexión representa menos del 3% de una longitud de onda, no siendo necesaria la caracterización de estas conexiones como guías de ondas. Por este motivo se ha considerado finalmente apropiado el uso del cable apantallado frente al coaxial.

Con respecto a la anterior imagen, ya solamente quedan por añadir la tapa posterior y la parte superior de la caja para tener el receptor totalmente montado y cerrado, como se ve en la primera imagen de la página siguiente. Como se puede ver se ha conseguido un uso óptimo del espacio, sobre todo gracias al uso de dos niveles de placas de circuitos, consiguiendo un tamaño total bastante pequeño, sobre todo teniendo en cuenta que solamente es un prototipo construido a mano. Así mismo se puede ver la sencillez que presenta el sistema hacia el exterior, ya que solamente se presenta la pantalla LCD por la parte anterior y los dos conectores (entrada RF y USB) en la parte posterior, como se muestra en la segunda imagen de la siguiente página.

En cuanto al diseño físico del sistema solo queda por añadir un pequeño detalle que se ha añadido por comodidad y seguridad, que son cuatro pequeñas piezas de goma que se han añadido en la parte inferior de la caja a forma de patas, que evitan el deslizamiento sobre superficies planas.

Figura 108: Foto del receptor totalmente montado y cerrado.

Figura 109: Foto de la parte trasera del receptor.

6.2. Herramientas utilizadas

Para el diseño de las placas PCB se ha utilizado la aplicación de diseño electrónico "*Protel Design Explorer 99*", que incluye tanto editores de esquemas eléctricos como de circuitos físicos PCB. Para el diseño del receptor los esquemas eléctricos se han creado de manera jerárquica, diseñando un solo fichero "*sch*" por placa, y dentro de éste

referencias a un número variable de otros esquemas. Esto es necesario porque la complejidad de los circuitos hace imposible el uso de una sola página por placa. Por ejemplo, para la placa principal se han usado seis páginas.

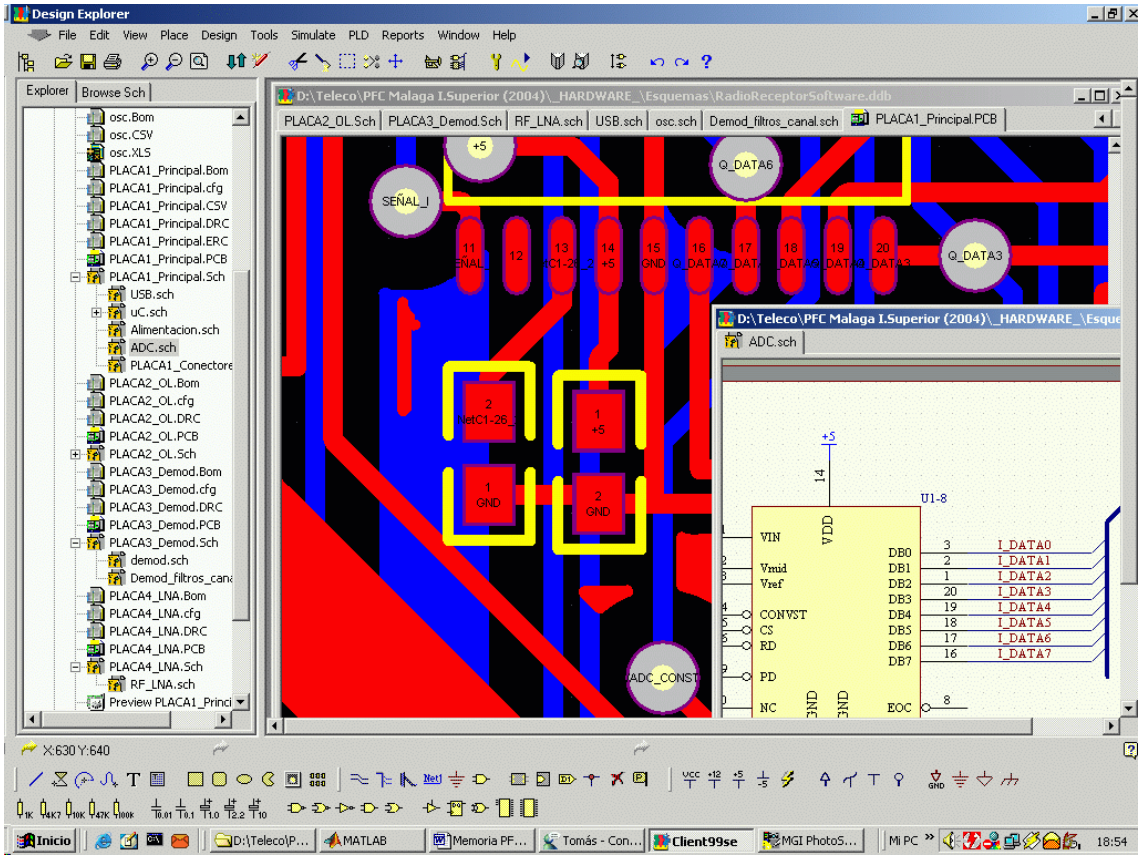


Figura 110: Imagen de la aplicación usada para la edición de los esquemas.

Entre las herramientas más útiles para este proyecto que se han usado están las de enrutado automático de las señales en pistas PCB, usando la información de los esquemas eléctricos y la comprobación automatizada y la aplicación de reglas de diseño para los PCB, como las distancias entre pistas o sus grosores máximos y mínimos.

Hay que señalar que al igual que con cualquier circuito no trivial, ha sido necesario el uso de dos niveles de pistas: por arriba y debajo de la placa. Las conexiones entre las dos capas se implementan a través de *vias*, o agujeros en la placa para enrutar una señal entre los dos niveles. Aunque este proceso se automatiza fácilmente en placas construidas profesionalmente, en este caso al tratarse de un prototipo se han usado pequeños conductores soldados a ambos lados a través de la *via*.

Otra herramienta muy útil ha sido la creación de polígonos de auto-rellenado de las capas asociadas a la señal de masa, lo que se ha usado para crear lo más parecido a un plano de masa en todas las áreas donde esto ha sido posible en todas las placas.

7. DISEÑO SOFTWARE

7.1. Diseño del protocolo de comunicaciones

7.1.1. Introducción

Hasta este punto de la memoria solamente se ha tratado del diseño del hardware del receptor y de las comunicaciones con el PC solamente se ha especificado que se usa el interfaz USB, a través de un integrado de la compañía FTDI que nos oculta todo el protocolo USB y ofrece de forma transparente un canal de datos bidireccional. Sin embargo en el lado del PC estos datos deben pasar por una serie de capas software hasta llegar a la aplicación principal (*SWRadioStudio*), como se muestra esquemáticamente en la figura:

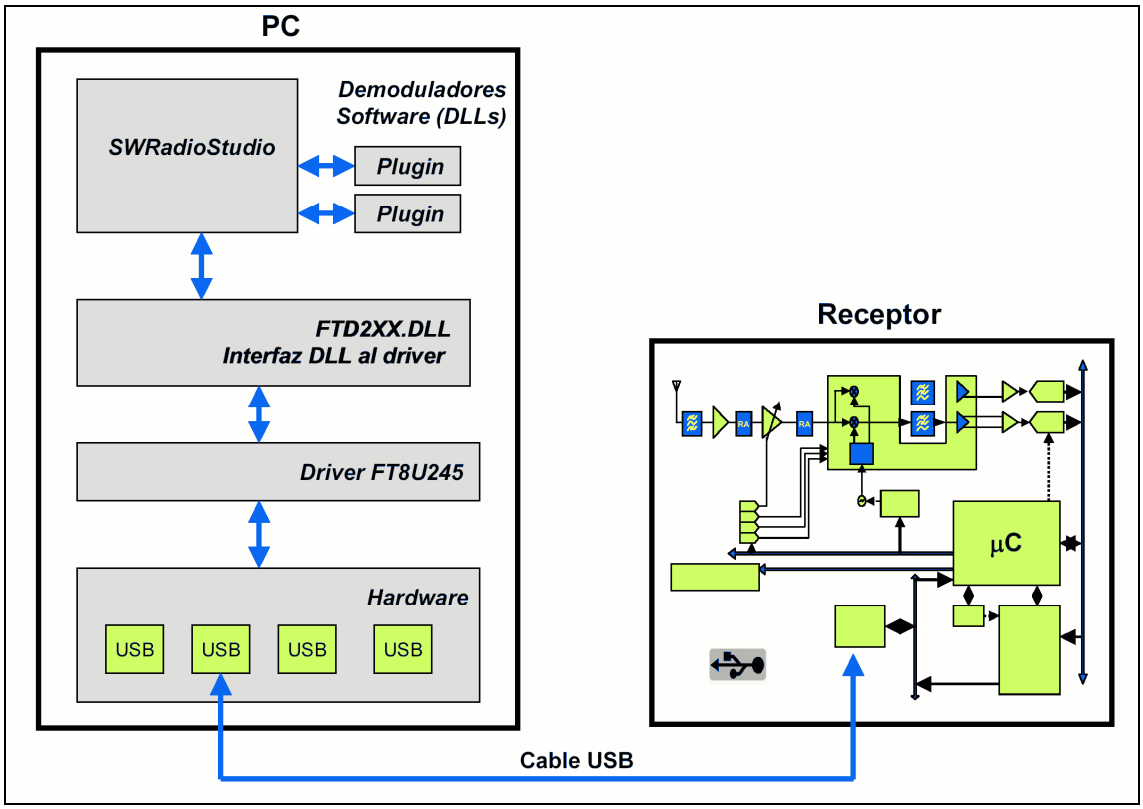


Figura 111:Esquema global del conjunto receptor-PC

Este esquema se explicará detalladamente más adelante, pero es evidente la necesidad de establecer una serie de reglas y un lenguaje común (protocolo) que asegure el

entendimiento del receptor con el software en el PC, cuyo objetivo es tanto prevenir posibles errores como normalizar el conjunto de datos que se pueden intercambiar.

Para definir el protocolo se usará un modelo de capas de comunicaciones, donde cada entidad a un nivel dado solamente se comunica físicamente con el elemento inmediatamente inferior, pero la comunicación lógica se produce con su pareja al mismo nivel del otro extremo.

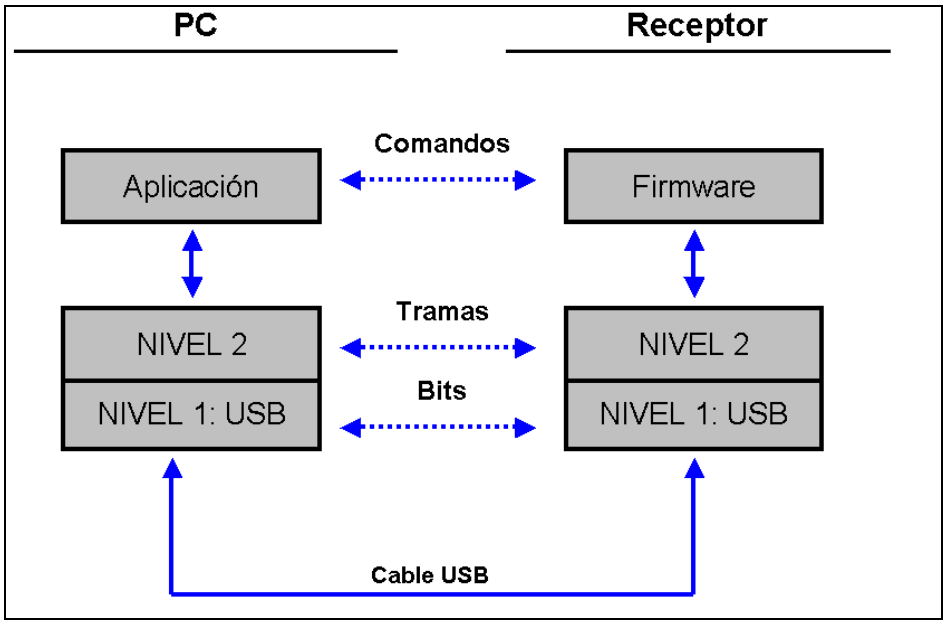


Figura 112:Los distintos niveles de la comunicación

En este caso solamente existen tres capas, que son:

- Nivel 1: Es el nivel físico y se implementa mediante el protocolo USB. Aunque este protocolo en sí dispone de varias capas, en este proyecto se tomará como si se tratara de un medio por el que se transmiten datos sin tener en cuenta sus capas internas ya que se usan de manera transparente.
- Nivel 2: Es el nivel de enlace. En esta capa se define el formato de las tramas que deben usarse para encapsular información entre los dos extremos. Sus objetivos son la sincronización de *byte* y la detección de errores.
- Nivel 3: Es el nivel de aplicación, donde en cada extremo las aplicaciones de usuario (el *firmware* en el radioreceptor y *SWRadioStudio* en el PC) utilizarán las primitivas de la capa de enlace para definir un lenguaje que define los distintos tipos de comandos y respuestas en ambos sentidos.

7.1.2. Definición del nivel 2: Capa de enlace

Los objetivos de esta capa son proveer un medio de sincronización entre los dos extremos y la detección de errores. El primer objetivo tiene que ver con un mecanismo para delimitar en el flujo de datos de la capa física dónde empieza y acaba un determinado segmento de datos, que en este nivel se llama trama. El segundo es necesario para detectar cuando los datos recibidos no son válidos. Aunque la transmisión USB dispone de su propio protocolo de enlace que detecta y corrige posibles errores de transmisión, existen más fuentes de posibles errores como la pérdida de datos por el desborde de la FIFO temporal del chip de la interfaz USB o ruidos en las líneas digitales que controlan esta interfaz. Para solucionar estos problemas se ha usado una trama para el nivel de enlace con este formato:

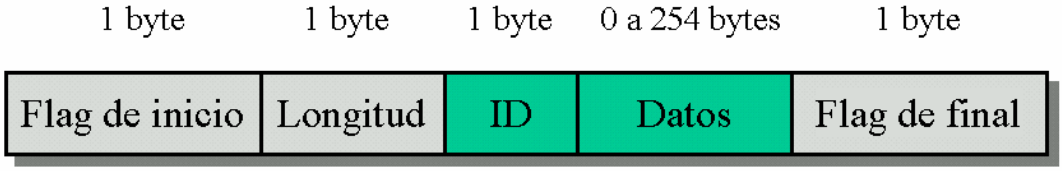


Figura 113:Formato de la trama de la capa de enlace

Como se puede ver la sincronización de byte y de trama se consigue mediante el uso de *flags* de inicio (valor constante a 0x96) y final de trama (valor constante a 0x69). La carga útil se compone de un mínimo de 1 y un máximo de 255 bytes, donde el primer byte debe ser un identificador de comando o respuesta. Este valor se utilizará para indicar el tipo de operación que se solicita, en el sentido PC-Receptor, o para indicar a que comando se está respondiendo en el sentido receptor-PC. Para tratar el tamaño variable de estas tramas se ha usado un campo de longitud que indicará el tamaño de la carga útil, y por lo tanto deberá tener un tamaño mínimo de 1 byte.

7.1.3. Definición del nivel 3: Capa de aplicación

Utilizando la comunicación basada en tramas definida en el nivel 2, se define a continuación las normas básicas que van a controlar las comunicaciones entre el software del PC y el *firmware* del receptor. Básicamente, el protocolo se puede resumir en un clásico maestro-esclavo, en el que el receptor actúa de esclavo y nunca toma la iniciativa en la comunicación, sino que solamente responde con tramas bajo petición del software del PC, que hace de maestro. Los tipos de comandos que se solicitan se pueden dividir en dos tipos:

- Comandos normales: Son la mayoría. En estos se realiza una petición al receptor mediante una trama y éste la ejecuta y responde con otra trama. Tanto las tramas de petición como de respuesta pueden tener o no datos.
- Comando de iniciar muestreo continuo.

Con un ejemplo gráfico se puede ver la diferencia entre estos tipos de comandos más fácilmente:

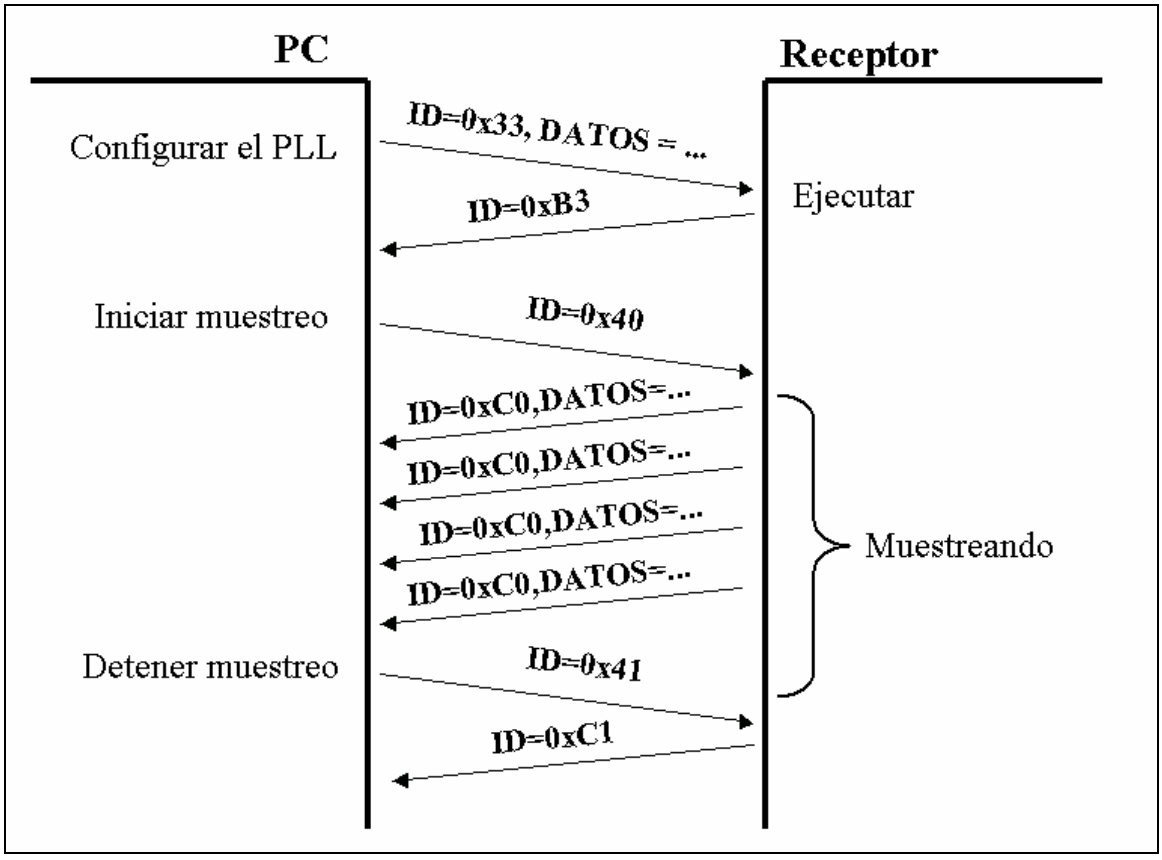


Figura 114:Ejemplo de intercambio de tramas.

En los comandos normales, se responde siempre con una sola trama, mientras que con el comando de iniciar muestreo se reciben un número indefinido de tramas conteniendo las muestras de los canales I y Q, hasta que se reciba el comando de detención del muestreo que detiene el flujo de tramas de muestras y genera una trama de respuesta.
En cuanto al código identificador (ID) de cada trama, en los comandos este indicará el tipo de acción a realizar, mientras que las respuestas siempre tendrán el mismo valor que el código del comando correspondiente, pero con el bit superior activado. Por ejemplo, la respuesta al comando 0x10 tendrá un código de 0x90. A continuación se detallan todos los comandos implementados en el firmware, junto con una descripción de los datos de entrada y de salida. Este es el nivel más alto del protocolo de comunicaciones, ya que el siguiente nivel de abstracción vendrá dado por el software del PC, como se verá en un apartado posterior.

COMANDO: Petición de identificador de *firmware*.

ID de comando: `0x10`

Descripción: Este comando tiene una doble finalidad. Por un lado sirve para comprobar que el receptor esté correctamente conectado y que las comunicaciones funcionan, ya que es el único comando que no tiene acción alguna asociada. Por otro lado, sirve para obtener la versión del firmware instalada en el recetor. Aunque actualmente es la 1.0, esto permitiría ampliar el conjunto de comandos disponibles en futuras ampliaciones.

Datos enviados: Nada.

ID de respuesta: `0x90`

Datos de la respuesta:

Longitud (bytes)	N
Contenido	Texto acabado en '\0'

COMANDO: Borrar el display

ID de comando: `0x20`

Descripción: Este comando borra la pantalla del display LCD.

Datos enviados: Nada.

ID de respuesta: `0xA0`

Datos de la respuesta: Nada.

COMANDO: Escribir en el display

ID de comando: `0x21`

Descripción: Permite escribir un texto en una posición del display LCD.

Datos enviados:

Longitud (bytes)	1	1	N
Contenido	Columna (0-1)	Fila (0-15)	Texto acabado en '\0'

ID de respuesta: `0xA1`

Datos de la respuesta: Nada.

COMANDO: Cambiar modo de ahorro de energía

ID de comando: `0x30`

Descripción: Al arrancar el receptor, la línea `/POWERDOWN` se encuentra activada por defecto, haciendo que el sistema esté en modo de ahorro de energía. Para poder usar el receptor normalmente, es necesario entrar en modo normal, desactivando este modo de bajo consumo.

Datos enviados:

Longitud (bytes)	1
Contenido	0x00: Modo normal 0x01: Modo ahorro de energía

ID de respuesta: `0xB0`

Datos de la respuesta: Nada.

COMANDO: Leer el estado actual de ahorro de energía

ID de comando: `0x31`

Descripción: Obtiene el estado actual de la línea de `/POWERDOWN`.

Datos enviados: Nada.

ID de respuesta: `0xB1`

Datos de la respuesta:

Longitud (bytes)	1
Contenido	0x00: Modo normal 0x01: Modo de ahorro de energía activado

COMANDO: Cambiar el valor de un conversor DAC

ID de comando: `0x32`

Descripción: Permite modificar el voltaje de salida de un conversor DAC.

Datos enviados:

Longitud (bytes)	1	1
Contenido	DAC a modificar (0-3)	Nuevo valor (0-255)

ID de respuesta: `0xB2`

Datos de la respuesta:

Longitud (bytes)	1
Contenido	0x00: Error en la comunicación con DAC 0x01: No hubo errores

COMANDO: Cambiar la configuración del PLL

ID de comando: `0xB3`

Descripción: Permite configurar uno de los registros del PLL. Como se vio en su apartado, al PLL entran bloques de 24 bits (3 bytes), donde en dos bits va codificado el registro destino de los datos de entre las cuatro opciones posibles. Este comando es transparente a esos detalles, ya que simplemente se envían los tres bytes y el software del PC es el encargado de prepararlos adecuadamente.

Datos enviados:

Longitud (bytes)	3
Contenido	Palabra de configuración

ID de respuesta: `0xB3`

Datos de la respuesta: Nada.

COMANDO: Muestrear la salida MUXOUT del PLL

ID de comando: `0x34`

Descripción: Este comando realiza un muestreo de la señal MUXOUT y envía el resultado. El bloque de datos de salida debe interpretarse como un flujo de bits individuales, empezando por el LSB del primer byte y acabando en el MSB del último, donde cada muestra corresponde al estado de la línea MUXOUT muestreada con un período de 2μs.

Datos enviados: Nada.

ID de respuesta: `0xB4`

Datos de la respuesta:

Longitud (bytes)	255
Contenido	Muestras

COMANDO: Iniciar modo de muestreo continuo

ID de comando: `0x40`

Descripción: Este comando inicia el modo de muestreo continuo de las señales I y Q. Mientras que no se reciba un comando 0x41, el receptor mandará de forma continua las tramas que se muestran abajo, conteniendo cada una de ellas 64 muestras de cada canal, intercaladas entre ellas. La frecuencia de muestreo es de 250KHz, dando lugar a 250000 / 64 = 3906.25 tramas por segundo.

Datos enviados: Nada.

ID de respuesta: `0XC0`

Datos de la respuesta:

Longitud (bytes)	128
Contenido	(Muestra I + Muestra Q) x 64

COMANDO: Detener modo de muestreo continuo

ID de comando: `0x41`

Descripción: Detiene el flujo de tramas de muestras de los canales I y Q.

Datos enviados: Nada.

ID de respuesta: `0xC1`

Datos de la respuesta: Nada.

COMANDO: Entrar en modo *bootloader*

ID de comando: `0x70`

Descripción: Como se explica en el apartado dedicado al firmware, se ha creado un modo de bootloader en el receptor, en el cual se habilitan una serie de comandos extra que permiten la lectura y escritura del código de programa directamente mediante el protocolo normal de tramas. Este comando es el necesario para activar estos otros comandos.

Datos enviados: Nada.

ID de respuesta: `0xF0`

Datos de la respuesta: Nada.

COMANDO DE *BOOTLOADER*: Reiniciar en el programa principal

ID de comando: `0x71`

Descripción: Reinicia el microcontrolador ejecutando el programa de usuario que se haya subido mediante el modo *bootloader*.

Datos enviados: Nada.

ID de respuesta: `0xF1`

Datos de la respuesta: Nada.

COMANDO DE *BOOTLOADER*: Leer memoria flash

ID de comando: `0x72`

Descripción: Este comando permite leer una página (64 bytes) de memoria flash (memoria de programa) del microcontrolador.

Datos enviados:

Longitud (bytes)	2
Contenido	Dirección de memoria a leer.

ID de respuesta: `0xF2`

Datos de la respuesta:

Longitud (bytes)	64
Contenido	Los datos leidos.

COMANDO DE *BOOTLOADER*: Grabar memoria flash

ID de comando: `0x73`

Descripción: Este comando graba una página (64 bytes) de memoria flash (memoria de programa) en el microcontrolador. Solo se puede grabar en las primeras 6Kb de flash, ya que los 2Kb superiores son código del bootloader que está actualmente en ejecución.

Datos enviados:

Longitud (bytes)	2	64
Contenido	Dirección de memoria a grabar	Datos a grabar

ID de respuesta: `0xF3`

Datos de la respuesta: Nada.

7.2. Diseño del firmware

En este apartado se abordan los objetivos del software que se ejecuta en el microcontrolador del receptor (firmware) y el diseño que se ha adoptado para cumplirlos.

7.2.1. Herramientas utilizadas

Al diseñar e implementar el firmware se presentan dos opciones principalmente: realizarlo en lenguaje C o directamente en ensamblador. Debido a la precisión temporal que se requiere para la tarea de sincronizar el muestreo continuo de las señales I y Q, y debido a que el resto de tareas a realizar no son demasiado complejas se adoptó la opción de realizarlo en lenguaje ensamblador.

El fabricante (Atmel) distribuye a través de su web un entorno de programación y simulación de forma gratuita, el *AVRStudio*, que es el que se ha utilizado para el desarrollo del firmware en este proyecto. El funcionamiento de este entorno es muy similar al de otros como Visual C++, ya que permite mantener proyectos consistentes en un grupo de ficheros, que serán ensamblados todos conjuntamente para generar una sola imagen ejecutable. Esta imagen ejecutable se puede simular posteriormente en el mismo

entorno, lo que ha sido de gran ayuda en la tarea de calcular el tiempo de ejecución de determinados bucles de temporización crítica.

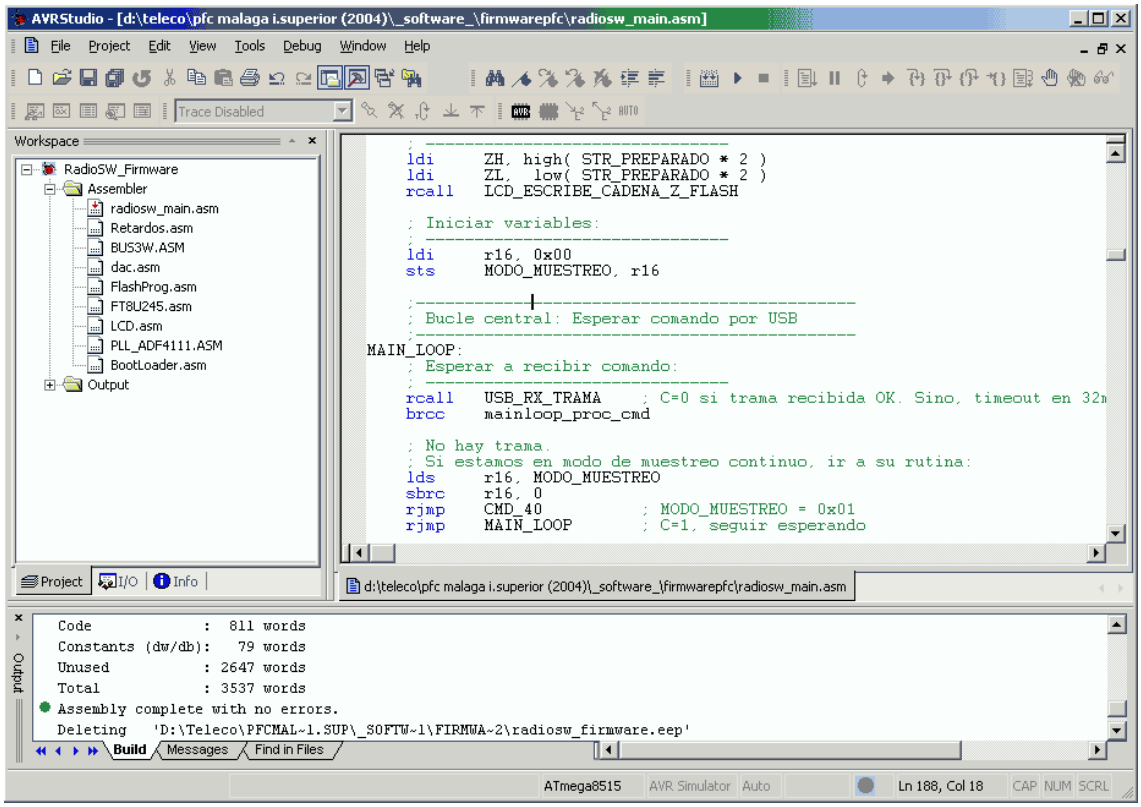


Figura 115:Para el desarrollo del firmware se ha usado el entorno AVRStudio 4.

Esta imagen ejecutable se puede obtener en distintos formatos, pero en este caso se ha elegido el formato *Intel HEX*, un formato textual consistente básicamente en un volcado de los datos en hexadecimal con alguna información extra. Este fichero HEX será el utilizado para programar el microcontrolador, ya sea mediante el programador hardware (que se verá en el capítulo 8) o mediante los comandos del modo *bootloader* desde el software del PC.

7.2.2. Esquema general

El diagrama de flujo simplificado del *firmware* del receptor se muestra en la siguiente figura. Como primer punto importante hay que señalar la separación en dos bloques diferentes de software:

- Programa principal: Este programa es el encargado de procesar las tramas que llegan del PC y ejecutarlas, lo que es la tarea principal del receptor.
- Bootloader: Este programa es el que se arranca automáticamente después del reset y es donde está la parte que procesa las tramas del modo *bootloader*.

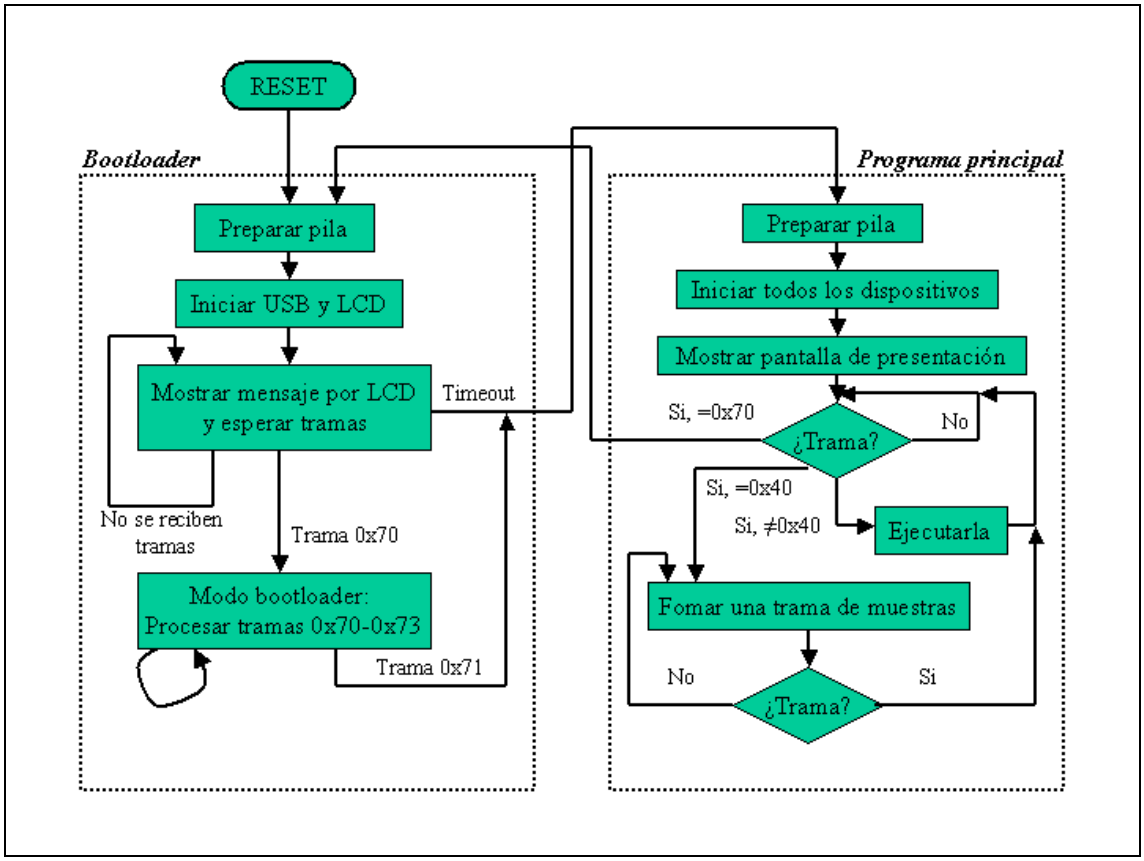


Figura 116: Diagrama de flujo simplificado para el *firmware* del receptor.

7.2.3. Descripción del *bootloader*

El código del bootloader se aloja en los 2Kb superiores de la memoria flash, mientras que los otros 6Kb de flash están destinados al programa principal. En realidad finalmente no se ha ocupado ni la mitad de estos 6Kb, por lo que hay espacio suficiente para posibles ampliaciones futuras. Esta separación del software en dos partes viene justificada por la seguridad: si el programa principal contuviese algún error (durante la

etapa de desarrollo y depuración) el firmware no respondería a los comandos del PC, y no pudiendo por lo tanto reprogramarlo mediante los comandos del bootloader para corregir el error y quedando como única alternativa el sacar el microcontrolador del zócalo y programarlo mediante el programador descrito en un capítulo posterior, lo que resulta mucho más engorroso que el método de reprogramación por comandos software.
Separando el *firmware* en dos partes y direccionando el reset al código del *bootloader*, lo que se hace con una opción del microcontrolador en sus *fuse-bits*, se tiene la seguridad de que siempre habrá un momento, justo al conectar el receptor, en el que se puede actualizar el software. Además esto permite deshabilitar las instrucciones de acceso a flash desde código fuera de la zona del código de bootloader, lo que es otra opción de los *fuse-bits*, evitando que la memoria de programa se escriba por error.
Lo primero que se hace nada más arrancar el receptor, en la parte del *bootloader*, es iniciar la pila. Esta se inicia apuntando a la máxima dirección de memoria RAM interna, que hay que recordar que era de 512 bytes. Como las únicas variables en RAM usadas en el firmware (el resto están en registros) son:

- Un buffer temporal para las tramas recibidas y a transmitir, de 261 bytes de largo.
- Un indicador de modo de muestreo continuo activo, de un solo byte.

Queda un total de 512 – 262 = 249 bytes libres para la pila, valor más que suficiente ya que solamente se utilizara para almacenar las direcciones de vuelta en las llamadas entre rutinas.
Una vez iniciada la pila, se inicializan solamente la interfaz USB y los puertos dedicados al display LCD, los únicos dispositivos usados en el código del *bootloader*. Estas rutinas de inicialización se describirán más detalladamente en el siguiente apartado. Una vez iniciados estos dos periféricos, se muestra un mensaje como el que se ve en la primera imagen de abajo durante unos segundos, tras los cuales si no se ha recibido una trama de iniciar modo *bootloader*, la de identificador 0x70 como se vio, se saltará a ejecutar el programa principal, ubicado a partir de la dirección 0x0000 de la memoria flash.

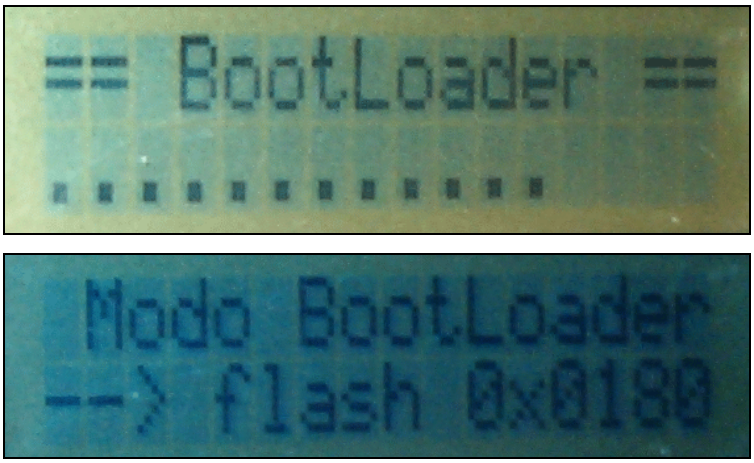


Figura 117: Imágenes del display durante (a) el tiempo de espera del bootloader, y (b) la grabación de la flash desde el PC.

Para ser exactos, en realidad en esta dirección están los vectores de interrupción, de los que el primero es el vector de reset que lanzará el comienzo del programa. Debido a la configuración de los *fuse-bits*, al lanzar el primer reset tras la alimentación del microcontrolador no se salta a este vector, sino al situado en la tabla alternativa de vectores de interrupción, en la zona de memoria del *bootloader* en los 2Kb superiores de memoria flash.

Si durante el tiempo de espera de la carga del *bootloader* se recibiese una trama de inicio de modo *bootloader* se llegará al bucle principal de ejecución del modo *bootloader*, como se ve en el diagrama de flujo anterior. En este bucle solamente se procesan las cuatro tramas del grupo del *bootloader* descritas en el apartado anterior, mientras que cualquier otra trama será ignorada. En la segunda pantalla de la imagen anterior se muestra un ejemplo de lo que se muestra en el display mientras se está subiendo desde el PC una nueva imagen ejecutable a la memoria flash, donde se ve como se va indicando la dirección de memoria que está siendo grabada en cada momento. Tanto el proceso de lectura como de grabación de la memoria de programa es muy rápido, y duran alrededor de 6 segundos. Esto es así porque, aunque la grabación de memorias flash es mucho más lenta que su lectura ya que requieren retardos de varios milisegundos para cada grabación (consultar el *datasheet* del ATMEGA8515 para información más detallada), ésta se ejecuta en bloques de páginas de memoria de 64 bytes, por lo que el número de retardos se reduce considerablemente que si se tuviesen que realizar para cada byte individual como ocurría con modelos anteriores de estos microcontroladores.

Existen dos formas en que se puede saltar al programa principal desde el código del bootloader:

- Por el *timeout* en la espera del comando de entrar en modo bootloader al comienzo: Por defecto, si se conecta el receptor y no se intenta grabar ni leer la memoria, en unos tres segundos se saltará automáticamente al programa principal, iniciando la ejecución normal.
- Si después de haber entrado en el modo bootloader, se recibe un comando con identificador 0x71, un comando de salto al programa principal. Este comando es necesario para volver a arrancar el programa principal tras haber subido un nuevo *firmware*, por ejemplo.

7.2.4. Descripción del programa principal

A continuación se repite por comodidad una vista parcial del diagrama de flujo mostrado anteriormente, donde solamente se muestra ahora la parte correspondiente al programa principal.

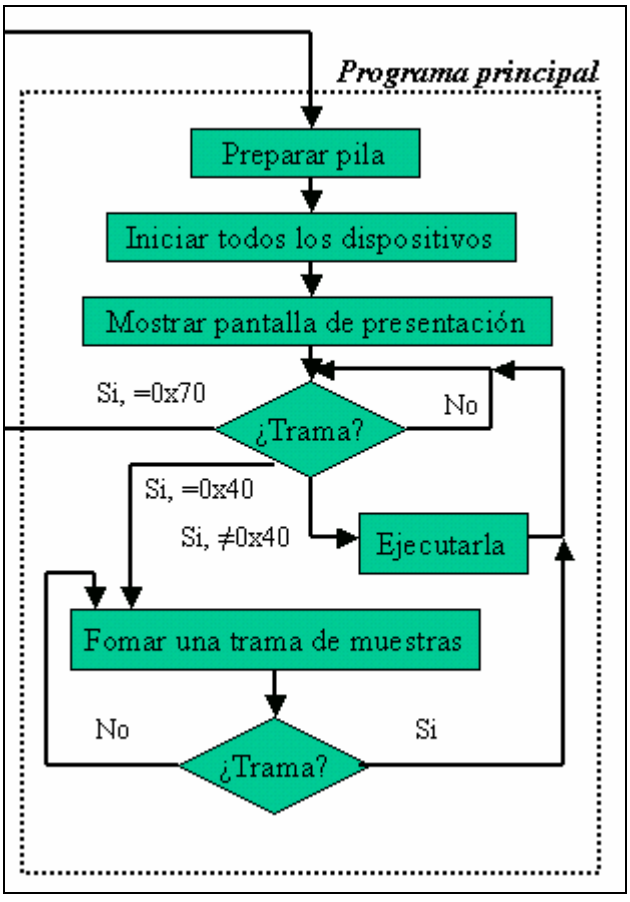


Figura 118:Diagrama de flujo simplificado para el programa principal.

Al igual que en la parte del bootloader, lo primero que se hace es inicializar la pila apuntando a la dirección más alta de la memoria RAM, ya que como la pila crece hacia abajo y las variables en RAM se sitúan en la parte inferior, así se tiene un máximo de espacio disponible. A continuación se inicializan todos los puertos de entrada y salida del microcontrolador necesarios para controlar todos los dispositivos del receptor, que son:

- La señal de `/POWER_DOWN`, inicialmente a nivel bajo y por lo tanto manteniendo a los dispositivos en modo de bajo consumo.
- La memoria FIFO, que es reiniciada para asegurar que esté vacía inicialmente.
- Las señales de control del integrado FT8U245BM, interfaz USB, para asegurar que ni se está escribiendo ni leyendo datos.
- Iniciación del bus SPI (3W, "*three wires"*), incluyendo las dos señales de *latch* para los conversores DAC y el PLL, encargadas de señalar que los datos del bus común se dirigen a estos dispositivos.
- Las señales de control de los conversores ADC, para mantenerlos en un estado de reposo.
- El display LCD. La inicialización del *display* conlleva una serie de comandos para asegurar que se inicia a un estado activo y conocido.

En el siguiente apartado se dan más detalles de bajo nivel de algunas de estas inicializaciones, además de otros procesos, y en el anexo A5.2 se lista el código fuente completo de todos los ficheros en ensamblador que forman el *firmware*, convenientemente comentado para poder consultar cómo se realiza cualquiera de estas tareas en detalle.

Tras iniciar todos los dispositivos, se muestra un mensaje en el display durante unos segundos con la versión del firmware, tras el cual ya se llega al bucle principal del programa. Como se ve en el diagrama de flujo anterior, este bucle consiste en la espera de tramas desde el PC y su procesado según su tipo. Se distinguen estos casos:

- Si es una trama de iniciar modo *bootloader*, con código 0x70, se salta al comienzo del código de *bootloader*. Así se procesarán a continuación correctamente todos los comandos del grupo *bootloader* que se reciban del PC.
- Si es una trama de inicio de modo de muestreo continuo de las señales I y Q, con código 0x40, se salta al bucle de muestreo, que se describe en más detalle a continuación, ya que es la parte más importante de todo el *firmware*.

- Si es cualquier otra trama, se salta a la rutina que la ejecuta extrayendo los parámetros de entrada de la trama entrante, genera una trama de salida, enviándola al PC y volviendo al bucle principal.

El diseño del bucle de muestreo se expondrá en el siguiente apartado con más detalle, pero básicamente lo que hace es lo siguiente:

- Forma la cabecera de una trama de resultado del muestreo, con código 0xC0. Esto se hace enviando la secuencia de bytes a la memoria FIFO para que, como ya se vio, estos se envíen mediante la lógica externa al chip de interfaz USB automáticamente.
- Activa las señales de control de los conversores ADC de la forma adecuada para hacer que 64 muestras de cada canal, de forma alternada, se vuelquen en la memoria FIFO.
- Escribe el *flag* de final de trama en la FIFO, dejando una trama completa en la memoria lista para ser enviada al PC. En realidad el proceso de envío ya habrá comenzado porque se ejecuta de manera asíncrona e independiente.
- Se comprueba si hay datos de entrada desde el PC por la interfaz USB. Si es así se salta al bucle principal para su procesado. Si no es así, se continuará en el bucle de muestreo.

En muchos sentidos este bucle es el más crítico de todo el *firmware*, ya que incluso la elección del modelo de microcontrolador estuvo principalmente condicionada por la capacidad de éste de ejecutar esta rutina cumpliendo con las estrictas condiciones temporales necesarias. Esto es así porque para que el muestreo de una señal tenga alguna validez se debe asegurar que las muestras se han tomado a intervalos totalmente idénticos entre ellos, mientras que como se ve, hay que realizar tareas distintas entre unas muestras y otras. En el siguiente apartado se explica exactamente como se ha resuelto este problema.

Hay un detalle que todavía queda por explicar del funcionamiento general del *firmware*: Si mientras se está en el modo de muestreo se recibe una trama, lo que será común por ejemplo, para cambiar un conversor DAC y ajustar la ganancia del receptor, hay que volver al bucle principal para procesarla, pero lo deseable es que inmediatamente después de procesarla se vuelva al bucle de muestreo para continuar cuanto antes enviando muestras al PC. Sin embargo las rutinas que se encargan de ejecutar los distintos comandos, al finalizar, saltan al bucle principal, no al de muestreo. Para solucionar esto se ha añadido una variable en memoria RAM, llamada

MODO_MUESTREO, y que contendrá un valor distinto de cero si se estaba en modo de muestreo cuando llegó la trama desde el PC. De esta forma, en el bucle principal se debe comprobar esta variable para saltar al bucle de muestreo no sólo cuando llegue un comando de inicio de muestreo (0x40) sino cuando esta variable sea distinta de cero. Para finalizar el modo de muestreo, en la rutina encargada de procesar el comando de finalización del modo muestreo (0x41) simplemente se pone a cero esta variable. Así, el bucle principal queda así:

```
  ;-------------------------------------------
  ; Bucle central: Esperar comando por USB
  ;-------------------------------------------
MAIN_LOOP:
   ; Esperar a recibir comando:
   ; ------------------------------
  rcall         USB_RX_TRAMA ; C=0 si trama recbda.Si timeout en 32ms C=1
  brcc          mainloop_proc_cmd
  ; No hay trama.
  ; Si estamos en modo de muestreo continuo, ir a su rutina:
  lds           r16, MODO_MUESTREO
  sbrc          r16, 0
  rjmp          CMD_40              ; MODO_MUESTREO = 0x01
  rjmp          MAIN_LOOP           ; C=1, seguir esperando
mainloop_proc_cmd:
   ; Ejecutar comando segun numero de comando:
   ...
```

Sobre la estructura global del firmware solo queda por decir que se ha considerado que un diseño sin hacer uso de interrupciones era el más adecuado para los objetivos deseados y el que resultaba en un código más sencillo, por lo que las interrupciones se mantienen deshabilitadas en todo momento.

7.2.5. Detalles de implementación del bucle de muestreo

El bucle de muestreo es el encargado de formar las tramas de muestras y enviarlas al PC, asegurando que se cumple estrictamente el período de muestreo. Este período se ha establecido en 4µs, es decir, 250.000 muestras por segundo y por canal, generando el flujo de 500Kb/s que ya se vio en el apartado dedicado al integrado USB y la frecuencia de Nyquist de 125KHz que se utilizó por ejemplo para el diseño de los filtros de banda base y *antialiasing*. A continuación se muestra el diagrama de flujo detallado de esta rutina, junto con el tiempo de ejecución de cada parte, medida en ciclos de reloj. Hay que recordar que el microcontrolador es RISC, por lo que cada instrucción se ejecuta normalmente en un ciclo de reloj, que al ser de 10Mhz resulta ser de 100ns.

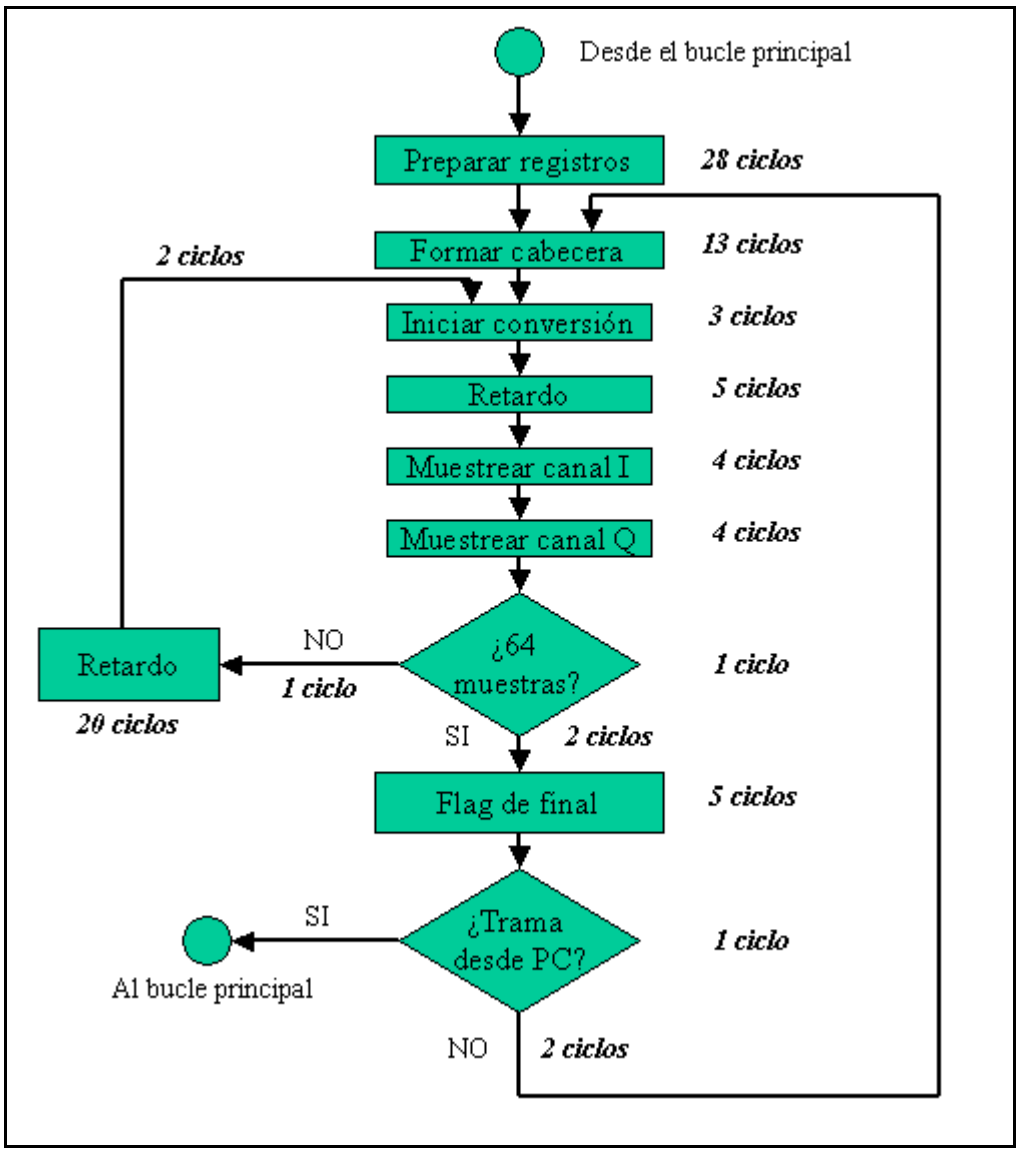


Figura 119:Diagrama de flujo detallado del bucle de muestreo.

El objetivo es generar tramas de 64 muestras de cada canal, muestreando cada canal a 250KHz, dando resultado a tramas con este contenido:

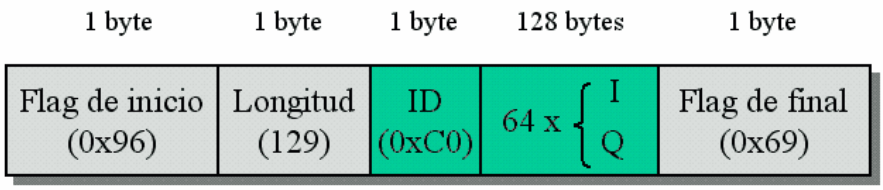


Figura 120: Tramas del tipo 0xC0.

Que es el formato de trama definido en el apartado 7.1.2. pero con los campos específicos de una trama del tipo 0xC0 correspondientes a tramas de datos muestreados. Por lo tanto se puede ver que la trama tiene unas partes inicial y final constantes:

- El flag de inicio, la longitud y el identificador de trama.
- El flag de final.

Los primeros tres bytes son los que se escriben en la parte del diagrama de flujo llamada "Formar cabecera", con una duración de 13 ciclos de reloj.

Para entender el proceso que se ejecuta dentro del bucle, primero hay que mostrar cómo funcionan los conversores ADC:

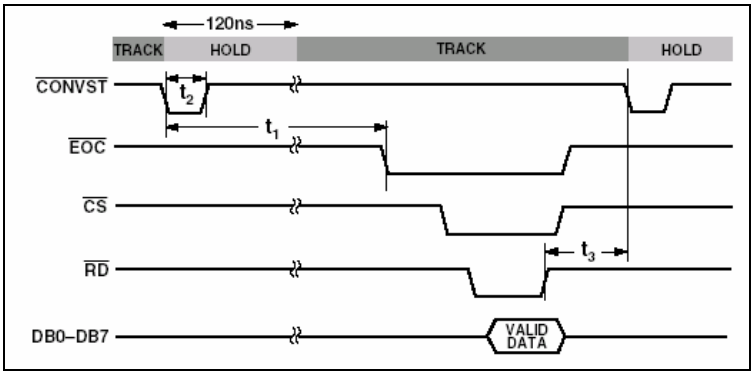


Figura 121: Temporización del ADC.

Como se ve en el cronograma, el inicio de las conversiones se realiza mediante una señal activa a nivel bajo `/CONVST`. Para que el tratamiento matemático de las señales en cuadratura tenga sentido, ambas señales I y Q deben ser muestreadas en el mismo instante. Por esto las dos señales de inicio de conversión de ambos conversores ADC se han conectado a una misma señal de control proveniente del microcontrolador, llamada `/ADC_CONVST` en los esquemas eléctricos. El tiempo en que esta señal se mantenga a nivel bajo será el tiempo de muestreo (*hold*) de los conversores, que debe ser de 20ns como mínimo. Como el ciclo de reloj del microcontrolador es de 100ns, éste es el

retardo mínimo, pero por seguridad y porque no influye negativamente éste ciclo perdido, se ha usado un retardo de 200ns, correspondiente a la ejecución de dos instrucciones. Esta parte se puede ver en el diagrama de flujo anterior como "Inicio de la conversión", de 3 ciclos de duración.

A continuación hay que esperar un tiempo mientras se realiza la conversión, indicado como t_1 en el cronograma de arriba y ligado a la activación de la señal EOC (*End Of Conversion*) que el receptor no se ha utilizado. En cambio se ha utilizado un retardo fijo que asegura que la conversión se ha completado antes de solicitar la palabra digital resultante a los conversores ADC. En el *datasheet* de los conversores AD7822 se indica un tiempo de conversión de 420ns como máximo. De nuevo debido a la granularidad que permite el tiempo de ciclo de 100ns, se ha usado un retardo fijo de 500ns tras la señal de comienzo de conversión. Este retardo se muestra en el diagrama de flujo como un retardo de 5 ciclos de ejecución.

Tras este retardo, hay que pedir las muestras a los conversores, activando la señal `/ADC_RD_I` (o `/ADC_RD_Q` para el caso del canal Q) a la vez que se activa la señal /FIFO_WR para que la muestra se guarde en la memoria FIFO. Este proceso debe repetirse primero para el canal I y luego para el canal Q, y en el diagrama de flujo se pueden ver como "Muestrear canal I" y "Muestrear canal Q", de 4 ciclos cada uno. Como muestra de cómo se ha implementado esta parte del código, se lista a continuación el fragmento de ensamblador encargado del inicio de la conversión, el retardo y el muestreo de los dos canales analógicos:

```
out          PORTD, r7                ; /ADC_CONVST = 0
nop          ; 2 ciclos
out          PORTD, r10               ; /ADC_CONVST = 1

nop          ; 4 ciclos
nop          ; 5 ciclos
nop          ; 6 ciclos
nop          ; 7 ciclos
nop          ; 8 ciclos

; Enviar muestra "I"
out          PORTD, r8                ; /ADC_RD_I = 0
out          FIFO_CTRL_PORT, r12      ; /FIFO_WR = 0
out          FIFO_CTRL_PORT, r11      ; /FIFO_WR = 1
out          PORTD, r10               ; /ADC_RD_I = 1

; Enviar muestra "Q"
out          PORTD, r9                ; /ADC_RD_Q = 0
out          FIFO_CTRL_PORT, r12      ; /FIFO_WR = 0
out          FIFO_CTRL_PORT, r11      ; /FIFO_WR = 1
out          PORTD, r10               ; /ADC_RD_Q = 1
```

Por lo tanto esta parte es la que deberá ejecutarse 64 veces para generar las 128 muestras que contiene la trama. Sin embargo hay que fijarse en un detalle respecto al tiempo de ejecución del bucle: en todas las iteraciones en que no se hayan llegado a 64 muestras por canal, la ejecución deberá reanudar por el punto del inicio de las conversiones, mientras que cada 64 iteraciones, deberá enviar el flag de final, comprobar que no hay datos desde el PC, y si no es así volver a formar otra cabecera y entonces sí, volver a iniciar las conversiones. Como es un requisito imprescindible el que las muestras se tomen cada 2.5ms en todos los casos, se ha tenido que añadir el retardo que se puede ver a la izquierda del diagrama, para compensar por los ciclos de ejecución perdidos en la formación de la cabecera, etc...

Por lo tanto esta parte del código ha sido meticulosamente diseñada para que la suma de todos los posibles tiempos de ejecución al cerrar el bucle sean de 4µs, es decir 40 ciclos de reloj. En este proceso se tienen que tener en cuenta tanto el tiempo de ejecución de las instrucciones como los ciclos perdidos por saltos o instrucciones condiciones, fruto de la arquitectura en *pipeline* del microcontrolador.

En los siguientes cronogramas se han resaltado con líneas más gruesas los procesos y saltos involucrados en cada uno de los dos posibles caminos que la ejecución puede seguir durante el proceso de muestreado de las señales I y Q:

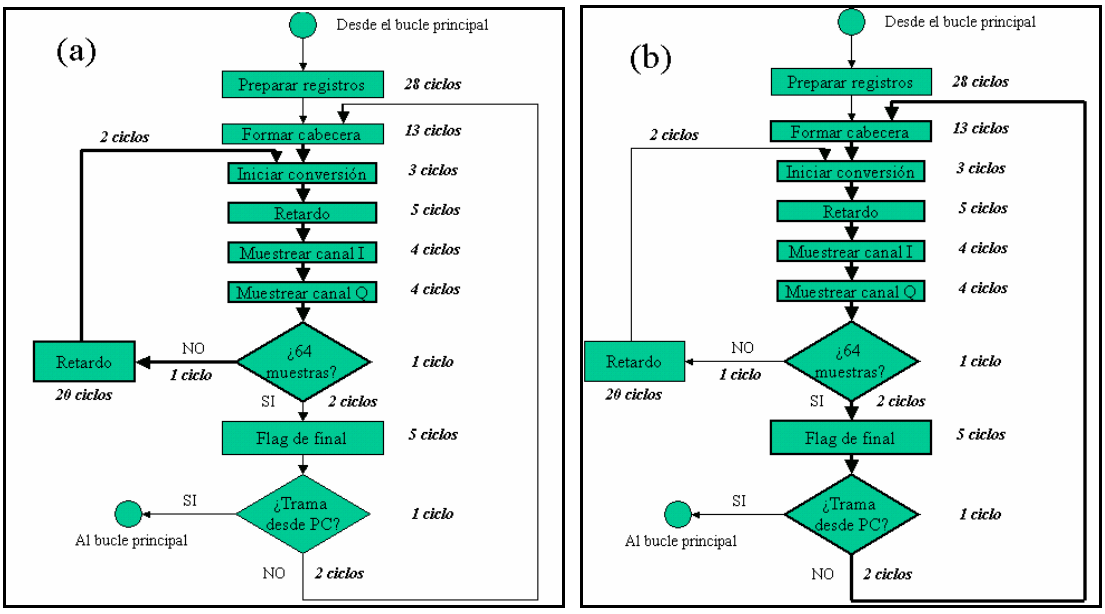


Figura 122: Los dos caminos posibles en la ejecución del bucle de muestreo.

En la siguiente tabla se muestra la suma de los tiempos de cada uno de los caminos de ejecución:

- Caso (a): Todavía no se ha completado una trama completa de 64 muestras por canal:

Proceso	**Duración en ciclos**
Iniciar conversión	3
Retardo	5
Muestrear canal I	4
Muestrear canal Q	4
Decisión	1
Salida de decisión por camino "NO"	1
Retardo	20
Salto	2
TOTAL	**40**

- Caso (b): Se ha completado una trama completa. Hay que añadir el *flag* de final, comprobar si hay comandos desde el PC y volver a formar la cabecera para la próxima trama de muestras:

Proceso	**Duración en ciclos**
Iniciar conversión	3
Retardo	5
Muestrear canal I	4
Muestrear canal Q	4
Decisión	1
Salida de decisión por camino "SI"	2
Flag de final	5
Decisión	1
Salida de decisión por camino "NO"	2
Formar cabecera	13
TOTAL	**40**

Para terminar este apartado, y como resumen de la información presentada tanto en este apartado como en los relativos al diseño digital de la máquina de estados, de la memoria FIFO y de la interfaz USB, se muestra el cronograma general que muestra el camino de los datos desde los conversores ADC hasta el interfaz USB.

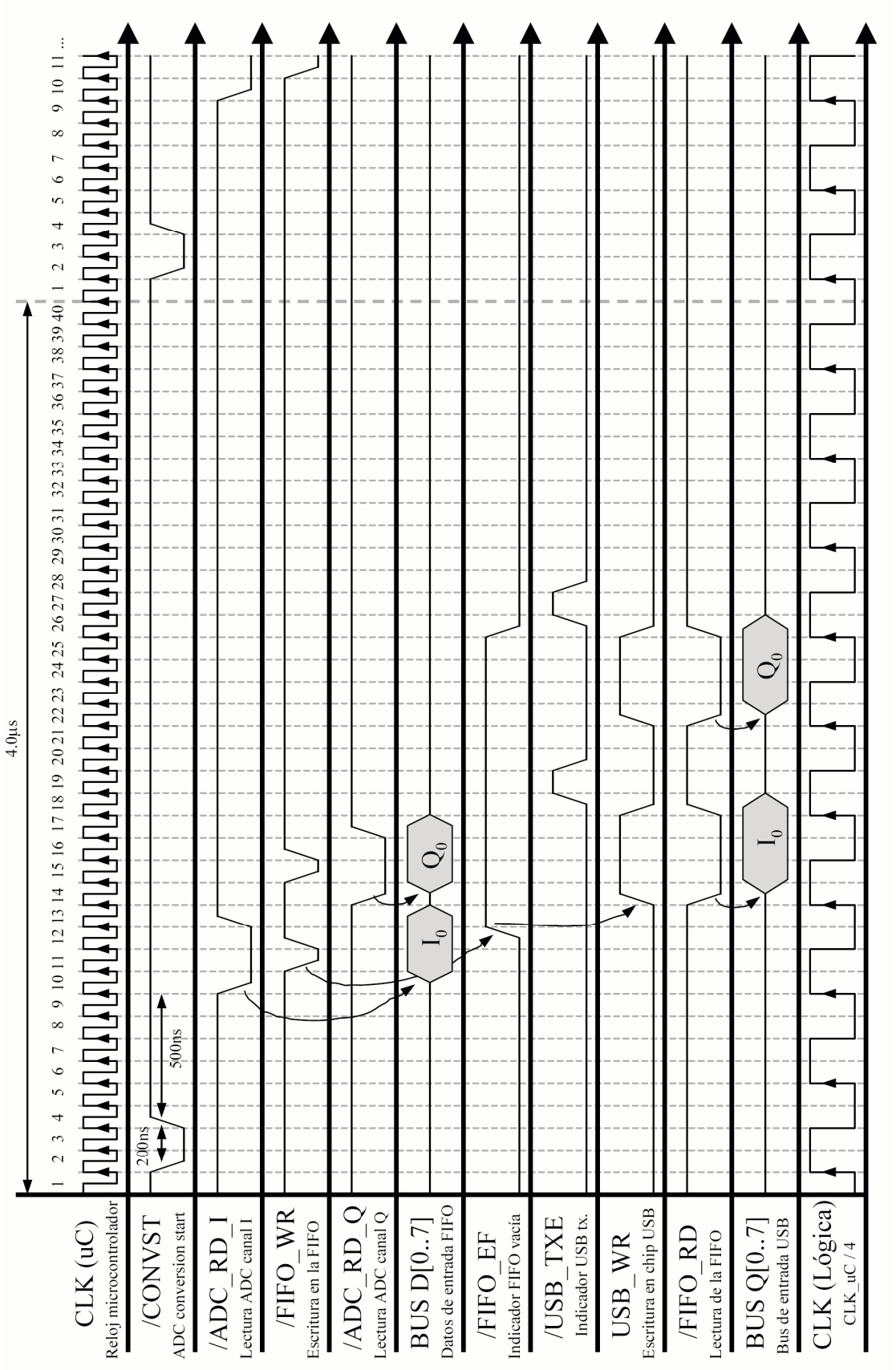


Figura 123:Cronograma de las principales señales involucradas en el muestreo de las señales analógicas.

7.2.6. Otros detalles de implementación

En este apartado se tratarán otros temas relaciones con la implementación del firmware en ensamblador, aparte de la rutina del bucle de muestreo, ya analizada extensamente en el apartado anterior.

Al igual que en los lenguajes C o C++ existen ficheros de cabecera destinados a declaraciones, en el lenguaje ensamblador Atmel existen unos ficheros de definiciones, o ficheros "*include*", que a diferencia que en los lenguajes de alto nivel, son específicos del modelo de microcontrolador usado. En el caso del ATMEGA8515, el fichero es el "*m8515def.inc*" y en él se definen con nombres simbólicos las direcciones de puertos de entrada y salida, periféricos, etc... Este fichero se incluye en un fichero de ensamblador utilizando la directiva:

```
.include "m8515def.inc"
```

Esta directiva es la que se ha utilizado para separar el código del firmware en distintos ficheros y luego volver a incluirlos todos realmente en uno sólo, ya que el ensamblador solo permite un solo fichero de código. La división que se ha hecho en ficheros ha sido la siguiente:

- "RadioSW_Main.asm": Este es el fichero principal que se pasa al ensamblador. En él está:
    - La tabla de vectores de interrupciones normales. Como ya se vio, sólo el vector de reset es llamado desde el código de bootloader, ya que el resto de interrupciones no se usan en este proyecto.
    - Las definiciones de las variables en RAM y de los puertos de entrada asociados a cada dispositivo. Mediante el uso de nombres simbólicos para cada puerto de cada dispositivo (como el PLL, el LCD, ...) se aumenta la claridad del código y se evitan errores.
    - Las llamadas las rutinas de inicialización de los puertos de los distintos dispositivos o buses alrededor del microcontrolador.
    - El bucle principal, encargado como ya se ha visto, de recibir las tramas desde el PC y ejecutar las rutinas correspondientes al tipo de comando recibido.
    - Al final de este fichero se incluyen todos los demás mediante directivas ".include".

- "Retardos.asm": En determinadas partes del *firmware* se hace necesaria la introducción de retardos de varios órdenes de magnitud superiores a los que se obtendrían con sencillas instrucciones `nop`, por ejemplo, cuando se muestran los mensajes por display al arrancar. Estos mensajes deben permanecer visibles unos segundos. Para estas y otras tareas, se han implementado en este fichero tres rutinas, cada una de ellas con un rango de retardos distintos:
    - RETARDO_R25_MAX100us: Ejecuta un retardo máximo de 100μs.
    - RETARDO_R25_MAX25ms: Un retardo máximo de 25ms.
    - RETARDO_R25x25ms: Ejecuta un retardo múltiplo de 25ms.
- "Bus3W.asm": En este fichero se implementa el manejo del bus SPI (o 3W, *three wires*). Como se ha visto a lo largo de la memoria, este bus se usa tanto por el conversor DAC como por el PLL. Las señales de *latch* individuales que indican a qué dispositivo en concreto se refiere el envío de datos no se manejan en la rutina `BUS3W_ENVIAR` (la principal rutina de este fichero), sino que su señalización es responsable del código que llama a esta rutina. Debido a que el PLL y el DAC tienen longitudes de palabra distintas, en esta rutina la longitud de palabra en bits es un parámetro más de entrada para asegurar la generalidad. La otra rutina que contiene este fichero es la encarga de configurar como salidas y a un valor conocido los puertos del microcontrolador destinados a ser usados como bus SPI.
- "DAC.asm": En este fichero existen dos rutinas. Una de ellas es la de inicialización, que simplemente configura como salida el puerto destinado a ser el *latch* en el protocolo SPI para el DAC. La otra es `DAC_CAMBIAR`, y directamente permite cambiar un DAC determinado a un valor dado de entrada. En esta rutina se llama a la rutina `BUS3W_ENVIAR` descrita arriba, usando una longitud de palabra de 12 bits.
- "FlashProg.asm": En este fichero solamente se implementa una rutina, la `FLASH_PROG`, que permite la grabación de una página de memoria flash de programa, a partir de un buffer en RAM y en la dirección indicada como parámetro.
- "FT8U245.asm": El nombre de este fichero es el del integrado encargado de la interfaz USB. En este fichero se implementan estas rutinas:
    - `USB_INICIAR`: Aquí se inicializan los puertos de control del integrado, es decir `/RD`, `WR`, `/TXE` y `/RXF`. Como el bus de datos es de uso compartido con otros dispositivos (ver esquemas eléctricos), su inicialización se deja para el programa principal.

- `USB_RX_BYTE`: Esta rutina espera hasta que se reciba un byte desde el PC por el interfaz USB, o hasta que se cumpla un *timeout* de 32ms. Con el *timeout* se pretende evitar que la rutina sea bloqueante en situaciones en que esto no sería deseable.
- `USB_TX_BYTE`: Esta rutina no envía realmente el dato al integrado, sino que por las razones de integridad en el flujo de los datos que ya se han comentado, escribe el byte en la memoria FIFO. La lógica externa asociada se encargará de enviar estos datos al integrado USB cuando éste se encuentre listo. De esta manera el microcontrolador también evita esperar cuando el chip USB no esté listo para aceptar más datos.
- `USB_TX_TRAMA_LONG_R18`: Con esta rutina se forma automáticamente una trama del formato que se ha descrito anteriormente y la envía a la memoria FIFO para su transmisión al PC. Solamente necesita como parámetros de entrada la longitud del campo de datos, ya que los datos se deben encontrar de forma fija en el *buffer* que a este propósito existe en RAM.
- `USB_RX_TRAMA`: Esta rutina se encarga del proceso de recepción de una trama completa desde el PC. Incluye varias medidas de seguridad para evitar errores o bloqueos, como un *timeout* entre dos bytes consecutivos y la comprobación de las cabeceras de comienzo y final y un *timeout* para la recepción de una trama completa si no llega ningún byte. Por lo tanto esta rutina tampoco es bloqueante.

- "`LCD`.asm": En este fichero se incluyen multitud de rutinas relacionadas con el control del display LCD. Por ejemplo, existen rutinas para configurar el modo del cursor (ver la tabla de comandos en el apartado 5.7 dedicado al LCD), cambiar la posición del cursor, borrar el display o escribir cadenas de texto, números en decimal o en hexadecimal, etc... Sin embargo la más compleja en este caso es la rutina de iniciación, ya que en el caso del LCD se requiere:
  - Configurar los puertos del microcontrolador destinados a las señales `E` y `/RS` como salidas.
  - Esperar un segundo para asegurar que el LCD se ha arrancado. Los displays son dispositivos lentos que suelen tardar mucho más que otros integrados en iniciarse.

  - Enviar tres veces consecutivas un comando que activa el modo de interfaz a ocho bits. Como se comentaba en el apartado 5.7, los LCD tienen dos modos de bus: a ocho y cuatro bits. Este comando se envía tres veces para asegurar que, aún en el caso de que al arrancar estuviese en el modo a cuatro bits que requiere dos envíos de cuatro bits para formar un comando, el comando se reciba y deje por lo tanto al LCD en un estado conocido de interfaz a ocho bits.
  - Enviar ahora el comando de configuración de bus a cuatro bits.
  - Habilitar el modo de dos líneas de texto y caracteres de 5x7 píxeles.
  - Activar el LCD. Por defecto, el display arranca con la pantalla desactivada.
  - Para evitar problemas que ocurren en ocasiones tras un *reset*, desactivar y volver a activar el LCD para asegurar que se activa correctamente.
- "PLL.asm": En este fichero solamente se incluyen dos pequeñas rutinas:
  - `PLL_INICIAR`: Inicializa el puerto de salida dedicado a ser el *latch* del bus SPI para el integrado del PLL.
  - `PLL_ENVIAR`: Esta rutina llama a la antes descrita `BUS3W_ENVIAR` indicando 24 bits como longitud de palabra del bus SPI, y activando la señal de *latch* para indicar al PLL que debe recibir los datos que van por el bus SPI. A este nivel el conjunto de bits, que deben ser interpretados según se vio en el apartado 4.8, se toman como un conjunto de tres bytes sin más, siendo el software del PC el encargado de construir las palabras de configuración a partir de los parámetros deseados.

Por último cabe señalar que al tratarse en general de rutinas con datos de entrada y salida sencillos, siempre se han utilizado los propios registros del microcontrolador para el paso y vuelta de parámetros, o incluso en algunas ocasiones algún *flag* del registro de estado como el `C` (bit de acarreo).

7.3. Diseño de "*SWRadioStudio*"

7.3.1. Introducción

Hasta ahora se ha explicado tanto todo el procesado analógico de las señales como su digitalización y el proceso que siguen hasta ser enviadas al PC. Además se ha definido el protocolo de comunicaciones con el que el firmware espera interactuar con el software del PC. En este apartado se va a describir tanto el objetivo, diseño y detalles de implementación del conjunto de software que ejecutándose en el PC permitirán hacer uso pleno del receptor. Hay que señalar que el sistema operativo para el que se ha diseñado todo el software es Windows 2000 o superior. Aunque el hardware USB utilizado es compatible también con la familia de sistemas operativos Linux desde la versión del kernel 2.40, sólo se ha diseñado para la familia de sistemas Windows por la mayor familiaridad con las herramientas de desarrollo necesarias.

El camino de los datos que entran desde el receptor en el PC se muestra esquemáticamente en la siguiente figura:

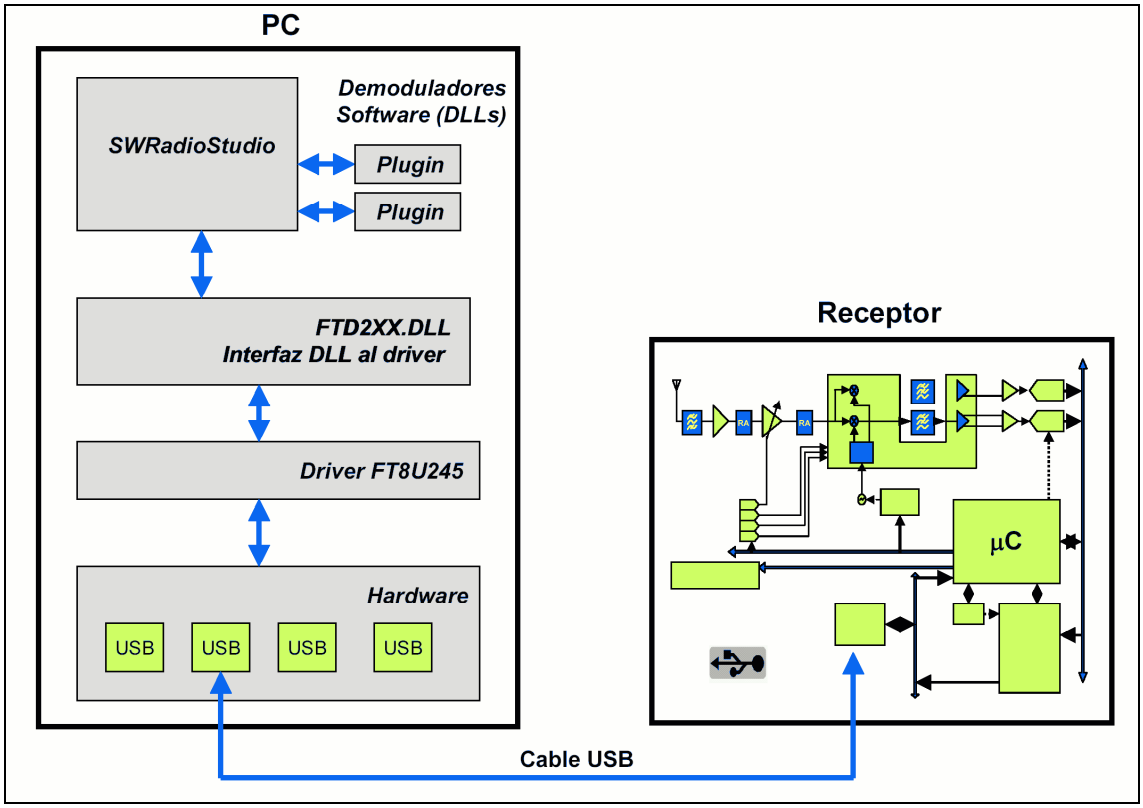


Figura 124:Esquema global del conjunto receptor-PC

La comunicación mediante la interfaz USB se basa en *pipes* (ver [3]), o canales de comunicación, donde los datos son flujos de *bytes* sin estructura. El manejo de estos canales está reservado a los drivers del dispositivo, que en el caso del chip FT8U245 se suministran de forma gratuita por el fabricante. Para acceder a los servicios ofrecidos por los drivers se ofrece junto con ellos una librería dinámica (FTD2XX.DLL) y ficheros de librería (.lib) para enlazar junto con los programas de usuario. De esta forma se puede acceder desde cualquier aplicación construida en Visual C++, C++ Builder, Delphi o Visual Basic al integrado USB.

La aplicación que hace uso de los servicios de los drivers USB es la aplicación principal de este proyecto, que se ha llamado *SWRadioStudio*. Este programa es una aplicación gráfica que pretende presentar de una forma amigable y completa todas las operaciones que se pueden realizar con el receptor. Básicamente, la aplicación se divide en dos interfaces gráficas y con un objetivo bien distinto cada una. Al arrancar el programa se muestra el "Panel de control", desde el que se puede acceder a estos dos módulos.

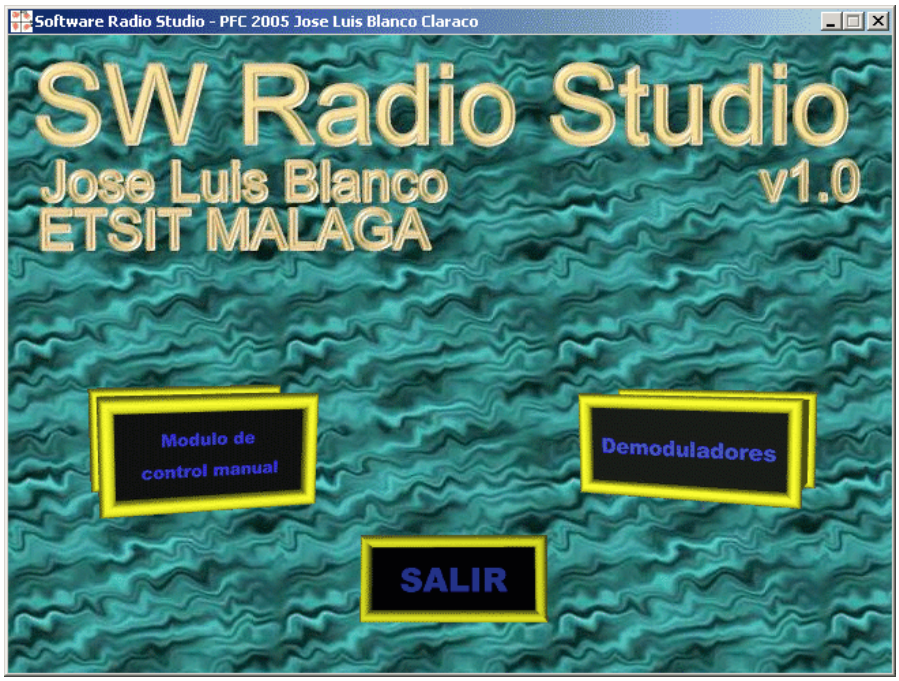


Figura 125:El panel de control de SWRadioStudio.

Este panel de control sólo contiene dos botones, que se han diseñado utilizando componentes gráficos personalizados como se ve en la imagen anterior. Cada uno de estos simplemente abre una de las dos ventanas correspondientes a los dos módulos en que se divide la aplicación.

El primero de estos módulos es el "módulo de control manual". Desde este módulo se tienen acceso a todos y cada uno de los dispositivos del receptor a través de unas sencillas interfaces gráficas y se puede arrancar y detener el modo de muestreo continuo, entre otras operaciones más que se verán en el siguiente apartado. Además se dispone de varias zonas gráficas donde se muestran las muestras recibidas en tiempo real, tanto en el dominio de las muestras como de la frecuencia. Este módulo ha sido diseñado con el objetivo principal de permitir de forma cómoda y flexible la realización de los diversos experimentos y medidas necesarios tanto para depurar como para caracterizar el receptor.

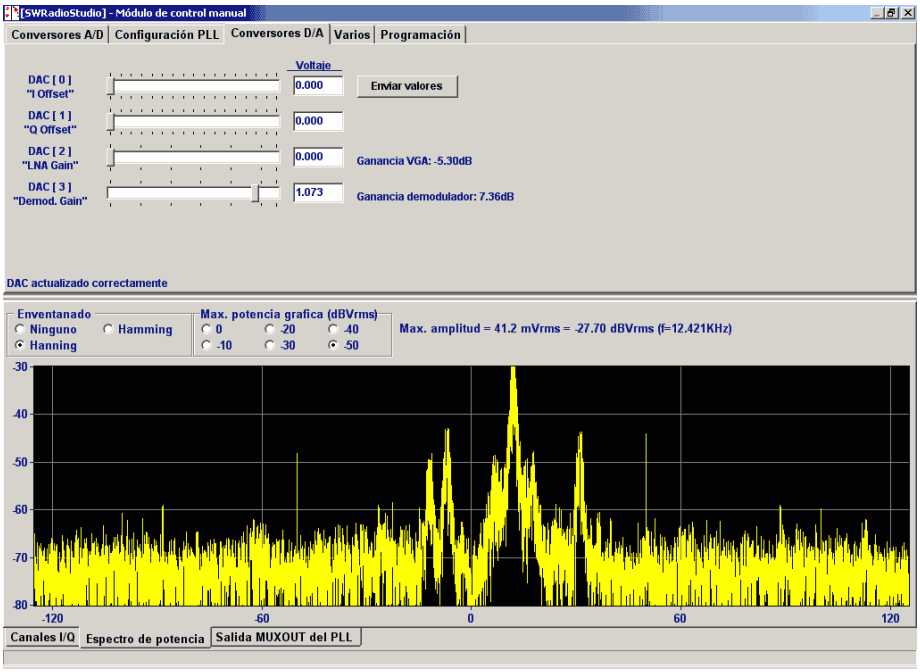


Figura 126:Captura de pantalla del módulo de control manual.

En cuanto a la otra parte de la aplicación, el "módulo de demodulación", es la encargada de ejecutar el procesado en tiempo real de la señal recibida y su demodulación, permitiendo entre otras cosas la visualización mediante gráficas de la señal y su espectro a su paso por las distintas partes del camino de la señal, la reproducción de la señal demodulada en forma de audio (para el caso de modulaciones analógicas) o el volcado de los datos a disco. Como se mostraba esquemáticamente en la figura del camino de los datos en el PC, los demoduladores software no forman parte de SWRadioStudio, sino que son librerías dinámicas externas, diseñadas con una determinada interfaz común que permite su uso a modo de *plugins* desde el módulo de demodulación de una forma fácil y sin necesidad de recompilación. Esta arquitectura

complica bastante el diseño de la aplicación, ya que además de las interfaces puramente dedicadas a la demodulación, se ha añadido la capacidad de que estos demoduladores software creen, mantengan y reaccionen a un interfaz gráfico que es ofrecido por la misma ventana del módulo de demodulación. Sin embargo, el incremento de complejidad merece la pena frente a la generalidad ganada y la capacidad de fácil ampliación futura con nuevos demoduladores.

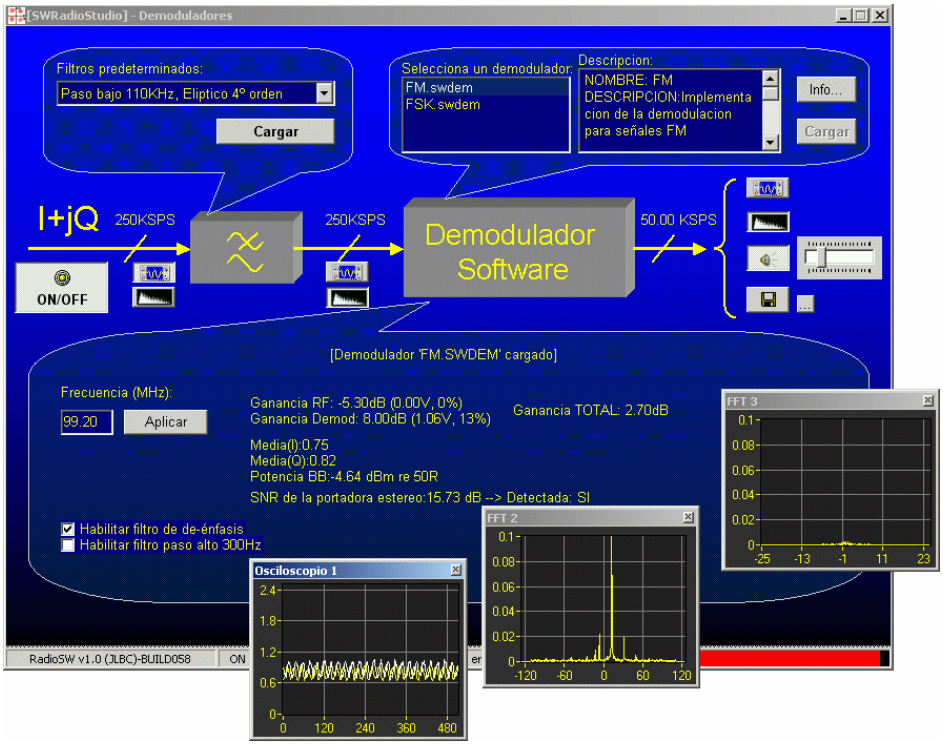


Figura 127:Captura de pantalla del módulo de demodulación.

Como se puede ver en la figura, la interfaz gráfica se ha mantenido alejada de la clásica de las aplicaciones estándar Windows, utilizándose los gráficos todo lo posible para aclarar todo lo posible al usuario la utilidad de los controles con sólo observar su situación en la pantalla.

En el siguiente apartado se expondrá el entorno C++ Builder 5.0 que ha sido el utilizado para desarrollar esta aplicación, pasando en los siguientes apartados a describir más detalladamente los dos módulos de que se compone ésta con todo detalle a nivel funcional, dejando para el último apartado los detalles de implementación.

7.3.2. Herramientas utilizadas

El desarrollo de SWRadioStudio se ha realizado íntegramente utilizando el entorno C++ Builder 5.0. Este entorno se compone de un compilador de lenguaje C++ y de un entorno de desarrollo integrado (IDE) totalmente visual. Los proyectos en C++ Builder se componen de "unidades" (*units*) que son el conjunto de un fichero de código (.cpp) más su fichero de cabecera (.h), lo que normalmente equivaldrá a una o varias clases. Este entorno está altamente adaptado para crear aplicaciones Windows de forma muy fácil y cómoda, ya que dispone de una librería de objetos C++, llamada VCL (*Visual Components Library*) que encapsulan prácticamente todo el API de Windows relacionado con objetos visuales: ventanas, botones, textos, controles, etc...

De hecho, un tipo especial de "unidad" son las que implementan ventanas ("forms" en la terminología de C++ Builder). Las ventanas se implementan a través de una clase que hereda del componente `TForm`. A una ventana se pueden añadir controles simplemente arrastrándolos desde la paleta de componentes predefinidos (zona 1 en la siguiente figura). Al hacer esto no solamente se añadirá el componente a la presentación visual (la zona 3 de la figura), sino que de manera automática se actualizará el código en los ficheros ".cpp" y ".h" correspondientes. Así mismo si se modifican los ficheros de código, se mostrarán los cambios en la interfaz visual en el momento.

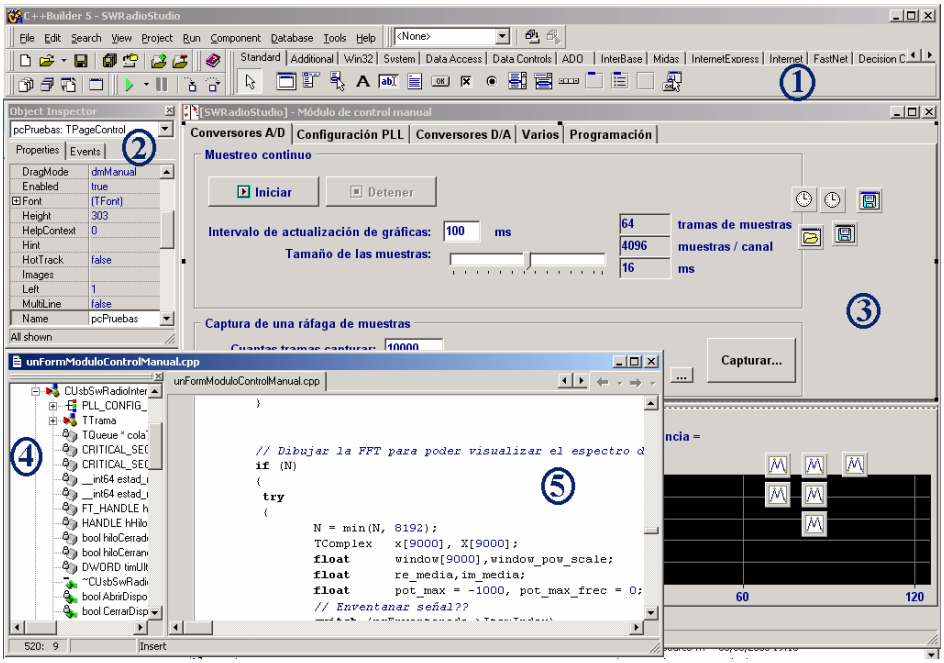


Figura 128:El entorno de trabajo de C++ Builder 5.

Los componentes de C++ Builder son clases de C++ pero con extensiones, como son los eventos y las propiedades. Tanto unos como otros pueden ser modificados de forma

visual desde el "explorador de objetos" (señalado como zona 2 en la figura). El interfaz también dispone de una zona de edición de código (indicada como zona 5) y de un explorador de clases, al estilo del existente en Visual Studio y otros entornos (la zona 4). Aunque C++ Builder introduce algunas ampliaciones más al lenguaje C++, éste tema no se tratará en esta memoria, ya que toda la implementación se ha realizado utilizando clases de C++ normales, en lugar de componentes de C++ Builder, por lo que la inmensa mayoría del código de SWRadioStudio es simplemente C++ estándar.

Debido a la integración de este entorno con Delphi, otro entorno de desarrollo rápido basado en el lenguaje *Object Pascal*, se da soporte a unidades escritas en lenguaje Pascal con objetos además de los escritos en C++. Esta característica se menciona porque sí que se hace uso de ella, aunque sólo para una "unidad".

Aunque este entorno es el único utilizado para desarrollar la aplicación, hay que decir que para el diseño de diversos filtros digitales se ha utilizado el entorno MATLAB, cuyos resultados se han incorporado en forma de datos constantes en la aplicación gráfica.

7.3.3. El módulo de control manual

En este apartado se va a describir al nivel funcional, sin entrar en cómo se ha implementado ninguna operación, lo que se deja para los apartados 7.3.5 y siguientes. La ventana se divide verticalmente en dos mitades. La parte inferior está dedicada a mostrar de manera gráfica las muestras recibidas del receptor, mientras que la superior está dedicada a los controles para configurar el receptor y otras opciones. Debido a la gran cantidad de acciones disponibles esta parte superior se ha dividido utilizando unas "pestañas" donde se han agrupado las acciones por grupos. Igualmente se puede elegir entre tres gráficas distintas en la parte inferior de la ventana mediante otras pestañas. A continuación se describen primero las distintas pestañas de controles, para posteriormente detallar la forma en que se representan los resultados en las distintas gráficas de datos recibidos.

Módulo de control manual, conversores ADC.

Esta pestaña presenta los controles necesarios para controlar el flujo de muestras de los canales analógicos I y Q, y por lo tanto es el canal de entrada principal desde el receptor.

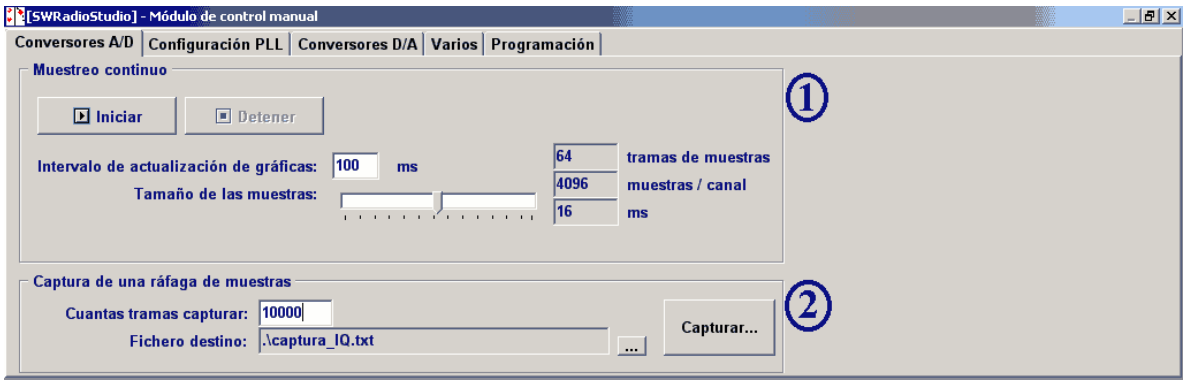


Figura 129:Módulo de control manual: Conversores ADC.

Como se puede ver en la figura, básicamente permite realizar dos tipos de acciones:

1. Muestreo continuo: Los botones de "Iniciar" y "Detener" tienen una relación directa con los comandos del firmware de iniciar y detener el modo de muestreo continuo. Aunque el hecho de iniciar el modo de muestreo continuo simplemente genera un flujo de tramas de muestras a una tasa fija desde el receptor, se permite mediante unos controles cambiar la forma y frecuencia en que estos se presentan gráficamente:
    - Se permite modificar el período de actualización de las gráficas. Mediante este parámetro, se puede conseguir una presentación gráfica más fluida reduciendo este período siempre que la capacidad del ordenador lo permita. Aunque el procesado de todas las tramas de datos no representa una carga significativa para un ordenador medio, su representación gráfica en tiempo real sí que lo hace, sobre todo teniendo en cuenta que no solamente se representa la señal en el dominio de las muestras, sino de la frecuencia, lo que requiere la ejecución de una FFT con su carga computacional asociada.
    - La cantidad de muestras a representar. Independientemente de la frecuencia de actualización de los gráficos se puede elegir la cantidad de muestras a representar cada vez. Esta cantidad se puede seleccionar

mediante un cursor móvil gráficamente, donde la unidad de medida es de tramas de muestras completas. Por lo tanto, se puede elegir entre representar 128 muestras (por canal) o 196, pero no 150. Como límite inferior se tiene una sola trama, es decir 64 muestras por canal, y como límite superior lógicamente se tiene el límite de representar todas las muestras en tiempo real, valor que resulta de calcular la cantidad de tramas que llegan durante un intervalo de actualización de gráficos.

2. Captura de una ráfaga de muestras: Esta operación permite el volcado directamente del flujo de muestras I y Q entrantes a un fichero, sin representación gráfica, lo que permite guardarlas en tiempo real sin ningún retardo ni pérdidas. Los parámetro a configurar son la cantidad de tramas de muestras a capturar y el fichero de destino. Las muestras se guardarán como un fichero de texto con dos columnas, donde las muestras I aparecerán en forma de valor de voltaje en la primera columna y las muestras Q en la segunda. Este modo de capturar ha resultado muy útil por ejemplo para realizar las pruebas que se mostrarán con diversas modulaciones digitales, ya que los datos así capturados pueden ser fácilmente procesados de una manera *offline* desde MATLAB para las modulaciones que no dispongan de demodulador software en tiempo real.

Módulo de control manual, configuración del PLL.

En esta pestaña se presentan desglosados casi todos los parámetros de que dispone el integrado sintetizador de frecuencias ADF4060-8.

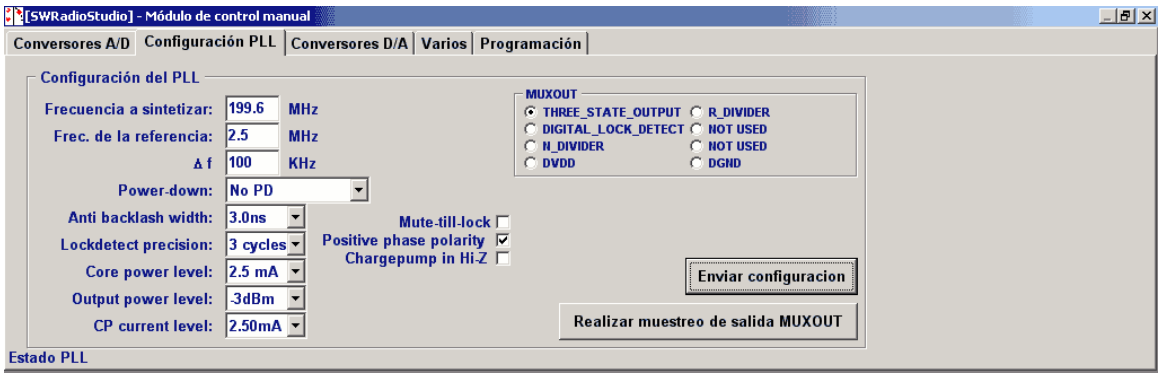


Figura 130:Módulo de control manual: Configuración del PLL.

Además de los controles que permiten la selección de los parámetros, se presentan dos botones con estos cometidos:

- Enviar configuración: Envía al receptor los comandos necesarios para cargar en el PLL la configuración seleccionada. Finalmente se muestra un mensaje informando si todo se ha realizado correctamente o si ha habido algún error en las comunicaciones.
- Realizar muestreo de la salida `MUXOUT`: Relacionado con la trama destinada a este fin, realiza al receptor una petición de muestrear esta salida del PLL, realizando 256 muestras cada 2μs. En el capítulo dedicado al plan de pruebas se muestran imágenes de los datos leídos según la señal seleccionada en el multiplexor de salida.

En cuanto a las distintas configuraciones disponibles, se encuentran:

- Frecuencia a sintetizar, frecuencia de la referencia y separación entre canales (Δf): Aunque en realidad los valores a configurar sean el registro B y R como se vio en el apartado dedicado al PLL, se permite introducir directamente la frecuencia a sintetizar deseada y la separación entre canales (la señal de referencia está fijada a 2.50Mhz por hardware y su valor no debe ser modificado) y el programa calculará automáticamente los valores de los registros B y R.
- Estado de *power-down*: Normalmente habrá que seleccionar "No PD" para que el integrado funcione y genere la señal deseada.
- *Core output power*: Es la potencia de salida RF deseada, regulable entre los niveles 0dBm, -3dBm, -6dBm y –9dBm. En el capítulo de medidas se mostrará realmente a qué potencias dan lugar cada una de estas configuraciones.
- *MUXOUT*: Se puede elegir entre un nivel lógico alto, uno bajo y la señal de detección digital de enganche, entre otras. En el capítulo de medidas se mostrarán ejemplos de las salidas que provocan cada una de estas y otras configuraciones.

Además de otras opciones que normalmente no se desearán variar, como por ejemplo la polaridad del detector de fase, que debe configurarse como positiva para que el PLL se enganche.

Módulo de control manual, conversores DAC.

En esta pestaña se permite variar mediante cursores deslizantes los valores de salida de cada uno de los cuatro conversores DAC del receptor.

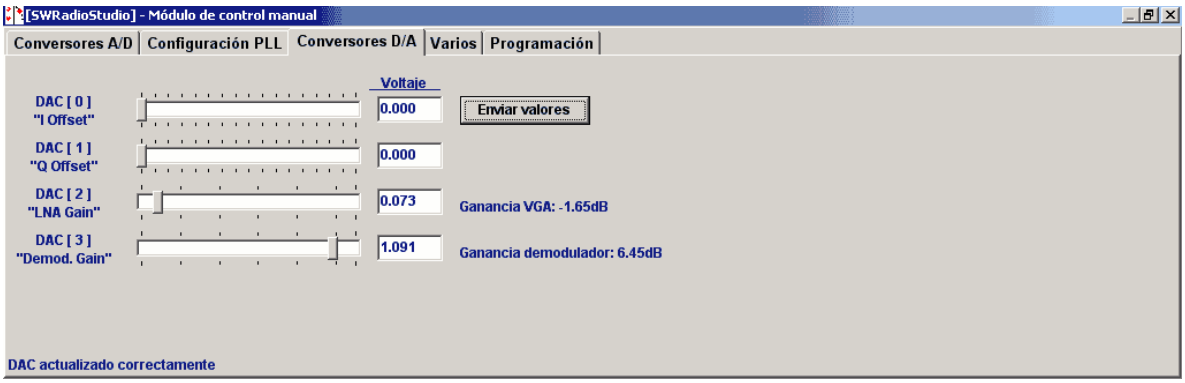


Figura 131:Módulo de control manual: Conversores DAC.

Como ya se ha comentado en la parte de diseño analógico, los dos primeros conversores inicialmente tenían el objetivo de anular las componentes de *offset* del demodulador, pero finalmente su uso se desechó y por lo tanto sus valores no tienen relevancia. En cuanto a los otros dos conversores, el tercero controla la ganancia del integrado VGA, amplificador de la señal de RF. Aunque los conversores DAC permiten salidas en todo el rango desde 0V a 5V, el rango permitido para la entrada del VGA es de 0V a 1V, pudiendo provocar daños al integrado de no cumplirse este límite. Por ello el control del cursor deslizando para este conversor solamente muestra este rango, por lo que el extremo derecho no corresponde a 5V, sino a 1V. De esta forma se evita el generar por error una señal que pudiera dañar el VGA. En cuanto al cuarto DAC, este controla la ganancia del demodulador, e igualmente su rango se ha limitado a los voltajes válidos, en este caso el rango que va desde 0.2V hasta 1.2V. En ambos casos se ha añadido a modo de guía un indicador textual que además del valor en voltaje generado, indica la ganancia en decibelios a la que, teóricamente, se están configurando tanto el VGA como el demodulador.

La actualización de los conversores se realiza en tiempo real mientras se desliza el cursor por la barra, enviando continuamente las tramas requeridas mientras el usuario cambie el valor deseado. De esta forma el efecto de variar la ganancia se puede observar instantáneamente en las gráficas inferiores, resultando esta herramienta de mayor utilidad.

Módulo de control manual, varios.

En esta pestaña se han añadido pequeñas operaciones de distinto tipo.

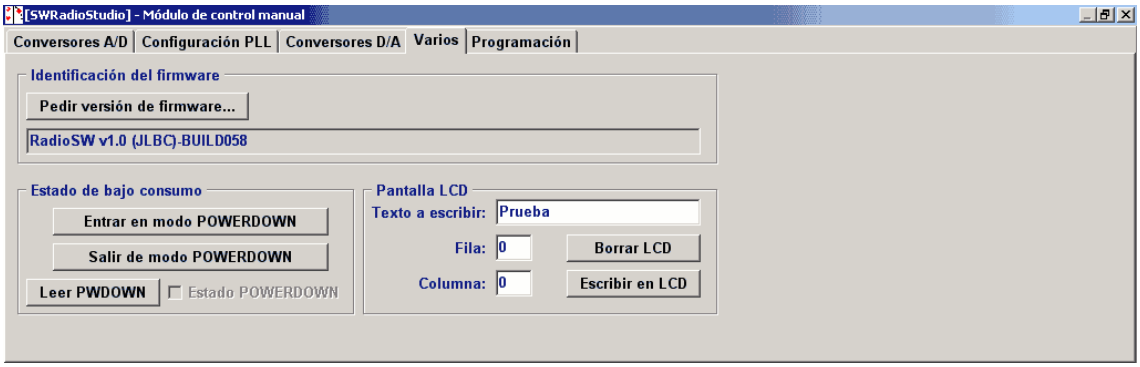


Figura 132:Módulo de control manual: Varios.

Estas operaciones son:

- Petición del identificador del *firmware*: Pulsando este botón se solicita al firmware la cadena de identificación, mediante una trama 0x10, y el resultado se muestra en un cuadro de texto. Este comando es útil para comprobar la correcta conexión del receptor al PC.
- Relacionados con el estado de bajo consumo: Existen tres botones, directamente relacionados con las dos tramas de manejo del estado de *power-down* del receptor:
    - Obtener el estado actual de *power-down*: Se pide al receptor el estado actual de esta línea, y se marca la casilla correspondiente si el resultado es que el receptor sí se encuentra en estado de bajo consumo.
    - Dos botones, uno para entrar y otro para salir del modo de *power-down*: Hay que recordar que el receptor siempre arranca en modo de ahorro de energía, por lo que es obligatorio ejecutar el comando de salir de este modo para realizar cualquier experimento desde el módulo de control manual (desde el módulo de demoduladores esto se realiza de forma automática y no es necesario hacerlo a mano).
- Relacionados con el control del display LCD: También directamente relacionados con las dos tramas existentes al respecto, existen dos botones para ejecutar:
    - Un borrado del display LCD.
    - La escritura de un texto en unas coordenadas, fila y columna, determinadas.

Módulo de control manual, programación.

En esta pestaña se encuentran las acciones relacionadas con el modo bootloader y la capacidad de auto-programación del receptor.

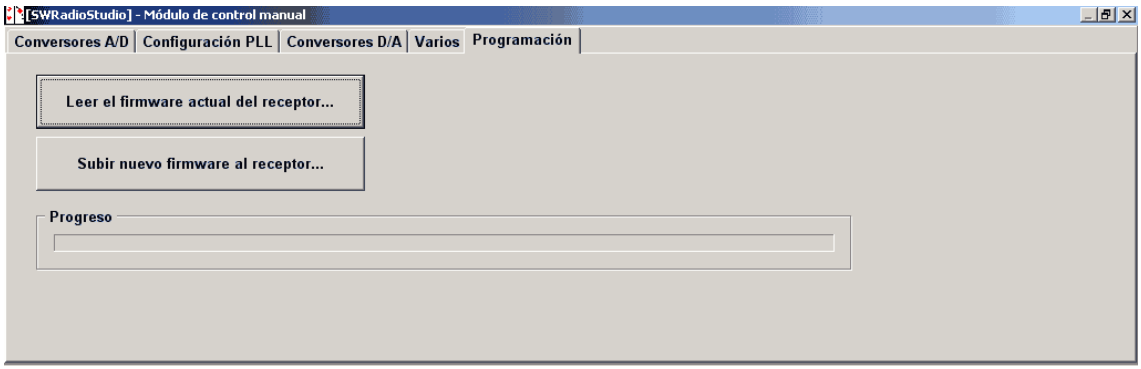


Figura 133:Módulo de control manual: Programación.

Aunque por transparencia y comodidad al usuario solamente aparecen dos botones relacionados, respectivamente, con la capacidad de descargar desde el receptor y subir a él un nuevo firmware, en realidad cada acción realiza estos pasos:

- Primero, entra en modo bootloader, mandando al receptor la trama 0x70 necesaria.
- Después realiza la lectura o grabación de la memoria flash de programa, según sea la acción seleccionada. Los ficheros de origen o destinos serán ficheros en formato *Intel HEX* que se seleccionan mediante un menú estándar de Windows de selección de ficheros.
- Tras realizar la operación, se envía una trama de salto al programa principal. De esta forma el receptor reinicia, con el nuevo firmware si se trataba de una grabación, y ya está preparado para responder a cualquier otro comando de las demás pestañas de este módulo de control manual. Hay que recordar que en modo bootloader no se procesan más que las cuatro tramas relacionadas con este modo, ignorándose el resto.

Ya se han examinado en detalle todas las pestañas de la parte superior de este módulo. A continuación se muestra el contenido de las pestañas inferiores, relativas a la representación de las muestras recibidas, tanto de datos como desde la señal MUXOUT del PLL.

Módulo de control manual, gráfico en el dominio de las muestras.

La primera de las tres pestañas se corresponde con las muestras de los canales I y Q representadas en el dominio de las muestras, o del tiempo discreto:

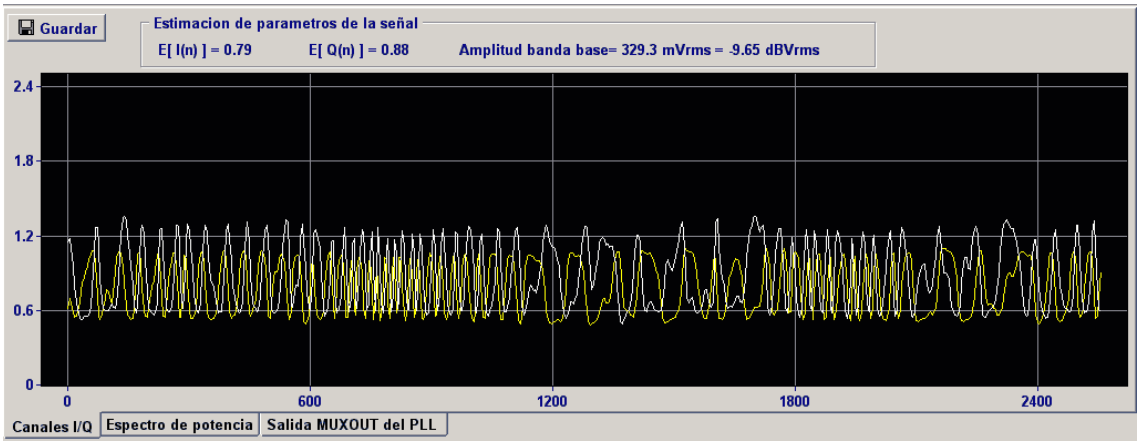


Figura 134:Módulo de control manual: gráfico en el dominio de las muestras.

Tanto la frecuencia de actualización de estas gráficas como la longitud, en número de muestras, son parámetros configurables desde la pestaña de los conversores ADC como se ha visto. Los valores mostrados son los voltajes medidos para cada canal (I y Q) en cada intervalo de tiempo, siendo la línea amarilla la del canal I y la blanca la del canal Q. Si llamamos t_k a los instantes de muestreo, I_{Bk} y Q_{Bk} son los valores binarios de ocho bits correspondientes a los valores de salida de los ADC, y I_k Q_k los valores en voltaje representados en las gráficas, se cumple que:

$$t_k = k \cdot 4\mu s$$

$$I_k = I_{Bk} \cdot 2.5 / 256 \text{ (Voltios)}$$

$$Q_k = Q_{Bk} \cdot 2.5 / 256 \text{ (Voltios)}$$

Donde la conversión de número de muestra a instante de tiempo se realiza multiplicando directamente por el periodo de muestreo, que es de 1 / 250KHz = 4µs, y la conversión a voltios se realiza conociendo que los valores de referencia de los conversores ADC eran 0V y 2.5V y su resolución de ocho bits.

En la parte superior de las gráficas se muestra un panel. En este se presenta un botón que permite guardar en un fichero de texto las muestras que actualmente se representan en las gráficas. Sin embargo, con el objetivo de guardar muestras a disco se recomienda usar la acción disponible en la pestaña de conversores ADC "Guardar ráfaga de muestras" ya que permite guardar una mayor cantidad de muestras sin necesidad de representarlas gráficamente, lo que ralentiza el proceso.

En este panel superior se muestran también tres medidas importantes. Dos de ellas son las medias de los canales I y Q, calculadas mediante:

$$\left|I_k\right| = \frac{\sum_{k=0}^{N-1} I_k}{N} \qquad \left|Q_k\right| = \frac{\sum_{k=0}^{N-1} Q_k}{N}$$

La tercera medida representa la amplitud cuadrática media (rms), no de las señales I y Q, sino de la señal en banda base compleja S_k:

$$S_k = I_k + j \cdot Q_k$$

Si representamos esta amplitud "rms" mediante σ_{Sk}, tenemos que (ver [11]):

$$\sigma^2{}_{Sk} = E\left(\left|S_k - \overline{S_k}\right|^2\right) \rightarrow \sigma_{Sk} = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1}\left|S_k - \overline{S_k}\right|^2}$$

$$\sigma_{Sk} = \sqrt{\frac{1}{N}\sum_{k=0}^{N-1}\left(I_k - \overline{I_k}\right)^2 \cdot \left(Q_k - \overline{Q_k}\right)^2} \text{ (Voltios rms)}$$

Esta es la fórmula utilizada para calcular la amplitud de la señal en banda base compleja en voltios *rms*. También se representa esta medida en unidades logarítmicas, para lo que se utiliza la relación:

$$\sigma_{Sk}\ (\mathrm{dBV_{rms}}) = 20 \cdot \log_{10}\left(\sigma_{Sk}\ (\mathrm{V_{rms}})\right)$$

Módulo de control manual, gráfico en el dominio de la frecuencia.

La segunda de las tres pestañas se corresponde con la estimación del espectro de la señal en banda base compleja S_k:

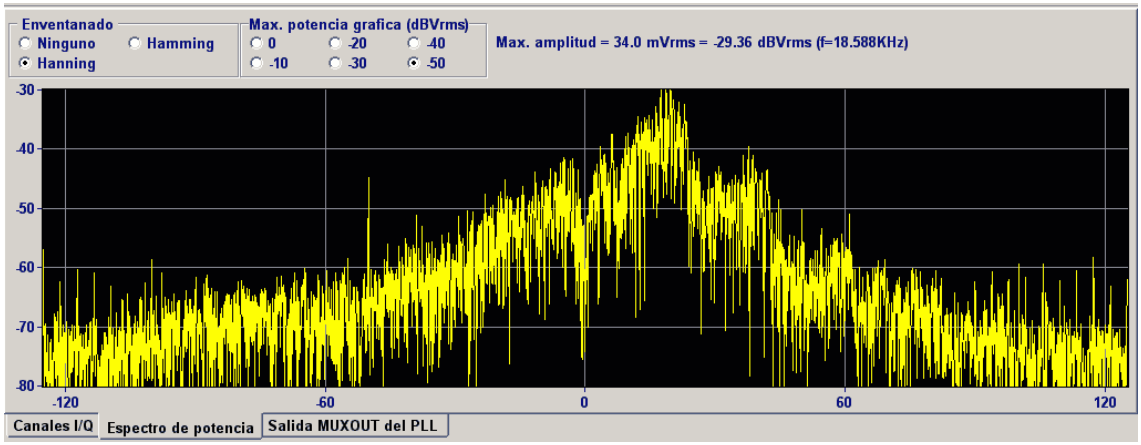


Figura 135:Módulo de control manual: gráfico en el dominio de la frecuencia.

Como se puede ver el eje de las frecuencias está normalizado a la frecuencia de muestreo, y abarca por lo tanto el intervalo [-125KHz, 125KHz]. Para realizar la estimación del espectro se utiliza la transformada de Fourier de la señal compleja enventanada:

$$SS_k = TF\{S_k \cdot W_k\}$$

Como se ve en la figura, en el panel superior se puede elegir entre tres tipos de ventanas:

- Ninguna: Se utilizará una ventana rectangular, es decir:

$$W_k = 1 \qquad k = 0,...,N-1$$

- Hanning: Es la seleccionada por defecto, ya que representa un compromiso entre resolución espectral y *spectral leakage*.

$$W_k = 0.5 + 0.5 \cdot \cos\left(\frac{2\pi}{N} \cdot k\right) \qquad k = 0,...,N-1$$

- Hamming: Es otra versión del coseno alzado, como la de hanning:

$$W_k = 0.54 + 0.46 \cdot \cos\left(\frac{2\pi}{N} \cdot k\right) \qquad k = 0,...,N-1$$

A partir de la transformada de Fourier, SS_k, se calcula la estimación de amplitud rms para cada frecuencia discreta k, como (ver [11]):

$$SS_k(V_{rms}) = \frac{\sqrt{2} \cdot \frac{|SS_k|}{N}}{\sqrt{F_P}}$$

Donde el factor corrector de potencia F_P depende del tipo de ventana usada, y vale 1.0, 0.5 y 0.54, respectivamente, para las ventanas rectangulares, hanning y hamming.

Gráficamente no se representa este valor directamente sino su valor en escala logarítmica:

$$SS_k\ (dBV_{rms}) = 20 \cdot \log_{10}\left(SS_k\ (V_{rms})\right)$$

Como ayuda a la hora de realizar medidas, se ha añadido además un indicador del pico máximo de potencia detectado, su valor y la frecuencia a la que se ha producido. Hay que señalar que al tratarse de una señal compleja, el espectro no tiene porqué ser simétrico.

Para mover esta gráfica en el sentido vertical se ha añadido un control para seleccionar la máxima potencia en dBV_{rms} a representar, a modo similar al control que a este efecto existe en los analizadores de espectros.

Módulo de control manual, gráfico de la señal MUXOUT.

La última de las tres pestañas gráficas sirve para mostrar el resultado del muestreo sobre la señal MUXOUT proveniente del PLL:

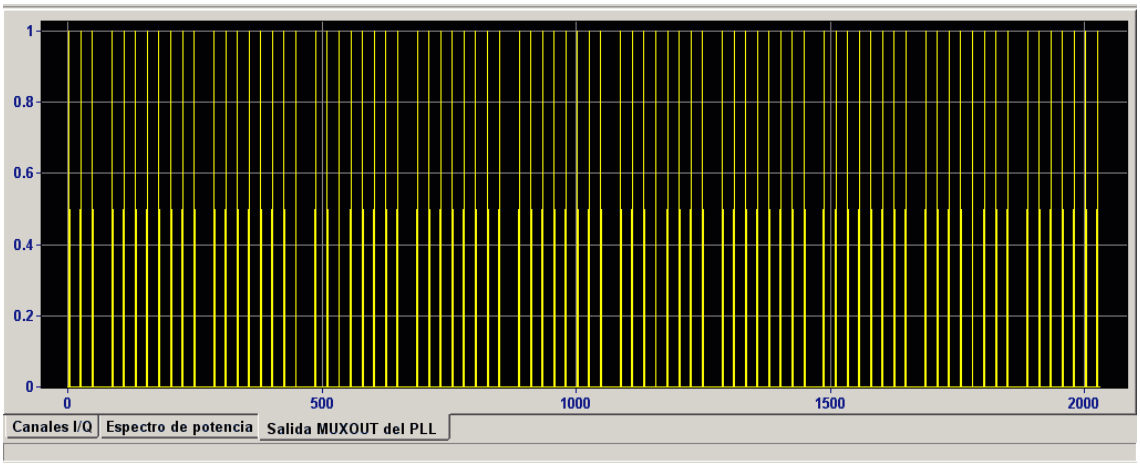


Figura 136:Módulo de control manual: gráfico de la señal MUXOUT.

Al representarse el contenido de una trama completa de muestras, en total se muestran:

256 bytes · 8 muestras / byte = 2048 muestras

En el capítulo dedicado al plan de pruebas se muestran las gráficas obtenidas para la señal MUXOUT para distintas configuraciones del PLL. De todas formas hay que indicar que este método de muestreo, aunque útil para averiguar si el PLL está enganchado por ejemplo, es demasiado lento como para muestrear correctamente algunas de las señales que se pueden seleccionar, como la del divisor de N. Estas señales se mostrarán más adelante capturadas con un osciloscopio en lugar de con éste método.

7.3.4. El módulo de demoduladores

Esta parte de *SWRadioStudio* es la encargada de demodular en tiempo real la señal sintonizada utilizando cualquiera de los demoduladores software disponibles. La ventana básicamente representa visualmente el camino de la señal, desde la entrada en banda base compleja que viene del receptor, su paso por un filtro digital paso bajo configurable y su demodulación mediante el demodulador seleccionado. Veamos detalladamente los controles de esta ventana:

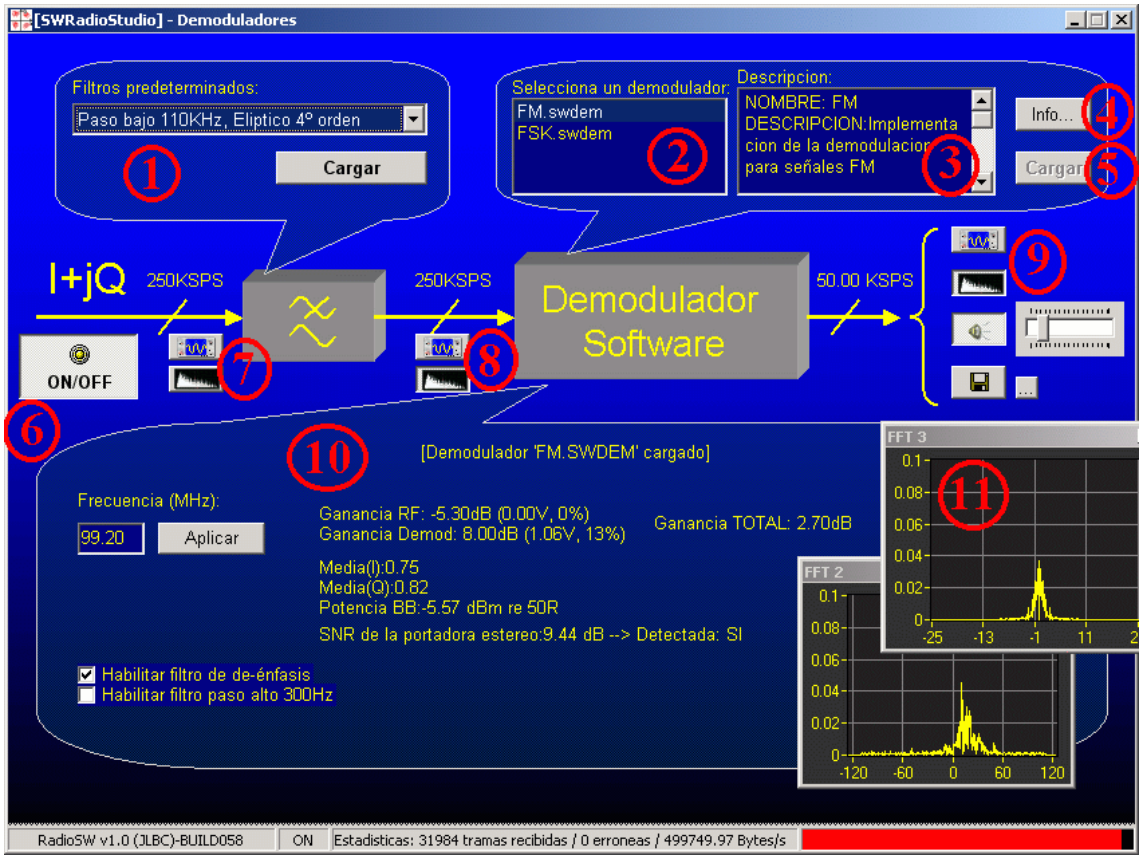


Figura 137:Partes del módulo de demoduladores.

1) Selección del filtro digital paso bajo. Este filtro es necesario para eliminar, al menos, la parte del espectro que no se puede asegurar que esté libre totalmente de aliasing. Por diseño, esta frecuencia es la de 110KHz. Además de esta frecuencia de corte, existen las de 100Khz, 75Khz, 50Khz y 25Khz. Todos ellos son filtros del tipo IIR, elípticos y de cuarto orden. El orden ha sido seleccionado para obtener un filtrado adecuado sin una carga computacional alta. Los filtros han sido diseñados usando MATLAB, usando el comando *elli*p, y usando como parámetros:

Atenuación de la banda eliminada: 20dB

Rizado en la banda de paso: 0.5dB

Las respuestas en frecuencia de estos filtros se muestran en las siguientes gráficas:

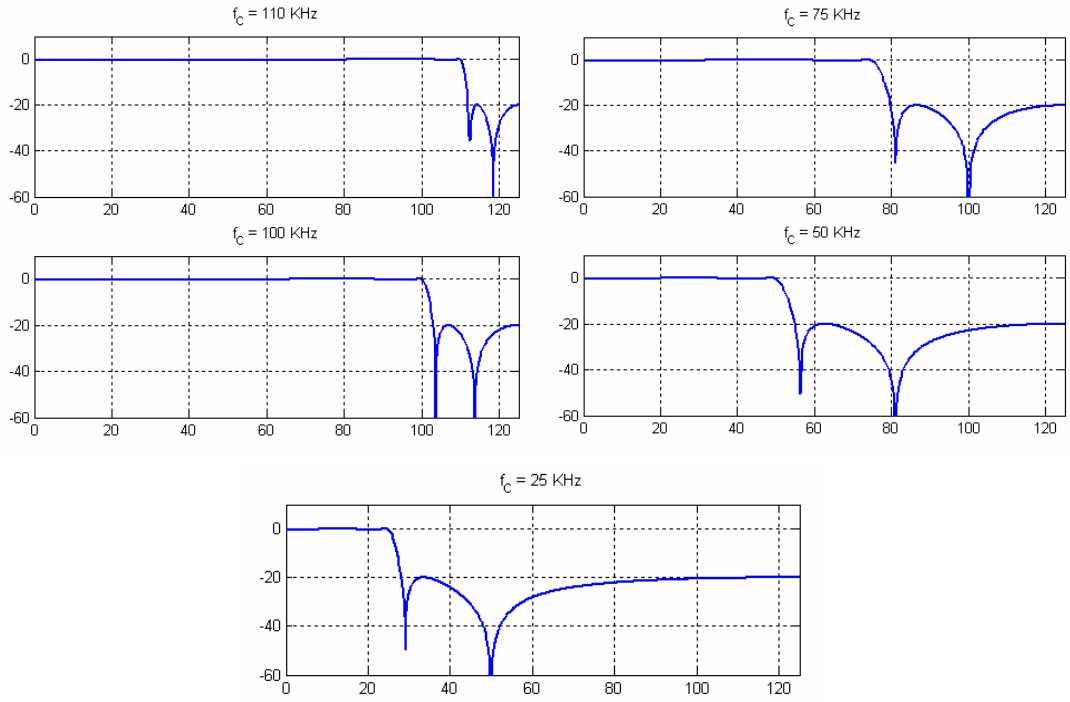


Figura 138:Filtros digitales paso bajo disponibles.

Para seleccionar cualquiera de estos filtros solo hay que seleccionarlos de la lista y pulsar el botón "cargar". Esta operación está disponible incluso durante el proceso de demodulación, pudiendo observarse el efecto del cambio de filtro en la señal demodulada.

2) Lista de demoduladores disponibles: Las librerías de demoduladores software se implementan mediante librerías DLL, pero con extensión ".swdem" en lugar de ".dll". El programa SWRadioStudio busca en el directorio "LibDemoduladores", desde el directorio de ejecución, para generar automáticamente esta lista.

3) Vista rápida del demodulador. Seleccionando un demodulador de la lista, éste se carga y se obtiene una descripción básica, que es la que se muestra en este cuadro. Si ocurriese algún error o el fichero no cumpliese el interfaz requerido, se mostrará un mensaje de error.

4) Pulsando el botón "Info...", se abrirá una ventana con todos los detalles del descriptor del demodulador, como se ve en la siguiente imagen:

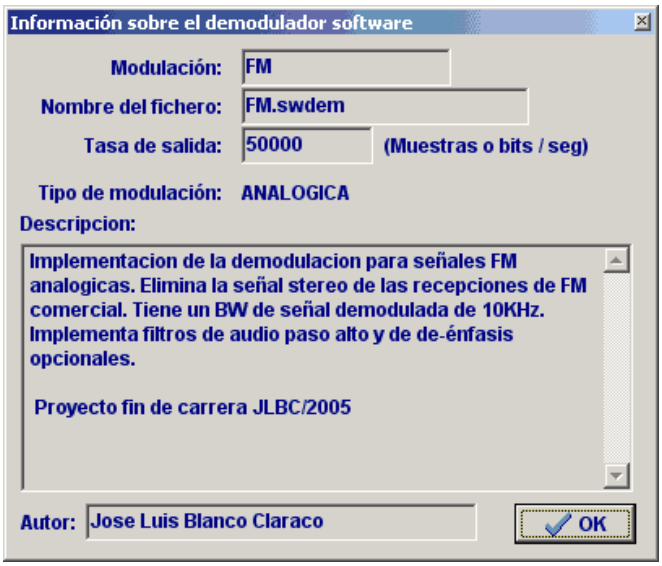


Figura 139:Ventana de información de demodulador.

5) Pulsando el botón "cargar", se carga e inicializa el demodulador marcado en la lista. Esto implica tanto la inicialización de sus variables internas, como la carga, si el demodulador software dispone de ella, de su interfaz gráfica en la parte inferior de la ventana.

6) Botón "ON/OFF". Este botón controla el arranque y parada del módulo de demodulación. Para poder arrancar, es necesario que se haya cargado un demodulador software. Si es así, se abrirá la conexión con el receptor, se sacará a este del modo de bajo consumo y se iniciará el modo de muestreo continuo. A su vez, se prepararán todos los hilos implicados en el camino de la señal, como se detallará más adelante.

7,8,9) Los botones de visualización del camino de la señal. En los tres puntos del camino de la señal (antes del filtro digital, tras él y después del demodulador), se pueden insertar unas "sondas virtuales", tanto en el dominio de las muestras como de la

frecuencia. Como ejemplo, se muestran a continuación tres de estas "sondas virtuales": la primera, en el dominio de la frecuencia, está tomada antes del filtro, mientras que la segunda, también en el dominio de la frecuencia, es de después del filtro. En este ejemplo el filtro tiene una frecuencia de corte de 75Khz. Se puede apreciar como en el segundo espectro la señal en frecuencias superiores a este valor se ha eliminado. La tercera señal muestra la señal demodulada, en el dominio del tiempo. Todas estas muestras se han tomado de una emisora de FM comercial, en la frecuencia 99.2MHz.

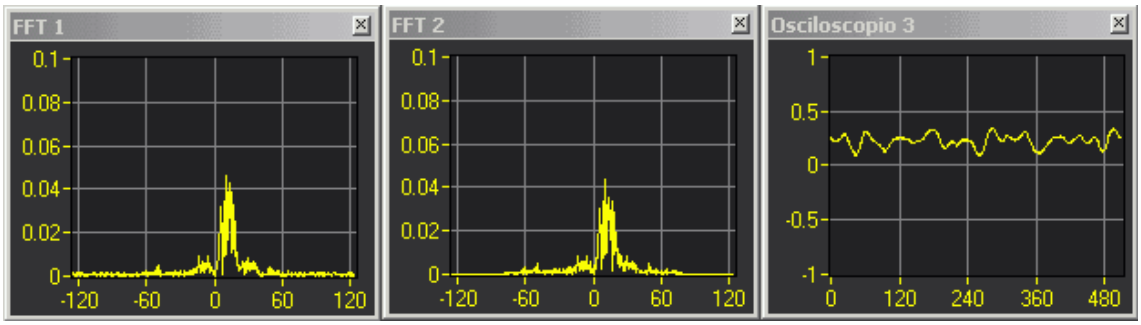


Figura 140:Ejemplo de "sondas virtuales" en el camino de la señal.

Para el caso de la señal demodulada existen otras dos opciones además de las "sondas virtuales":

- Salida de audio: Si se habilita, se abrirá el dispositivo de onda por defecto del ordenador, a la frecuencia de muestreo correspondiente a la tasa de salida del demodulador, y las muestras demoduladas se interpretarán como una señal de audio mono de 16 bits, siendo el fondo de escala el valor 1.0.
- Salida a disco: Se permite el volcado del flujo de muestras demoduladas a un fichero de texto. Esto puede ser útil para posteriores análisis de la señal en MATLAB, por ejemplo.

10) Esta es la zona de interfaz gráfica del demodulador. Su contenido depende del demodulador cargado.

11) Estas son las ventanas de "sondas virtuales" que ya se han visto. Aparecen como ventanas siempre visibles ("*always-on-top*") de forma que se pueden colocar donde menos molesten a la visualización de otras partes de la ventana.

7.3.5. El manejo de la interfaz USB

A partir de este apartado se van a tratar con detalle algunos de los puntos más relevantes de la implementación de *SWRadioStudio*. Debido a la extensión de esta aplicación, solamente se discutirán algunas partes. Todo el código fuente que puede resultar importante ha sido añadido en el anexo A5.3, ya que al estar ampliamente comentado es relativamente fácil de seguir.

En cuanto a este apartado, ha que decir que todo lo relacionado con la interfaz USB se ha encapsulado dentro del objeto `CUsbSwRadioInterfaz`. Esto incluye:

- Acceso a las funciones de la librería *FTD2XX.DLL* que hace de interfaz con los drivers del integrado FT8U245BM.
- Apertura y cierre del dispositivo automáticamente bajo demanda. Para la apertura del dispositivo es necesario utilizar un nombre, al igual que cuando se abre un fichero. Para esto se utiliza el descriptor que, como se vio en su apartado, se grabó en la memoria EEPROM externa del integrado y que lo identificaba en el administrador de dispositivos de Windows. El cierre del dispositivo se hace automáticamente tras unos segundos sin actividad.
- Un hilo de ejecución, que de forma paralela se encarga de la recepción de los bytes provenientes del receptor. En este hilo se realizan las comprobaciones para asegurar que se ha recibido una trama completa y con los *flags* de inicio y final correctos. Si esto es así, la trama se añade a una cola FIFO de tramas. Todas estas operaciones se realizan con estructuras de datos y memoria dinámica, para evitar el definir un tamaño máximo de forma rígida.
- Métodos para leer de la cola FIFO de tramas. Entre estos método y el hilo de recepción de tramas se utilizan secciones críticas para evitar conflictos en la escritura y lectura simultánea de la cola de tramas.
- Métodos, tanto síncronos como asíncronos, para ejecutar los distintos comandos disponibles sobre el receptor, como por ejemplo, borrar el LCD, cambiar un DAC, iniciar el modo de muestreo continuo, ... La diferencia entre los métodos síncronos y los asíncronos es que los primeros se esperan a recibir (vía la cola de tramas) la respuesta del receptor, mientras que los segundos no. Básicamente los asíncronos serán utilizados solo durante el modo de demodulación en tiempo real, en el que se cambian parámetros como la ganancia del receptor pero es más importante continuar

recibiendo tramas de muestras para que no se pierdan que esperar la respuesta de confirmación del cambio.

7.3.6. La carga de los demoduladores software

Los demoduladores software se han implementado mediante librerías DLL, las cuales deben tener una interfaz común para permitir el intercambio de demoduladores sin problemas. La interfaz estándar que estos demoduladores deben cumplir consta de estas funciones:

```
// ------------------------------------------------------------------------
// Declaracion de las funciones NECESARIAS que debe exportar una libreria
//   que implementa un demodulador software:
// ------------------------------------------------------------------------
extern "C" __declspec(dllexport) void  __fastcall SWDEMOD_GetInfo(
                        DemodSW::DescripcionDemodulador *info_out );
extern "C" __declspec(dllexport) void __fastcall SWDEMOD_Demod(
                        float *in,
                        DWORD nSamplesIn,
                        float *out,
                        DWORD &nSamplesOut,
                        DWORD nMaxSamplesOut,
                        CUsbSwRadioInterfaz     *USBSwRadioInterfaz );
extern "C" __declspec(dllexport) void __fastcall SWDEMOD_GUICallback(
                        char *name,
                        WORD event,
                        DWORD param1,
                        DWORD param2 );
extern "C" __declspec(dllexport) void __fastcall SWDEMOD_SetGUI_Functions(
                        DemodSW::GUI_Funciones *functions );
void SWDEMOD_Close();   // No se exporta, solo se llama desde la DLLEntryPoint
void SWDEMOD_Init();    // No se exporta, solo se llama desde la DLLEntryPoint
```

A continuación se describen brevemente:

- SWDEMOD_GetInfo: Esta función se utiliza para rellenar una estructura de datos con una descripción del demodulador. Entre otros datos se incluyen:
    - Una descripción textual del demodulador y sus características.
    - La tasa de salida, en muestras por segundo.
    - Si es un demodulador analógico o digital.

```
// Estructura usada para describir al demodulador.
// ------------------------------------------------------------------------
struct DescripcionDemodulador
{
        char    nombreModulacion[100];  // Texto para el nombre del modulo
        char    version[10];            // Texto para la version
        char    autor[100];             // Texto para el autor
        char    comentarios[2000];      // Comentarios varios.

        bool    analogica;              // Si se trata de una mod. analogica o
digital
        int     rateSalida;             // Muestras por segundo a la salida.
};
```

- SWDEMOD_Demod: Es donde se realiza la demodulación en sí. Como entrada hay un bloque de longitud variable de muestras I y Q, y como salida un bloque también variable de muestras de salida. Se puede observar cómo uno de los parámetros de entrada es una referencia al objeto encargado de la interfaz USB. Usando este objeto se realizan las tareas de realimentación hacia el receptor, como el control automático de ganancia o la selección de la frecuencia a sintonizar.

- SWDEMOD_SetGUI_Functions: A esta función se llama justo después de cargar un demodulador en el módulo de demoduladores, con una estructura como parámetro que contiene una lista de punteros a funciones para manejar el interfaz gráfico del demodulador. En esta función es donde el demodulador debe crear su interfaz gráfico si es que quiere crear uno. En esta lista de funciones se incluyen funciones como:
    - GUI_Create_Button: Utilizada para crear un botón con un nombre, un texto y una posición determinados.
    - GUI_Free_Component: Llamado cuando se va a descargar la DLL, para destruir todos los controles que se hayan creado al cargarse para la interfaz gráfica.
    - GUI_Change_Text: Usada para cambiar el texto que aparece sobre un control. Se ha utilizado por ejemplo para mostrar la ganancia del bucle de control de ganancia automática en cada momento.

```
// Estructura usada para pasarle al demodulador todas las funciones
//   que necesita para interaccionar con el entorno gráfico.
// ---------------------------------------------------------------------
struct  GUI_Funciones
{
        // Creacion de componentes:
        void (*GUI_Create_Button)(char *name,char *text, int X, int Y, int LX, int LY
);
        void (*GUI_Create_Label)(char *name,char *text, int X, int Y );
        void (*GUI_Create_Edit)(char *name,char *text, int X, int Y, int LX, int LY );
        void (*GUI_Create_CheckBox)(char *name,char *text, int X, int Y, bool
init_state);

        // Destruccion de componentes:
        void (*GUI_Free_Component)(char *name);

        // Cambio de componentes.
        void (*GUI_Change_Text)(char *name,char *new_text);
        void (*GUI_Get_Text)(char *name,char *new_text);
        bool (*GUI_Get_CheckBox_State)(char *name);
};
```

- SWDEMOD_GUICallback: A esta función se llama cuando ocurren eventos sobre los controles del interfaz gráfico del demodulador. Como parámetros se pasan el

nombre del control afectado y otros parámetros como por ejemplo el estado de marcado o no para controles de casilla.

La carga, utilización y descarga de DLLs de demoduladores se ha encapsulado en el objeto `CCargadorDemodDLL`. Este objeto carga automáticamente la DLL que se le indique como parámetro en su constructor, permitiendo su llamada de forma transparente a través de sus métodos, del mismo nombre que las funciones invocadas en la DLL, y liberando la librería al destruir el objeto. También se añaden métodos de seguridad, como bloques "`try { ... } catch (...)`" para capturar posibles excepciones o errores en accesos de memoria dentro de la librería cargada, evitando que se cierre la aplicación completa.

7.3.7. Hilos involucrados en el proceso de demodulación

Cuando se arranca el proceso de demodulación, desde el botón "ON/OFF" del módulo de demodulación, los datos provenientes del receptor pasan por varios hilos que trabajan de manera asíncrona. Con esto se consigue que la interfaz gráfica no reduzca la capacidad de proceso, que debe realizarse en tiempo real ya que sino los datos se acumularían sin límite. En la siguiente figura se muestra esquemáticamente este camino de los datos:

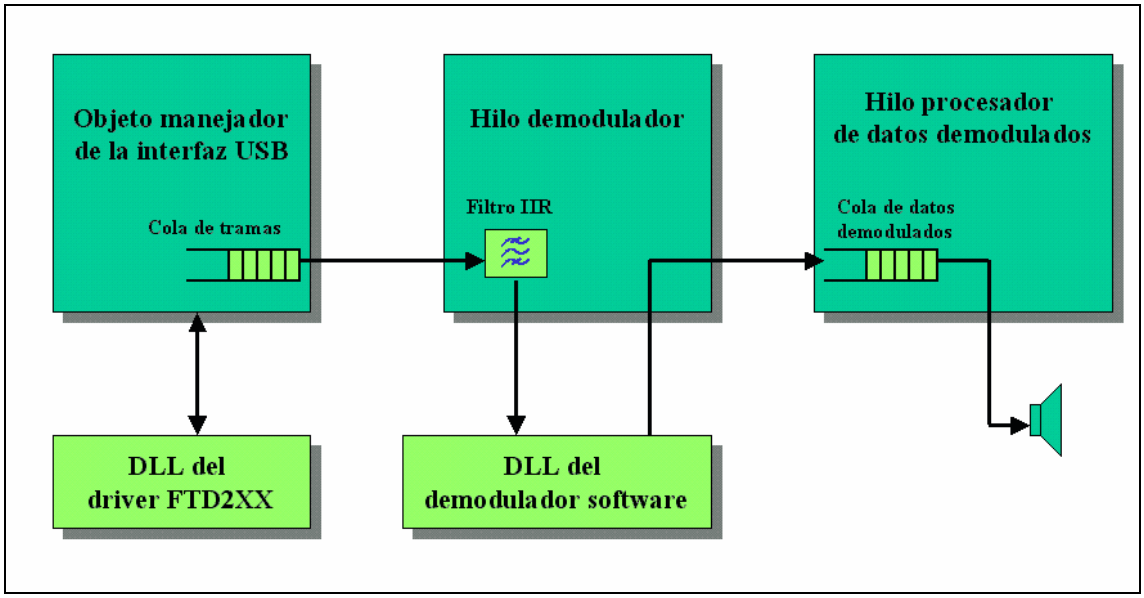


Figura 141:Hilos involucrados en el demodulado de la señal.

Ya se ha visto como existe una clase encargada de leer los bytes provenientes del integrado USB e ir formando tramas en un hilo independiente, para ir guardando éstas en una cola de tramas recibidas. Pues en el modo de demodulación, existe otro hilo, el hilo demodulador, que va cogiendo las tramas de esta cola, las filtra usando el filtro IIR seleccionado por el usuario, pasa el bloque de datos al demodulador software que se haya cargado y pasa los bloques de salida, con muestras demoduladas a la cola de un tercer hilo, el hilo procesador de datos demodulados. En este hilo es donde se forman bloques de un determinado tamaño y son enviados a reproducirse como audio a través de la clase CWaveOut. Un detalle a resaltar con respecto a la salida de audio es el *buffering* que se realiza para los primeros bloques de datos: no se comienza la reproducción hasta que no se tengan preparados un numero fijo (30 en este caso), de *buffers* de audio. Con esto se evita el indeseable efecto del sonido cortado, inevitable en estos casos en los que la tasa de entrada de datos es exactamente la misma que la de reproducción. El problema viene provocado por la necesidad de enviar *buffers* a reproducir al ritmo exacto con que se generan. Si no se tienen *buffers* de reserva, en cuanto uno de los bloques tarde un poco más en procesarse a la salida no habrá nada que reproducir, produciéndose un corte en el audio. Utilizando esta técnica de *buffering* se evita este efecto. La cantidad de *buffers* en cola de reproducción se puede visualizar en la esquina inferior derecha de la ventana de demodulación, en forma de barra roja. Normalmente esta barra estará rondando el máximo, indicando que hay suficientes *buffers* encolados y no hay problemas de corte de audio.

Aunque no se muestre en el esquema, cada hilo realiza más tareas. Por ejemplo, el hilo demodulador guarda muestras en unos *buffers* auxiliares destinados a la representación de la señal antes y después del filtrado en las “sondas virtuales”.

7.4. Demoduladores software

En este apartado se va a tratar de la estructura genérica que deben cumplir los demoduladores software para ser usados en el *SWRadioStudio*, así como del caso concreto del demodulador en tiempo real de FM que se ha implementado. A continuación se muestran otros demoduladores de modulaciones digitales que también se han implementado con éxito, pero en estos casos en MATLAB de una manera *offline*.

7.4.1. Partes comunes

Para crear un demodulador software compatible con SWRadioStudio se usará un proyecto de C++ Builder del tipo DLL. El esqueleto de todo demodulador constará de estos dos ficheros:

- "DLLEntry.cpp": En este fichero, común para todos los demoduladores, solamente se encuentra la función "DllEntryPoint" que es la que se llama cuando se carga y se libera una DLL por una aplicación anfitriona, en este caso *SWRadioStudio* a través del objeto `CCargadorDemodDLL`. Al cargarse se llama a la función `SWDEMOD_Init()`, y al liberarse al `SWDEMOD_Close()`, que ya no son comunes, sino que realizan tareas dependiendo del demodulador en particular.
- "Implementation.cpp": En este fichero es donde se implementan las cuatro funciones obligatorias que definen al demodulador, que ya se han enumerado en el apartado 7.3.6. El contenido de estas funciones es totalmente dependiente de la implementación.

Cada demodulador podrá contener tantos objetos, funciones o ficheros como sean necesarios, pero toda la relación con la aplicación *SWRadioStudio* se llevara a cabo solamente a través de estas funciones interfaces de la forma definida.

7.4.2. Demodulador FM (En SWRadioStudio)

En este apartado se describe el diseño conceptual de un demodulador en tiempo real para el entorno *SWRadioStudio*, para modulaciones analógicas en FM. En los siguientes sub-apartados se describirán diferentes aspectos de este módulo. En el anexo A5.4 se lista el código fuente de este demodulador extensamente comentado.

A) Demodulación de la señal

Dada una señal en banda base compleja en forma de partes reales e imaginarias:

$$S(k) = I(k) + j \cdot Q(k)$$

Su representación en módulo y fase viene dada por:

$$S(k) = A(k) \cdot e^{-j \cdot \phi(k)}$$
$$A^2(k) = I^2(k) + Q^2(k)$$
$$\varphi(k) = \text{atan2}\big(Q(k), I(k)\big)$$

Donde por tratarse de una modulación de fase, estamos interesados en el término $\varphi(k)$. Debido a que las señales I(k) y Q(k) no tienen media cero, al estar centradas alrededor de 1.0V, en realidad la señal $\varphi(k)$ deberá calcularse así:

$$\varphi(k) = \text{atan2}\Big(Q(k) - \overline{Q(k)}, I(k) - \overline{I(k)}\Big) \qquad k = 0,..., N-1$$

Como la señal de información va modula en frecuencia, no es fase, debemos derivar la fase para estimar la frecuencia. Como lo que tenemos son muestras de la fase, lo que se hace es:

$$d(k) = \varphi(k) - \varphi(k-1) \quad k = 1,..., N-1$$

Para corregir los casos especiales en que la diferencia angular sea en valor absoluto mayor que π, hacemos:

$$d(k) = 2\,\pi - d(k), \quad \text{si } d(k) > \pi$$
$$d(k) = -2\,\pi - d(k), \quad \text{si } d(k) < -\pi$$

Por lo tanto, ya tenemos la señal demodulada, pero a la misma frecuencia de muestreo que a la entrada, mientras que ya se puede reducir sensiblemente ésta por tener un ancho de banda más reducido que la señal modulada, es decir, diezmar la señal.

En este caso, la señal entra a una tasa de 250.000 muestras por segundo y canal, por lo que tenemos 250.000 muestras demoduladas por segundo a la salida. Para simplificar el proceso de diezmado, se va a utilizar un diezmado por un número entero. En este caso un diezmado por 5 deja una tasa de 50KSPS, con un ancho de banda de 25Khz, más que suficiente para una señal de audio.

B) Diezmado

Ahora bien, antes de diezmar una señal hay que asegurar que no se producirá *aliasing* espectral. Esto se asegura utilizando un filtro paso bajo anterior al diezmado de muestras.

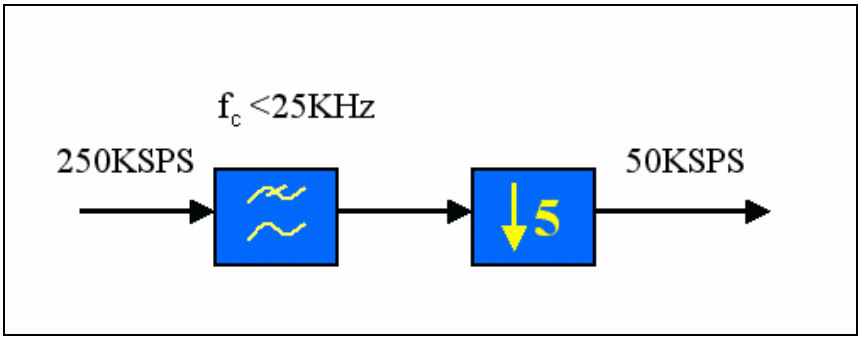


Figura 142:Diezmado de la señal a la salida del demodulador.

Aunque este filtro paso bajo debe tener una frecuencia máxima de 25Khz para evitar el aliasing, en este caso se utilizará un filtro más restrictivo. El objetivo es utilizar este filtro para dejar solamente la parte del espectro audible para señales de música, y así eliminar más ruido fuera de banda. En concreto, se ha utilizado un filtro con estas características:

Tipo de filtro: IIR, elíptico.

Orden: Sexto orden.

Frecuencia de corte: 9KHz.

Atenuación en banda eliminada: 40dB

Rizado en banda de paso: 1dB

Utilizando estos datos y la función *ellip* de MATLAB, se han obtenido los coeficientes del filtro que se han introducido en el demodulador, y cuya respuesta en frecuencia es:

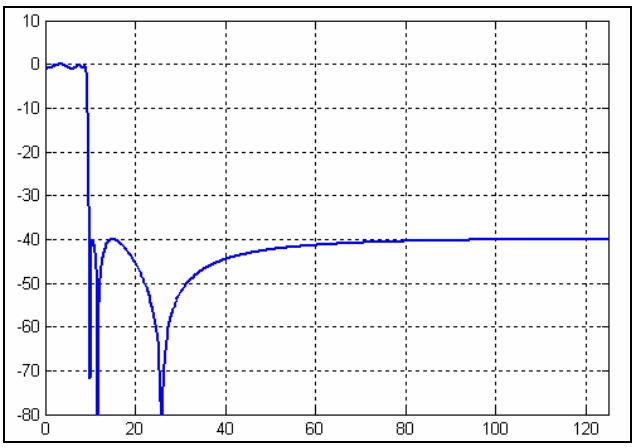


Figura 143:Respuesta del filtro diezmador.

C) Control automático de ganancia

El receptor provee de dos parámetros para controlar la ganancia total, y por tanto, la potencia recibida en banda base: la ganancia del VGA en RF y la ganancia del demodulador. Para realizar un control automático de ganancia (CAG) es necesario estimar la potencia de señal recibida. Esto se realiza en dos pasos. Primero se estima potencia instantánea del bloque de datos que está siendo demodulado. Para eso se calcula la varianza de la señal I y de la señal Q. Una estimación de la potencia de la señal compleja en banda base se calcula de esta forma:

$$\mathrm{Pot(dBm)} = 10 \cdot \log\left(\frac{\sigma^2{}_I + \sigma^2{}_Q}{50\Omega} \cdot 1000 \right)$$

Una vez obtenida esta potencia instantánea, se procede a filtrarla para obtener una estimación de la potencia media:

$$\hat{\mathrm{P}}(\mathrm{dBm}) = \hat{\mathrm{P}}(\mathrm{dBm}) \cdot 0.95 + \mathrm{Pot(dBm)} \cdot 0.05$$

Este valor es el que se muestra en la interfaz gráfica del demodulador. Experimentalmente se ha comprobado que un margen de potencias óptimo para la señal demodulada es el de –7dBm a –3dBm. Para mantener la potencia recibida dentro de este margen, se ejecuta el siguiente algoritmo de control de ganancia:

- Inicialmente, se comienza con una ganancia mínima.
- Si la potencia recibida es menor que la mínima deseada:
    - Aumentar en 1dB la ganancia del demodulador, o en caso de que ya sea máxima, la del VGA.
- Si la potencia recibida es mayor que la máxima deseada:
    - Disminuir en 1dB la ganancia del VGA, o en caso de que ya sea mínima, la del demodulador.

Se ha utilizado la equivalencia de 1dB de ganancia con un escalón de los conversores ADC (aproximadamente 20mV) para establecer el incremento de voltaje en ese valor exactamente, resultando un CAG por lo tanto de 1dB de resolución. Aparte de esto, se incluye un contador para que este CAG sólo se ejecute cada 100ms, aproximadamente. Esto se ha hecho así para dar tiempo tanto a que el comando de cambio de ganancia se ejecute, como para que las nuevas estimaciones de potencia media se actualicen con el nuevo nivel de la señal.

D) Detección de portadora de estéreo

Las emisiones de FM comercial en estéreo utilizan una subportadora, dentro de la señal ya demodulada, a una frecuencia de 19Khz. Esta se utiliza para demodular la señal estéreo que se sitúa en torno a los 38Khz. En este proyecto no se ha tratado la señal estéreo, pero sí que se ha diseñado un mecanismo para la detección de la presencia de esta portadora, cuya presencia aparece de forma muy nítida como se verá en el capítulo de resultados.

Para detectar esta portadora se utiliza el algoritmo de Goertzel. Este algoritmo se ha implementado mediante una clase ("`CGoertzel`") que extrae la componente frecuencial para una frecuencia discreta determinada. Para determinar la presencia de la portadora de estéreo se utilizan dos objetos `CGoertzel` a la vez: Uno, sobre la frecuencia de 19.0KHz, y el otro sobre la 18.5Khz. En ambos casos se utiliza un tamaño de bloque de 2500 (en este algoritmo no tienen porqué usarse potencias de dos, como en la FFT), dando lugar a una resolución espectral de 250KHz / 2500 = 100 Hz.

Cada vez que se obtiene un resultado de estos procesos, lo que ocurrirá una de cada 2500 muestras, se calculará el módulo de las componentes frecuenciales complejas, se filtrarán usando un filtro paso bajo (con idea de utilizar valores de potencia medios), y se calculará la potencia relativa de la componente a 19KHz con respecto a la de 18.5Khz, en decibelios. Este valor se mostrará en la interfaz gráfica del demodulador, además de una decisión sobre cuando se ha detectado o no portadora. El criterio adoptado es decidir que sí hay portadora si hay una diferencia de, al menos, 3dB.

7.4.3. Demodulador ASK (En MATLAB)

En el caso de los demoduladores *offline*, se han desarrollado en MATLAB, simplemente como "prueba de concepto", para demostrar que la demodulación de este tipo de señales es factible con el receptor desarrollado, pero sin llegar a desarrollar un demodulador en tiempo real. Por lo tanto, tanto este como los siguientes demoduladores no son tan elaborados como el de FM, sino que simplemente se han desarrollado a modo de "prototipos" de posibles demoduladores para *SWRadioStudio*.

En el caso del demodulador de ASK, la característica de la señal que nos interesa en este caso es la amplitud de la envolvente compleja. Este valor puede calcularse como:

$$a(k) = \sqrt{(I(k) - \overline{I(K)})^2 \cdot (Q(k) - \overline{Q(K)})^2}$$

Tras obtener la envolvente la señal es filtrada por un filtro paso bajo, elíptico, de cuarto orden, 40dB de rechazo en la banda eliminada y frecuencia de corte ligeramente por encima de la tasa de símbolo, que para este caso tendrá el valor fijo de 5KBaud.

Tras el filtrado la señal es muestreada a la tasa de símbolo, decidiendo si el símbolo es un cero o un uno según la amplitud sea superior al valor medio de la señal. Los resultados de este demodulador se muestran, junto con los del resto de siguientes demoduladores, el capítulo dedicado a medidas y resultados.

7.4.4. Demodulador 2-FSK (En MATLAB)

En este caso se ha implementado un demodulador de una señal 2-FSK. El proceso de demodulación es idéntico al descrito para el demodulador de FM, hasta el filtro de diezmado. En este caso ese filtro se sustituye por uno de frecuencia de corte ligeramente superior a la tasa de símbolo, que de nuevo en este caso es de 5KBaud. En este caso la señal filtrada también se muestrea a la tasa de símbolo, pero en este caso se decide si es un cero o un uno según la señal esté por debajo o por encima, respectivamente, de cero.

7.4.5. Demodulador BPSK (En MATLAB)

Como ejemplo de una modulación digital coherente, se ha realizado un demodulador para señales BPSK. En esta modulación, al ser coherente, es necesario recuperar la frecuencia exacta de la portadora. Aunque la demodulación tenga una frecuencia intermedia nominal nula, en realidad siempre habrá una pequeña frecuencia intermedia, y normalmente será tan pequeña que se notarán variaciones con el tiempo debido, por

ejemplo, a pequeñas variaciones en los osciladores locales. Por lo tanto lo primero que se ha hecho en esta modulación es estimar la frecuencia intermedia residual, lo que se ha hecho a mano directamente en MATLAB, por tratarse solamente de un prototipo de demodulador. Una vez estimada esta frecuencia, ω_{FI}, se multiplica la señal en banda base compleja por una portadora compleja para desplazar la señal en torno a la frecuencia cero:

$$S'(k) = S(k) \cdot e^{-jk \cdot \omega_{FI} / \pi}$$

Señal de la que se puede extraer la fase mediante:

$$\varphi(\mathrm{k}) = \mathrm{atan2}\left(\mathrm{Imag}\{\mathrm{S'(k)}\}, \mathrm{Real}\{\mathrm{S'(k)}\}\right)$$

En este demodulador los símbolos se han decidido estableciendo un umbral en esta fase, y, tras muestrearla a la frecuencia de símbolo, decidiendo los símbolos como cero o uno según caigan a un lado u otro del umbral de fase.

También para este demodulador coherente, aún no estando automatizado, se han obtenido muy buenos resultados, como se mostrará en el capítulo de medidas y resultados.

8. DISEÑO DEL PROGRAMADOR

8.1. Introducción

En este apartado se trata el diseño de una herramienta auxiliar, cuyo desarrollo ha sido necesario durante la construcción del proyecto. Se trata del programador para el microcontrolador ATMEGA8515. Su objetivo principal es grabar en la memoria flash del microcontrolador la imagen generada por el ensamblador usado para desarrollar el firmware (apartado 7.2).

8.2. Diseño hardware

En el *datasheet* del microcontrolador ATMEGA8515 se describen las dos formas de programarlo en las páginas 177-194. Básicamente, existen dos formas: usando un bus paralelo de ocho bits o mediante un bus serie SPI. Por la mayor simplicidad del hardware necesario se ha optado por el modo de programación serie.

Este modo de programación está accesible mientras la señal `/RESET` del microcontrolador está a nivel bajo y por tanto, el programa no se está ejecutando. Son necesarias solamente cinco señales, que son:

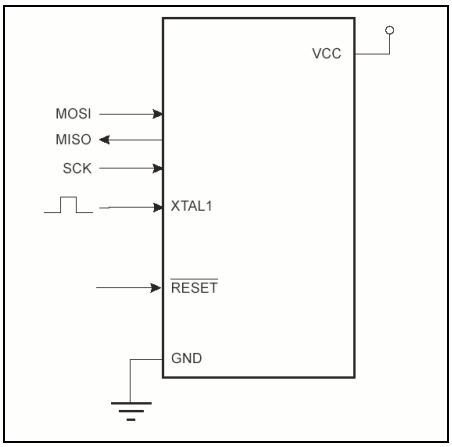


Figura 144: Señales usadas para programar el microcontrolador en el modo SPI

- `MOSI` (*Master Out Slave In*): Señal de datos de entrada del bus SPI.
- `MISO` (*Master In Slave Out*): Datos de salida del bus SPI.
- `SCK` (*Serial Clock*): Señal de sincronización del bus SPI.

- `XTAL`: Reloj para funcionamiento del microcontrolador. Debe tener una frecuencia mínima del doble de la señal SCK.
- `/RESET`: Esta señal debe ser mantenida a nivel bajo para usar el modo de programación.

Para manejar estas señales, se ha creado una placa donde estas señales están conectadas a un puerto paralelo de PC, como se ve en el esquema de abajo. De este puerto se obtiene también la alimentación a partir de salidas digitales a nivel alto, a través de tres resistencias de 100Ω para evitar posibles cortocircuitos.

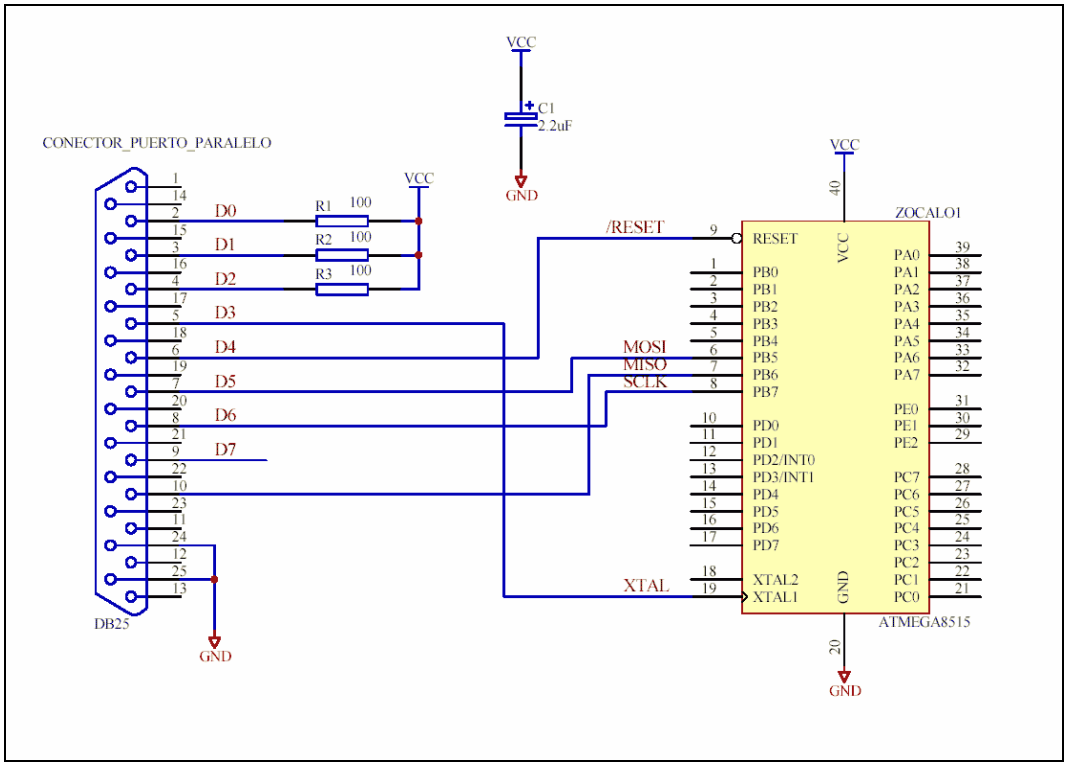


Figura 145: Esquema del programador del microcontrolador.

Para entender la conexión de las señales al puerto del PC a continuación se expone el uso de cada uno de los pines utilizados del puerto paralelo:

Señales	Pin	Uso
D[0..7]	2-9	Es un puerto de ocho bits de sólo salida (Al menos si no se usan las extensiones ECP/EPP)
GND	19-25	Masa
ACK	10	"*Acknowledge*" en el protocolo para impresoras. Para este proyecto, se usa como una entrada digital al PC, que ira conectado a "`MISO`"

Con todo esto ya se puede controlar cada una de las señales necesarias para programar el microcontrolador. En el siguiente apartado se expone el protocolo necesario para esta programación y en el siguiente la creación de una interfaz gráfica que permite usar el programador.

8.3. Protocolo de programación

Las operaciones básicas que se pueden hacer sobre el microcontrolador en el programador se listan a continuación, y son un reflejo de los conjuntos de comandos de programación que se definen en el *datasheet*:

- Identificarlo: Ya que todos los microcontroladores de la familia "*Atmel AVR*" (entre la que está la familia *ATMEGA*) usan el mismo protocolo de programación, cada uno dispone de una firma de tres *bytes* que identifica el modelo exacto de microcontrolador. Aunque en este proyecto solo se usa el modelo ATMEGA8515, cuya firma es 0x1E9306, este comando es útil para comprobar la correcta conexión del microcontrolador al programador y de este al puerto paralelo del PC.
- Borrarlo: La memoria flash del microcontrolador solamente se puede volver a grabar tras un borrado completo de esta. Por lo tanto este comando debe ser ejecutado siempre antes de una grabación de la memoria flash. Con este comando todos los *bytes* de la memoria flash y la EEPROM se borran a un estado de 0xFF.
- Grabar la flash: Se graba en el microcontrolador los 8Kb de flash desde un fichero con formato *Intel HEX*, que es uno de los formatos en que el ensamblador produce la imagen de memoria de programa.
- Leer la flash: Este comando es útil para verificar que la grabación se ha realizado correctamente. Se lee toda la memoria flash y se graba en un fichero *Intel HEX*.
- Grabar la EEPROM: Permite grabar el contenido de los 512 *bytes* de la memoria EEPROM. Esta memoria sí se puede volver a grabar sin tener que borrar todo el chip. En este proyecto esta memoria no se ha utilizado, pero se ha dejado la opción de grabarla desde el programador para posibles usos futuros.
- Leer la EEPROM: Permite leer esta memoria y volcarla a un fichero *Intel HEX*.

- Leer y grabar los bits de configuración: Este microcontrolador dispone de 16 bits de configuración, cuyo contenido y uso en este proyecto se expuso en el apartado 6.9.

El protocolo de comunicación para la programación es el del bus SPI, bus serie con una línea de datos de entrada (`MOSI`) y otra de salida (`MISO`). En este tipo de bus la transmisión y recepción de datos van asociados ya que siempre se realizan simultáneamente, aunque según el caso solamente serán significantes los datos de entrada o los de salida. En el caso de la programación del microcontrolador, los datos se transmiten siempre en grupos de cuatro *bytes*. El contenido del primero indica el tipo de comando pedido al microcontrolador, mientras que el resto de los *bytes* pueden llevar datos de entrada o de salida. En la siguiente tabla se resumen todos los comandos posibles y que se han implementado en la interfaz gráfica del programador:

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx	Enable Serial Programming after RESET goes low.
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase EEPROM and Flash.
Read Program memory	0010 **H**000	0000 **aaaa**	**bbbb bbbb**	**oooo oooo**	Read **H** (high or low) data **o** from Program memory at word address **a**:**b**.
Load Program memory Page	0100 **H**000	0000 xxxx	xxx**b bbbb**	**iiii iiii**	Write **H** (high or low) data **i** to Program memory page at word address **b**. Data low byte must be loaded before Data high byte is applied within the same address.
Write Program memory Page	0100 1100	0000 **aaaa**	**bbb**x xxxx	xxxx xxxx	Write Program memory Page at address **a**:**b**.
Read EEPROM Memory	1010 0000	00xx xxx**a**	**bbbb bbbb**	**oooo oooo**	Read data **o** from EEPROM memory at address **a**:**b**.
Write EEPROM Memory	1100 0000	00xx xxx**a**	**bbbb bbbb**	**iiii iiii**	Write data **i** to EEPROM memory at address **a**:**b**.
Read Lock bits	0101 1000	0000 0000	xxxx xxxx	xx**oo oooo**	Read Lock bits. “0” = programmed, “1” = unprogrammed. See Table 81 on page 177 for details.
Write Lock bits	1010 1100	111x xxxx	xxxx xxxx	11**ii iiii**	Write Lock bits. Set bits = “0” to program Lock bits. See Table 81 on page 177 for details.
Read Signature Byte	0011 0000	00xx xxxx	xxxx xx**bb**	**oooo oooo**	Read Signature Byte **o** at address **b**.
Write Fuse bits	1010 1100	1010 0000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 84 on page 179 for details.
Write Fuse High Bits	1010 1100	1010 1000	xxxx xxxx	**iiii iiii**	Set bits = “0” to program, “1” to unprogram. See Table 83 on page 178 for details.
Read Fuse bits	0101 0000	0000 0000	xxxx xxxx	**oooo oooo**	Read Fuse bits. “0” = programmed, “1” = unprogrammed. See Table 84 on page 179 for details.
Read Fuse High Bits	0101 1000	0000 1000	xxxx xxxx	**oooo oooo**	Read Fuse high bits. “0” = programmed, “1” = unprogrammed. See Table 83 on page 178 for details.
Read Calibration Byte	0011 1000	00xx xxxx	0000 0000	**oooo oooo**	Read Calibration Byte

Figura 146: Tabla de comandos de programación.

En la tabla, los bits marcados como "i" son datos de entrada, "o" datos de salida, "x" bits cuyo valor no importa y "a" y "b" los bits altos y bajos, respectivamente, de una dirección de memoria a leer o escribir. El *bit* "H" en comandos de acceso a la memoria flash indica si leer el *byte* alto (1) o el bajo (0) de la dirección dada, ya que las direcciones de flash son direcciones de palabra, que son de 16 bits.
La memoria flash está dividida en páginas de 64 bytes, y se programa de página en página. Esto se hace mediante un buffer temporal de que dispone el chip internamente. Primero se debe ejecutar 64 veces el comando "*Load program memory page*" para cargar esta página temporal, y posteriormente ejecutar al comando "*Write program memory page*" para que realmente se graben en la memoria flash los datos enviados. Este proceso lleva unos 4.5ms durante los cuales el programador debe esperar sin enviar ningún comando.

8.4. Diseño software

Para crear la aplicación se ha usado el entorno *Delphi* de Borland, basado en el lenguaje de programación "*Object Pascal*". La aplicación se ha dividido en dos objetos (clases), cada uno en un fichero de pascal. Estos objetos son:

- Clase `TVentanaPrincipal` (fichero `unPrincipal.pas`)

Esta clase hereda de la clase `TForm`, que en Delphi significa que representa una ventana. Esta es la única ventana del programador, donde se encuentran los botones que permiten ejecutar los diferentes comandos sobre el programador. También se incluyen los controles que permiten modificar los bits de configuración de forma fácil para el usuario, transformándose después al correspondiente conjunto de bits. En este objeto solamente se encuentra el código necesario para manejar la interfaz de usuario, y para ejecutar cada comando utiliza la otra clase, que se describe en el siguiente punto. En lugar de botones estándar de Windows se han usado unos botones gráficos transparentes para mejorar el aspecto visual de la interfaz. Estos botones se implementan en el componente "`TTransBtn`", cuyo código fuente también se incluye en el anexo A5.8 junto con el resto del código de esta aplicación.

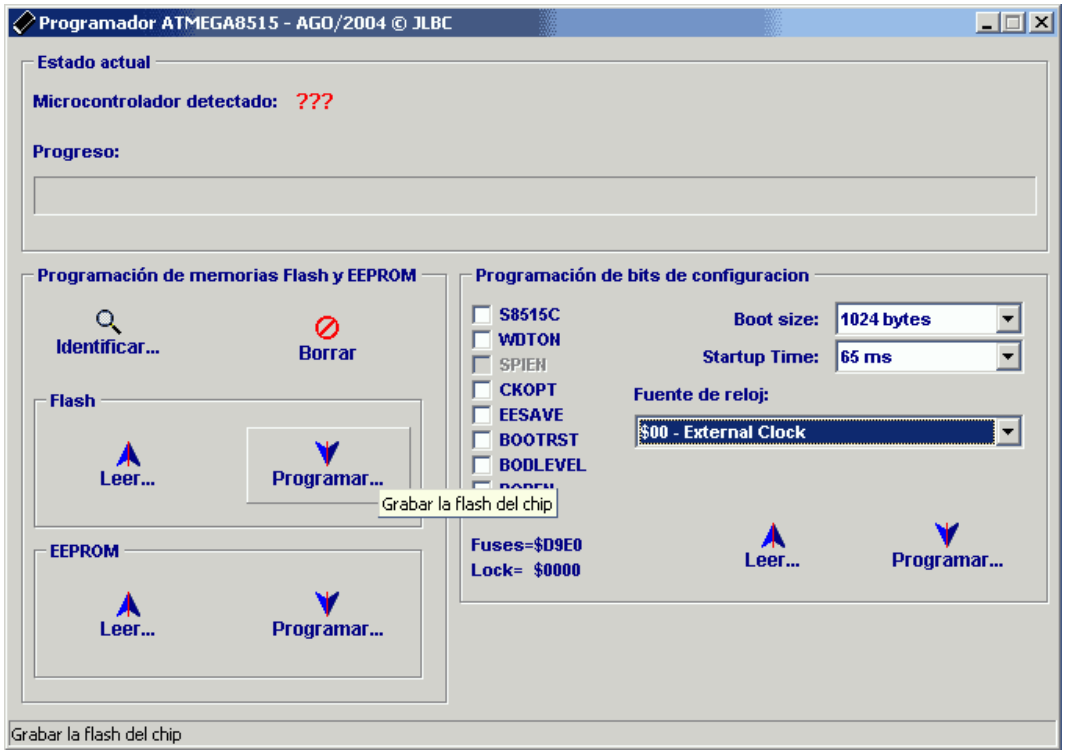


Figura 147: Captura de pantalla de la interfaz gráfica programadora del microcontrolador.

- Clase `TProgramadorAtmel` (fichero `unTProgramadorAtmel.pas`)

En este objeto se ha encapsulado toda la funcionalidad directamente relacionada con el programador. En cuanto al acceso al puerto paralelo del PC, este se ha realizado accediendo directamente a los puertos de E/S del PC correspondientes, que son:

Dirección de E/S	Uso	Pins correspondientes
0x0378	Puerto de salida	El registro se escribe directamente en D[0..7]
0x0379	Puerto de entrada	El bit 6 de este registro se corresponde con el valor lógico en el pin ACK.

Por lo tanto, escribiendo en el puerto 0x378 se pueden modificar todas las salidas hacia el programador, mientras la única entrada (`MISO`) se obtiene leyendo del puerto 0x379. Sin embargo en el sistema operativo Windows, desde la versión Windows 2000, las instrucciones en ensamblador para acceder a los puertos de E/S son instrucciones privilegiadas y no están permitidas en una aplicación de usuario que no se ejecute en el *kernel* del sistema operativo. Para solucionar este problema se ha utilizado el driver "*DriverLINX Port I/O Driver*" [8] que consiste en un módulo que se ejecuta en modo

kernel y una DLL que ofrece los servicios de leer y escribir puertos de E/S. Es esta DLL la que se usa desde la aplicación Delphi para manejar directamente los puertos de E/S. La forma de importar las funciones de la DLL en la aplicación es mediante estas dos sencillas líneas de código:

```
function  ReadPort(Port:longInt): Integer;stdcall; external 'dlportio.dll';
procedure WritePort(Port:longInt;Value:byte);stdcall; external 'dlportio.dll';
```

A continuación se implementan una serie de métodos que realizan acciones básicas usando estas dos funciones importadas de la DLL, y estos se usan como acciones básicas para otros métodos más complejos. Algunos de los métodos implementados más importantes son, por orden de complejidad creciente:

- `procedure CambiarPin( pin: Integer ; valor : Boolean );`
  Con este método se cambia la señal de salida indicada al valor digital alto o bajo según se indique.
- `procedure DarPulsosCLK( n : Integer );`
  Este método se usa para generar una secuencia de pulsos de reloj (pin `XTAL` del microcontrolador), lo que es necesario regularmente para que el chip procese los comandos que se le van enviando. Usando este método software se evita tener que usar un cristal en el programador.
- `procedure TXRX( tx : Byte; var rx : Byte);`
  Este método realiza una transacción SPI, enviando el *byte* indicado y devolviendo el valor recibido del microcontrolador. Como ya se dijo arriba, en unos casos el valor a trasmitir o el recibido será ignorado, lo que se debe tomar en cuenta en la función que llame a este método.
- `function CargarBytePagina(dir_pagina:WORD;dato:byte):Boolean;`
  Una llamada a este método realiza las cuatro transacciones SPI necesarias para implementar el comando "*Load program memory page*".

En esta clase también se implementan dos métodos que procesan ficheros HEX y extraen el array de *bytes* que contiene, usado para programar el microcontrolador desde un fichero, y otro método que realiza el proceso contrario, usado para leer la memoria y volcarla a un fichero. Por no alargar en exceso este apartado no se describe el método de entrar en modo de programación ni otros muchos detalles de bajo nivel, ya que se pueden consultar en el código fuente del programador, que con a este propósito está convenientemente comentado.

9. ESTIMACIÓN TEÓRICA DE PARÁMETROS DEL SISTEMA

En este capítulo se obtendrán valores teóricos para ciertos parámetros del radiorreceptor, basándose solamente en las especificaciones de los integrados, los componentes y en simulaciones y cálculos matemáticos. Los parámetros realmente medidos sobre el sistema físico se presentan en un capítulo posterior. El cálculo de los parámetros ideales servirá para saber qué esperar del receptor y también para comprobar la imprecisión de casi todos estos cálculos debido a múltiples factores que no se suelen contemplar en los cálculos teóricos.

9.1. Ganancia total

En el siguiente esquema se ha representado el camino de la señal, con todas las etapas que intervienen y las potencias de entrada a cada una de estas etapas:

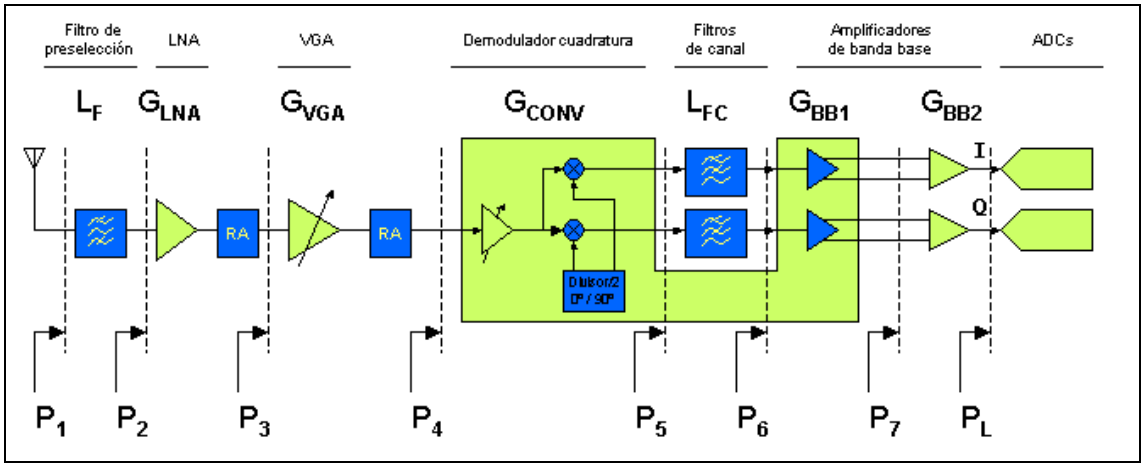


Figura 148: Cálculo global de ganancias.

En cada etapa se define una ganancia o una pérdida (en caso de los filtros) de potencia, cuyos valores se obtienen a continuación.

- Filtro de preselección: Al ser un filtro compuesto solamente de componentes pasivos, no presenta ganancia. Idealmente tampoco presentaría pérdidas en la banda de paso, pero mediante medidas (apartado 11.4) se ha estimado que presenta una pérdida de potencia de:

$$L_F = 3 \text{ dB}$$

- Amplificador de bajo ruido: Como se expuso en el apartado 6.2 la ganancia de este amplificador está fijada a 18dB para la banda de trabajo:

$$G_{LNA} = 18 \text{ dB}$$

- Amplificador VGA: La ganancia de este amplificador es regulable, y según las hojas de características, para el rango de frecuencias de trabajo está en el intervalo:

$$G_{VGA} = [-3.5 , 42.5] \text{ dB}$$

- Ganancia de conversión: En este caso, hay que tener en cuenta tanto la ganancia de conversión, por tratarse de mezcladores activos, como la ganancia del VGA interno al demodulador. Con los datos de las hojas de características, se llega a:

$$G_{CONV} = [-18.5 , 25.5] \text{ dB}$$

- Filtros de banda base: También se trata de filtros pasivos, por lo que idealmente no deberían tener pérdidas, pero se como aproximación a priori se podría tomar:

$$L_{FC} = 1 \text{ dB}$$

- Primer amplificador banda base: Este amplificador interno al integrado del demodulador proporciona una ganancia fija de:

$$G_{BB1} = 20 \text{ dB}$$

- Segundo amplificador banda base: Como se vio en su apartado correspondiente, este amplificador está diseñado para una ganancia de:

$$G_{BB2} = 20.8 \text{ dB}$$

Despreciando las pérdidas por desadaptación de impedancias entre diferentes etapas, que son del orden de décimas de decibelios, y del anterior esquema, se deduce que la ganancia total es de:

$$G(dB) = G_{LNA} + G_{VGA} + G_{CONV} - L_F - L_{FC} + G_{BB1} + G_{BB2}$$

Debido a que hay elementos con ganancia variable, hay que calcular la ganancia global máxima y mínima:

$$\mathbf{G_{MAX} = 122.8 \text{ dB}}$$

$$\mathbf{G_{MIN} = 32.8 \text{ dB}}$$

Por lo tanto el sistema tiene un margen de control de ganancia de 90dB, que como se calculó en los apartados 4.4 y 4.5 es regulable digitalmente con una precisión ligeramente mejor que un decibelio.

9.2. Punto de compresión de 1dB

El cálculo del punto de compresión es importante, pues es una indicación de la máxima potencia de entrada que soporta el receptor sin dejar de tener un funcionamiento totalmente lineal en todas sus etapas. Para su cálculo, se toma el punto de compresión del último elemento en la cadena de la señal, y se irá propagando hacia atrás. Por comodidad, se repite de nuevo el esquema con los bloques que intervienen en el camino de la señal:

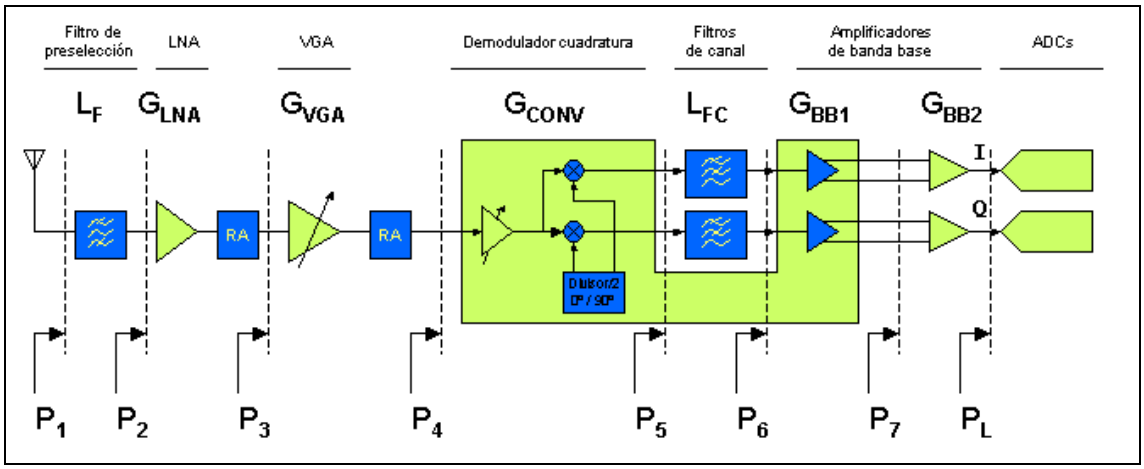


Figura 149: Balance de potencias para punto de compresión de 1dB.

Empezando por el demodulador, su punto de compresión de 1dB es de 13dBm. Este valor es para el caso de mínima ganancia. Tanto en el demodulador como en el amplificador variable se tomará el caso del punto de compresión para la mínima ganancia, ya que se quiere obtener el máximo valor de potencia a la entrada que obliga al receptor a entrar en su zona no lineal. El punto de compresión de IP_{P4}=13dBm está referenciado al punto P_4, la entrada del conversor, con una Z_0=50Ω. Calculando la potencia correspondiente en P3 (restando la mínima ganancia del VGA) obtenemos IP_{P3}=16.5dBm.

Buscando el punto de compresión del VGA en las hojas de características se obtiene el valor de 8.4dBm, referenciado a la salida cuando la ganancia vale 20dB, con lo que se tiene una limitación para IP_{P3} = 8.4dBm – 20 dB = -11.6 dBm, para Z_0=200Ω. Como este valor es más restrictivo que el calculado anteriormente, es el que se toma como máxima potencia en el punto P_3.

Como P_3 es la salida del amplificador de bajo ruido, hay que comprobar que éste no imponga una restricción mayor. Consultando las hojas de características de éste, se obtiene que para la frecuencia de trabajo el IP_1 a la salida es de 3.5dBm, con Z_0=50Ω. Como este valor es mayor que IP_{P3}, no representa un factor limitante, por lo que se puede calcular IP_{P2} simplemente restando la ganancia de este amplificador, que era de 18dB, quedando IP_{P2}= –11.6dBm – 18dB = – 29.6 dBm. Ya sólo queda contar con la atenuación del filtro de preselección para obtener IP_{P1} = – 29.6dBm + 1dB = –28.6dBm.

El resultado es, por tanto, que el sistema se comporta de forma lineal hasta entradas aproximadamente de potencia igual a **IP_{P1} = –28.6 dBm**

9.3. Potencia de ruido a la salida

En este apartado se estima la potencia de ruido que existe a la salida del receptor (en los datos digitalizados), considerando el ruido térmico a la entrada del receptor y estableciendo así el "*noise floor*" mínimo que siempre existirá a la salida. A continuación se muestra el conjunto de las ganancias, pérdidas y factores de ruido de cada elemento de la cadena de señal, hasta llegar a los últimos amplificadores de banda base:

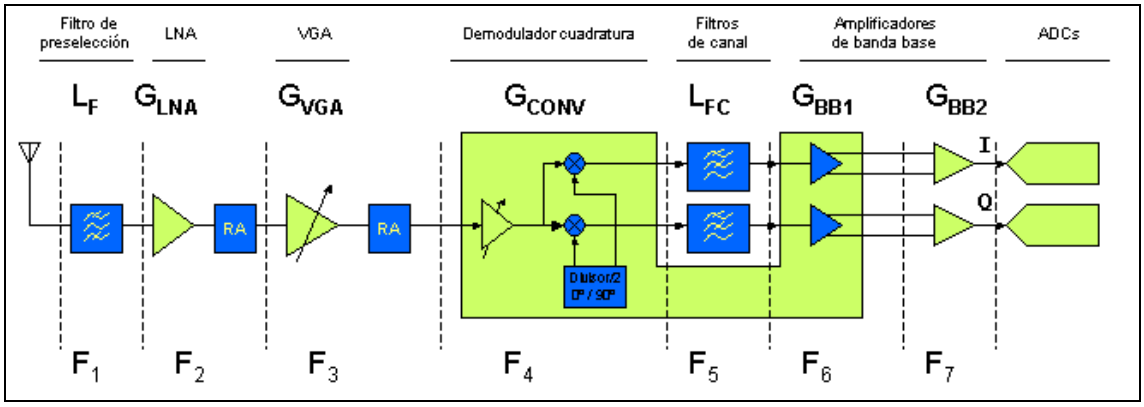


Figura 150: Diagrama de bloques para cálculo del factor de ruido global

En la siguiente tabla se resumen estos datos, tanto en decibelios como en unidades lineales. Para el caso de los filtros, su factor de ruido se aproximará con su pérdida de potencia.

Elemento	**Ganancia en potencia**		**Factor de ruido**	
	dB	**Lineal**	**dB**	**Lineal**
(1) Filtro de preselección	-3	0.5	3	2
(2) LNA	18	63.1	3.5	2.23
(3) VGA	42.5	17782	7.2	5.25
(4) Demodulador	25.5	354.8	11	12.6
(5) Filtros de banda base	-1	0.79	1	1.25
	Ganancia en voltaje		**Densidad espectral de ruido a la entrada**	
	20 log G_V	**G_V**		
(6) Primer amplificador banda base	20	10	6.9 nV/√Hz	
(7) Segundo amplificador banda base	20.8	11	4.5 nV/√Hz	

Desde la entrada de antena hasta los filtros de banda base se puede calcular el factor de ruido de estas cinco etapas utilizando la fórmula de *Friis*:

$$F_{1\text{-}5} = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \frac{F_4 - 1}{G_1 G_2 G_3} + \frac{F_5 - 1}{G_1 G_2 G_3 G_4} =$$
$$1.26 + \frac{1.23}{0.794} + \frac{4.35}{50.1} + \frac{11.6}{890903} + \frac{1.51}{316 \cdot 10^6} = 2.89 = 4.6\text{dB}$$

En los dos últimos amplificadores de banda base el fabricante no proporciona el factor de ruido, sino las densidades espectrales de ruido referidas a la entrada, siendo de $6.9\,\text{nV}/\sqrt{\text{Hz}}$ en el caso del primer amplificador y de 4.5 $\text{nV}/\sqrt{\text{Hz}}$ en el segundo. Para calcular la potencia media de ruido producida por estos amplificadores a sus salidas, hay que tener en cuenta el ancho de banda de ruido, que podemos aproximar por el ancho de banda de los filtros banda base, de 110KHz. Usando este dato y los valores de ganancias de voltaje se llega a las varianzas de voltaje del ruido a las salidas de estos amplificadores, originado solamente por el ruido interno:

$$\sigma^2_{BB1} = 5.23\ 10^{-10}\ V^2$$
$$\sigma^2_{BB2} = 2.2\ 10^{-12}\ V^2$$

Además hay que tener en cuenta el efecto del muestreo de la señal con los conversores ADC de ocho bits. Para un conversor uniforme se tiene que la varianza en voltaje del ruido se puede modelar como:

$$\sigma^2 = \frac{\Delta^2}{12} = \frac{\left(\frac{5V}{2^8}\right)^2}{12} = 3.18 \cdot 10^{-5}\ V^2$$

Ya se tienen todos los datos para reconstruir la evolución de la señal de ruido desde la entrada hasta la salida del receptor. Empezando por la entrada de la antena, se tiene que la potencia de ruido térmico a la entrada viene dado por la ecuación:

$$N_{IN} = K\ T\ B$$

Donde K es la constante de Boltzman, T la temperatura en grados Kelvin y B en ancho de banda de ruido equivalente a la entrada. Este último viene marcado por el ancho de banda del filtro de preselección. Aproximando este ancho de banda por su ancho de banda a tres decibelios, que es de 32.4MHz, se tiene:

$$N_{IN} = 1.38 \cdot 10^{-38}\ \frac{W \cdot seg}{K} \cdot 298\ K \cdot 32.4\ MHz = 1.33 \cdot 10^{-28} W = -249 dBm$$

Como se tiene el factor de ruido de las cinco primeras etapas del receptor y la ganancia global es la suma en decibelios de todas ellas por separado, se puede estimar la potencia de ruido a la entrada del primer amplificador de banda base mediante:

$$N_{BBin}\,(dBm) = N_{IN}\,(dBm) + G_{1\text{-}5}\,(dB) + F_{1\text{-}5}(dB) =$$

$$-249dBm + 122.8dB + 4.6dB = -121.6dBm$$

Por lo tanto la varianza del ruido a la entrada del primer amplificador de banda base será de $\sigma^2_n = 3.56 \cdot 10^{-14}\ V^2$. La varianza de ruido a la salida del primer amplificador será el resultado de la amplificación de este ruido más el generado por él:

$$\sigma^2_{sal.BB1} = \sigma^2_n \cdot G_{BB1} + \sigma^2_{BB1} = 5.23 \cdot 10^{-10}\ V^2$$

Operando de igual forma con el segundo amplificador de banda base se tiene:

$$\sigma^2_{sal.BB2} = \sigma^2_{sal.BB1} \cdot G_{BB2} + \sigma^2_{BB2} = 6.32 \cdot 10^{-8}\ V^2$$

Este último valor del mínimo ruido que existe teóricamente a la entrada de los conversores ADC.

Sin embargo la alta ganancia del receptor hará que en la práctica no se pueda utilizar en el máximo de su ganancia, debido a la gran cantidad de ruido que amplifica y a las múltiples fuentes de ruido originadas en el propio receptor, seguramente en su mayoría por la parte digital del sistema. Por lo tanto, este valor teórico no se da en la práctica.

9.4. Sensibilidad

La sensibilidad de un receptor se define como la mínima potencia de señal necesaria a su entrada para conseguir a su salida un determinado nivel de calidad en la señal recibida. Debido a la generalidad de este receptor, se calculará la sensibilidad para una modulación analógica genérica, que requiera una relación S/N a la salida (entrada de los conversores ADC) de 10dB.

Como resultado del apartado 8.3 se obtuvo un valor para la potencia de ruido, de origen solamente térmico más el generado en los integrados, reflejado a la salida del receptor. Sin embargo ese valor es totalmente irreal ya que no toma en cuenta otras fuentes de ruido que al parecer predominan sobre las que se han tenido en cuenta para los cálculos.

Aún así, se puede utilizar como mínima señal de ruido a la entrada de los conversores la potencia del ruido de cuantificación de los ADC de ocho bits, que como se ha visto anteriormente, es de:

$$\sigma^2_{ruido} = 3.18 \bullet 10^{-5}\ V^2$$

$$\sigma_{ruido} = 5.64mV$$

Para comparar con potencias de señales dadas en unidades logarítmicas, se puede expresar esta potencia como N_o= –32dBm respecto a una Z_0 de 50Ω. Por lo tanto, dado que la relación señal a ruido a la salida debe ser de 10dB, establecemos una potencia de señal a la salida de S_o=–22dBm. Usando la máxima ganancia global del receptor calculada en el apartado 9.1, que era de G_{MAX}=122dB, podemos calcular la potencia de señal necesaria a la entrada, que es justamente la definición de sensibilidad:

S = Si = –100dBm

9.5. Margen dinámico

El margen dinámico (MD) de un receptor es el rango de potencias de señal de entrada que producen una señal recibida de calidad aceptable. Este parámetro también se puede definir de distintas formas según el receptor. En este caso se va a utilizar:

- Límite superior: Se usará el límite de la linealidad del receptor, es decir el punto de compresión de un decibelio, que se vio en el apartado 9.2 era de –28.6dBm

- Límite inferior: Como señal de potencia mínima aceptable se va a usar la sensibilidad calculada en el apartado anterior, que es de –100dBm

Por lo tanto, se tiene un margen dinámico de **71.4dB**. Se puede observar que este margen es inferior al rango de 90dB con que se puede controlar la ganancia. Esto es deseable ya que es otro factor (el punto de compresión) el que limita el margen dinámico y no la falta de rango de control de ganancia.

9.6. Rechazo a canal adyacente

En este caso se podría pensar que la naturaleza genérica de este receptor tampoco permite definir de forma clara este parámetro, ya que a priori se desconoce el ancho de banda del canal a recibir (BW_C) y la separación entre canales. Sin embargo, sí que se puede estimar siguiendo el siguiente razonamiento. Sea f_{RF} la frecuencia donde está el canal deseado, y por tanto la frecuencia que el receptor sintoniza. Se tiene que independientemente del ancho de banda del canal BW_C, el receptor siempre muestrea y envía al PC el rango de frecuencias $\left[f_{RF} - \frac{BW}{2}, f_{RF} + \frac{BW}{2}\right]$ donde BW es el ancho de banda en RF de la señal muestreada, cuyo ancho viene determinado por los filtros de banda base y es 250KHz. Supongamos en primer lugar que el canal no deseado cae fuera del rango de BW, como se ilustra en la siguiente figura, donde también se ha dibujado la respuesta del filtro banda base equivalente:

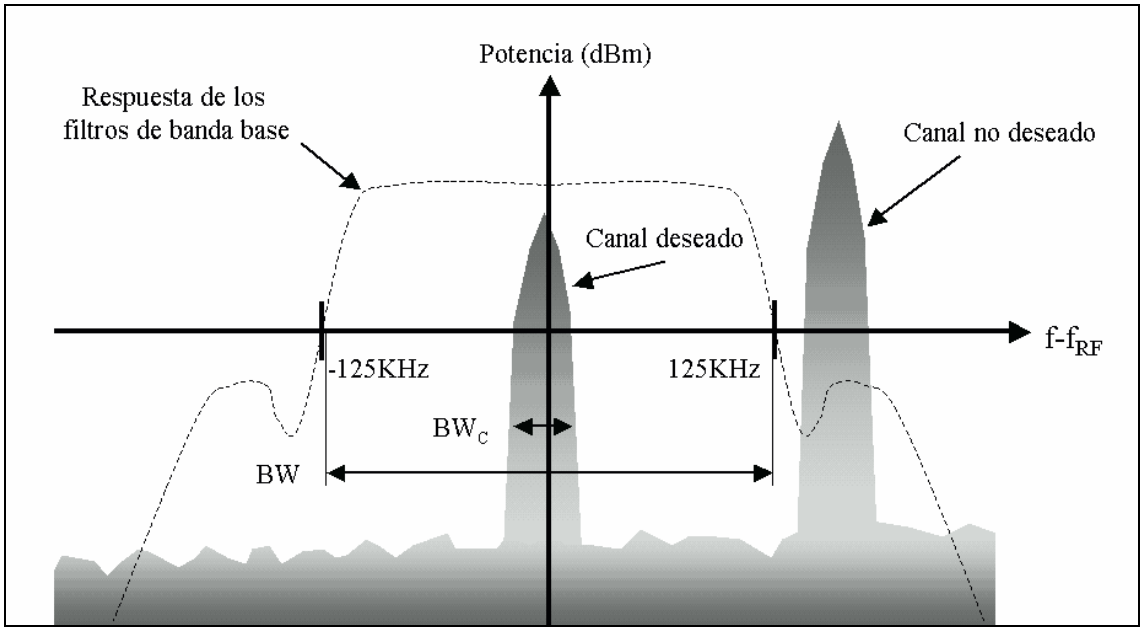


Figura 151: Ejemplo de canal no deseado fuera de BW

Aunque al PC se envía la señal muestreada en el rango $\left[f_{RF}-\frac{BW}{2}, f_{RF}+\frac{BW}{2}\right]$, el efecto del aliasing de frecuencias en los conversores ADC provoca que si entran componentes frecuenciales por encima de 125KHz, éstas se mezclen de forma irreversible con otras frecuencias en el rango deseado. Lo único que evita esto son los filtros de banda base, descritos en el apartado 6.6. En estos se estableció una frecuencia de corte de 110KHz, lo que quiere decir que los efectos del aliasing solamente serán perjudiciales a partir de 125KHz + (125KHz – 110KHz) = 140KHz. Mirando la respuesta de los filtros de banda base en la gráfica al final del apartado 6.6 se puede ver que la atenuación, y por lo tanto el rechazo a un supuesto canal no deseado en esas frecuencias es de **45dB** como mínimo.

En el caso de que el canal no deseado esté dentro del ancho de banda BW, se tendría la situación que se ilustra a continuación:

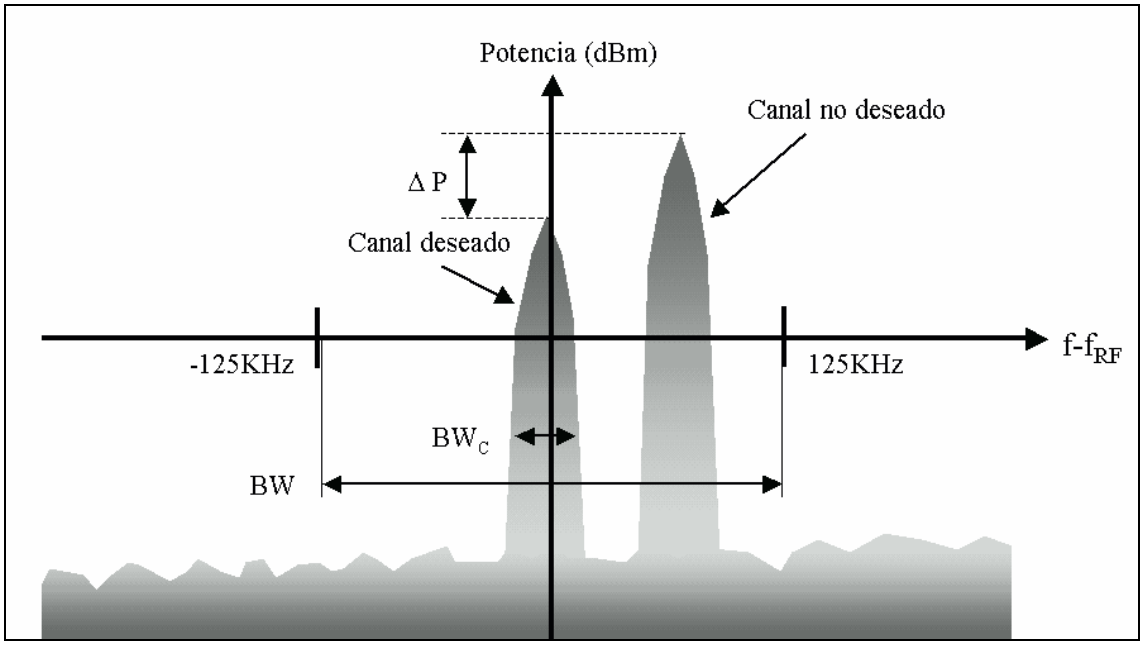


Figura 152: Ejemplo de canal no deseado dentro de BW

En este caso el canal no deseado cae en el rango $\left[f_{RF}-\frac{BW}{2}, f_{RF}+\frac{BW}{2}\right]$ y por lo tanto es inevitable que se envíe al PC sin ser eliminado. Sin embargo en el software del PC una de las primeras etapas incluirá un filtro digital FIR paso bajo que solamente deje pasar el ancho de banda del canal que realmente se está usando, BW_C. Por lo tanto, el rechazo depende de este filtro digital y en principio se puede hacer **ilimitada** aumentando el orden del filtro FIR. En realidad sí hay dos límites. El primero está directamente

relacionado con la capacidad de proceso disponible para realizar el filtrado en la máquina. La segunda tiene que ver con el rango dinámico de los conversores ADC y la diferencia relativa de potencias de los dos canales, ΔP en la figura, donde $\Delta P = P_{ND} - P_D$, y siendo P_{ND} la potencia del canal no deseado y P_D la del canal deseado. Mientras P_{ND} sea bastante menor que P_D, no existe problema. Sin embargo si la potencia del canal no deseado es bastante mayor que la del canal deseado se puede llegar a perder la señal deseada.

Esto se puede explicar con un sencillo ejemplo gráfico: supongamos que la señal a la entrada de los conversores ADC se compone de dos tonos, siendo el de baja frecuencia el correspondiente al canal deseado y el de alta frecuencia al canal no deseado. Si la potencia de ambos es del mismo orden de magnitud se obtendría una señal como la mostrada a la izquierda:

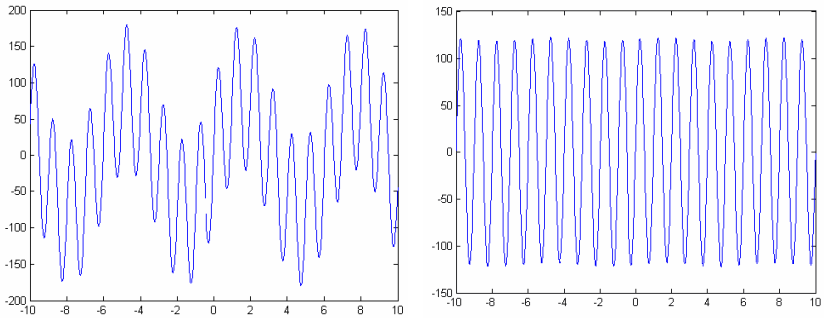


Figura 153: Ejemplo del problema del margen dinámico

En este caso se digitalizarían ambas señales y mediante filtrado digital se podrían separar sin problemas. Pero si la señal de alta frecuencia (canal no deseado) tiene mucha más potencia, la ganancia del receptor deberá reducirse para evitar la saturación de los conversores ADC, reduciendo también la amplitud de la señal deseada hasta el extremo de hacerla menor que un escalón de voltaje del conversor, y por tanto, hacerla indetectable (el caso de la gráfica de la derecha).

Por lo tanto existe un máximo para la diferencia de potencias ΔP que permite seguir recibiendo el canal deseado, y que depende de la resolución de los ADC. Para el caso de los conversores de ocho bits, la diferencia de potencia entre la máxima y mínima señal que se puede digitalizar (margen dinámico del conversor) se podría estimar por:

$$\mathrm{MD}_{\mathrm{ADC}} = 10 \cdot \log \frac{128^2}{1^2} = 20 \cdot \log 128 = 42\mathrm{dB}$$

Sin embargo el valor exacto de ΔP es menor que $\mathrm{MD}_{\mathrm{ADC}}$ en una medida que depende del tipo concreto de modulación, aunque como aproximación sería un buen valor

cualquiera comprendido entre 20 y 30dB. A continuación se resumen todos los casos expuestos en este apartado:

10. PLAN DE PRUEBAS

En este capítulo se mostrarán las pruebas básicas que se han realizado para, de una forma ordenada e incremental, comprobar el buen funcionamiento de algunas partes del receptor. Las pruebas que implican medida de parámetros o características del receptor se dejan para el capítulo siguiente.

10.1. Prueba de la circuitería digital

La principal prueba que se puede hacer a nivel básico de la circuitería digital es comprobar que los osciladores funcionan. Para ello se ha medido con un osciloscopio el oscilador del microcontrolador, a 10MHz, y el del integrado USB, a 6Mhz. Como se ve en las siguientes imágenes ambos funcionaban correctamente.

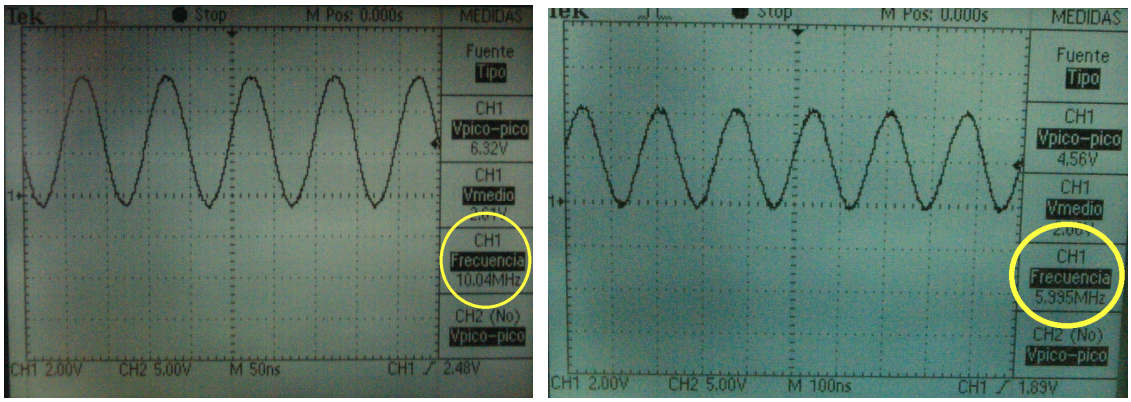


Figura 154: Comprobación del funcionamiento de los osciladores a cristal.

A continuación se comprobó el funcionamiento de los *flip-flops* encargados de realizar la división de frecuencias hasta obtener la señal de 2.5Mhz, que también se observó que se obtenía sin problemas.

El siguiente paso implica la comprobación del funcionamiento del integrado de interfaz USB, y su correcta conexión con el PC. Para esto se examinaron las líneas de datos D(+) y D(-) de este integrado, pudiéndose ver como se recibían correctamente las tramas desde el PC cada 1ms, signo de que se había establecido conexión correctamente.

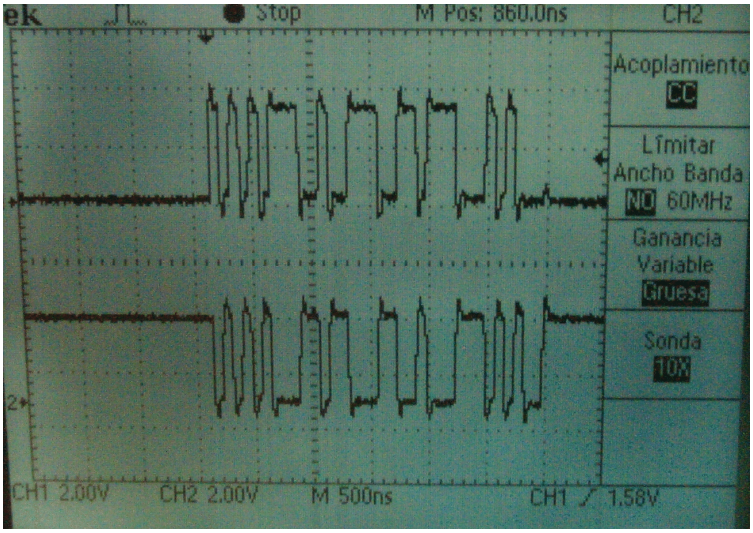


Figura 155: Comprobación mediante osciloscopio del funcionamiento de la interfaz USB.

En cuanto al resto de pruebas de circuitería digital, estas implican el funcionamiento de parte del *firmware* para enviar y recibir tramas al PC, lo que demuestra que la máquina de estados y la memoria FIFO funcionan correctamente. Una vez establecida la comunicación correctamente con el PC, se pueden ejecutar comandos mandando tramas desde el módulo de control manual de *SWRadioStudio*. Por ejemplo, la siguiente imagen es una captura de la ejecución de un comando de cambio de un conversor DAC, que genera la transacción que se puede observa en el bus SPI. Como se ve se generan correctamente los 12 pulsos de reloj y los bits de datos, pudiéndose comprobar además que todos los conversores se cambian al valor indicado desde la aplicación.

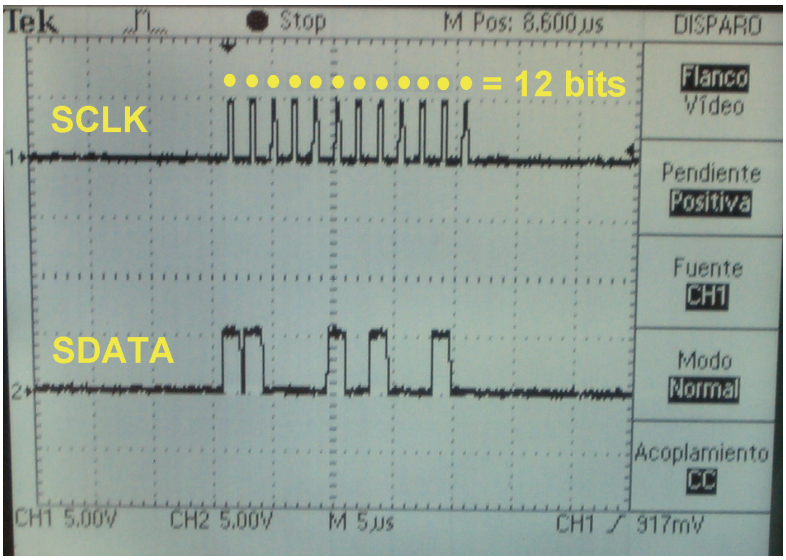


Figura 156: Visualización del funcionamiento del bus SPI

10.2. Pruebas al PLL

La prueba de la correcta comunicación con el PLL se realizó de esta forma: desde el módulo de control manual de SWRadioStudio, se seleccionó la salida de MUXOUT a un nivel lógico bajo, enviando la configuración al PLL. Tras esto, se solicitó un comando de muestreo de la línea MUXOUT, obteniéndose todas las muestras a nivel bajo. A continuación se repitió el proceso pero seleccionan como salida de MUXOUT un nivel lógico alto. Al obtenerse la señal muestreada con todas las muestras a nivel alto, se puede asegurar que la comunicación con el PLL funciona.

Para asegurar a continuación que la señal de referencia llega correctamente al PLL se realizaron estas mediciones. Primero, se midió directamente a la entrada del pin para la señal de referencia, obteniéndose la esperada señal cuadrada a 2.5Mhz y con los niveles lógicos transformados de 3.3V, como se ve en esta captura.

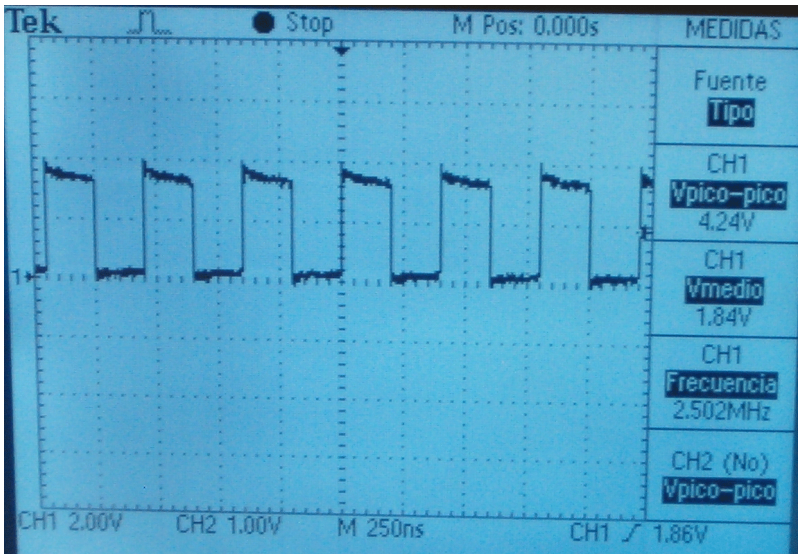


Figura 157: La señal de referencia de 2.5Mhz convertida a niveles lógicos de 3.3V

A continuación, se seleccionó alternativamente como salida de MUXOUT el divisor N y el divisor R, a la vez que se establecía la configuración adecuada para que el PLL se enganchase en la frecuencia de 200Mhz con un Δf=100KHz. En las siguientes imágenes se pueden ver las mediciones.

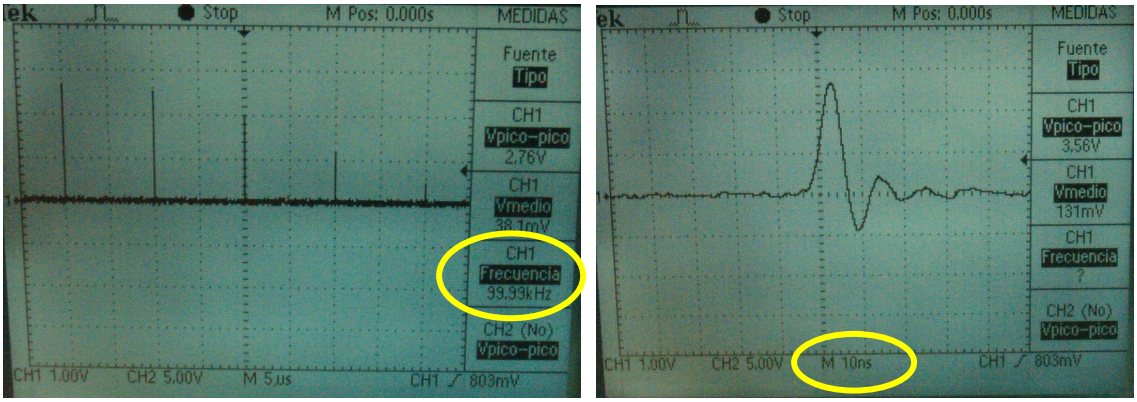


Figura 158: La salida MUXOUT en modo "N divisor"

En ambos casos se obtuvieron señales de período 100KHz, pero en el primer caso con un período a nivel alto de tan solo unos 10ns, mientras que en el segundo caso se mantenía por 400ns, justo un ciclo de reloj a 2.5Mhz. Con estas mediciones queda comprobado que el PLL esta correctamente enganchado y que recibe correctamente la señal de referencia.

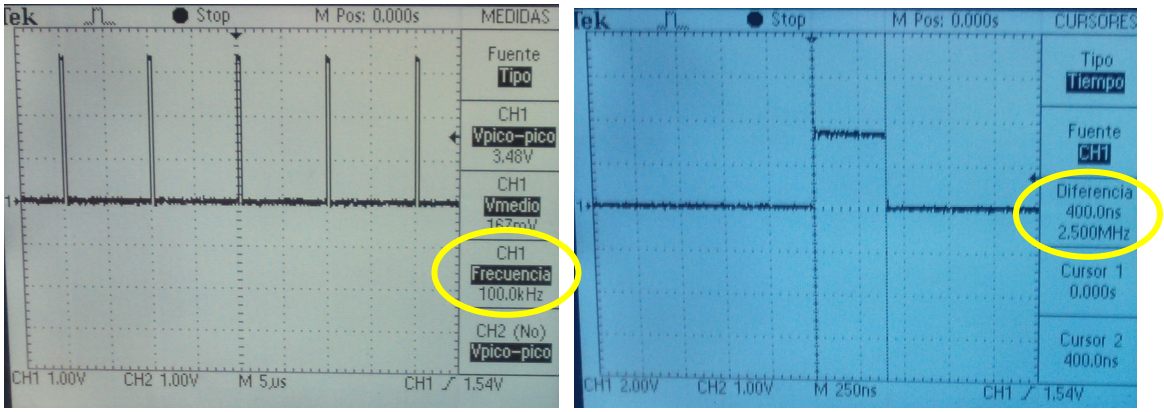


Figura 159: La salida MUXOUT en modo "R divisor"

11. MEDICIONES Y RESULTADOS

En este capítulo se recogen todas las mediciones realizadas para caracterizar el receptor y su correcto funcionamiento, así como los resultados de la ejecución de los distintos demoduladores software.

11.1. Flujo de tramas

En estas pruebas, que fueron las primeras en realizarse en el receptor, fueron en las que se estableció el límite superior de 500Kb/s como flujo de datos máximo, usando un reloj de 2.5Mhz, que se podía transmitir al PC con total seguridad de que no se perdería ningún *byte* por saturaciones de colas. Tras estas pruebas, se procedió a establecer la frecuencia de Nyquist en 125KHz y a realizar todo el proceso de diseño analógico consiguiente.

11.2. Sintetizador de frecuencias: VCO y PLL

Para realizar las pruebas sobre la señal del oscilador local se ha aprovechado el hecho de que el oscilador tiene dos salidas, de las que una va al demodulador y la otra hacia una resistencia de carga de 50Ω. Sin embargo, utilizando un puente en la placa diseñado para este fin, se permite desconectar la salida de la carga resistiva y conectar un cable hacia el analizador de espectros, con lo que la medida no requiere desconectar la salida que va hasta el demodulador.
A menos que se diga lo contrario, todas las medidas con el PLL se han realizado con estos valores para las distintas opciones del integrado:

Parámetro	Valor
Frecuencia	200 MHz
Δf	100 KHz
Anti backslash	3 ns
Lock-detect	3 ciclos
Core-power	2.5 mA
Potencia de salida	-3 dBm
Corriente del CP	2.50 mA

11.2.1.Conversión de niveles lógicos de las señales

Como se vio en el apartado correspondiente al diseño del PLL, éste funciona con niveles lógicos CMOS de 3.3V, mientras que el microcontrolador funciona con niveles también CMOS pero de 5V, por lo que se tuvo que intercalar un adaptador de niveles lógicos, el MAX3002. Para comprobar el correcto funcionamiento de este conversor, basta comprobar que las comunicaciones digitales del microcontrolador con el PLL funcionan, lo que asegura que las líneas del bus SPI (datos, reloj y señal de sincronía) se transforman correctamente. La única señal cuya transformación es más crítica es la correspondiente a la señal de reloj de referencia para el PLL, la señal cuadrada a 2.5Mhz. Esta señal es crítica porque según las especificaciones del integrado del PLL, se requiere que si la entrada de referencia está acoplada en corriente continua y es una señal digital, como es el caso, cumpla que tenga un *slew rate* en los flancos de al menos 21V/μs. Anteriormente al uso del integrado conversor de niveles lógicos se intentó adaptarlos mediante unas sencillas redes resistivas a modo de divisores de voltaje. Esta adaptación funcionó para las señales del bus SPI, pero no para la señal de referencia, por lo que finalmente se adoptó la opción del integrado adaptador de niveles. Utilizando el adaptador se procedió a medir el *slew rate* de los flancos, como se muestra en las siguientes imágenes:

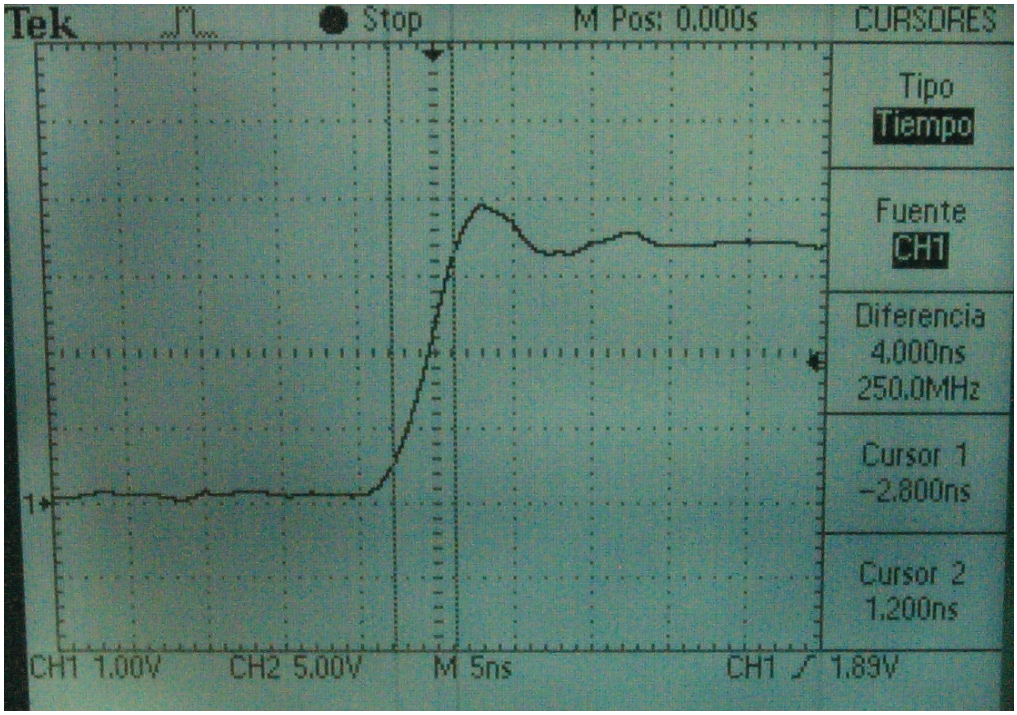


Figura 160: Medición con osciloscopio del *slew rate* de la señal REFCLK del PLL.

Se observa que en la transición desde el 10% hasta el 90% del escalón a la salida (señal CMOS a 3.3V) transcurren 5ns. Como este intervalo representa 3.16V, se obtiene que:

$$\text{SlewRate} = \frac{\Delta V}{\Delta T} = \frac{3.16V}{5ns} = \mathbf{632\ V/\mu s} >> 21\ V/\mu s$$

Que como se ve cumple sobradamente el requisito del *slew rate* mínimo.

11.2.2.Frecuencias máximas y mínimas de funcionamiento

Para comprobar el rango de frecuencias de funcionamiento del sintetizador de frecuencias se han ido enviando valores de frecuencias desde el módulo de "Control Manual" del programa *SWRadioStudio*, seleccionando para la salida MUXOUT la señal de enganche digital ("*Digital Lock Detect*") y posteriormente pulsando sobre el botón de muestrear la salida MUXOUT del PLL. Con esto se consigue ver claramente cuando el PLL se ha enganchado correctamente a la frecuencia deseada (se ve la salida con un valor lógico alto constantemente) o si no se ha enganchado (con un valor lógico bajo).

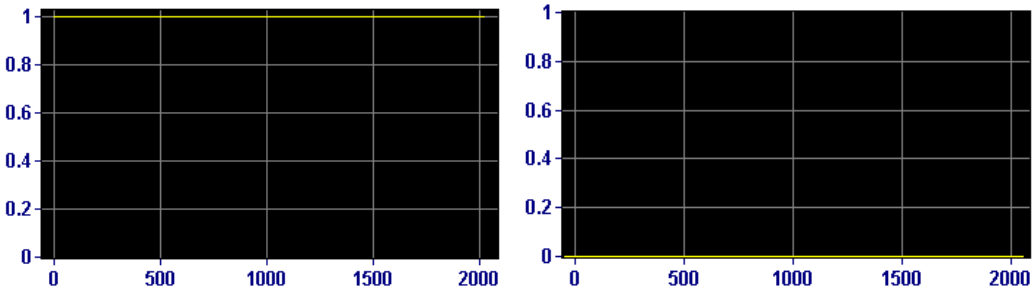


Figura 161: Aspecto de la gráfica MUXOUT para el caso de que el PLL se enganche (izquierda) o que no lo haga (derecha)

Como resultado de estas pruebas se han obtenido estos valores, con una resolución de la frecuencia de referencia (Δf=100KHz):

$$\mathbf{f_{MIN} = 167.9\ MHz}$$
$$\mathbf{f_{MAX} = 245.1\ MHz}$$

Comparándolos con el objetivo de diseño, que era el margen de frecuencias 170MHz a 222MHz, se ve que los objetivos se han cumplido sobradamente, ya que el correspondiente margen de frecuencias de RF sintonizables va desde **83.95Mhz** hasta **122.55MHz**, mientras que el objetivo general del proyecto era incluir al menos el rango de las emisiones de FM comercial (88-108MHz).

11.2.3.Potencia de salida

En este apartado se exponen los valores de potencia realmente medidos por el analizador de espectros con respecto a la potencia de salida deseada teórica, configurable mediante un registro interno al PLL. Esta potencia se puede establecer en cuatro valores: 0dBm, –3dBm, –6dBm y –9dBm. Estos son los valores realmente medidos para cada una de estas configuraciones de la salida:

$P_{teórica}$ (dBm)	P_{medida} (dBm)
0	-4.9
-3	-7.8
-6	-11.1
-9	-14.5

A continuación se representan estos valores gráficamente:

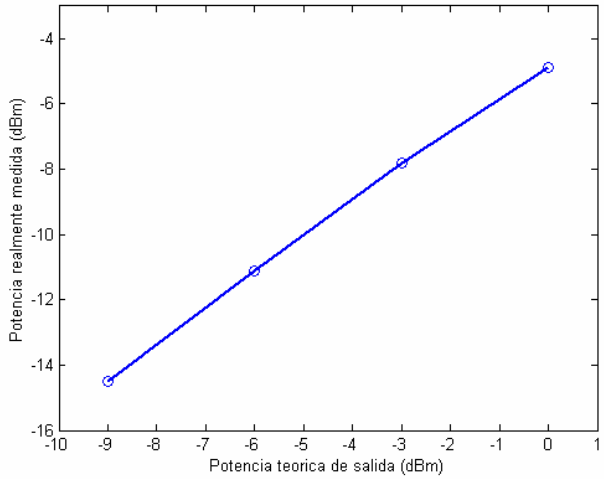


Figura 162: Potencia de salida del PLL respecto a la teórica.

Aproximando la potencia medida en función de la potencia de salida teórica de forma lineal, se obtiene:

$$P_{real}\,(dBm) = 1.07 \cdot P_{teorica}\,(dBm) - 4.76\ dB$$

Es decir, que como es de esperar la relación es prácticamente lineal salvo una diferencia constante representada por la pérdida de 4.67dB con respecto al valor de salida teórico. Aunque la salida se ha adaptado según las indicaciones del *datasheet* para adaptar la impedancia de salida, es posible algunas causas de no conseguir la máxima potencia de salida sean:

- Que el valor de la inductancia usada para adaptar la impedancia de salida no concuerde exactamente con la requerida.
- Pérdidas por capacidades e inductancias parásitas en el trazado de las pistas.
- Pérdidas en el cable coaxial, en la conexión con la placa, y calibración no perfecta de medición de potencia del analizador de espectros. Como indicador de esto decir que, por ejemplo, al conectar directamente la salida de un generador RF del laboratorio al analizador de espectros, se miden entre 2 y 3dB menos potencia que la que indica el generador.

Aún así el requisito de entrada del demodulador es una señal entre –12dBm y 0dBm, por lo que se ve que al menos con las tres potencias más altas de salida se cumple este requisito perfectamente.

Todas estas medidas se han hecho para la frecuencia de 200Mhz. En cuanto a la variación de la potencia de salida dentro de todo el rango de frecuencias esta es

despreciable ya que la potencia de salida varía solamente en torno a 1dB en todo el margen de frecuencias de funcionamiento, como se ve en las siguientes imágenes, tomadas cada una en un extremo del rango de funcionamiento del sintetizador:

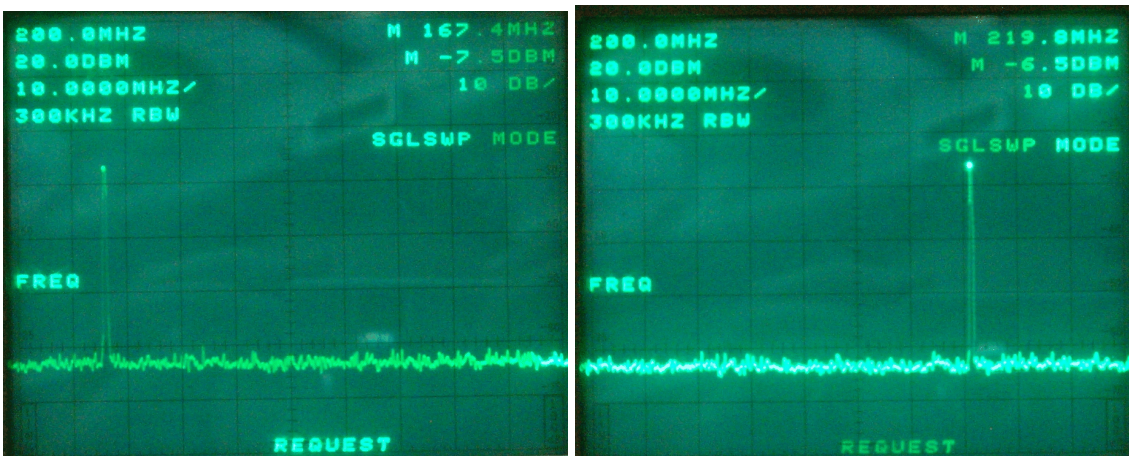


Figura 163: La potencia de salida del PLL se mantiene prácticamente constante en todo el rango de funcionamiento

11.2.4.Ruido de fase y espúreos

Para realizar la medición del ruido de fase de un oscilador lo que se hace es estimar la densidad espectral de potencia (d.e.p.) en las frecuencias cercanas a la de oscilación. Esto es así porque un ruido en la fase de un oscilador sinusoidal se transforma en una distribución de potencia en el dominio del espectro en las frecuencias cercanas. Sin embargo la d.e.p. no es directamente mensurable, sino que se utilizan analizadores de espectros que miden la potencia en intervalos tan pequeños como 1 Hz. Sin embargo para este proyecto se ha hecho uso de un analizador de espectros cuya resolución de ancho de banda mínima es de 300Hz. Aunque esto supone una pérdida de resolución, es válido para el propósito de estimar el ruido de fase teniendo en cuenta esta transformación:

$$\text{d.e.p.}(f) \cong \frac{P(f)_{\text{Watios}}}{\text{RBW}_{\text{Hz}}}$$

Donde P(f) es la potencia medida por el analizador para una frecuencia *f*, y RBW es la resolución de ancho de banda de este, en este caso de 300 Hz. Sin embargo la potencia

medida por el analizador de espectros está en unidades logarítmicas, por lo que hay que transformarla en unidades lineales. Se llega fácilmente a:

$$\text{d.e.p.}(f)_{\text{dBm}/Hz} \cong P(f)_{\text{dBm}} - 10 \cdot \log 300 = P(f)_{\text{dBm}} - 24.77\text{dB}$$

Como el ruido de fase se aprecia en frecuencias bastante cercanas a la de oscilación, en el rango de pocos kilohercios, se ha configurado el analizador de espectros de forma que la frecuencia central coincida con la del oscilador, que aunque es de 200.0Mhz en el analizador aparece como 200.02578Mhz, usando un *span* de 1Khz y el ancho de banda mínimo de 300Hz. En la siguiente imagen se puede ver el aspecto de esta medición.

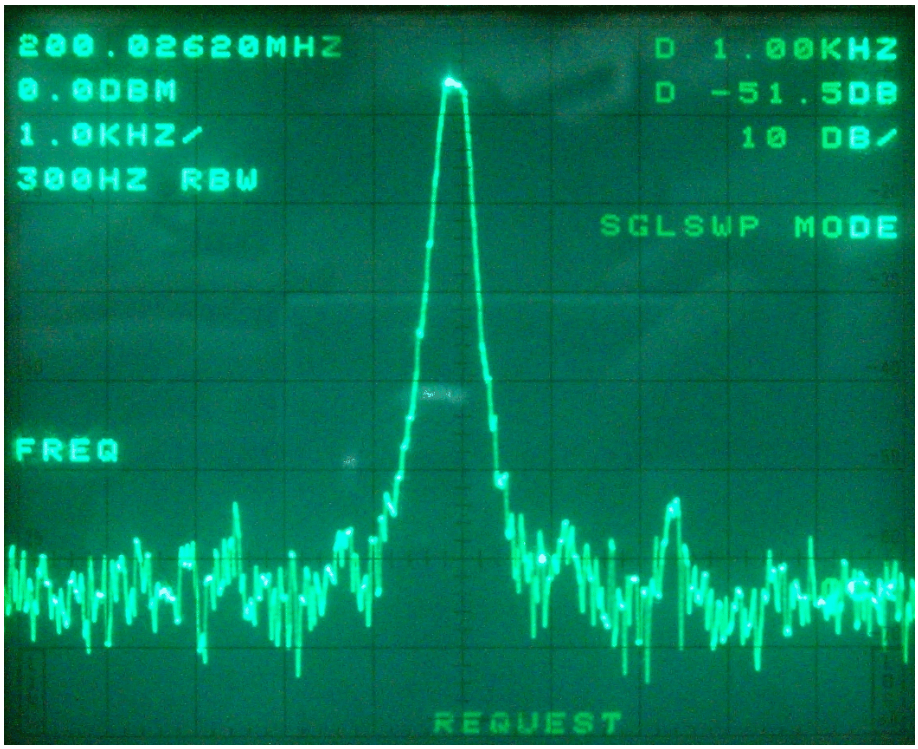


Figura 164: Imagen del analizador de espectros midiendo el ruido de fase del oscilador.

Para caracterizar el ruido de fase se han hecho varias mediciones de potencia a distintos desplazamientos de frecuencia desde el oscilador, midiendo la potencia en "dBm", y realizando cuatro medidas distintas para calcular la media. Con los datos de la potencia medida en 300Hz se ha estimado valor de la d.e.p. utilizando la fórmula obtenida arriba, y la densidad de potencia relativa a la portadora, medida en "dBc/Hz".

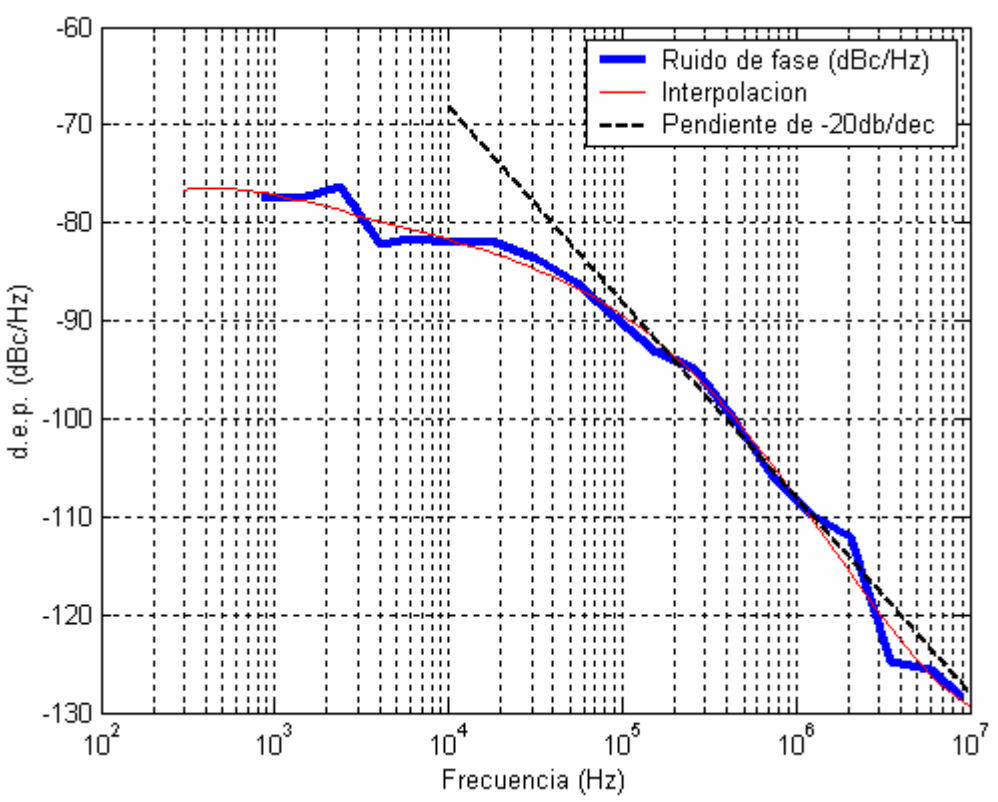


Figura 165: Densidad espectral de ruido medido respecto a la portadora.

Según la teoría, esta densidad espectral de ruido debería caer con una pendiente de 20dB por década a partir de cierto punto, y en la gráfica se puede observar que así es.

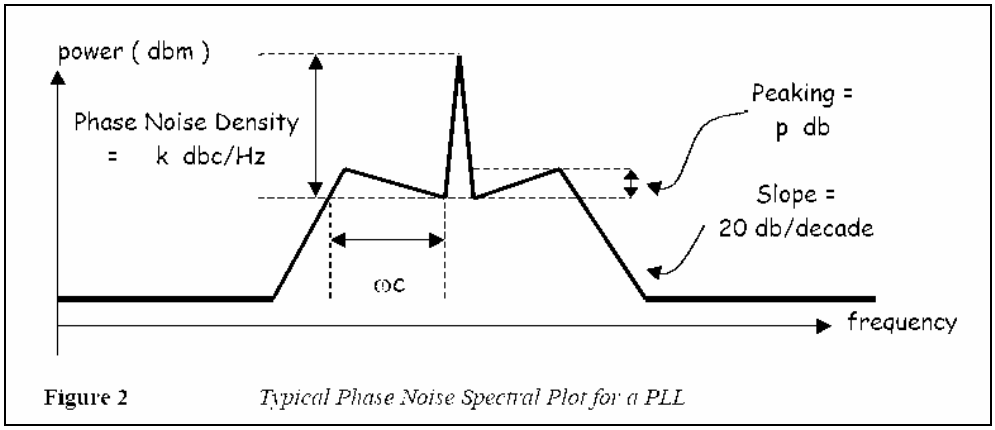


Figura 166: Aspecto que debería tener teóricamente la d.e.p. de ruido.

En cuanto a los espúreos, estos son componentes que aparecen como consecuencia de las modulaciones residuales de que sufre la salida del oscilador. Por lo tanto cualquier señal periódica externa que llegue por ejemplo a la alimentación del PLL producirá una

modulación en amplitud y frecuencia que producirá espúreos en la señal sintetizada. En las primeras mediciones se encontraron unos espúreos a la frecuencia de 56Khz, y a sus armónicos. Este espúreo se ha conseguido eliminar mediante el filtrado adicional de la línea de alimentación no sólo en la placa del PLL sino en todo el sistema, utilizando condensadores de 47µF, valor alto de capacidad dada la baja frecuencia del ruido.

Otro espúreo detectado y no eliminado se da para la frecuencia de 2.5Mhz y sus armónicos, es decir, originada por la señal de referencia procedente del divisor de frecuencias basado en *flip-flops* que ya se vio. En la siguiente imagen se muestra el aspecto que estos tienen en el analizador de espectros. Hay que señalar que el span utilizado para medir estos espúreos (2MHz) es varios órdenes de magnitud mayor que el usado en la anterior imagen para medir el ruido de fase cerca del oscilador (1KHz), por lo que la forma visualizada es muy distinta, como es lógico.

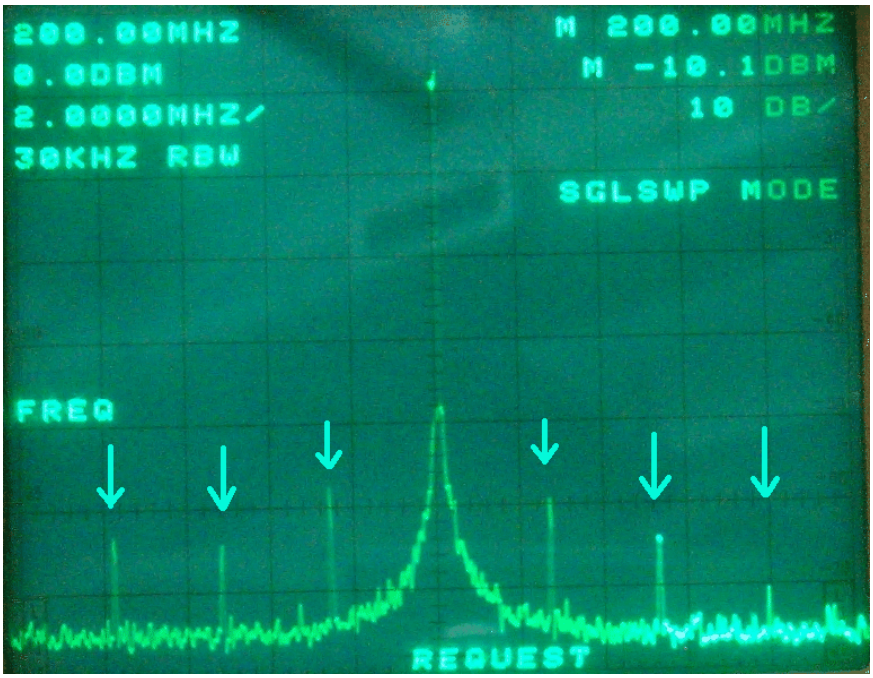


Figura 167: Imagen del analizador de espectros midiendo los espúreos del oscilador.

La potencia de estos espúreos decae con la distancia a la frecuencia del oscilador, y sus valores se han medido para los nueve primeros armónicos de 2.5Mhz.

$\lvert f - f_0 \rvert$ (Mhz)	N x 2.5 MHz	$dep(f) - dep(f_0)$ (dBc)
2.5	1	-48.7
5	2	-51.6
7.5	3	-54.3
10	4	-54
12.5	5	-57
15	6	-57.3
17.5	7	-61.3
20	8	-67
22.5	9	-61.6

11.2.5.Armónicos

Un oscilador sinusoidal ideal debería generar un tono puro sin componentes armónicas. Pero normalmente todos los osciladores generan componentes en los armónicos de la frecuencia principal, en múltiplos enteros de ésta. Para medir la cantidad de armónicos generados por el sintetizador de frecuencias, se ha configurado para que se enganche a la frecuencia de 200Mhz y se ha medido usando el analizador de espectros la potencia de señal en las frecuencias armónicas, obteniéndose estos datos:

Armónico	Frecuencia (MHz)	Potencia (dBm)
1 (Fundamental)	200	-4.2
2	400	-18.5
3	600	-16.2
4	800	-25.2
5	1000	-31.1
6	1200	-36.7
7	1400	-54.8
8	1600	-50.8
9	1800	-61.5

Que representados gráficamente quedan así:

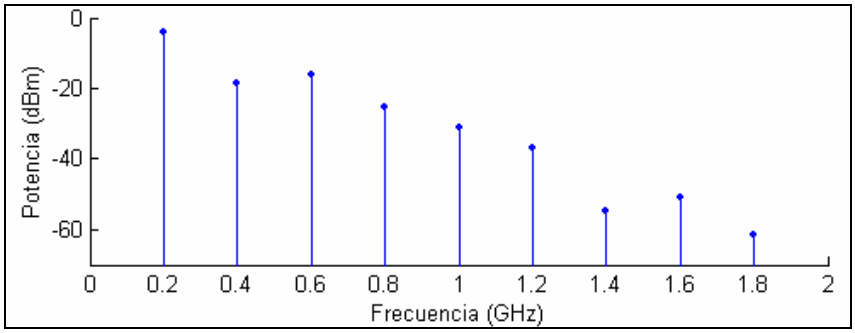


Figura 168: Potencia de los distintos armónicos del oscilador.

Utilizando estos datos se puede calcular la distorsión armónica total de la señal. Esta se define como el porcentaje de potencia contenida en los armónicos con respecto a la potencia total de la señal, armónicos y componente fundamental. Por lo tanto se puede calcular así:

$$DAT(\%) = \frac{\sum_{i=2}^{9} P_i}{\sum_{i=1}^{9} P_i} \cdot 100$$

Solo hay que tener en cuenta que las potencias deben sumarse en unidades lineales, no logarítmicas, para llegar fácilmente al resultado de:

DAT = 9.97%

Un valor bastante bajo, que es lo deseable. Otra manera de visualizar lo parecido a un tono puro que es la salida del oscilador es reconstruir lo que sería un cronograma de la señal a partir del análisis espectral que se ha realizado. Esta visualización no se puede realizar normalmente con un osciloscopio, al menos en la mayoría de ellos, debido a la alta frecuencia de la señal. Sin embargo a partir de las potencias de cada armónico se puede calcular la amplitud en voltaje de cada componente armónica. Esto se ha hecho con un *script* en MATLAB, con el resultado que se muestra abajo. Como se ve el resultado no es una sinusoidal perfecta pero tampoco presenta demasiada distorsión.

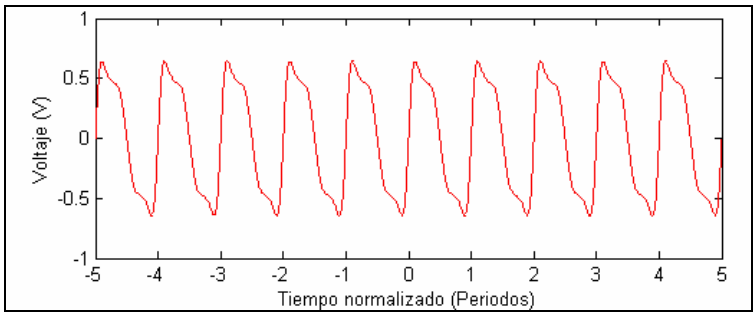


Figura 169: Reconstrucción de la salida del oscilador.

11.2.6.Tiempo de enganche

Este parámetro (*time-to-lock*) indica el tiempo que el PLL tarda en cambiar desde estar enganchado a una frecuencia hasta volver a engancharse a otra nueva. Para medir este tiempo se ha realizado el siguiente experimento: Utilizando el módulo de control manual de la aplicación *SWRadioStudio*, se ha configurado el PLL para engancharse a 170MHz, un valor prácticamente en el extremo inferior del rango de frecuencias funcionamiento. También se ha seleccionado la salida MUXOUT en el modo de detector de enganche digital. En este estado, se ha medido usando un osciloscopio digital la señal MUXOUT, disparado mediante flanco de bajada, mientras en el *SWRadioStudio* se volvía a mandar al PLL un comando para cambiar la frecuencia de funcionamiento a 220MHz, prácticamente en el extremo superior del rango de funcionamiento. Así se ha obtenido la siguiente imagen, donde se ve el tiempo que el PLL permanece en estado de *unlocked*, representado por un nivel lógico bajo. Este tiempo es de aproximadamente 500µs y se ha comprobado que aunque no siempre se obtiene exactamente el mismo tiempo, hay poca variación si se cambian las frecuencias inicial y final.

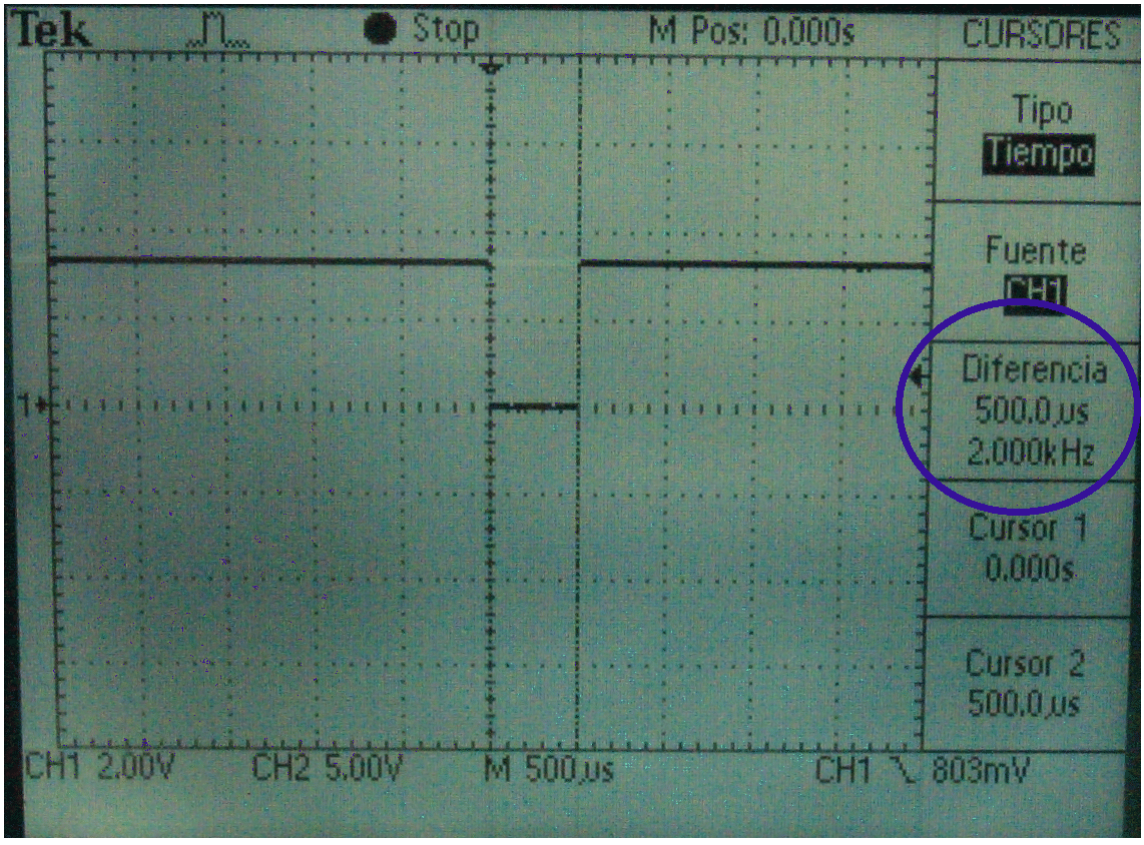


Figura 170: Medición con el osciloscopio del tiempo de enganche del PLL.

Este mismo experimento se puede realizar midiendo la señal de control del VCO, obteniéndose aproximadamente la esperada respuesta al escalón de un sistema paso bajo infra-amortiguado:

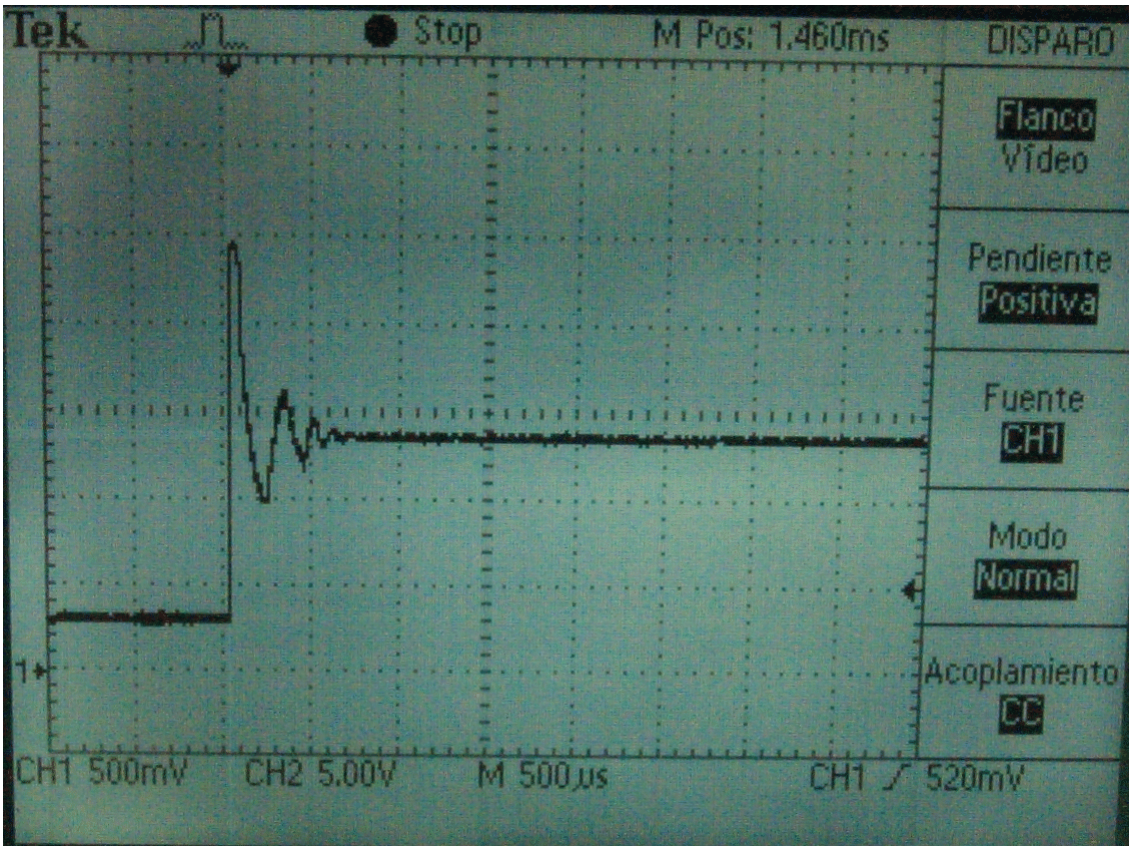


Figura 171: Medición con el osciloscopio de la señal de control del VCO durante el enganche.

Aunque en esta gráfica no esté tan claro como en la anterior, sí que se puede comprobar también que el intervalo hasta el enganche es de aproximadamente una división horizontal que resulta de nuevo en unos 500µs.

11.3. Conversores A/D

En este apartado se pretende caracterizar el comportamiento de los conversores analógico digitales que son la entrada de las señales en banda base I / Q del radiorreceptor. Para ello se han conectado directamente sendos generadores de señales a las entradas de los conversores, desconectándolos de su entrada normal proveniente del demodulador. Hay que recalcar que esta prueba no incluye los filtros *antialiasing*, por estar estos en la parte del demodulador. Estas mediciones sirven para caracterizar la respuesta de los conversores frente a situaciones de submuestreo, entre otras cosas.

11.3.1.Respuesta en frecuencia

Para caracterizar la respuesta de los conversores ADC en frecuencia, se han tomado muestras, usando la opción de capturar una ráfaga de muestras de *SWRadioStudio*, de los canales I y Q, aunque solamente se introduce una señal sinusoidal por el canal Q. Al ir variando la frecuencia de esta sinusoidal variará la amplitud del tono en el dominio de las muestras. Para medir la potencia del tono digitalizado se ha ejecutado el comando *psd* (*power spectral density*) en MATLAB sobre las muestras, convenientemente enventanadas con una ventana de hanning para evitar los efectos de las diferentes frecuencias en la medida de la potencia. Los resultados son estos:

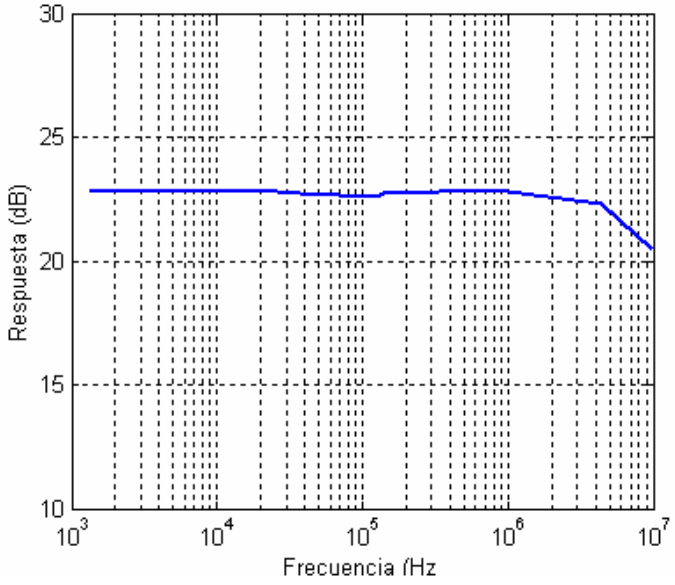


Figura 172: Respuesta en frecuencia de los ADC.

Como se puede ver, al tratarse de confesores con un alto ancho de banda, la respuesta no empieza a decaer hasta los 10Mhz, límite del generador de funciones de que se disponía para hacer las pruebas, por lo que ni siquiera se ha podido llegar al punto de -3dB.

11.3.2.SNR en función de la frecuencia

Realizando la misma prueba que antes, pero midiendo esta vez la diferencia en potencia entre el tono principal y el siguiente máximo entre el ruido, incluyendo en este los posibles efectos de la distorsión, se obtiene esta respuesta:

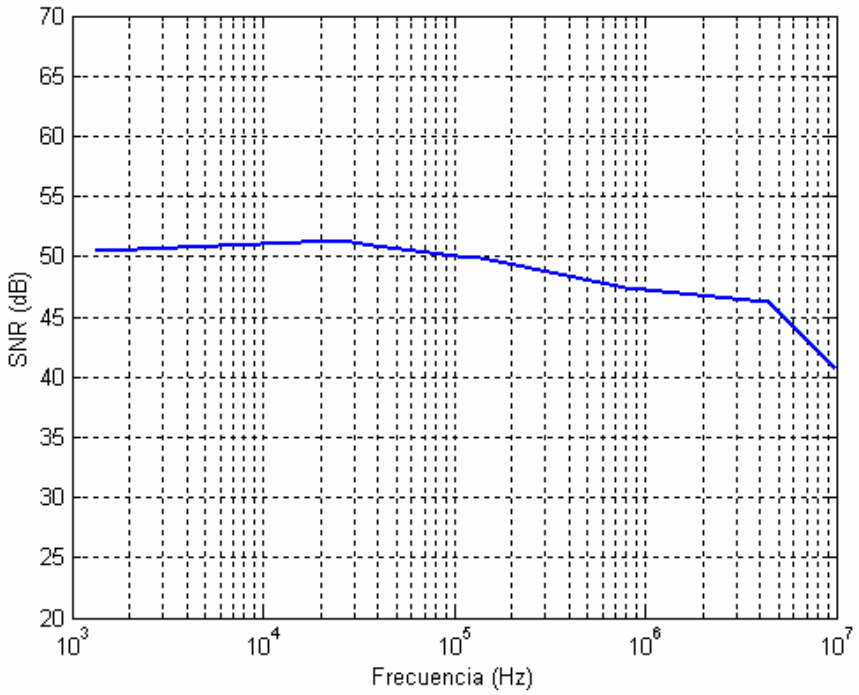


Figura 173: Relación SNR en función de la frecuencia.

11.3.3.Aislamiento entre canales

Para caracterizar el aislamiento entre canales se ha introducido por el canal I una señal de 0.75Vpp a 90KHz, y por el canal Q una de igual amplitud pero a 70Khz. Ambas señales están en el rango de frecuencias normal que se espera a la entrada del receptor, y tienen unas amplitudes moderadas. Estas son las densidades espectrales de potencias de las dos señales recibidas, donde se puede comprobar como parte de la potencia de cada canal pasa al otro:

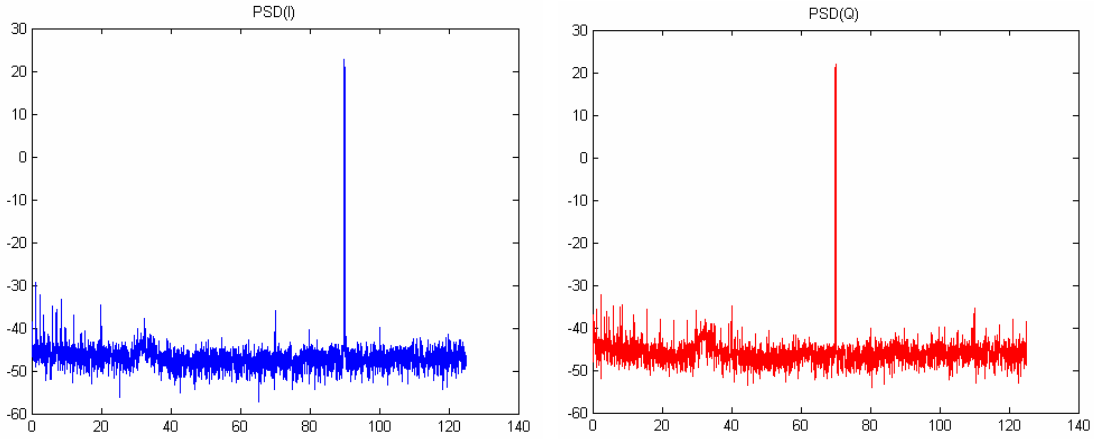


Figura 174: Densidades espectrales de potencia para las dos señales I y Q.

En el espectro correspondiente al canal I, de 90KHz, se puede observar un pico en la frecuencia del canal Q, de 70KHz. Midiendo la diferencia en potencia entre los dos picos se puede estimar, para este caso, el aislamiento entre canales, que resulta ser de 59dB.

11.4. Filtro de preselección y LNA

En este apartado se muestran las medidas de la respuesta de los dos primeros subsistemas en el camino de la señal en RF, que son el filtro de preselección y el amplificador LNA, para obtener la respuesta en frecuencia para el filtro y la ganancia del amplificador. En este caso, tanto las impedancias de entrada como de salida son de 50Ω por lo que se han conectado directamente un generador de RF y un analizador de espectros a la entrada del receptor y la salida del LNA, respectivamente.

Las medidas se han hecho realizando un barrido en la frecuencia del generador y apuntando la potencia tanto la recibida por el analizador de espectros como la generada. La potencia de entrada no se ha mantenido constante debido a las grandes variaciones (hasta de 75dB) de la respuesta, que hace que en ciertos rangos de frecuencia la señal a la salida sea tan poco potente que cueste distinguirla fácilmente de otras señales interferentes. En las siguientes tablas se muestran los resultados.

Frecuencia (MHz)	Potencia de entrada (dBm)	Potencia de salida (dBm)
5	-12	-67
15	-12	-46
25	-12	-35.5
35	-30	-46.8
45	-30	-40.1
55	-30	-33.8
65	-50	-47
75	-50	-40.8
85	-50	-37.3
90	-50	-35.9
95	-50	-36.2
100	-50	-35.9
105	-50	-37.7
110	-50	-40.1
115	-50	-42.5
120	-50	-44.8
140	-50	-48.4
180	-30	-45.2
230	-12	-53.4
270	-12	-56.2

Frecuencia (MHz)	Potencia de entrada (dBm)	Potencia de salida (dBm)
310	-12	-58.7
380	-12	-69.0
400	5	-56.6
420	5	-48.5
430	5	-43.5
440	5	-38.1
450	5	-31.8
460	5	-30.8
470	5	-34.9
480	5	-39.0
520	5	-37.1
580	5	-29.1
650	-10	-31.1
700	-10	-39.0
750	-10	-32.1
800	-10	-27.5
850	-10	-27.1
900	-10	-26.2
950	-10	-24.5
1000	-10	-22.2

La ganancia para cada frecuencia se obtiene como la diferencia entre las potencias de salida y de entrada. Si se representa esta ganancia gráficamente, se puede estimar su valor máximo, en la banda de paso y el ancho de banda de tres decibelios.

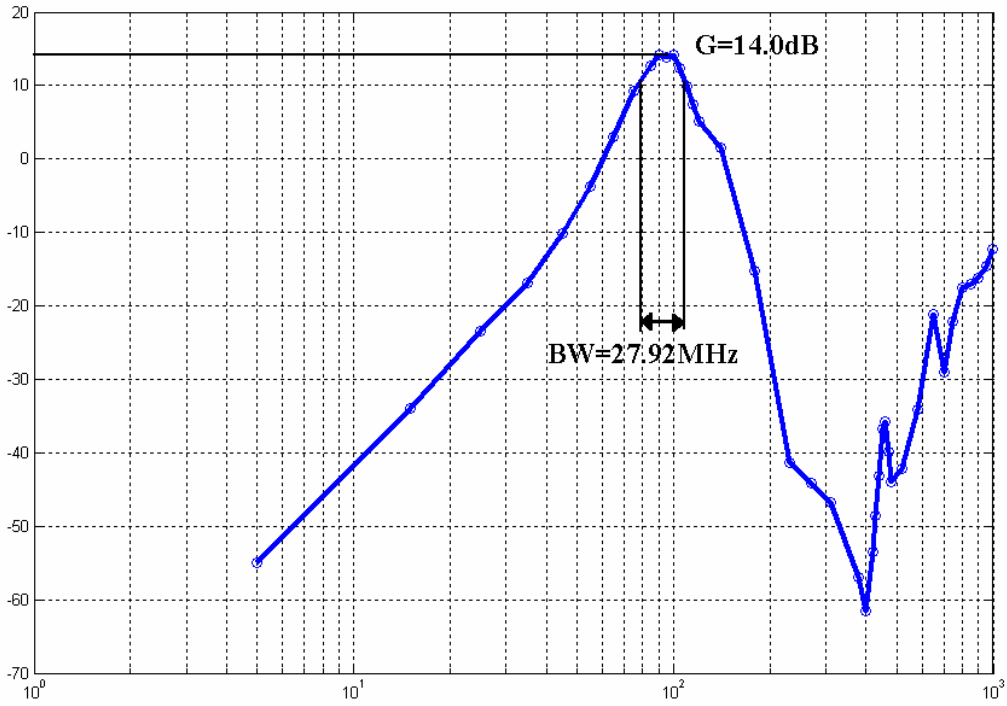


Figura 175:Respuesta en frecuencia del filtro y el LNA

Se ha obtenido una ganancia de potencia de 14.0dB, aunque hay que tener en cuenta que uniendo directamente el generador al analizador se observa una diferencia de 1dB-2dB, por lo que la ganancia realmente podría estar en torno a 15dB. Teniendo esto en cuenta, y asumiendo que las pérdidas se producen principalmente en el filtro de preselección, se puede tomar como pérdidas de éste:

$$L_{\text{Filtro preselec}} = 18\ \text{dB} - 15\text{dB} = 3\text{dB}$$

En cuanto a la banda de paso, situada entre los cortes de la ganancia con 11dB (ancho de banda a tres decibelios del máximo), se obtienen los extremos 79.98MHz y 107.9MHz. A continuación se resumen los datos medidos y los valores para los que se diseñaron idealmente estos subsistemas (ver apartado 6.1).

Parámetro	Valor de diseño	Valor medido
Ganancia del LNA	18dB	15dB
Banda de paso	85.8MHz – 118.2MHz	79.98MHz – 107.9 MHz
Frecuencia central	102.0 MHz	93.94MHz
Ancho de banda a 3dB	32.4MHz	27.92MHz

Se pueden observar dos hechos importantes:

- La respuesta del filtro ha resultado desplazada hacia frecuencias más bajas. Aunque el ancho de banda mínimo de diseño global se cumple, de 88Mhz a 108Mhz, este hecho justifica la decisión tomada en 4.1 de diseñar para un margen más amplio del que realmente se necesitaba.
- Mientras que en bajas frecuencias el filtro se comporta según lo esperado, en la gráfica se puede ver que a partir de 400MHz la atenuación comienza a disminuir, observándose picos de resonancia. Este efecto se puede achacar al aspecto tecnológico de selección de componentes, especialmente los materiales dieléctricos de los condensadores y al material de las inductancias. Aunque todos los componentes que se han usado son de montaje superficial (SMD, concretamente del tamaño 0805), estos efectos no se pueden evitar sin recurrir a materiales más caros. El problema es que los condensadores e inductancias se comportan como tales solamente hasta una determinada frecuencia, tienen una frecuencia de resonancia, y a partir de ahí presentan una impedancia de signo contrario al normal.

11.5. Amplificador VGA

En este apartado se caracteriza la ganancia en potencia de este subsistema según la frecuencia y el voltaje de control. Para ello se conectó el generador de RF a la entrada del receptor, y el analizador de espectros a la salida de la red adaptadora de impedancias que está tras este amplificador. En realidad se mide la respuesta en frecuencia del conjunto de todas estas primeras etapas de RF, pero como la ganancia en la banda de paso del LNA ya se caracterizó en el apartado anterior, la de este amplificador resultará de la diferencia entre la ganancia observada y la de la etapa anterior. Fuera de la banda de trabajo, se observará también el efecto combinado del filtro de preselección y de las dos redes adaptadoras, formando un filtro paso banda. En cuanto a las impedancias, a la entrada no hay problema para conectar el generador de RF, ya que la entrada del receptor tiene una impedancia de 50Ω, pero la red adaptadora a la salida está diseñada para una carga de 200Ω, por lo que se ha añadido una resistencia de 150Ω en serie a la entrada del analizador de espectros. Hay que tener en cuenta el efecto de esta resistencia en la potencia detectada por el analizador de espectros para corregir estas medidas y calcular la potencia entregada a una carga adaptada de 200Ω.

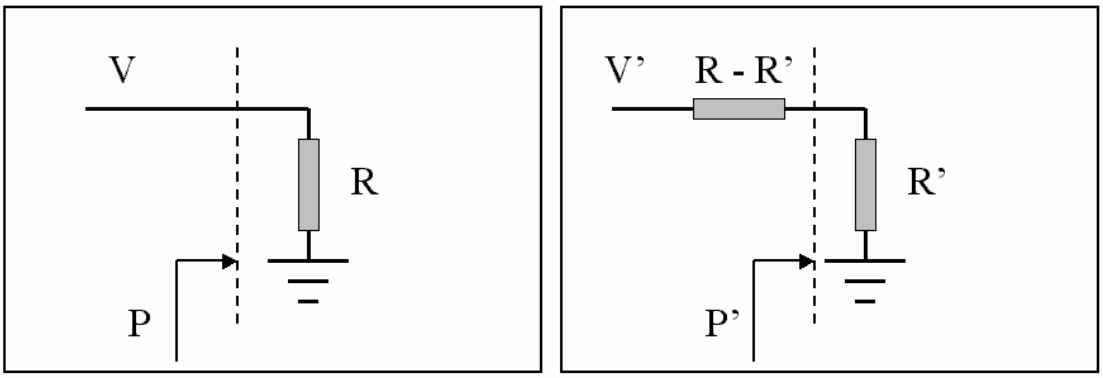


Figura 176: Efecto de la resistencia en serie a la entrada del analizador de espectros

En la primera parte de la figura se ve el caso de que a la salida del VGA se conectara una carga resistiva adaptada, de valor R=200Ω. La potencia entregada en este caso dependería del voltaje de salida (supuesto el valor en r.m.s. aunque no es relevante):

$$P = \frac{V^2}{R}$$

Ahora bien, en estas medidas lo que hay a la salida en realidad es una resistencia en serie con la entrada del analizador de espectros (segunda parte de la figura), de forma que entre las dos sumen R=200Ω. Como en las dos situaciones la impedancia de salida es idéntica, se tiene que V es constante en ambos casos. Llamando V' al voltaje que cae en la resistencia R' (que representa la entrada al analizador de espectros), su relación con V viene dada por el divisor de voltaje:

$$V' = V\frac{R'}{R}$$

Y la potencia disipada en R', es decir la potencia detectada por el analizador de espectros, viene dada por:

$$P' = \frac{V'^2}{R'} = \frac{V^2}{R'}\left(\frac{R'}{R}\right)^2 = \frac{V^2}{R^2/R'}$$

Siendo la relación entre ambas potencias:

$$\frac{P}{P'} = \frac{\frac{V^2}{R}}{\frac{V^2}{R^2/R'}} = \frac{R}{R'} = \frac{200\Omega}{50\Omega} = 4 = 6dB$$

Es decir, la potencia medida con el analizador de espectros será 6dB inferior a la que teóricamente recibirá el demodulador que sí presenta una impedancia de entrada de 200Ω.

A continuación se muestran las medidas en las que se ha caracterizado la ganancia de potencia en función del voltaje de control y el punto de compresión de ganancia. Para ello se ha usado el generador de radiofrecuencia a una frecuencia fija dentro del rango de trabajo y de la banda de paso, a 102.5Mhz. En la tabla se indica tanto la potencia detectada por el analizador de espectros como la potencia de salida real que se entregaría a una carga de 200Ω, obtenida sumándole 6dB a la anterior, la potencia de entrada para cada caso, el voltaje de control usado y la ganancia resultante:

V_{GAIN} (V)	P_{IN} (dBm re 50Ω)	P_{AE} (dBm re 50Ω)	P_{OUT} (dBm re 200Ω)	G (dB)
1	-90	-41.7	-35.7	54.3
	-80	-33.8	-27.8	52.2
	-70	-23.5	-17.5	52.5
	-60	-13.5	-7.5	52.5
	-50	-2.5	+3.5	53.5
	-40	-0.4	+5.6	45.6
0.5	-90	-61.8	-55.8	34.2
	-80	-53.1	-47.1	32.9
	-70	-42.8	-36.8	33.2
	-60	-32.0	-26.0	34.0
	-50	-22.4	-16.4	33.6
	-40	-12.0	-6.0	34.0
	-30	-3.1	+2.9	27.1
	-20	-0.4	+5.6	14.4
0	-90	-90	-84	6
	-80	-76.2	-70.2	9.8
	-70	-65.9	-59.9	10.1
	-60	-55.9	-49.9	10.1
	-50	-46.2	-40.2	9.8
	-40	-36.2	-30.2	9.8
	-30	-25.2	-19.2	10.8
	-20	-15.2	-9.2	10.8
	-10	-6.8	-0.8	9.2

Para los valores de la tensión de control VGAIN se han usado los dos extremos (el de máxima ganancia, que es de 1V y el de mínima, de 0V) y un punto medio (el de 0.5V). Estas medidas se pueden representar de forma gráfica de dos formas. La primera, que se presenta abajo, muestra la ganancia de potencia en función del voltaje de control y la potencia de entrada.

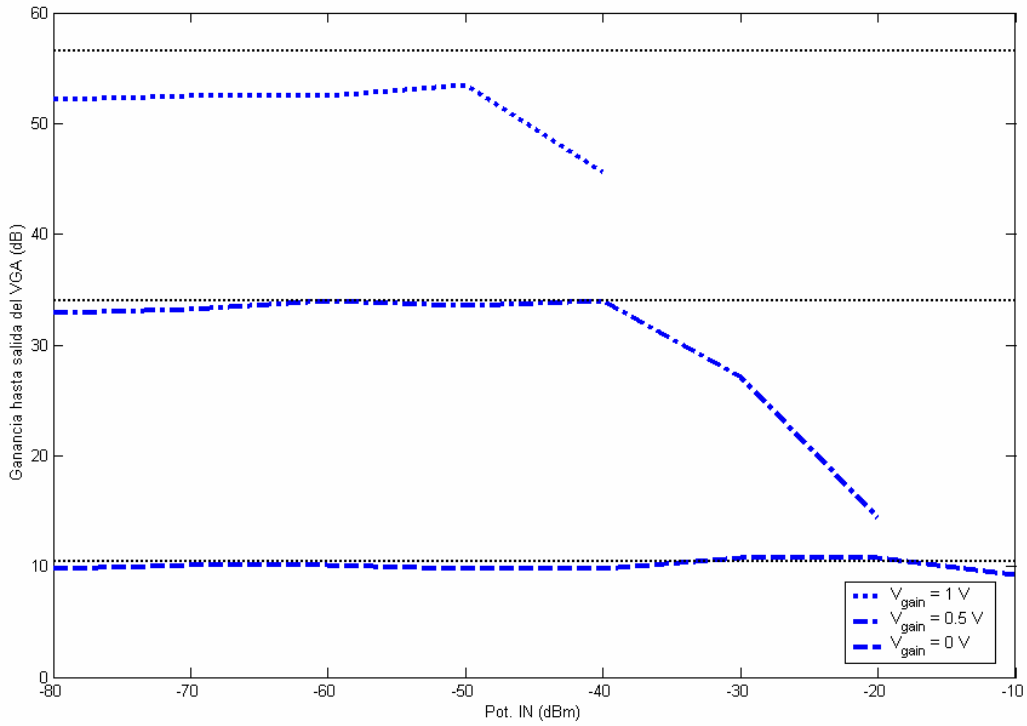


Figura 177: Ganancia hasta la salida del VGA

Se puede apreciar perfectamente cómo para cada valor de ganancia existe una zona de comportamiento lineal, es decir, de ganancia constante hasta llegar al punto de saturación del dispositivo en que la ganancia empieza a disminuir. Como el límite realmente consiste en un valor máximo de ganancia a la salida es lógico que este punto de compresión de la ganancia esté más a la izquierda (menores potencias de entrada) cuanto mayor sea el voltaje de control (mayor la ganancia).

Sabiendo del apartado anterior que la ganancia desde la entrada de radiofrecuencia hasta la entrada del VGA es de 14dB para la banda de paso, se puede estimar la ganancia esperada para cada voltaje de control en función de esta ganancia y de la ganancia de potencia ideal descrita en el *datasheet* del VGA:

$$G_{VGA}\,(dB) = -5.6dB + V_{GAIN} \cdot 50.2\ dB/V$$

Utilizando esta interpolación lineal se puede estimar la ganancia del VGA para 1V, 0.5V y 0V como 42.5dB, 20dB y –2.5dB, respectivamente. Sumando estas ganancias a los 14dB de la etapa anterior, se puede estimar la ganancia esperada, y esto es lo que se ha hecho en la gráfica anterior, donde esta predicción está representada por las tres líneas discontinuas horizontales. Se puede comprobar como para dos de los casos el error es menor de 1dB, mientras que para el otro es de unos 4dB.

La otra forma de representar gráficamente los resultados de estas medidas es mediante una gráfica de potencia de salida respecto a la potencia entrante.

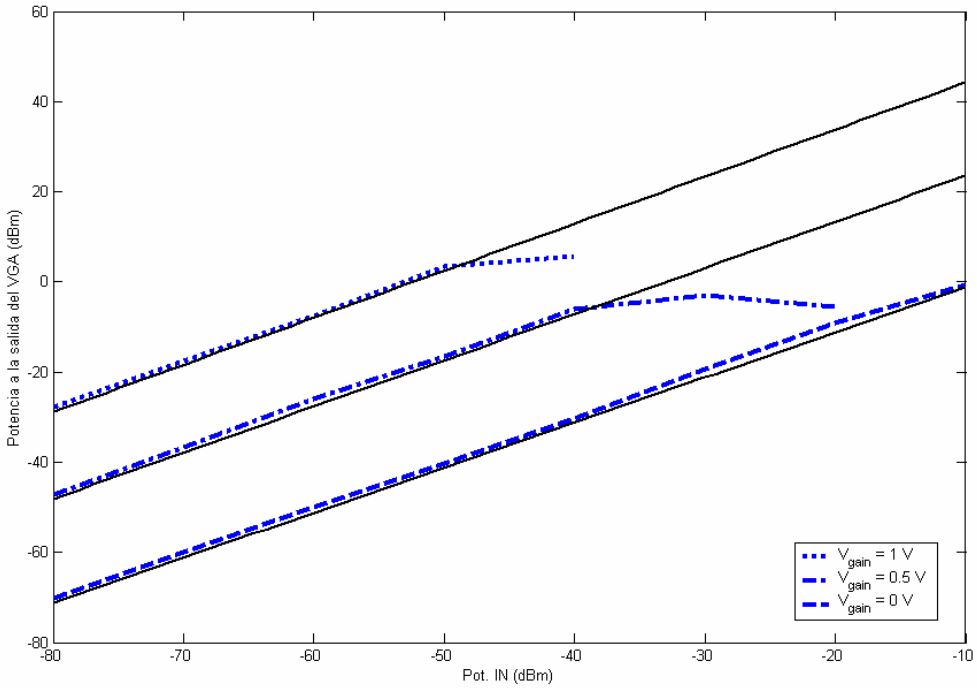


Figura 178: Potencia de salida en función de la potencia de entrada

Sobre las tres gráficas se ha calculado la linealización de la potencia de salida suponiendo comportamiento lineal, es decir, ganancia constante. Las tres líneas rectas que aparecen en la gráfica corresponden a estas extrapolaciones lineales, menos 1dB. Buscando las intersecciones de estas rectas con las curvas, mirando la potencia de entrada correspondiente y sumando la ganancia del VGA, se obtiene que el punto de compresión de 1dB a la salida del amplificador está en torno a los 0dBm.

11.6. Filtros de banda base

El objetivo de los dos filtros de banda base, ambos idénticos para las señales I y Q, es doble. Por un lado, limita el ancho de banda de la señal para evitar el *aliasing* en los conversores analógico digitales, y por otro el ancho de banda eliminado significa ruido eliminado y por lo tanto disminuye la cantidad de ruido antes de que las señales sean amplificadas por el último amplificador dentro del integrado del demodulador en cuadratura.

Para caracterizar estos filtros, se ha conectado un generador de radiofrecuencia a la entrada del receptor, generando una portadora sin modular en 92.0MHz. Se ha sintonizado el receptor a esa misma frecuencia, y desde el módulo de control manual de *SWRadioStudio* se ha medido la potencia de señal recibida en banda base, mientras se variaba la frecuencia de la portadora. El efecto de variar la portadora es que en banda base se recibe un tono a la frecuencia de la diferencia de frecuencias. Por lo tanto, se utiliza esta diferencia de frecuencias para comprobar la respuesta en amplitud para cada desplazamiento de frecuencia. Se han obtenido estas medidas:

Frecuencia con respecto a la portadora (KHz)	Potencia relativa (dB)
20	0
30	-1.0000
40	-1.4000
50	-1.5000
60	-1.4800
70	-1.3000
80	-1.3500
90	-1.6000
100	-2.4000
110	-4.7000
120	-10.6000
130	-18.6000
140	-22.6000
150	-30.6000
200	-72.6000

Que se presentan a continuación de forma gráfica:

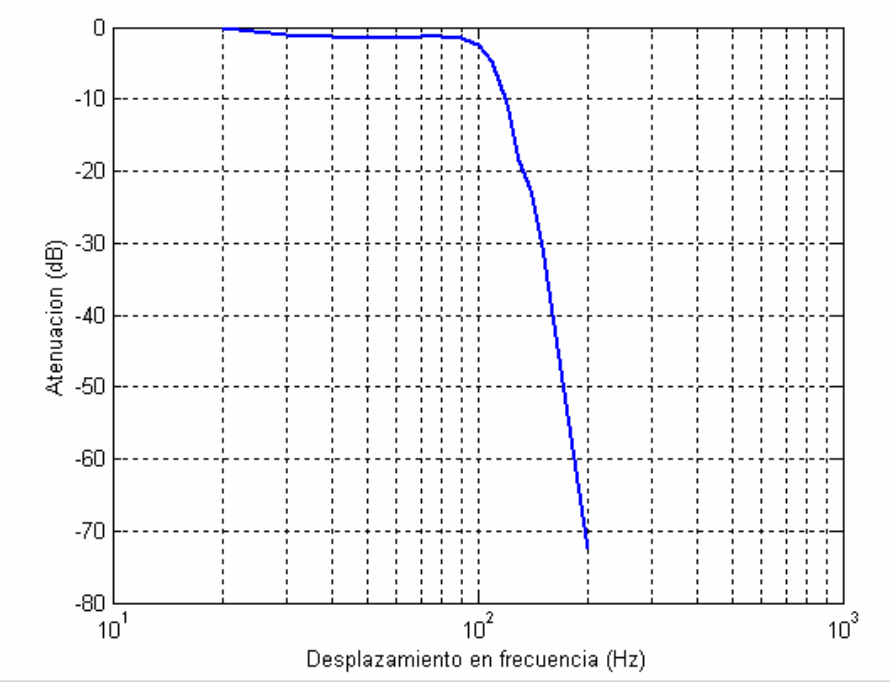


Figura 179. Respuesta en frecuencia de los filtros en banda base

Aunque el diseño del filtro especificaba una frecuencia de corte a tres decibelios de 110KHz, la frecuencia de corte medida realmente es la de 102KHz. Esto se puede justificar por:

- Las limitadas tolerancias de los componentes, aún siendo componentes de tolerancias de 2% y 5%, pueden modificar la posición exacta de los polos y ceros del filtro, llevando a grandes variaciones de la respuesta en frecuencia.
- Las inductancias reales presentan una resistencia no modelada en el diseño teórico.

Este último punto parece haber resultado determinante y haber incidido muy negativamente incluso en características globales del receptor. Una bobina real se puede modelar como una bobina ideal, junto a una resistencia en serie que modela la resistencia, tanto óhmica como dependiente de la frecuencia, y una capacidad que representa las capacidades parásitas de los hilos o el encapsulado.

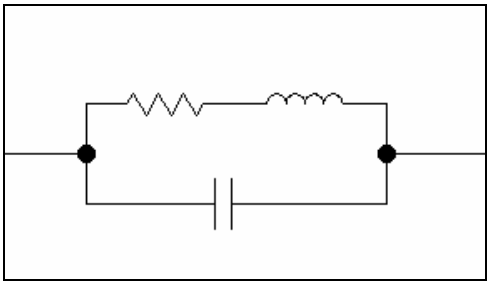


Figura 180: Modelo de una inductancia real

El efecto de la capacidad es que las bobinas reales tienen una frecuencia de resonancia, la cual hay que evitar con un cierto margen de seguridad por abajo. Esta frecuencia se ha tenido en cuenta para la elección de las inductancias de estos filtros, y no representa un problema en esta situación. Sin embargo, los efectos de las resistencias son varios:

- Aumentan las pérdidas, incluso en la banda de paso ya que influyen en el camino de corriente continua.
- Desajustan la respuesta en frecuencia global del filtro, debido al desplazamiento de polos y ceros en la función de transferencia ideal de éste, provocando normalmente mayores pérdidas aún y mayores rizados de la respuesta en frecuencia.

Para confirmar el efecto de estas resistencias se han medido en las inductancias reales, resultando en unos 15Ω. Utilizando la aplicación de simulación electrónica *Electronics Workbench 5.12*, se ha modelado el filtro de banda base, incluyendo en el modelo estas resistencias para las inductancias.

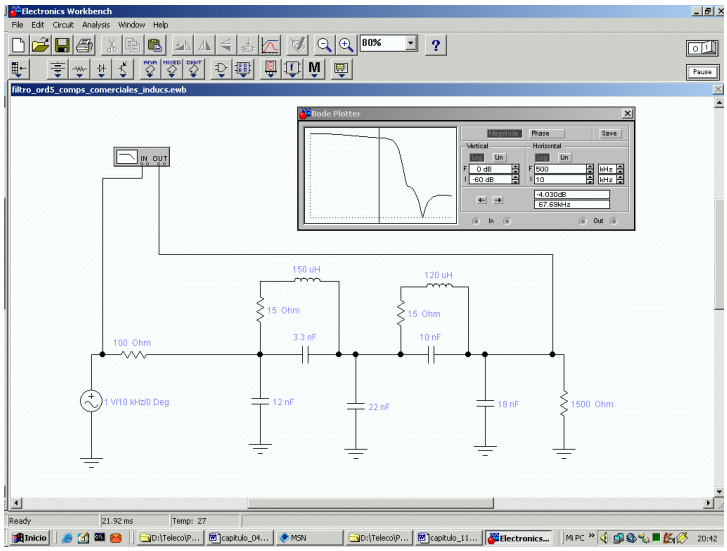


Figura 181: Modelado del filtro incluyendo las resistencias de las inductancias.

A continuación se muestran los gráficos con los resultados de simular la respuesta en frecuencia del filtro usando un modelo de inductancias ideales y usando un modelo de inductancias con una resistencia de 15Ω en serie. Se pueden observar dos hechos relevantes para la aplicación actual: se añaden 1.8dB de pérdidas en la banda de paso y la frecuencia de corte pasa de 109KHz a 88KHz. Aunque en esta simulación la frecuencia de corte ha disminuido más incluso que en el filtro real según las mediciones, esto demuestra que la causa principal de este desacuerdo con lo diseñado muy posiblemente sea la no-idealidad de las inductancias utilizadas.

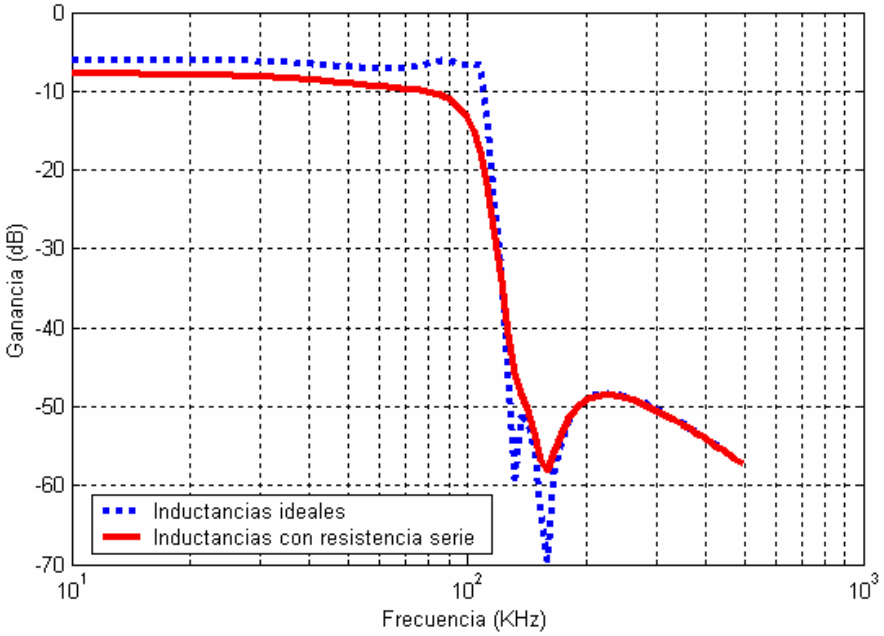


Figura 182: Simulación de la respuesta para inductancias ideales y reales.

11.7. Medidas del receptor completo

11.7.1.Ganancia total

Para estimar la ganancia total del receptor, se ha conectado un generador de radiofrecuencia a la entrada del receptor, en la frecuencia de 92Mhz (en la mitad de la zona de paso), y se ha sintonizado el receptor a esa misma frecuencia. Debido al desvío de frecuencias, que siempre ha resultado ser de 11KHz, entre el oscilador local del receptor y el generador, aparece un tono a esta frecuencia en la señal base banda compleja. Utilizando el módulo de control manual de *SWRadioStudio*, y las gráficas de espectro de frecuencias en escala logarítmica de la señal en banda base, se ha podido medir la potencia de la señal recibida para distintos valores de potencias de entrada, con lo que se puede estimar la ganancia del receptor. La configuración de las ganancias ha sido:

- Amplificador VGA de RF: Mínima ganancia.
- Demodulador: Máxima ganancia.

En la siguiente tabla se muestran los valores de potencias de entrada, en dBm referidas a 50Ω, la potencia del tono recibido, en $dBVrms^2$, y la ganancia de potencia resultante. Para calcular esta ganancia de potencia se han pasado ambas potencias a valores de voltaje *rms*, sabiendo que:

$$V_{out}\ (V_{rms}) = 10^{\,Pout\ (dBVrms2)\ /\ 20}$$

$$Vin\ (V_{rms}) = \sqrt{50 \cdot 10^{Pin(dBm)/10} \cdot 10^{-3}}$$

Con lo que la ganancia queda simplemente:

$$G(dB) = 20 \cdot \log_{10}(V_{out} / V_{in})$$

Pin (dBm)	Pout (dBVrms^2)	Ganancia (dB)
-102	-30	85.01
-95	-21	87.01
-90	-17	86.01
-85	-12.2	85.81
-80	-7.8	85.21
-75	-3.8	84.21
-70	-1.14	81.87
-65	-0.15	78.16

Por lo tanto se puede decir que para la parte de comportamiento lineal, la ganancia está en torno a **85dB**. Sin embargo, el receptor no está configurado para la máxima ganancia, ya que el VGA de radiofrecuencia está a mínima ganancia. Esto representa una ganancia teórica de de -3.5dB en lugar del máximo de 42.5dB, y por lo tanto una ganancia 46dB por debajo de la máxima posible en teoría. Esto deja la máxima ganancia teórica del receptor en 85dB + 46dB = **131 dB**. Sin embargo al aumentar la ganancia total por encima de **100dB** aproximadamente**,** el ruido a la salida aumenta considerablemente, inutilizando estas configuraciones para la recepción de señales. Estos valores de ganancia se pueden comparar con el valor de ganancia máxima estimado teóricamente en el capítulo 9, donde se llegó a una estimación de unos 123dB para la ganancia máxima.

11.7.2.Sensibilidad

Para la configuración anterior, se mide un nivel de ruido de fondo de -33dBVrms^2. Si definimos la sensibilidad como la potencia de entrada para la que la señal en banda base está 10dB por encima del nivel de ruido, ésta se puede calcular representando gráficamente los valores anteriores:

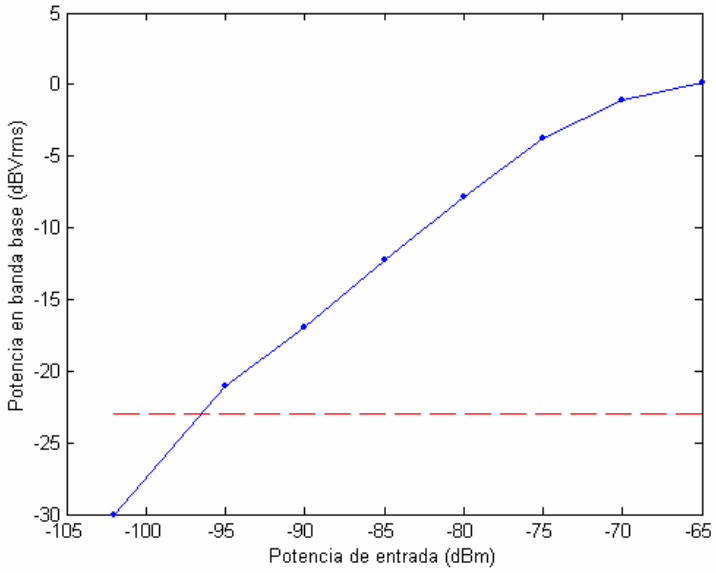


Figura 183: Estimación de la sensibilidad del receptor.

Donde la línea discontinua representa el nivel de 10dB por encima del nivel de ruido. Se puede comprobar que para $\mathbf{P_{in}}$ **= -96dBm** esta condición se cumple, por lo que esta es la sensibilidad del receptor para la configuración en que se ha realizado el experimento.

Este valor está basta próximo al valor de la sensibilidad estimado teóricamente en el capítulo 9, que resultó de ser de -100dBm.

11.7.3.Punto de compresión de 1 dB

Realizando las mismas medidas que en el apartado 11.7.1, pero en el caso de mínimas ganancias tanto para el VGA como para el demodulador, se puede estimar el punto de compresión de 1dB del receptor. Interesa realizar la prueba para la mínima ganancia ya que así se tendrá el punto de compresión más alto de todas las configuraciones posibles, y por lo tanto el limitante. Estos son los datos recogidos, donde la ganancia en potencia se ha calculado de forma idéntica a los apartados anteriores.

Pin (dBm)	Pout ($dBVrms^2$)	Ganancia (dB)
-85	-53	45.01
-80	-48.7	44.31
-75	-44.3	43.71
-70	-39.2	43.81
-65	-34.5	43.51

-60	-29.6	43.41
-55	-24.5	43.51
-50	-19.9	43.11
-45	-15.22	42.79
-40	-11.2	41.81
-35	-7.48	40.53
-30	-2.9	40.11
-25	-0.8	37.21

Se puede observar que la ganancia mínima media está en torno a los **43dB**, mientras que la ganancia mínima estimada teóricamente era de 32.8dB.

Representando los resultados de forma gráfica, se puede apreciar la alta linealidad de la ganancia para la mayor parte del rango de potencias de entrada, como se ve en la siguiente figura.

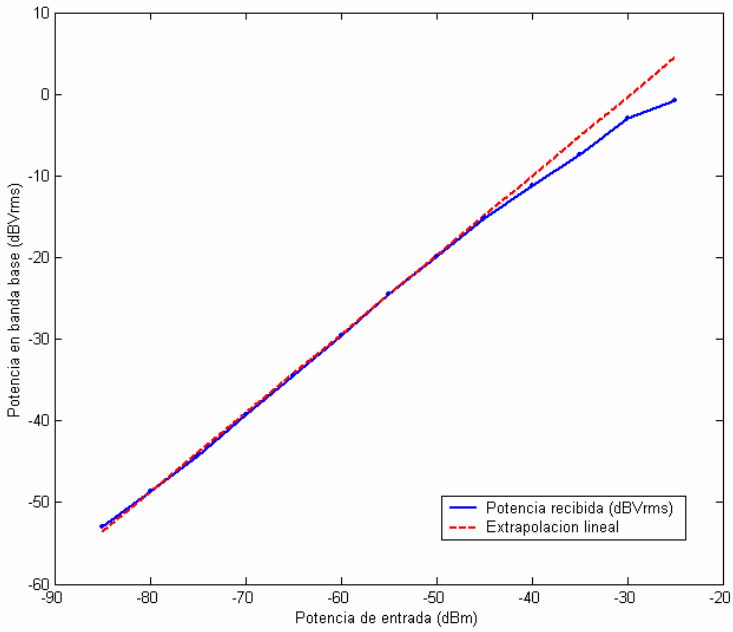


Figura 184: Potencia recibida en banda base en función de potencia entrante de RF.

En línea discontinua se ha representado la extrapolación lineal de la potencia recibida suponiendo ganancia lineal y constante. Para calcular el punto de compresión de 1dB, se representa gráficamente la diferencia entre esta extrapolación lineal y la ganancia realmente medida:

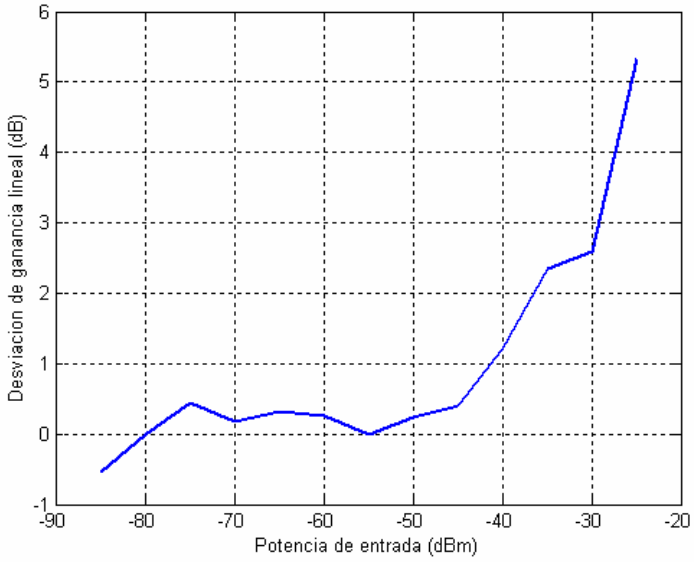


Figura 185: Diferencia de la ganancia con su linealización.

Se puede ver cómo la diferencia pasa por 1dB en torno a los **-40dBm** re 50Ω de potencia de entrada, por lo que este valor es el punto de compresión del receptor. Este valor se puede comparar con el estimado en la parte teórica, donde se obtuvo un valor de -28.6dBm para este punto de compresión. Por lo tanto el valore realmente medido es algo peor al estimado.

11.7.4.Margen dinámico

Si tomamos como límite superior del margen dinámico la potencia de entrada al receptor que provoca un 1dB de pérdida de ganancia, y como límite inferior la mínima potencia de entrada que permite una relación señal a ruido de 10dB, es decir, la sensibilidad, tenemos un margen comprendido entre:

$$-96\text{dBm} \leftrightarrow -40\text{dBm}$$

Por lo que se tiene un margen de **56dB** de margen dinámico. En este caso el margen es inferior que el estimado teóricamente, que se calculó en 71.4dB, debido en la mayor parte por la diferencia en el valor del punto de compresión de 1dB con respecto al estimado teóricamente.

11.8. Pruebas con los demoduladores software

11.8.1.FM

En este apartado se muestran algunas imágenes y resultados de medidas que demuestran el buen funcionamiento del demodulador software para señales FM en tiempo real para el entorno *SWRadioStudio*, cuyo diseño se expuso en el apartado 7.4.2.

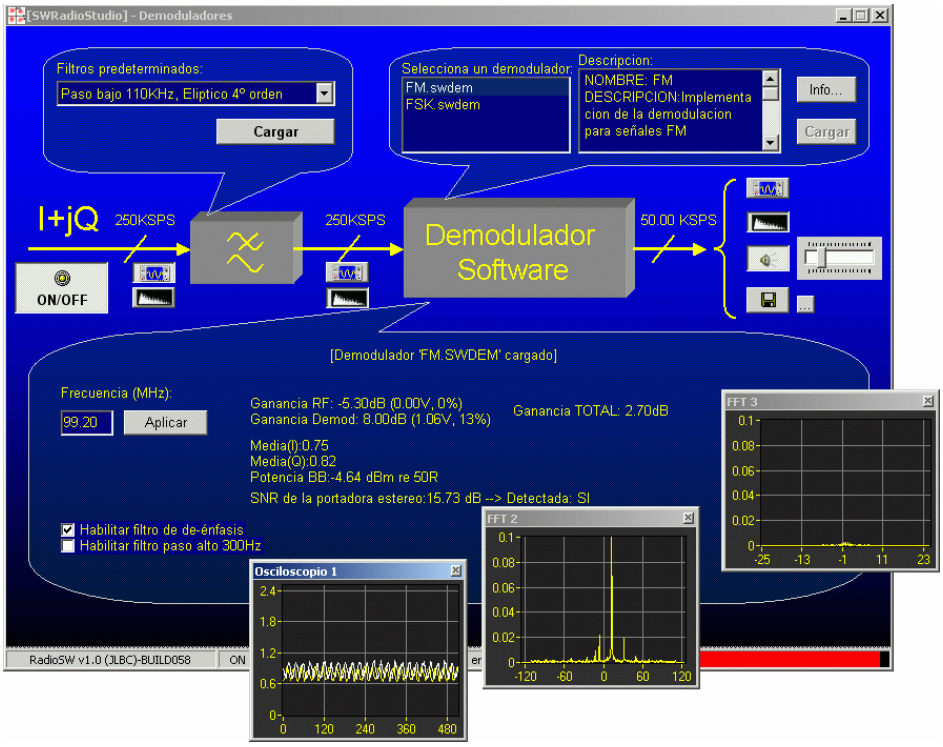


Figura 186: Aspecto del módulo de demodulación mientras se recibe una señal de FM comercial.

Prueba subjetiva de calidad del sonido y CAG

Para esta prueba se ha utilizado el generador de radiofrecuencia *Marconni* disponible en el laboratorio, configurado con estos parámetros:

Frecuencia portadora: 93.0MHz, modulación FM

Tono modulador: 1KHz

$\Delta f = 25KHz$

En la parte del *SWRadioStudio*, se ha seleccionado el demodulador software de FM, y el filtro de banda base de 50KHz, que es suficiente para recibir esta señal modulada. A continuación se ha comenzado colocando el generador en su mínima potencia, -110dBm y se ha ido aumentando la potencia generada y describiendo lo que va ocurriendo con la potencia recibida, la ganancia controlada mediante un bucle de AGC y la calidad subjetiva del sonido demodulado:

Potencia entrada (dBm)	Calidad del sonido	Potencia en banda base (dBm)	Ganancia (solo del VGA y demodulador) (dB)
-90	Empieza a oírse entre el ruido	-6	26
-80	Se escucha casi sin ruido	-4.3	25.7
-76	Sin ruido notable	-4	23.7
-65	"	-3.1	13.7
-55	"	-3.2	4.7
-45	"	-2	-4.3
-35	"	5.6	-4.3
-30	"	10	-4.3
-25	Se empieza a distorsionar	12.78	-4.3

Hay que aclarar que la potencia en el caso de este demodulador software se mide en dBm con respecto a 50Ω, de acuerdo con esta fórmula:

$$\text{Potencia recibida (dBm)} = 10 \cdot \log\left(\left(\sigma^2_I + \sigma^2_Q \right) / 50\Omega \right)$$

En cuanto al control automático de ganancia, si se muestra gráficamente la suma de la potencia de entrada con la ganancia establecida automáticamente, se tiene esta gráfica:

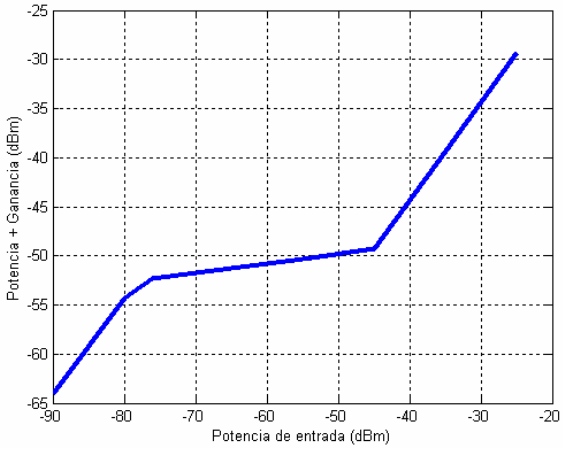


Figura 187: Efecto del control automático de ganancia.

Se puede ver cómo hay un rango, entre -80dBm y -40dBm, aproximadamente, en el que la potencia recibida se mantiene aproximadamente constante gracias a este bucle de control de ganancia.

Recepción de la portadora de estéreo

Al sintonizar cualquier emisora de FM comercial, se ha podido comprobar como siempre que hay señal suficiente se indica por el demodulador la detección de la portadora de estéreo de 19Khz. El margen de "dB" por el que se detecta varía entre unos 3dB hasta 15dB, aunque lo normal es que mantenga en torno de 9-10dB.

Como curiosidad, a continuación se muestra el espectro de una señal de audio demodulada de una emisora de FM comercial, a la izquierda.

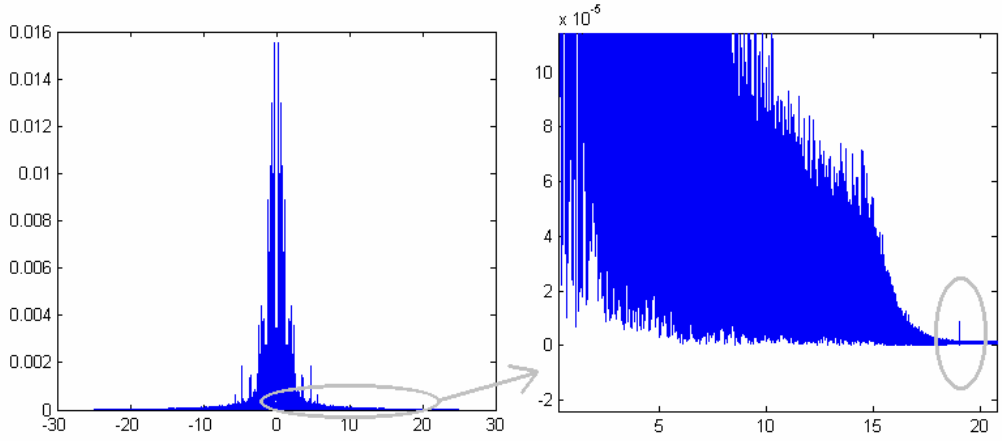


Figura 188: Aún atenuada varios órdenes de magnitud debido al filtro de audio, la portadora de estéreo aún se puede apreciar a la salida.

Esta señal ha sido volcada desde *SWRadioStudio* a disco, y abierta posteriormente en MATLAB. Ampliando la zona alrededor de 19Khz se puede ver que aún después de haber pasado por un filtro que ha atenuado la señal en 40dB, se puede apreciar muy claramente con respecto a su alrededor este tono portador del estéreo, lo que da una idea de la claridad con que se recibe antes de ser atenuado para reproducir el audio.

11.8.2.ASK

Se han realizado dos pruebas con este demodulador. Tanto en estas dos pruebas como en todas las pruebas de modulaciones digitales se ha utilizano el generador de señales de radiofrecuencia "Rohde & Schwarz Signal Generador SMIQ 02B". El nivel de la señal generada en todos los casos ha sido de 210μV.

En este caso se ha configurado para generar una portadora de radio en la frecuencia 92Mhz, usando una modulación digital ASK con profundidad de modulación del 100%, y una tasa de símbolo de 5KBaud.

Para la primera prueba se ha seleccionado una fuente de bits basada en la repetición del patrón 0-1 indefinidamente. Tras ejecutar el *script* MATLAB del demodulador sobre la señal compleja en banda base capturada por *SWRadioStudio*, se obtienen entre otros estos resultados:

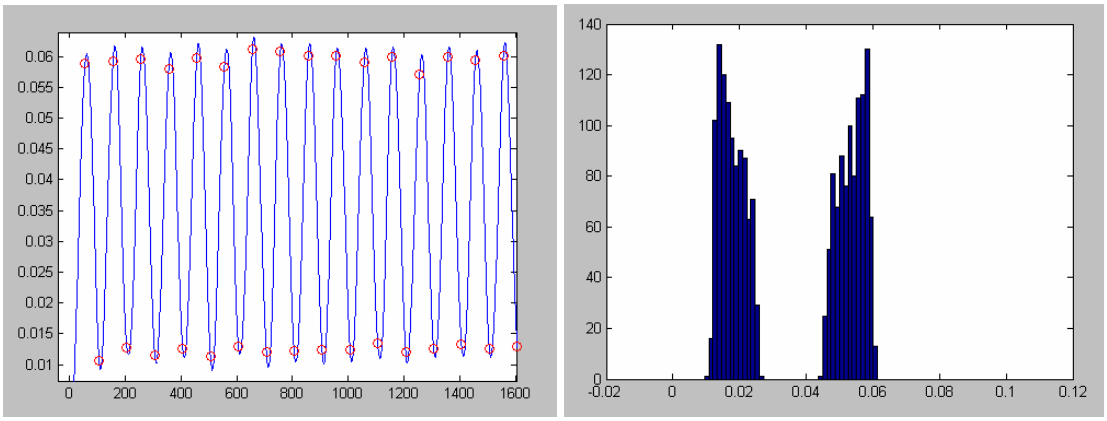


Figura 189: Resultados para demodulador ASK y patrón de bits 0-1.

En la izquierda se puede ver la envolvente compleja de la señal ya filtrada paso bajo, y muestreada a 5KHz. Se puede comprobar que hay un símbolo adquirido cada 50 muestras, debido a que 250KHz / 5KBaud = 50 muestras / símbolo.

A la derecha se muestra el histograma de estas muestras, donde se ve claramente que se encuentran separadas en dos grupos claramente, los asociados al símbolo 0 y los asociados al símbolo 1.

La segunda prueba se ha realizado utilizando los mismos parámetros, pero usando una fuente de símbolos basada en un generador de bits pseudoaleatorio de 9 bits. Conociendo el funcionamiento de estos generadores se sabe que la secuencia de bits es de 2^9-1 = 511 bits antes de volver a repetirse.

Los resultados se muestran a continuación. En la primera gráfica se muestra de forma ampliada la señal envolvente compleja demodulada, donde se ven ahora zonas más irregulares que antes, correspondiendo a grupos de bits a uno o a cero consecutivos. En la segunda gráfica (arriba a la derecha) se muestra esta envolvente compleja al completo junto con las muestras que se han tomado a la tasa de símbolo.

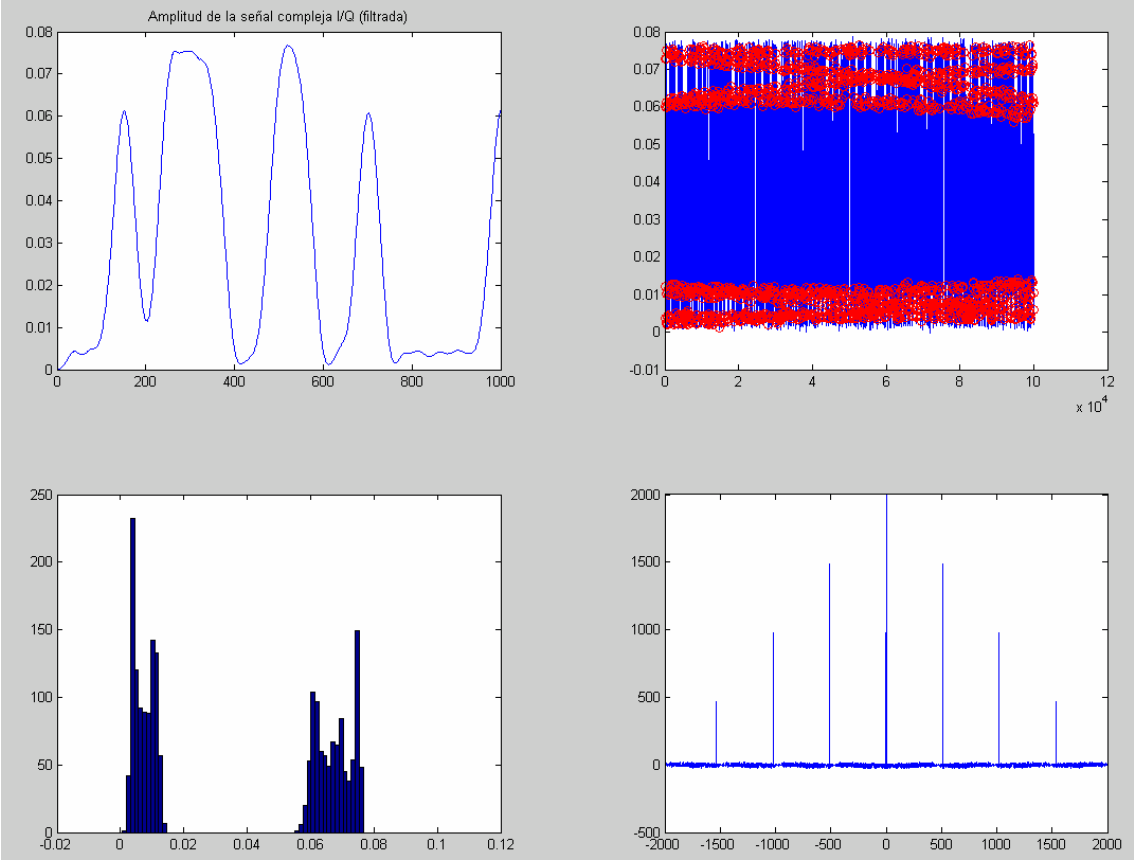


Figura 190: Resultados para demodulador ASK y patrón PRBS de 9 bits.

Se puede apreciar claramente como existe una división clara entre valores altos, símbolo uno, y valores bajos, símbolo cero. Esto también se aprecia claramente en el histograma, abajo a la izquierda. Para asegurar que la secuencia de bits se ha recibido correctamente, y para evitar generar esta secuencia para compararlos bit a bit, se ha calculado la autocorrelación de la señal. Los picos en los desplazamientos 511 y sus múltiplos indican claramente que se ha recibido la secuencia de bits correspondiente a la secuencia PRBS de 9 bits.

11.8.3.FSK

Para el caso del demodulador software 2-FSK, se ha configurado el generador de señales en esta modulación, también a 5KBaud y con un Δf de 50KHz. Como fuente de bits se ha utilizado el generador PRBS de 9 bits, obteniendo tras la demodulación estos resultados.

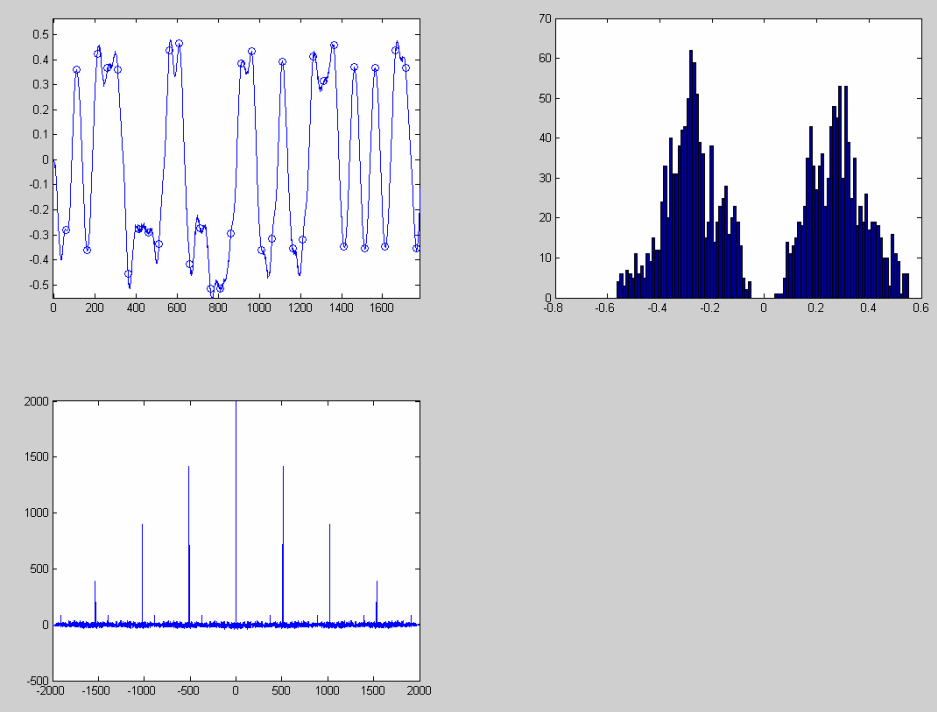


Figura 191: Resultados para demodulador FSK y patrón PRBS de 9 bits.

En la primera figura se puede ver la señal demodulada en FM, junto con los puntos en que se ha muestreado esta señal para obtener los símbolos, que como es deseable han resultado en dos conjuntos bien separados, como se ve en el histograma, figura de la derecha.

Al igual que en el apartado anterior, se ha utilizado una autocorrelación para demostrar que la señal recibida es la señal pseudoaleatoria de 511 bits de período.

11.8.4.BPSK

Para probar este demodulador se ha utilizado una señal modulada en BPSK a 5KBaud y usando como fuente de bits el generador pseudoaleatorio de 9 bits.

En este caso, y al tratarse de una modulación coherente, no solo se pueden representar las señales I y Q con respecto al tiempo (primera figura de la izquierda), sino que se puede dibujar la constelación de fase asociada, como se ve en la figura de arriba a la derecha.

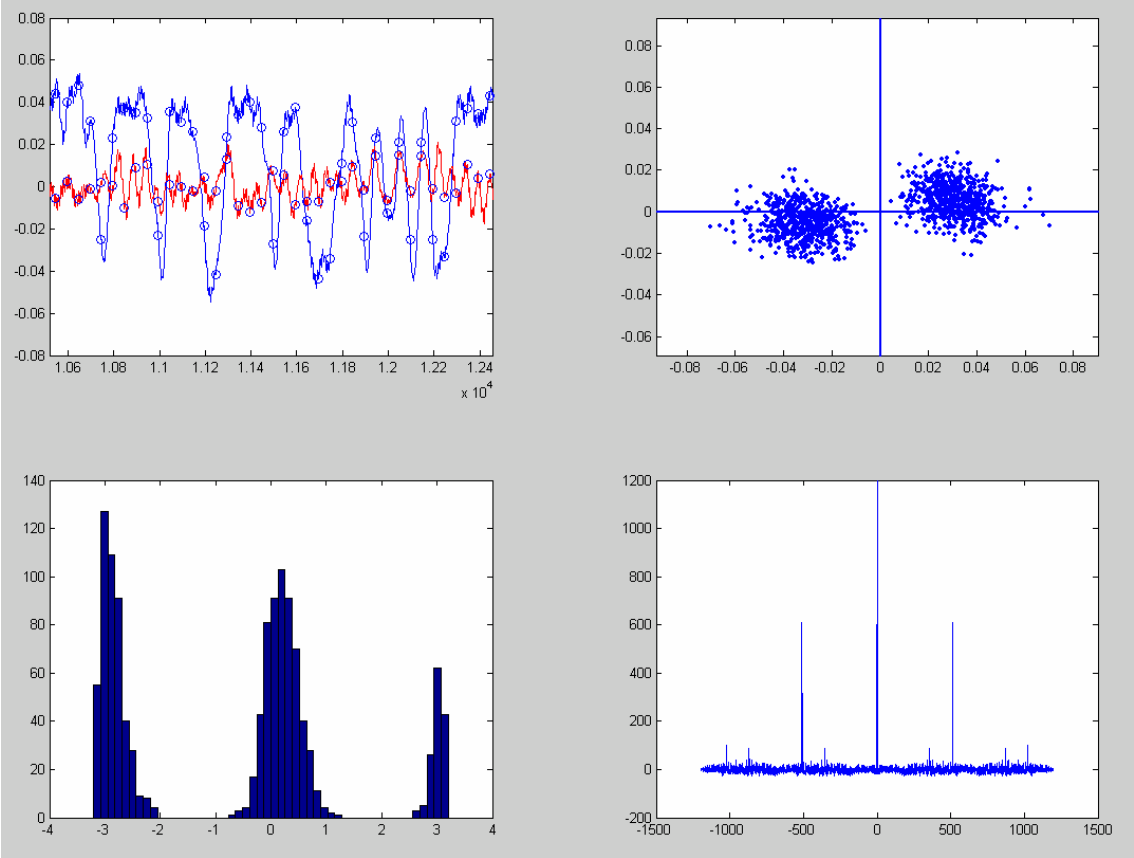


Figura 192: Resultados para demodulador coherente BPSK y patrón PRBS de 9 bits.

Se ve claramente que los dos símbolos forman grupos con fases bien diferenciada, con una diferencia de π radianes, como es propio de esta modulación. Esto se aprecia igualmente en el histograma de fases de los símbolos muestreados, donde se ve que los dos símbolos se agrupan alrededor de 0 y de π radianes, respectivamente.

De nuevo, y al igual que en los casos anteriores, se ha calculado la autocorrelación del flujo binario demodulado para demostrar que este flujo es la secuencia pseudoaleatoria de 511 bits de duración que se esta realmente transmitiendo.

12. CONCLUSIONES Y LINEAS DE FUTURO

En este proyecto se ha propuesto el diseño, implementación y caracterización de un sistema completo de adquisición de señales de radiofrecuencia basada en una técnica que permite la demodulación por software de forma flexible. Como se ha demostrado en el capítulo de mediciones y resultados, el sistema funciona y sus características, si bien en algunos aspectos difieren de los planteados como objetivos teóricos de diseño, han demostrado ser lo suficientemente válidos como para proveer el entorno hardware y software necesario para la demodulación de varias demodulaciones analógicas y digitales de forma totalmente satisfactoria. Este entorno software y sobre todo, su característica de ser un sistema abierto, es posiblemente uno de los puntos más atractivos para proponer futuras líneas de trabajo como continuación de este proyecto. Desde la implementación en el entorno de tiempo real *SWRadioStudio* de nuevas modulaciones, tanto analógicas como digitales, como la ampliación de algunas de las ya existentes, como por ejemplo el dotar al demodulador FM de capacidad de procesar señales estéreo.

13. REFERENCIAS

[1] “Software radio: a modern approach to radio engineering”, Prentice Hall, Jeffrey H. Reed, 2002

[2] “Communications Receivers: Principles & Design”, McGraw-Hill, U.L.Rohde; J.C. Whitaker; T.T.N. Bucher.

[3] “Universal Serial Bus Specification”, Revision 2.0, 27-Abril-2000. Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips.

[4] Apuntes de clase de “Circuitos y Subsistemas de comunicaciones”. Mariano Fernández Navarro, E.T.S.I. Telecomunicación (Universidad de Málaga)

[5] “Communication Systems”. Simon Haykin.

[6] “PLL Performance, Simulation and Design”, Dean Banerjee, 1998. Guía de *National Semiconductors*.

[7] “Impedance Matching Network Designer”, John Wetherell, Berkeley University, http://bwrc.eecs.berkeley.edu/Research/RF/projects/60GHz/matching/ImpMatch.html

[8] “DriverLINX Port I/O Driver”, John Pappas, http://diskdude.cjb.net/

[9] “A digital AM, FM, SSB demodulator”, http://thierry.leconte.chez.tiscali.fr/

[10] “Demodulation of CPFSK and GMSK Signals using Digital Signal Processing DPLL with Sequence Estimator”, Yasunori Iwanami. IEICE Trans. Commun. Enero 2001.

[11] “The Fundamentals of FFT-Based Signal Analysis and Measurement in LabVIEW and LabWindows/CVI”, www.ni.com

[12] “Stereo FM Radio”, http://www.st-andrews.ac.uk/~www_pa/Scots_Guide/RadCom/part21/page1.html

[13] “The Goertzel Algorithm”, www.embedded.com

[14] Technical Notes: “PSK Modulation (Part 1)” y “PSK Modulation (Part 2)”, WJ Communications, J.Mark Steber.

[15] “Theory of Digital Communications”, Wong, Lok. Capítulo 3: “Synchronization”.

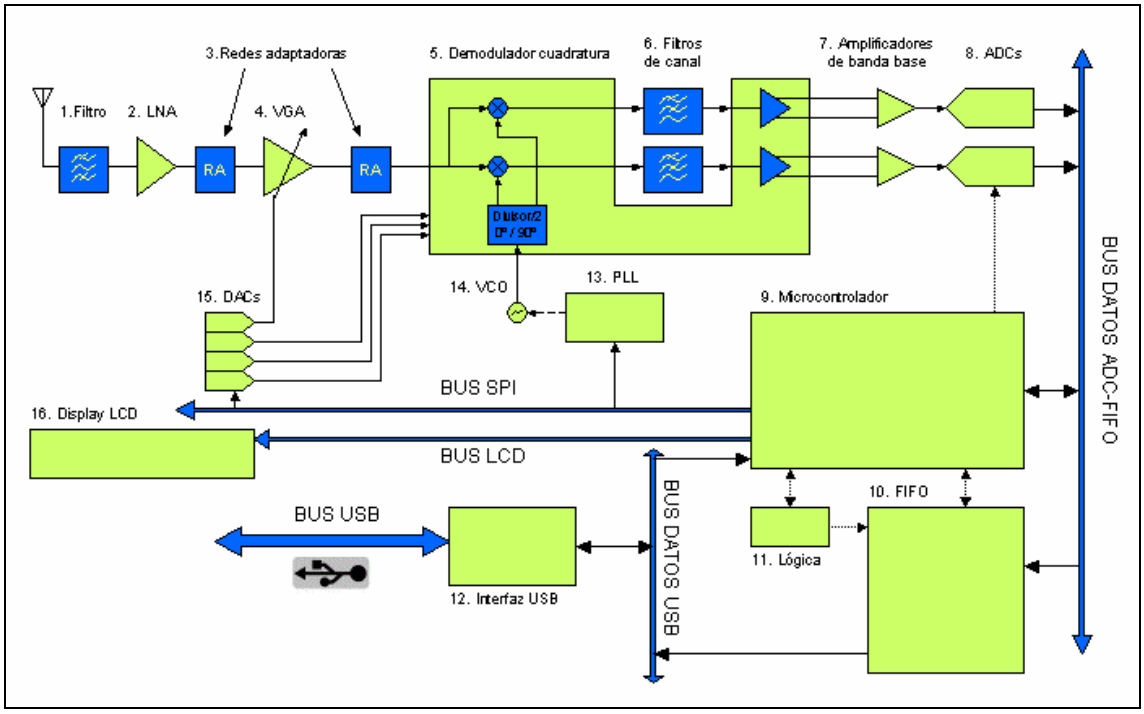
Además, se han usado *datasheets* (que se incorporan en el anexo A-5) de las siguientes compañías:

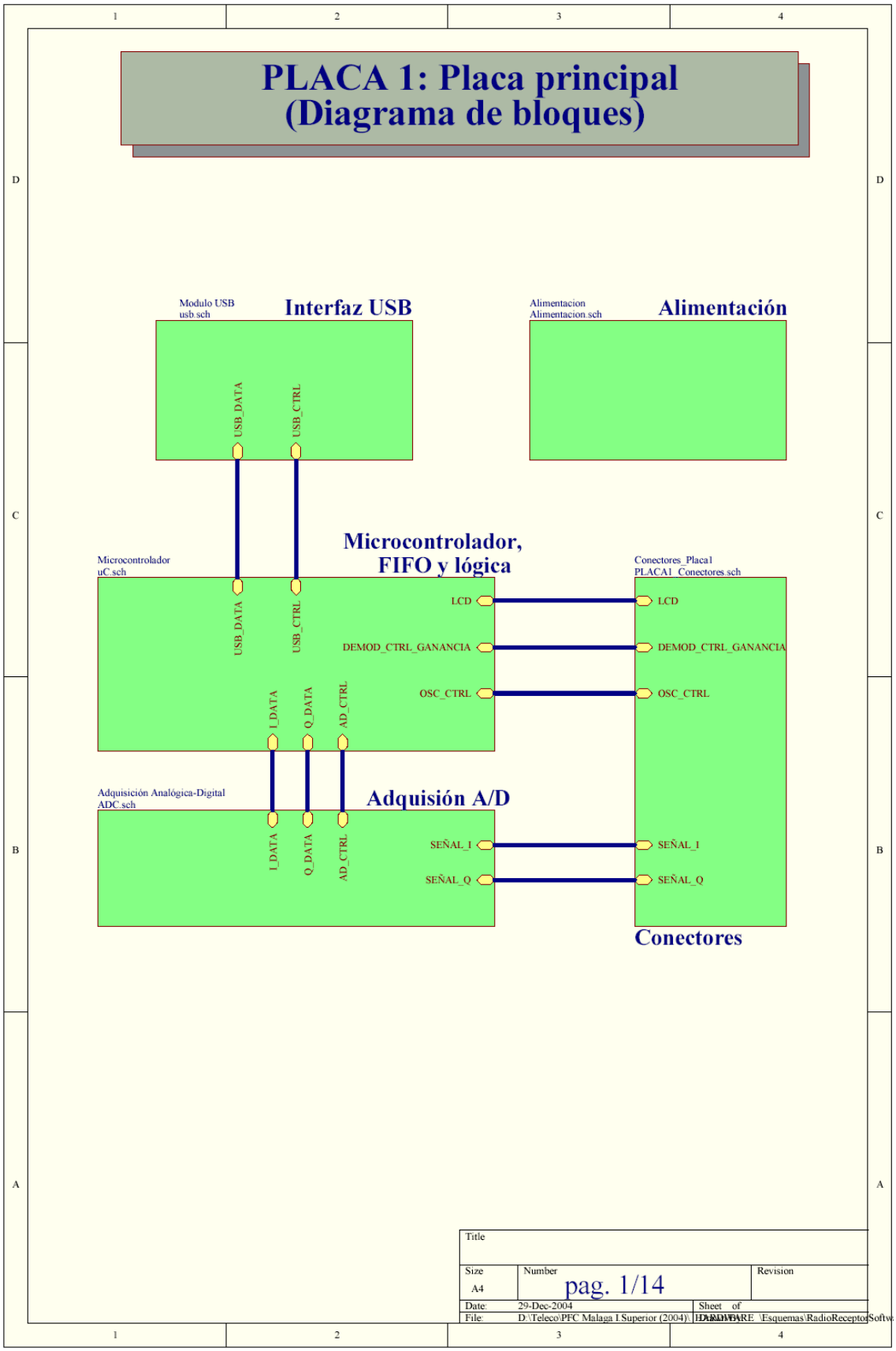
- Analog Devices Inc. (www.analog.com)
- Dallas Semiconductors Maxim (www.maxim-ic.com)
- Atmel (www.atmel.com)
- Cypress Semiconductor Corporation (www.cypress.com??)
- Infineon Technologies
- AMP Incorporated
- SERPAC
- Texas Instruments (www.ti.com)

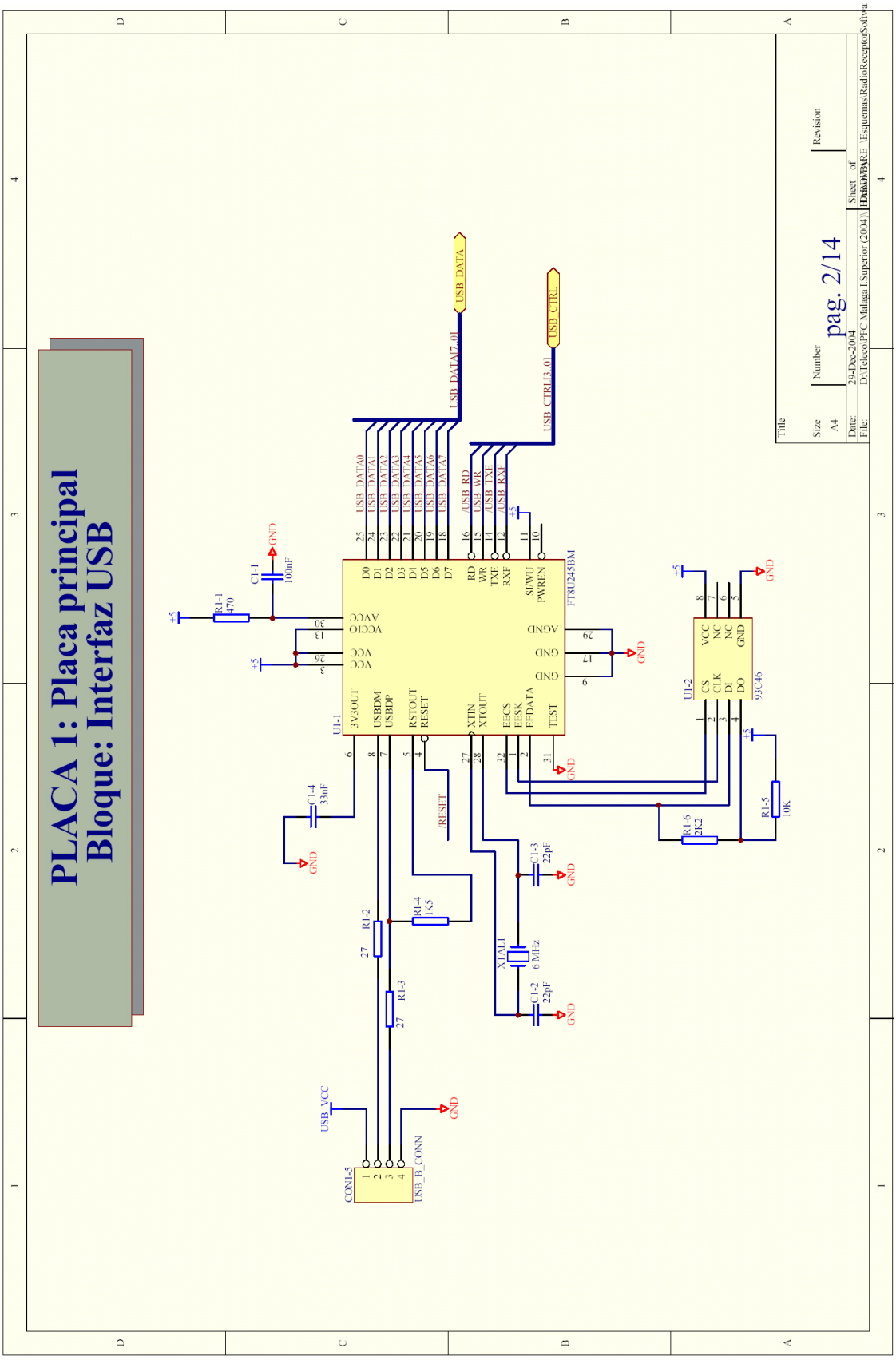
ANEXOS

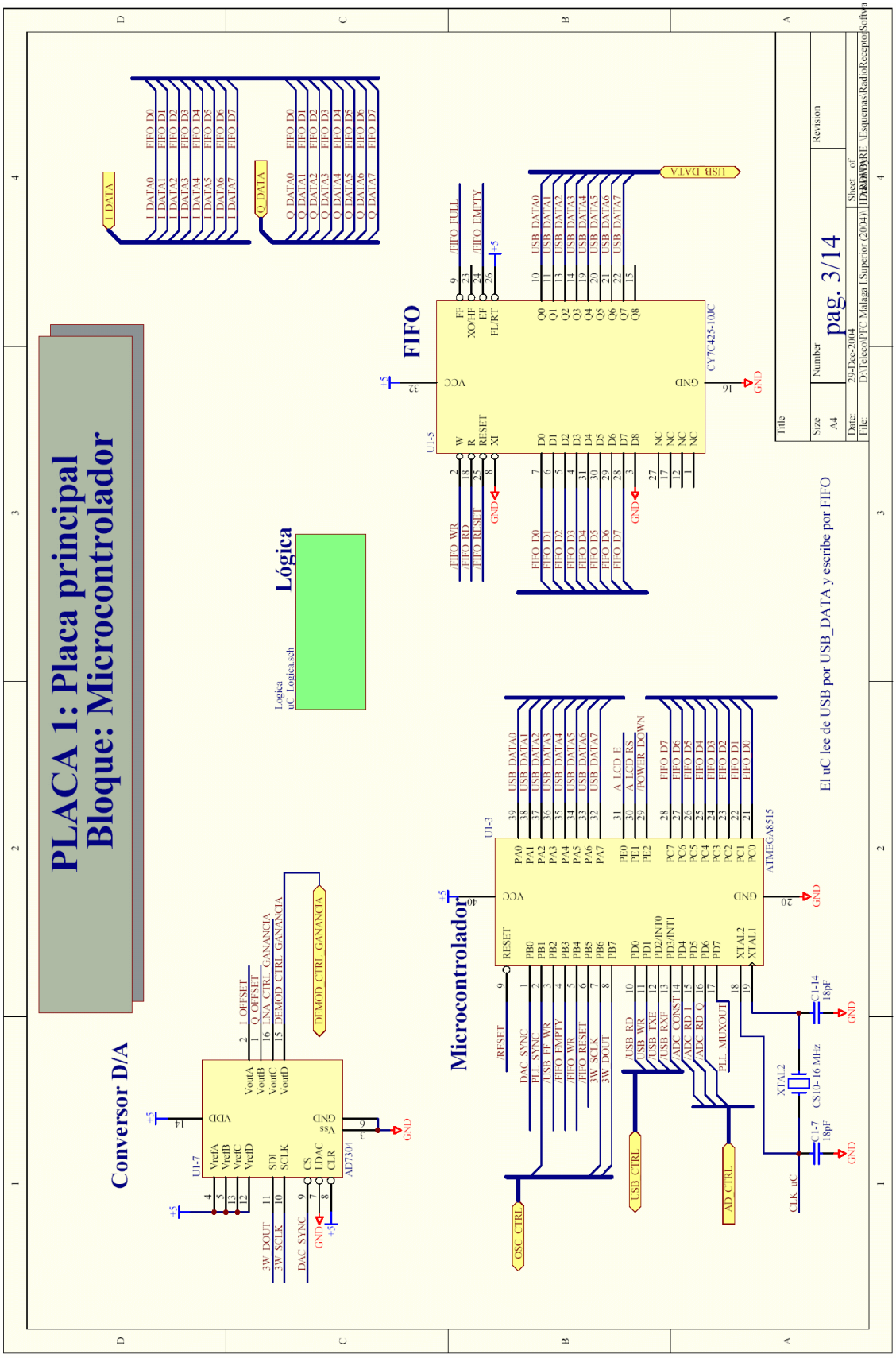
A1. Planos eléctricos y mecánicos

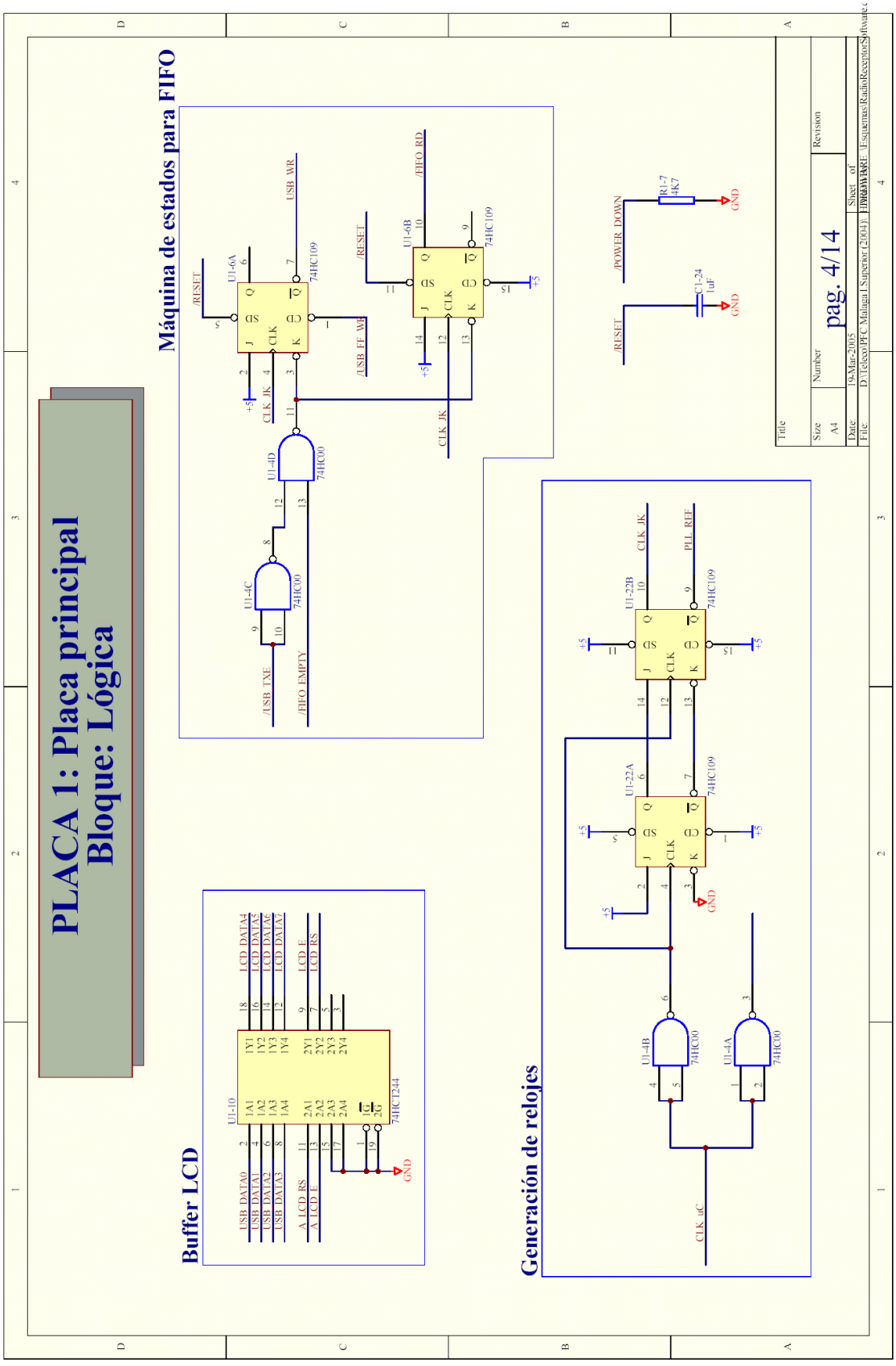
En las siguientes catorce páginas se muestran los esquemas eléctricos completos del sistema desarrollado. Para cada una de las cuatro placas en que se ha dividido el diseño se incluye una primera página donde se muestran los subsistemas ubicados en esa placa. Por comodidad se repite a continuación el diagrama de bloques del receptor:

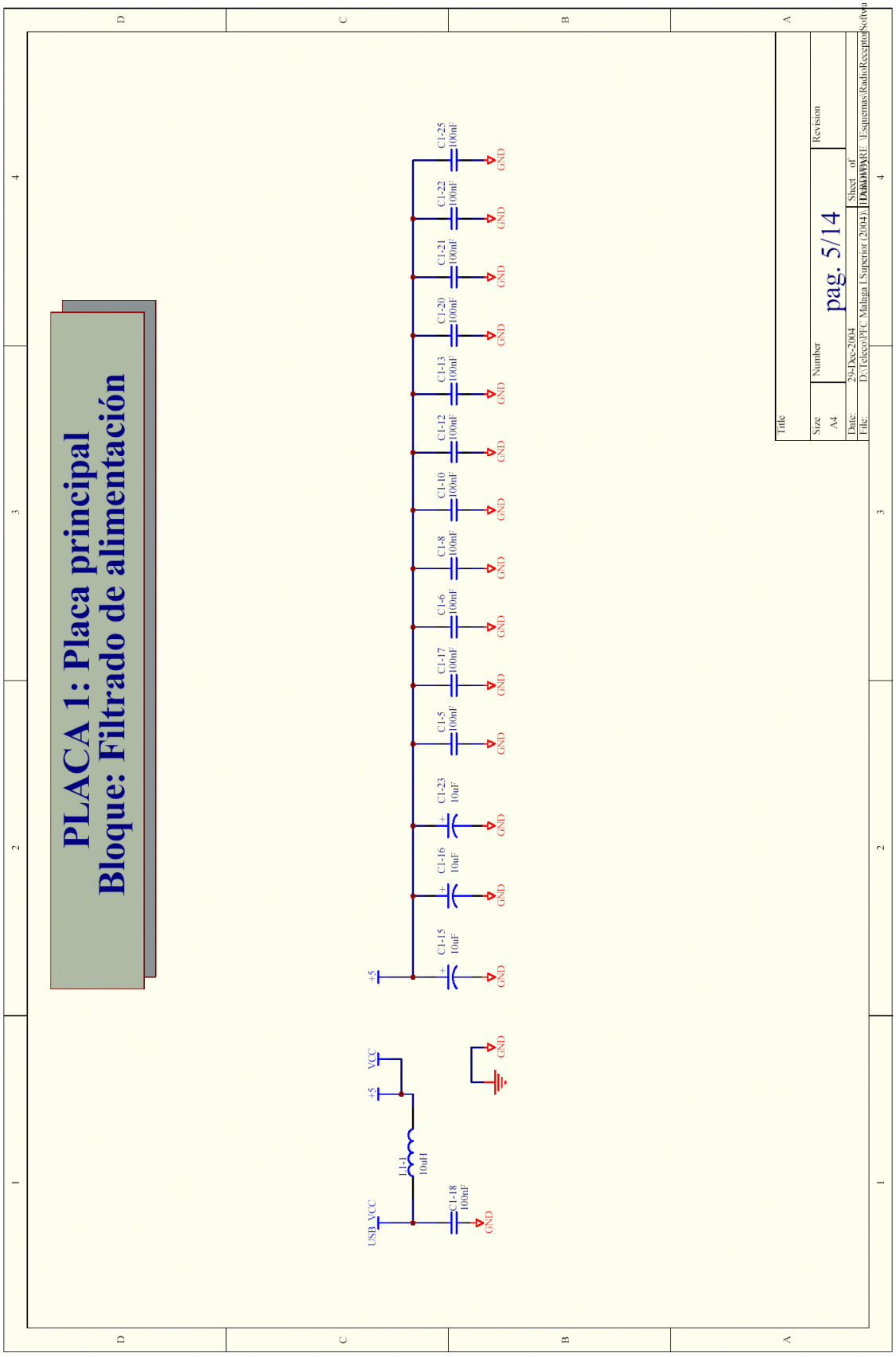


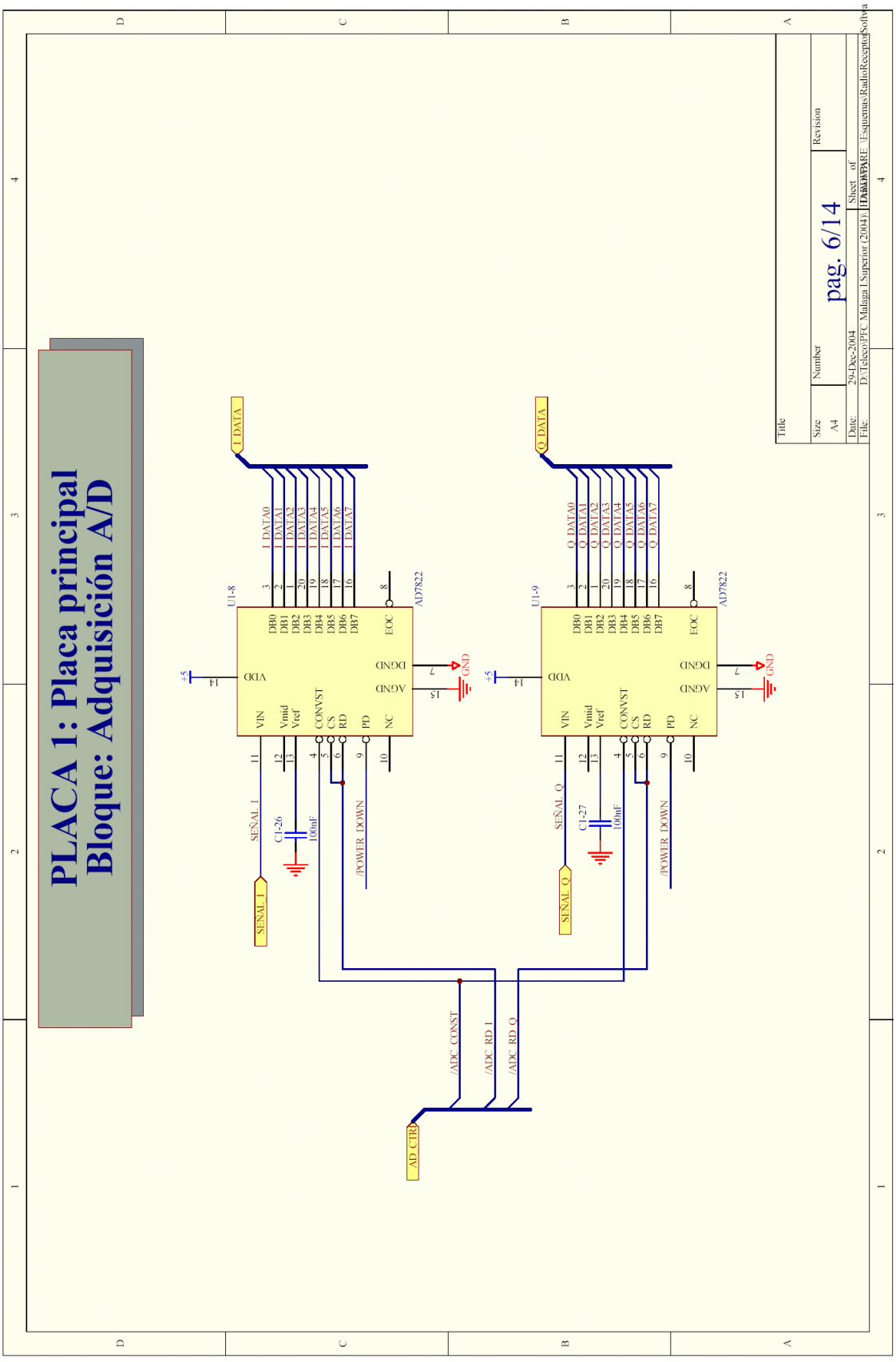

PLACA 1: Placa principal
(Diagrama de bloques)
Modulo USB
usb.sch
Interfaz USB
USB_DATA
USB_CTRL
Alimentacion
Alimentacion.sch
Alimentación
Microcontrolador,
FIFO y lógica
Microcontrolador
uC.sch
USB_DATA
USB_CTRL
LCD
DEMOD_CTRL_GANANCIA
OSC_CTRL
I_DATA
Q_DATA
AD_CTRL
Conectores_Placa1
PLACA1_Conectores.sch
LCD
DEMOD_CTRL_GANANCIA
OSC_CTRL
Adquisición Analógica-Digital
ADC.sch
Adquisión A/D
I_DATA
Q_DATA
AD_CTRL
SEÑAL_I
SEÑAL_Q
SEÑAL_I
SEÑAL_Q
Conectores
Title
Size
A4
Number
pag. 1/14
Revision
Date: 29-Dec-2004
Sheet of
File: D:\Teleco\PFC Malaga I.Superior (2004)\...\HARDWARE\Esquemas\RadioReceptorSoftw

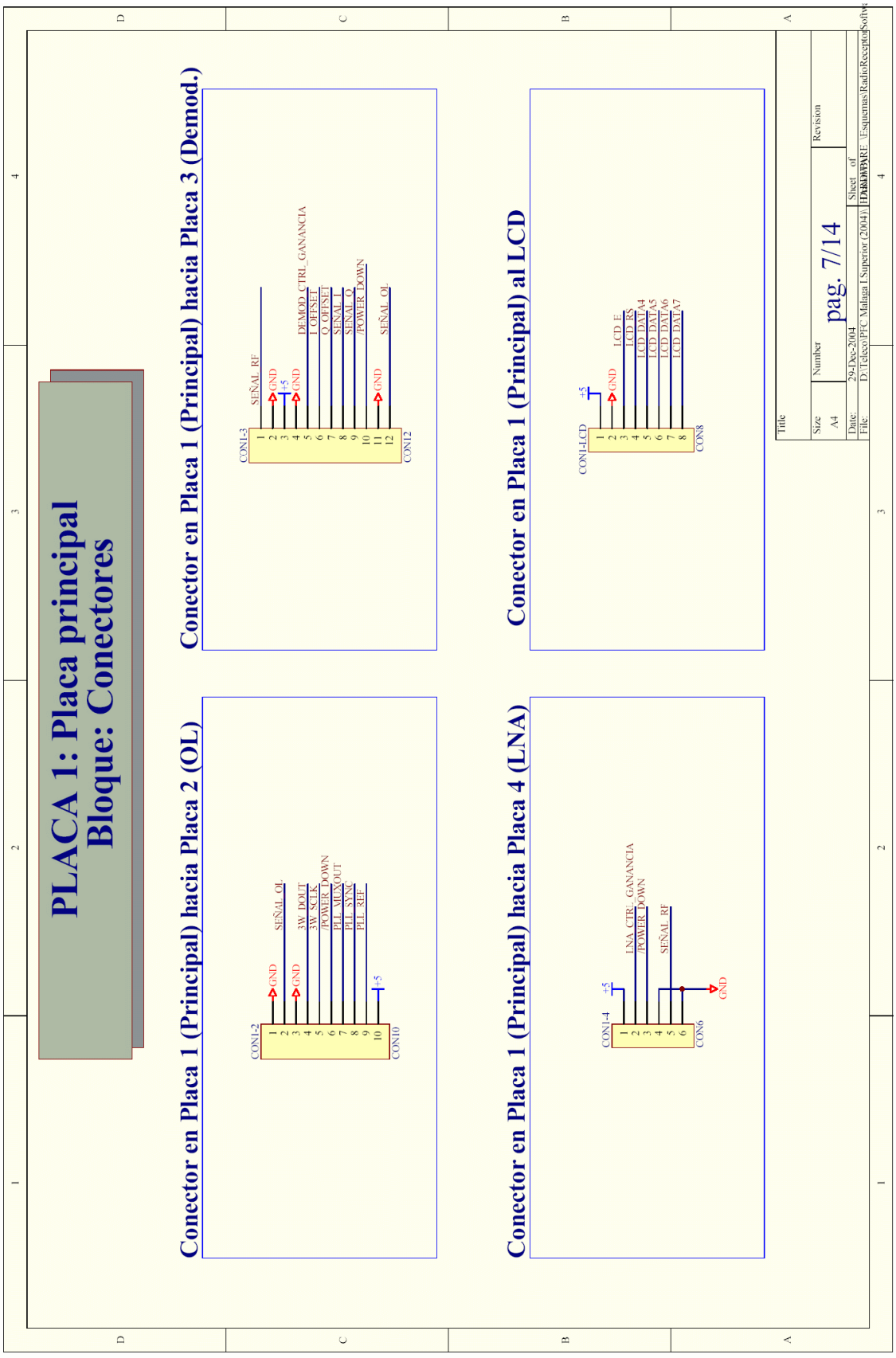

PLACA 1: Placa principal
Bloque: Interfaz USB
pag. 2/14
USB_DATA
USB_CTRL
FT8U245BM
93C46
USB_B_CONN
XTAL1
6 MHz
/RESET


PLACA 1: Placa principal
Bloque: Microcontrolador
Conversor D/A
Lógica
Logica
uC_Logica.sch
FIFO
Microcontrolador
AD7304
CY7C425-10JC
ATMEGA8515
I_DATA
Q_DATA
USB_DATA
OSC_CTRL
USB_CTRL
AD_CTRL
DEMOD_CTRL_GANANCIA
CLK_uC
El uC lee de USB por USB_DATA y escribe por FIFO
pag. 3/14
Title
Size
A4
Number
Revision
Date: 29-Dec-2004
Sheet of
File: D:\Teleco\PFC Malaga I.Superior (2004)\DATA\SOFTWARE\Esquemas\RadioReceptorSoftwa


PLACA 1: Placa principal
Bloque: Lógica
Máquina de estados para FIFO
Buffer LCD
Generación de relojes
pag. 4/14


PLACA 1: Placa principal
Bloque: Filtrado de alimentación
pag. 5/14
Title
Size
A4
Number
Revision
Date: 29-Dec-2004
File: D:\Telecos\PFC Malaga I Superior (2004)\LaTeX\Esquemas\RadioReceptorSoftwa
Sheet of
C1-25 100nF
C1-22 100nF
C1-21 100nF
C1-20 100nF
C1-13 100nF
C1-12 100nF
C1-10 100nF
C1-8 100nF
C1-6 100nF
C1-17 100nF
C1-5 100nF
C1-23 10uF
C1-16 10uF
C1-15 10uF
C1-18 100nF
L1-1 10uH
+5
VCC
USB_VCC
GND

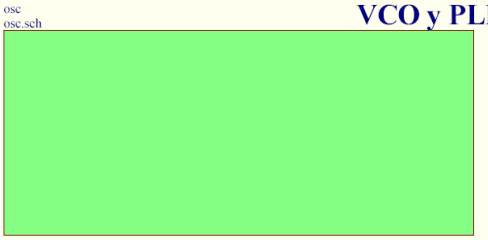

PLACA 1: Placa principal
Bloque: Adquisición A/D
pag. 6/14
I_DATA
Q_DATA
U1-8
U1-9
AD7822
I_DATA0
I_DATA1
I_DATA2
I_DATA3
I_DATA4
I_DATA5
I_DATA6
I_DATA7
Q_DATA0
Q_DATA1
Q_DATA2
Q_DATA3
Q_DATA4
Q_DATA5
Q_DATA6
Q_DATA7
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
EOC
VDD
DGND
AGND
VIN
Vmid
Vref
CONVST
CS
RD
PD
NC
+5
GND
SEÑAL_I
SEÑAL_Q
C1-26
C1-27
100nF
/POWER_DOWN
/ADC_CONST
/ADC_RD_I
/ADC_RD_Q
AD_CTRL


PLACA 1: Placa principal
Bloque: Conectores
Conector en Placa 1 (Principal) hacia Placa 3 (Demod.)
CON1-3
SEÑAL_RF
GND
+5
GND
I_OFFSET
Q_OFFSET
SENAL_I
SENAL_Q
/POWER_DOWN
GND
SEÑAL_OL
DEMOD_CTRL_GANANCIA
CON12
Conector en Placa 1 (Principal) al LCD
CON1-LCD
+5
GND
LCD_E
LCD_RS
LCD_DAT_A4
LCD_DAT_A5
LCD_DAT_A6
LCD_DAT_A7
CON8
Conector en Placa 1 (Principal) hacia Placa 2 (OL)
CON1-2
GND
SEÑAL_OL
GND
3W_DOUT
3W_SCLK
/POWER_DOWN
PLL_VLOUT
PLL_SYNC
PLL_REF
+5
CON10
Conector en Placa 1 (Principal) hacia Placa 4 (LNA)
CON1-4
+5
LNA_CTRL_GANANCIA
/POWER_DOWN
SEÑAL_RF
GND
CON6
Title
Size
A4
Number
pag. 7/14
Revision
Date
29-Dec-2004
Sheet of
File
D:\Telecos\PFC-Malaga I Superior (2004)\[illegible]\Esquemas\RadioReceptor.Sch

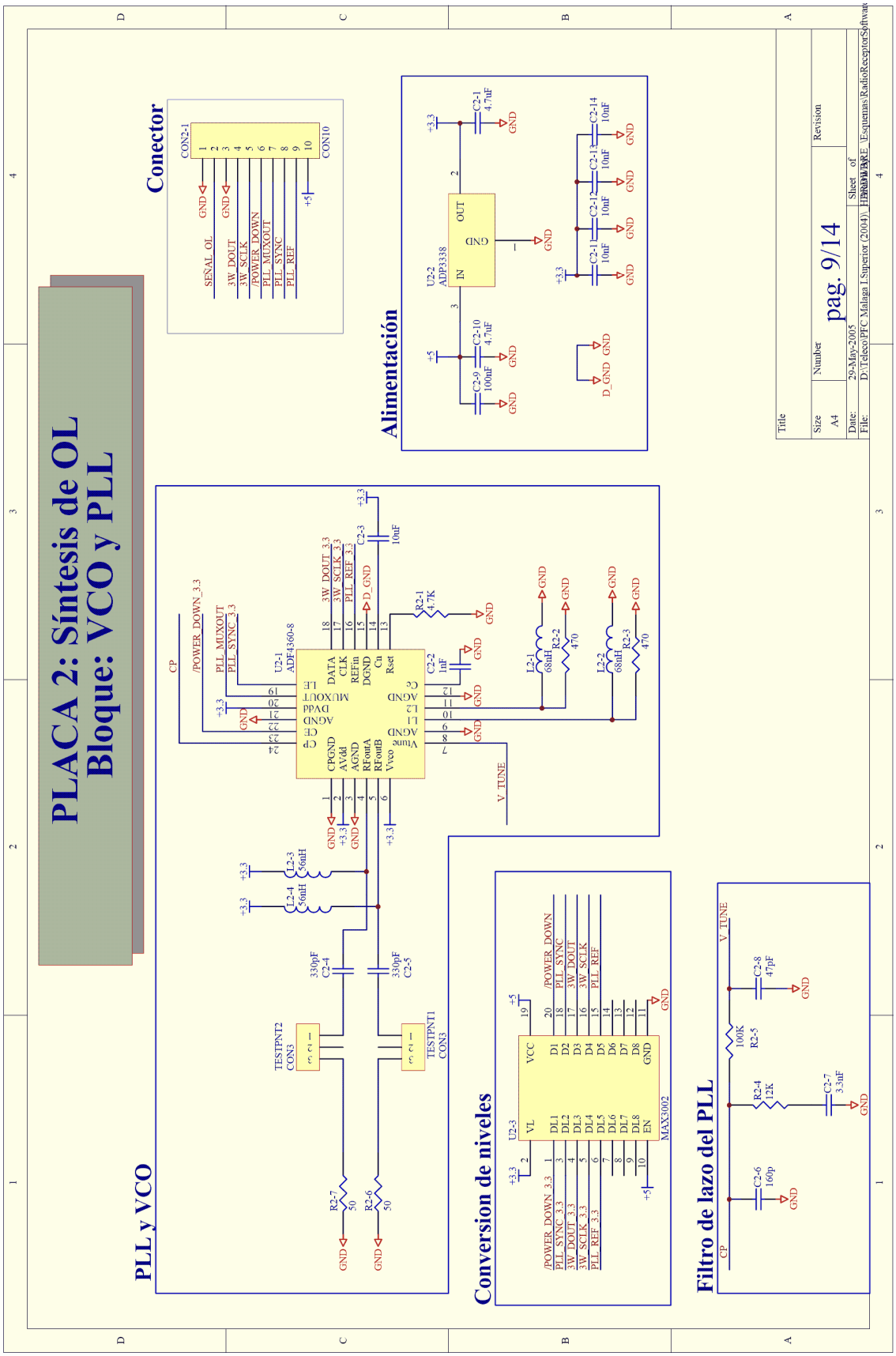
PLACA 2: Síntesis del OL (Diagrama de bloques)

osc
osc.sch

VCO y PLL



Title		
Size A4	Number pag. 8/14	Revision
Date: 29-Dec-2004		Sheet of
File: D:\Teleco\PFC Malaga I.Superior (2004)\...\Esquemas\RadioReceptorSoftwa		


PLACA 2: Síntesis de OL
Bloque: VCO y PLL
Conector
Alimentación
PLL y VCO
Conversion de niveles
Filtro de lazo del PLL
pag. 9/14

PLACA 3: Demodulador (Diagrama de bloques)

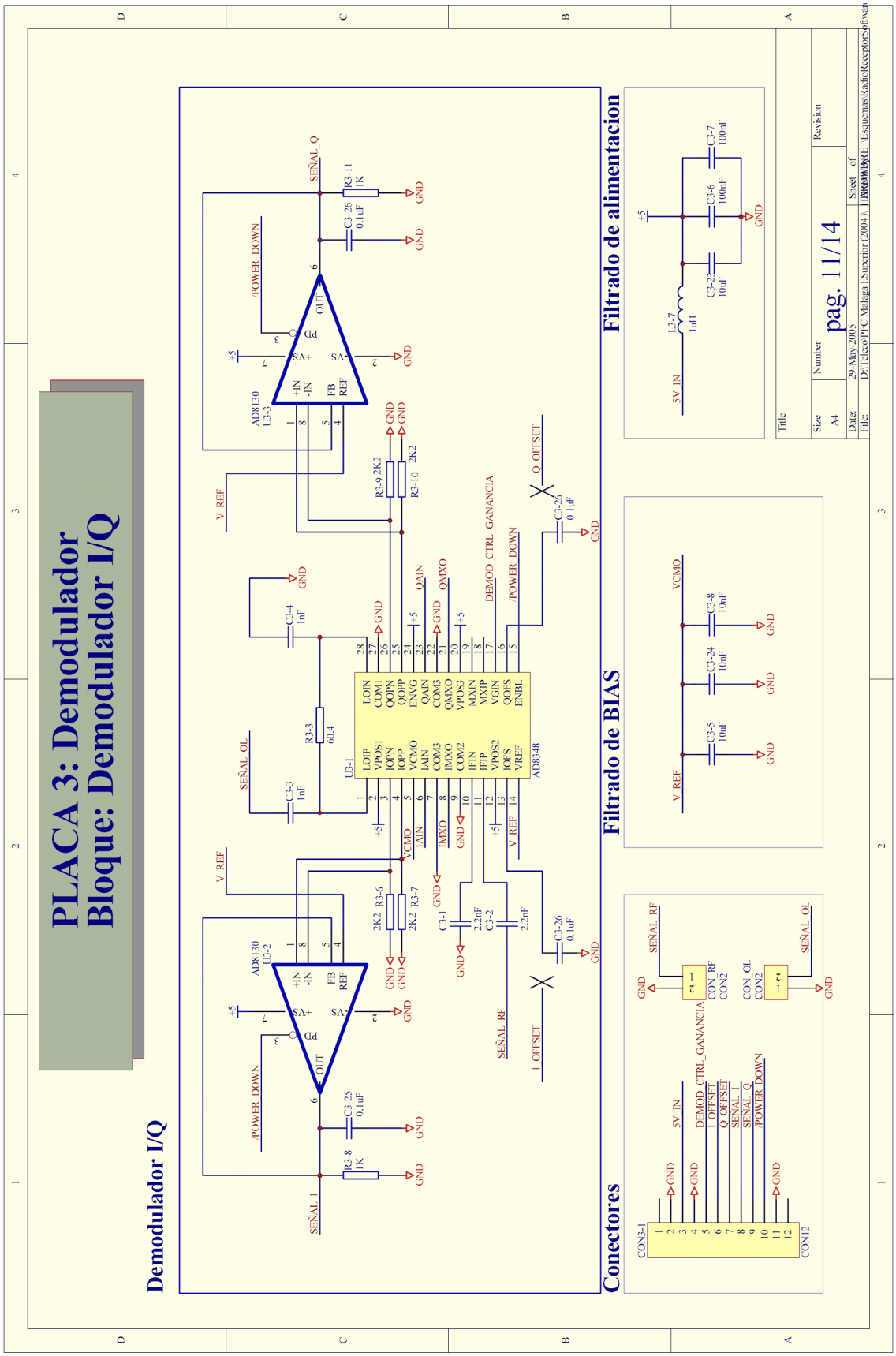
demod
demod.sch

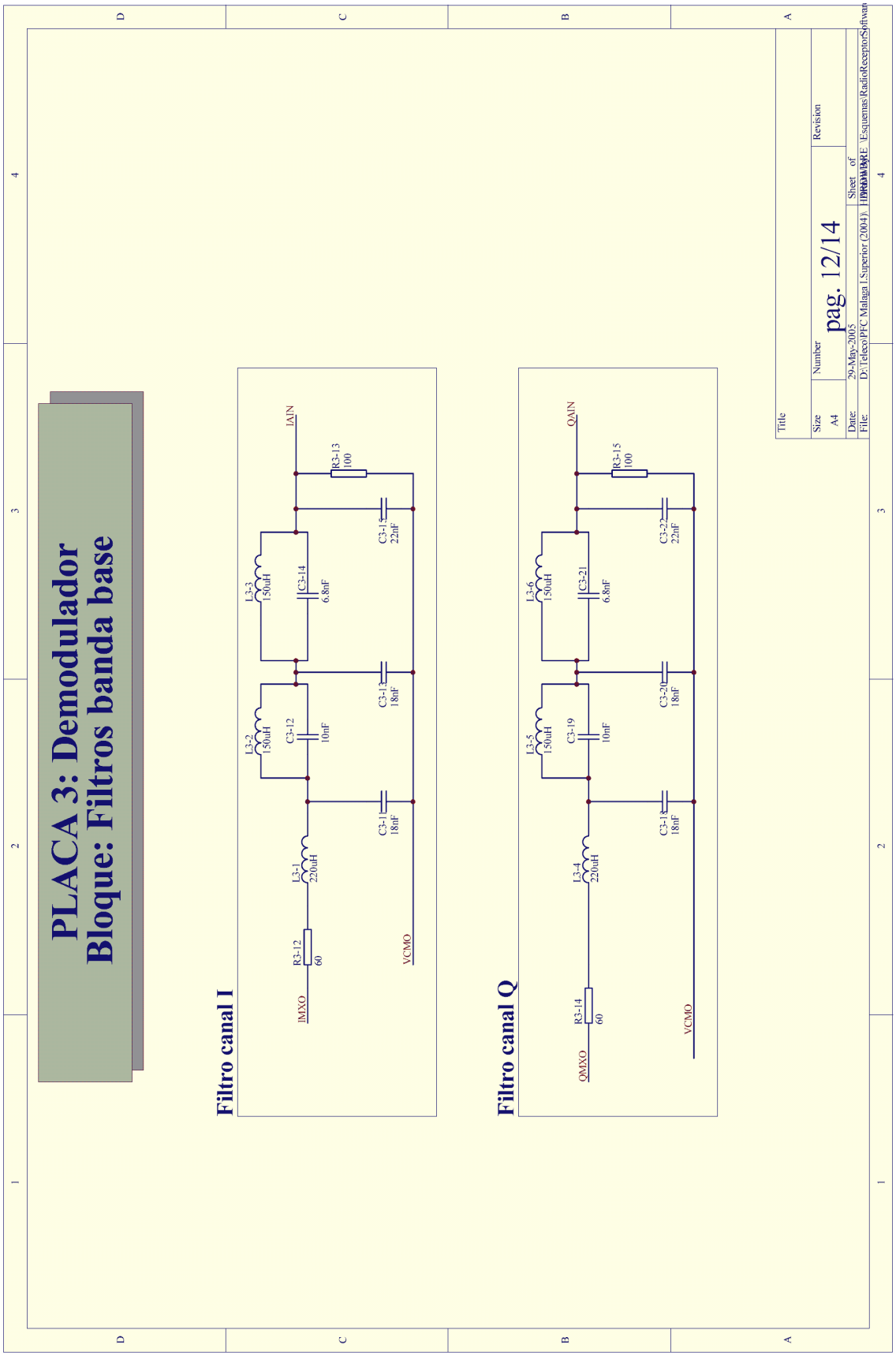
Demodulador I/Q

Filtros de canal
Demod_filtros_canal.sch

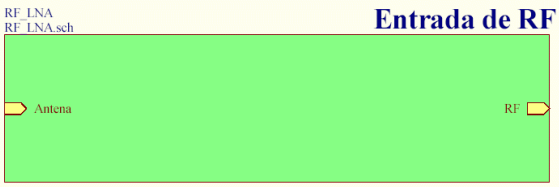
Filtros banda base

Title		
Size A4	Number pag. 10/14	Revision
Date: 29-Dec-2004	Sheet of	
File: D:\Teleco\PFC Malaga I.Superior (2004)\...\HARDWARE\Esquemas\RadioReceptor\Softwa		

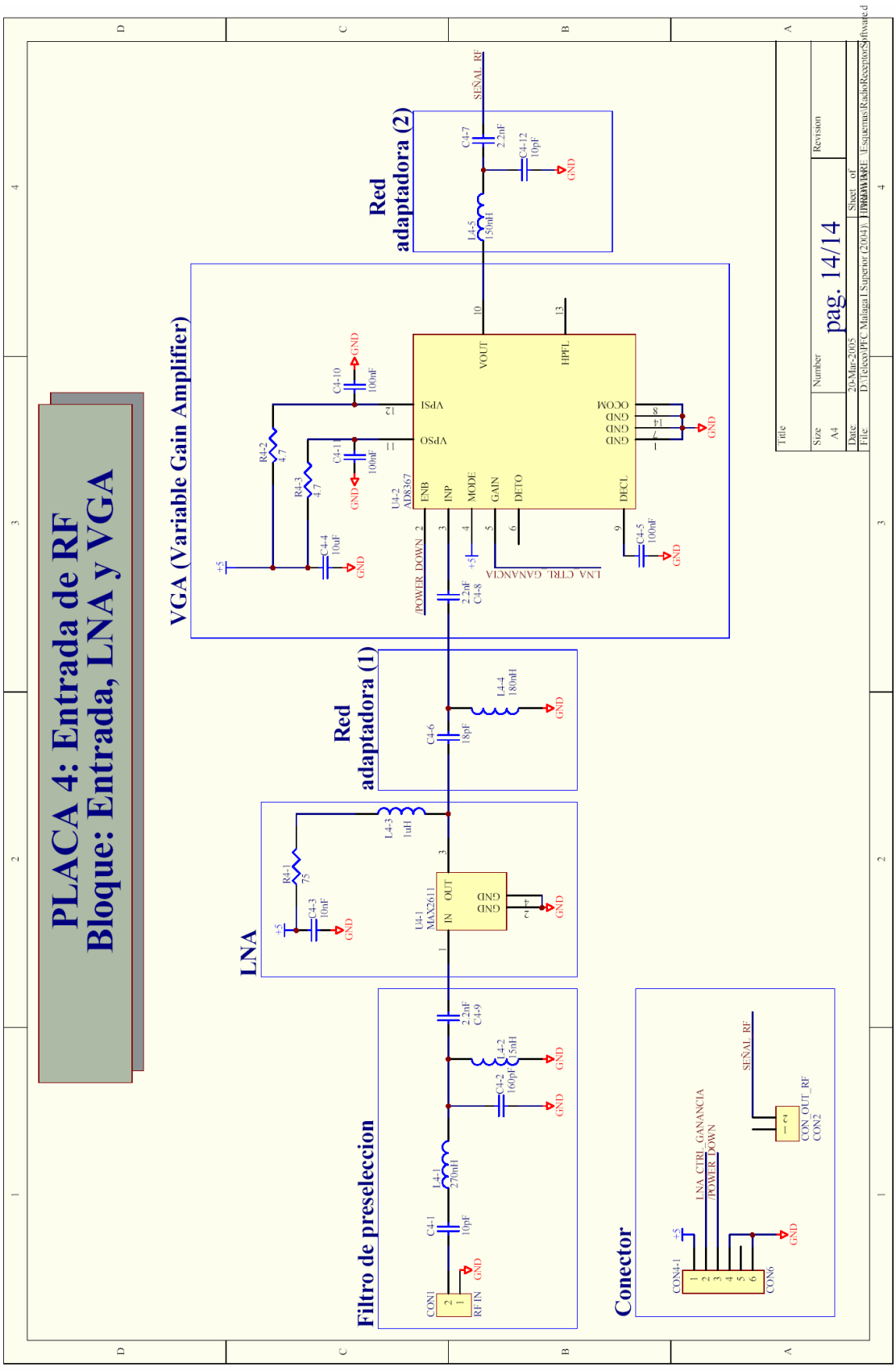

PLACA 3: Demodulador
Bloque: Demodulador I/Q
Demodulador I/Q
Filtrado de alimentacion
Filtrado de BIAS
Conectores
pag. 11/14


PLACA 3: Demodulador
Bloque: Filtros banda base
Filtro canal I
IAIN
R3-13
100
C3-15
22nF
L3-3
150uH
C3-14
6.8nF
L3-2
150uH
C3-12
10nF
C3-13
18nF
L3-1
220uH
C3-11
18nF
R3-12
60
IMXO
VCMO
Filtro canal Q
QAIN
R3-15
100
C3-22
22nF
L3-6
150uH
C3-21
6.8nF
L3-5
150uH
C3-19
10nF
C3-20
18nF
L3-4
220uH
C3-18
18nF
R3-14
60
QMXO
VCMO
Title
Size
A4
Number
pag. 12/14
Revision
Date:
29-May-2005
Sheet of
File:
D:\Teleco\PFC Malaga I.Superior (2004)\...\Esquemas\RadioReceptorSoftwar

PLACA 4: Entrada de RF (Diagrama de bloques)

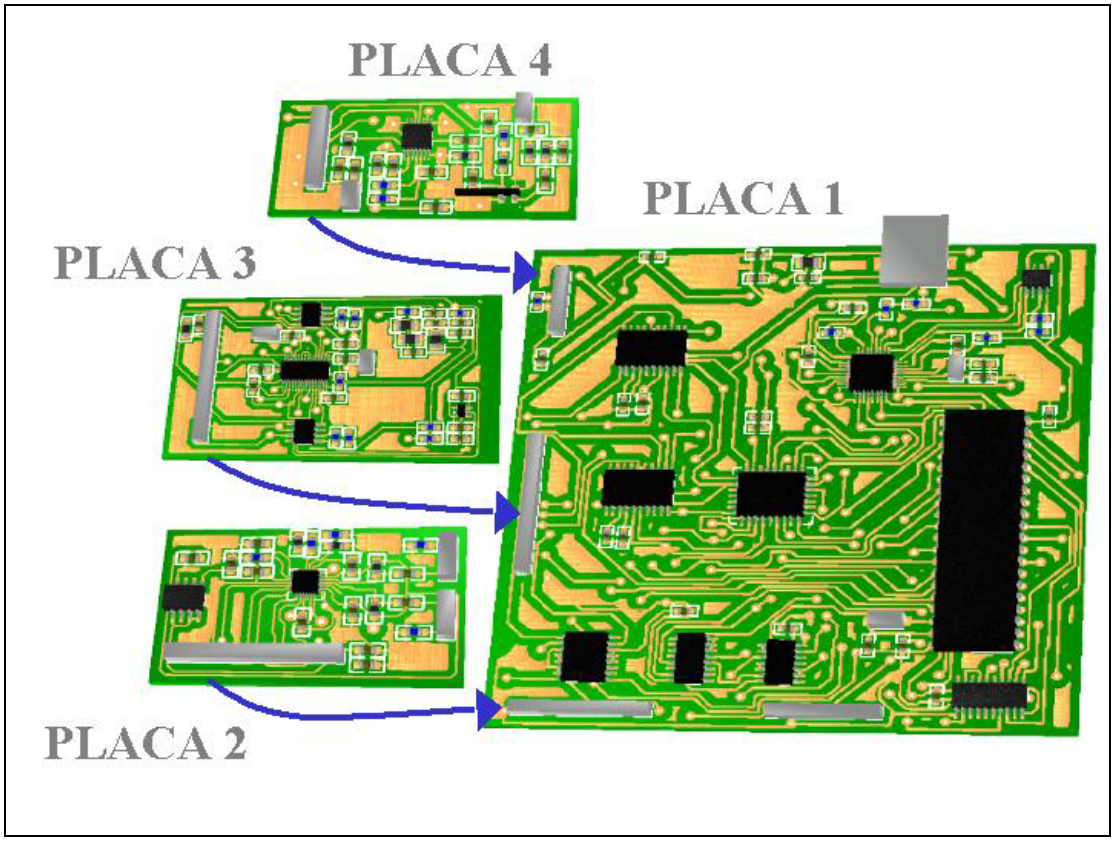


Title		
Size A4	Number pag. 13/14	Revision
Date: 29-Dec-2004		Sheet of
File: D:\Teleco\PFC Malaga I.Superior (2004)\...\Esquemas\RadioReceptorSoftwa		


PLACA 4: Entrada de RF
Bloque: Entrada, LNA y VGA
Filtro de preseleccion
CON1
RF IN
C4-1
10pF
L4-1
270nH
C4-2
160pF
L4-2
15nH
C4-9
2.2nF
LNA
U4-1
MAX2611
IN
OUT
GND
L4-3
1uH
R4-1
75
C4-3
10nF
Red
adaptadora (1)
C4-6
18pF
L4-4
180nH
VGA (Variable Gain Amplifier)
U4-2
AD8367
ENB
INP
MODE
GAIN
DETO
DECL
VPSI
VPSO
VOUT
HPFL
OCOM
POWER_DOWN
LNA_CTRL_GANANCIA
C4-8
2.2nF
C4-10
100nF
C4-11
100nF
C4-4
10uF
C4-5
100nF
R4-2
4.7
R4-3
4.7
Red
adaptadora (2)
L4-5
150nH
C4-7
2.2nF
C4-12
10pF
SEÑAL_RF
Conector
CON4-1
CON6
LNA_CTRL_GANANCIA
/POWER_DOWN
CON_OUT_RF
CON2
+5
GND
Title
Size
A4
Number
pag. 14/14
Revision
Date:
20-Mar-2005
Sheet of
File:
D:\Teleco\PFC Malaga I Superior (2004)\...\Esquemas\RadioReceptorSoftware.d

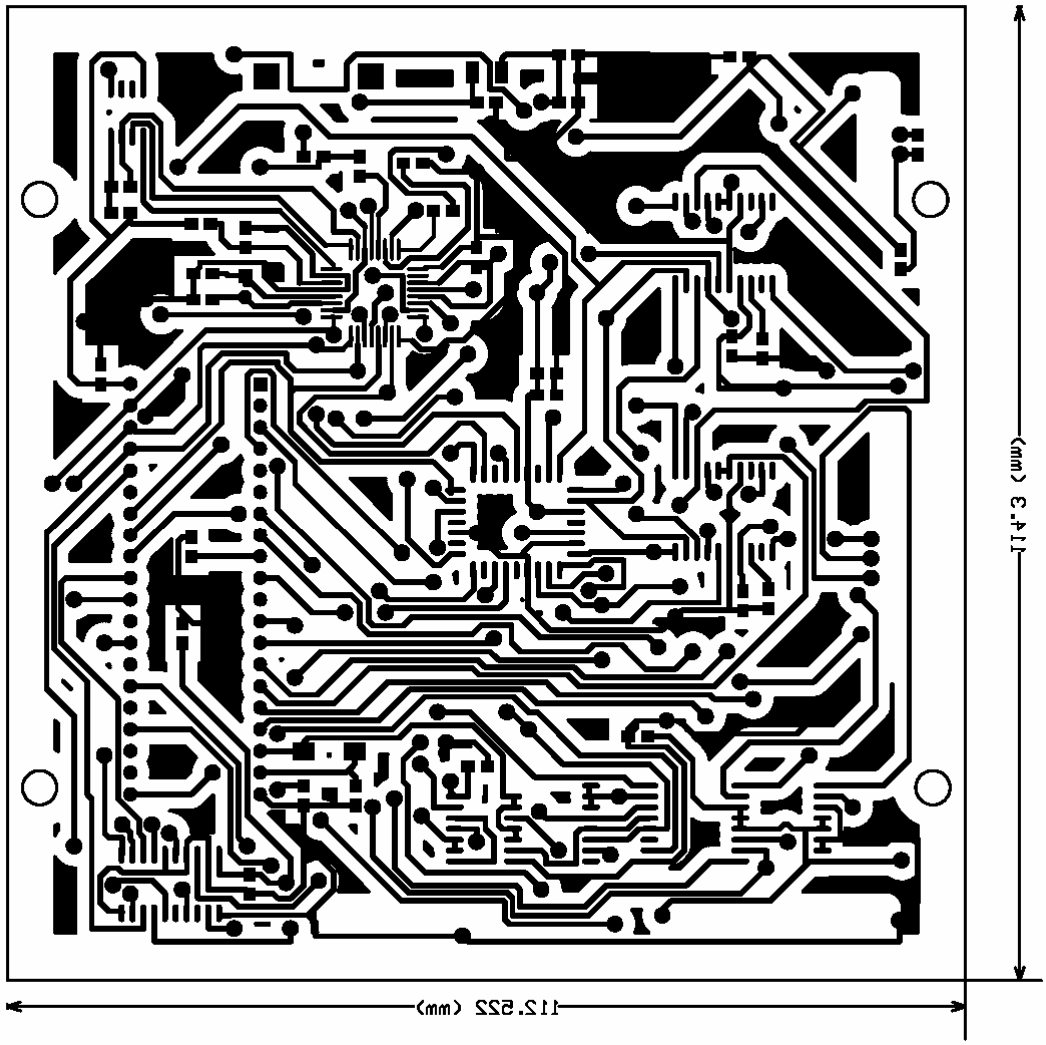
A2. Placas PCB

Por comodidad se repite aquí el esquema en 3D mostrando la disposición física de las cuatro placas que forman parte del receptor. Las siguientes páginas contienen las imágenes de los fotolitos usados para insolar cada una de las placas. No se han representado a tamaño real, sino ampliadas para poder observarlas con más comodidad.

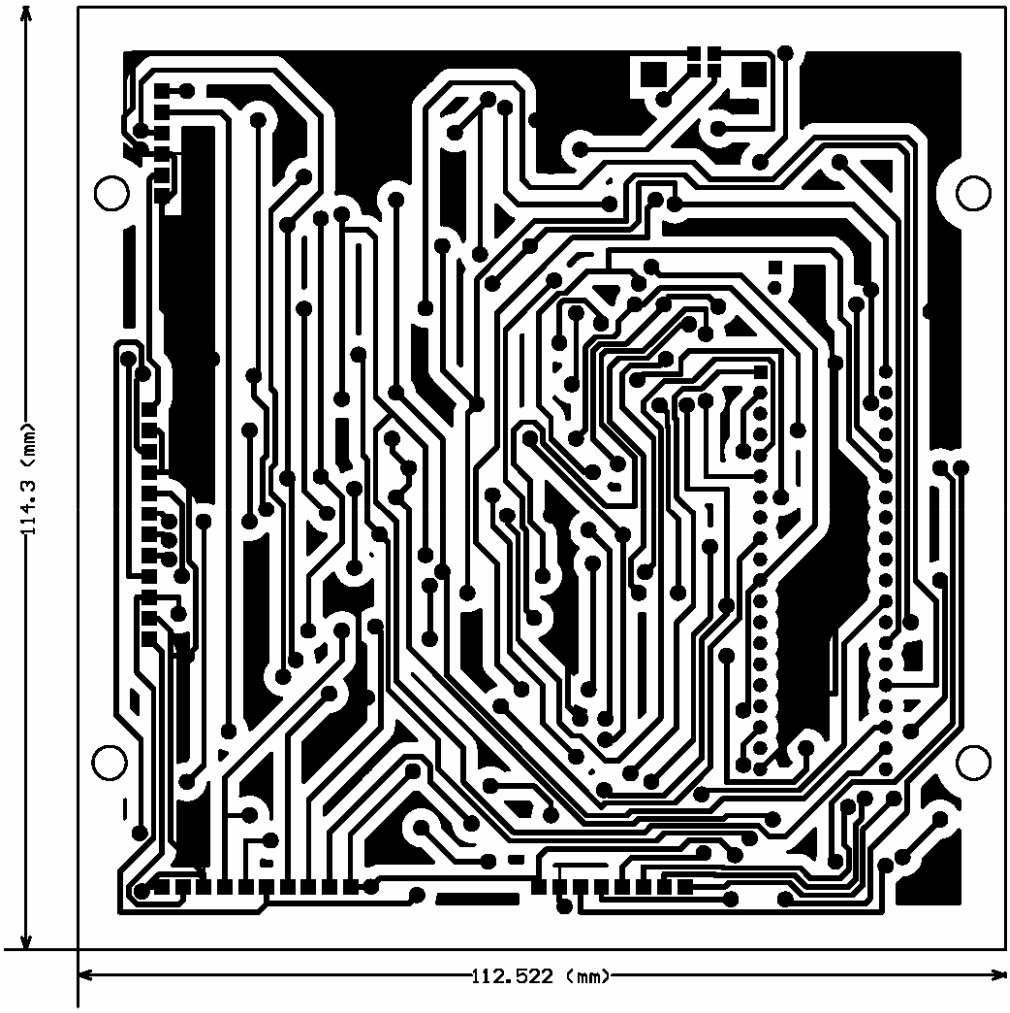


Placa 1: Principal

- Cara superior, invertida horizontalmente.

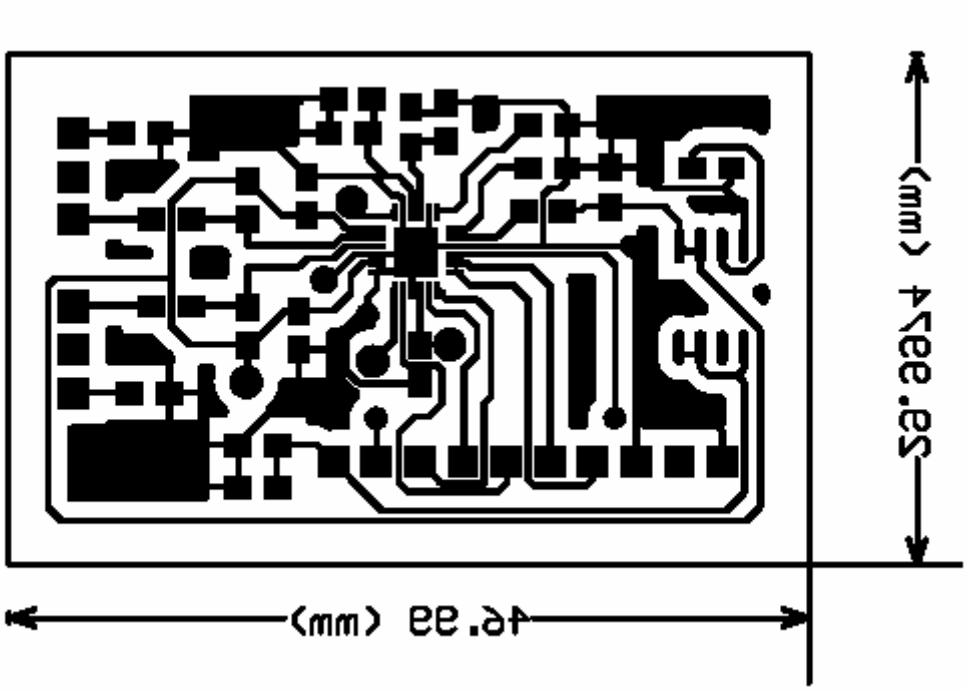


- Cara inferior, no invertida.

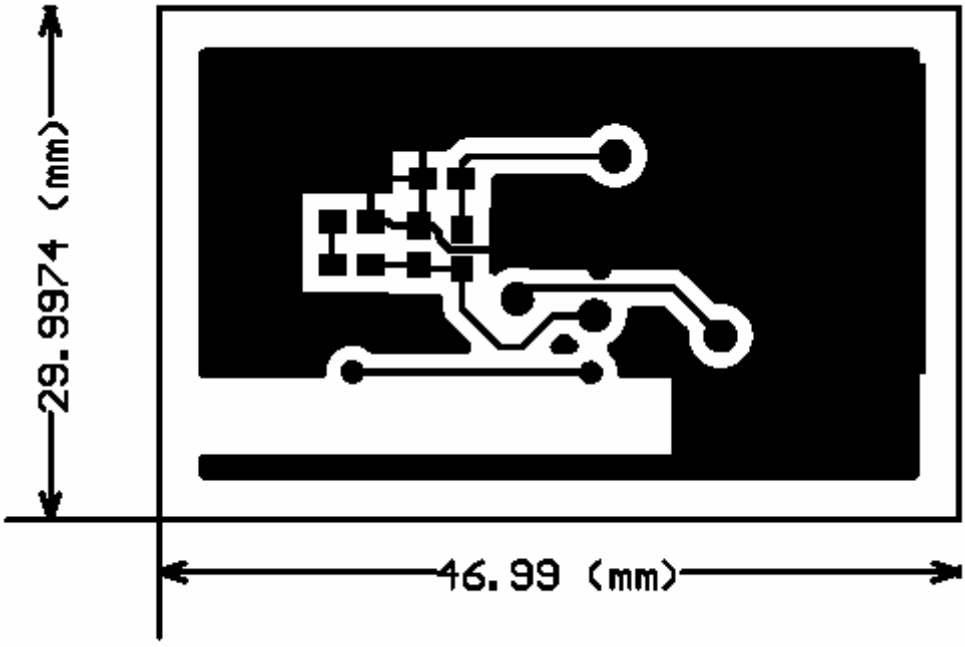


Placa 2: Oscilador

- Cara superior, invertida horizontalmente.

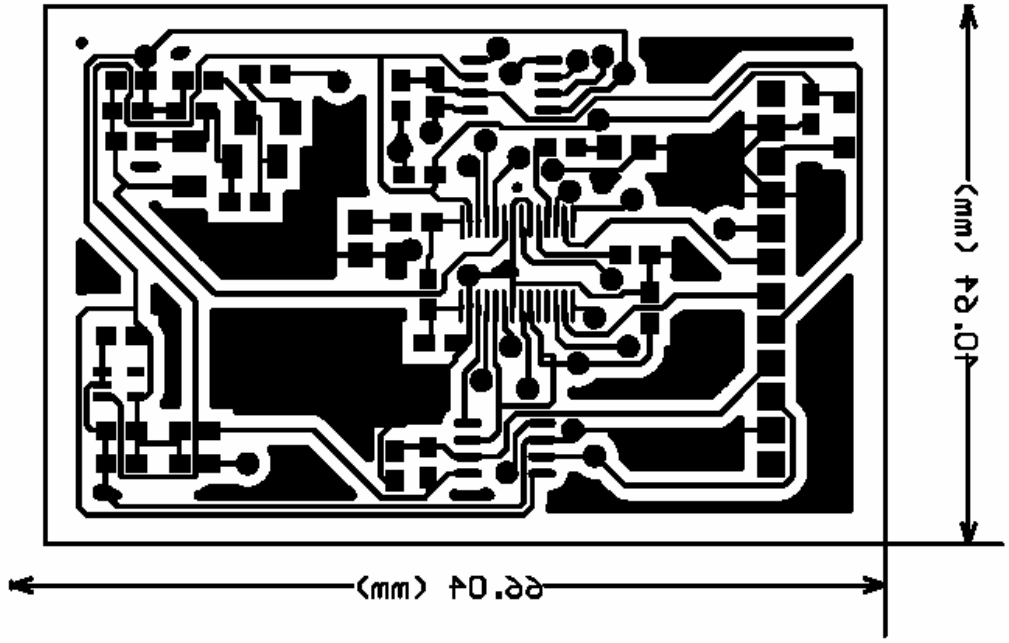


- Cara inferior, no invertida.

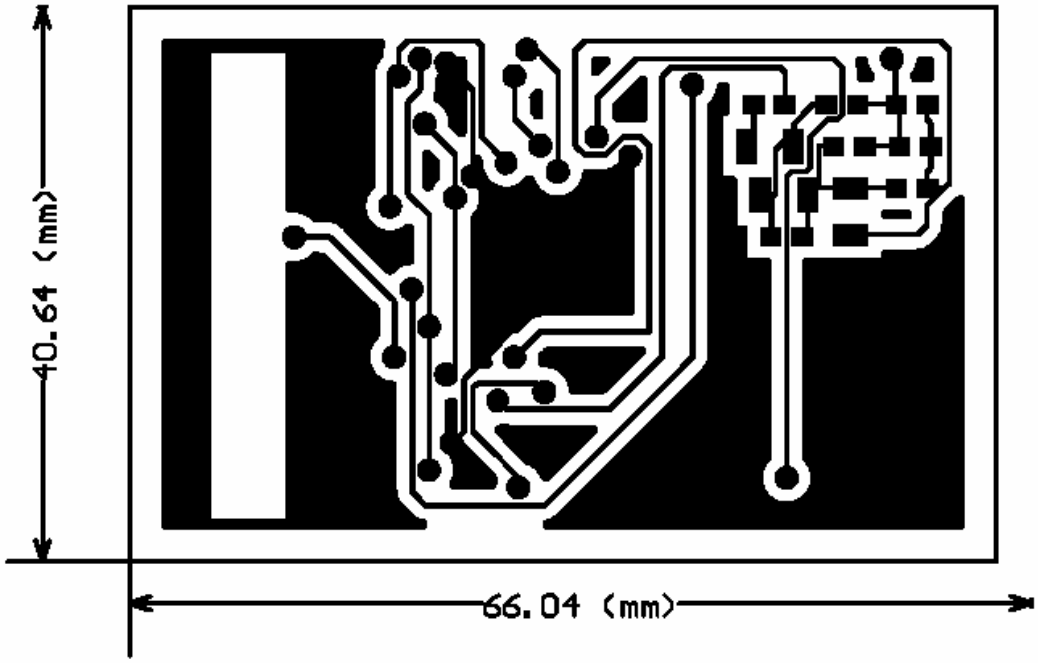


Placa 3: Demodulador

- Cara superior, invertida horizontalmente.

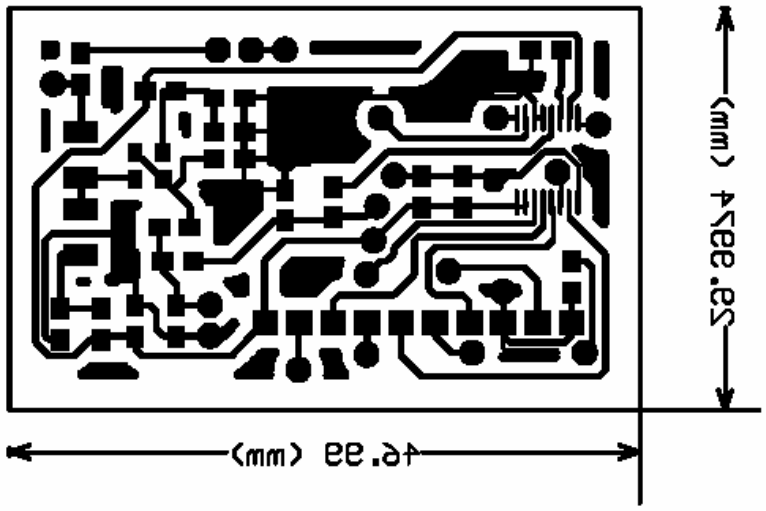


- Cara inferior, no invertida.

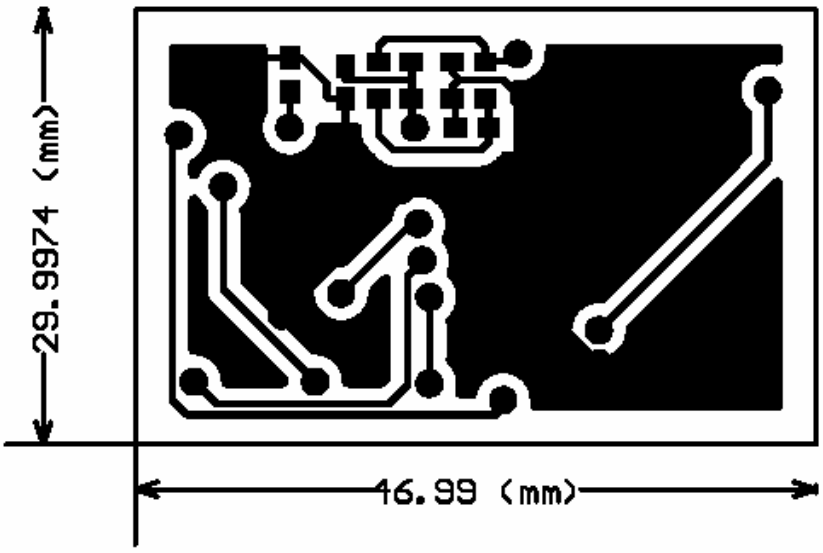


Placa 4: Amplificadores RF

- Cara superior, invertida horizontalmente.



- Cara inferior, no invertida.



A3. Hojas de características

A continuación se presentan las primeras páginas de todas las hojas de características de los integrados usados en el diseño del receptor. Solamente se incluye las primeras páginas por espacio, ya que el resto de la información es fácilmente accesible a través de Internet, pero así se incluye al menos la información más relevante sobre cada integrado que suele presentarse en la primera página de los *datasheets*.

Los integrados de lógica discreta (puertas lógicas y flip-flops) no se han incluido, ya que están lo suficientemente estandarizados como para ser ampliamente conocidos sin necesidad del *datasheet*.

MICROCHIP

93AA46A/B/C, 93LC46A/B/C, 93C46A/B/C

1K Microwire®-Compatible Serial EEPROM

Device Selection Table

Part Number	Vcc Range	ORG Pin	Word Size	Temp Ranges	Packages
93AA46A	1.8-5.5	No	8-bit	I	P, SN, ST, MS, OT
93AA46B	1.8-5-5	No	16-bit	I	P, SN, ST, MS, OT
93LC46A	2.5-5.5	No	8-bit	I, E	P, SN, ST, MS, OT
93LC46B	2.5-5.5	No	16-bit	I, E	P, SN, ST, MS, OT
93C46A	4.5-5.5	No	8-bit	I, E	P, SN, ST, MS, OT
93C46B	4.5-5.5	No	16-bit	I, E	P, SN, ST, MS, OT
93AA46C	1.8-5.5	Yes	8 or 16-bit	I	P, SN, ST, MS
93LC46C	2.5-5.5	Yes	8 or 16-bit	I, E	P, SN, ST, MS
93C46C	4.5-5.5	Yes	8 or 16-bit	I, E	P, SN, ST, MS

Features

- Low power CMOS technology
- ORG pin to select word size for '46C version
- 128 x 8-bit organization 'A' ver. devices (no ORG)
- 64 x 16-bit organization 'B' ver. devices (no ORG)
- Self-timed ERASE/WRITE cycles (including auto-erase)
- Automatic ERAL before WRAL
- Power on/off data protection circuitry
- Industry standard 3-wire serial I/O
- Device status signal (READY/$\overline{\text{BUSY}}$)
- Sequential READ function
- 1,000,000 E/W cycles
- Data retention > 200 years
- Temperature ranges supported
  - Industrial (I) -40°C to +85°C
  - Automotive (E) -40°C to +125°C

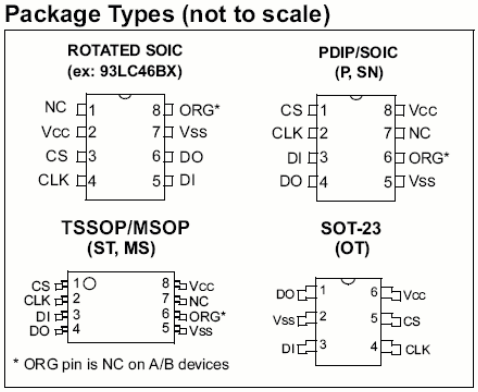
Pin Function Table

Name	Function
CS	Chip Select
CLK	Serial Data Clock
DI	Serial Data Input
DO	Serial Data Output
Vss	Ground
NC	No internal connection
ORG	Memory Configuration
Vcc	Power Supply

Description

The Microchip Technology Inc. 93XX46A/B/C devices are 1K bit low voltage serial Electrically Erasable PROMs (EEPROM). Word-selectable devices such as the 93AA46C, 93LC46C or 93C46C are dependent upon external logic levels driving the ORG pin to set word size. For dedicated 8-bit communication, the 93AA46A, 93LC46A or 93C46A devices are available, while the 93AA46B, 93LC46B and 93C46B devices provide dedicated 16-bit communication. Advanced CMOS technology makes these devices ideal for low power, non-volatile memory applications. The entire 93XX Series is available in standard packages including 8-lead PDIP and SOIC, and advanced packaging including 8-lead MSOP, 6-lead SOT-23, and 8-lead TSSOP. Pb-free (Pure Matte Sn) finish is also available.

Package Types (not to scale)



Microwire is a registered trademark of National Semiconductor.

© 2003 Microchip Technology Inc.

ANALOG DEVICES

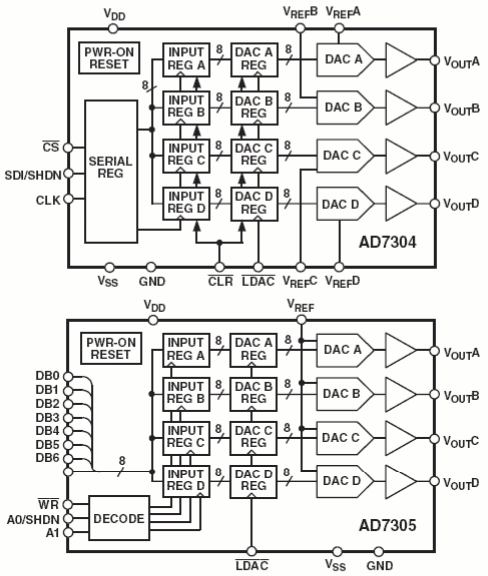
3 V/5 V, Rail-to-Rail Quad, 8-Bit DAC

AD7304/AD7305*

FEATURES
Four 8-Bit DACs in One Package
+3 V, +5 V, and ±5 V Operation
Rail-to-Rail REF Input to Voltage Output Swing
2.6 MHz Reference Multiplying Bandwidth
Compact 1.1 mm Height TSSOP 16-/20-Lead Package
Internal Power-On Reset
SPI Serial Interface Compatible—AD7304
Fast Parallel Interface—AD7305
40 μA Power Shutdown

APPLICATIONS
Automotive Output Span Voltage
Instrumentation, Digitally Controlled Calibration
Pin Compatible AD7226 Replacement when V_{DD} < 5.5 V

FUNCTIONAL BLOCK DIAGRAMS



GENERAL DESCRIPTION

The AD7304/AD7305 are quad, 8-bit DACs that operate from a single +3 V to +5 V supply or ±5 V supplies. The AD7304 has a serial interface, while the AD7305 has a parallel interface. Internal precision buffers swing rail-to-rail. The reference input range includes both supply rails allowing for positive or negative full-scale output voltages. Operation is guaranteed over the supply voltage range of 2.7 V to 5.5 V, consuming less than 9 mW from a 3 V supply.

The full-scale voltage output is determined by the external reference input voltage applied. The rail-to-rail V_{REF} input to DAC V_{OUT} allows for a full-scale voltage set equal the positive supply V_{DD}, the negative supply V_{SS}, or any value in between.

The AD7304's doubled-buffered serial data interface offers high speed, 3-wire, SPI, and microcontroller compatible inputs using data in (SDI), clock (CLK), and chip select ($\overline{CS}$) pins. Additionally, an internal power-on reset sets the output to zero scale.

The parallel input AD7305 uses a standard address decode along with the $\overline{WR}$ control line to load data into the input registers. The double buffered architecture allows all four input registers to be preloaded with new values, followed by a $\overline{LDAC}$ control strobe that copies all the new data into the DAC registers, thereby updating the analog output values. When operating from less than 5.5 V, the AD7305 is pin compatible with the popular industry-standard AD7226.

An internal power-on reset places both parts in the zero-scale state at turn ON. A 40 μA power shutdown (SHDN) feature is activated on both parts by tristating the SDI/SHDN pin on the AD7304 and tristating the A0/SHDN address pin on the AD7305.

The AD7304/AD7305 are specified over the extended industrial –40°C to +85°C and the automotive –40°C to +125°C temperature ranges. AD7304s are available in a wide-body 16 lead SOIC (R-16) package. The parallel input AD7305 is available in the wide-body 20 lead SOIC (R-20) surface-mount package. For ultracompact applications, the thin 1.1 mm 16 lead TSSOP (RU-16) package is available for the AD7304, while the 20 lead TSSOP (RU-20) houses the AD7305.

*Protected under Patent No. 5684481.

REV. B

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 www.analog.com
Fax: 781/326-8703 © 2004 Analog Devices, Inc. All rights reserved.

3 V/5 V, 2 MSPS, 8-Bit, 1-, 4-, 8-Channel Sampling ADCs

AD7822/AD7825/AD7829

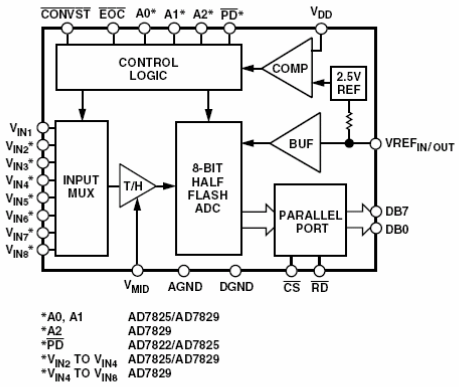
FEATURES

8-Bit Half-Flash ADC with 420 ns Conversion Time
1, 4 and 8 Single-Ended Analog Input Channels Available with Input Offset Adjust
On-Chip Track-and-Hold
SNR Performance Given for Input Frequencies Up to 10 MHz
On-Chip Reference (2.5 V)
Automatic Power-Down at the End of Conversion
Wide Operating Supply Range
3 V ± 10% and 5 V ± 10%
Input Ranges
0 V to 2 V p-p, V_{DD} = 3 V ± 10%
0 V to 2.5 V p-p, V_{DD} = 5 V ± 10%
Flexible Parallel Interface with EOC Pulse to Allow Stand-Alone Operation

APPLICATIONS

Data Acquisition Systems, DSP Front Ends
Disk Drives
Mobile Communication Systems, Subsampling Applications

FUNCTIONAL BLOCK DIAGRAM



*A0, A1	AD7825/AD7829
*A2	AD7829
*PD	AD7822/AD7825
*V_{IN2} TO V_{IN4}	AD7825/AD7829
*V_{IN4} TO V_{IN8}	AD7829

GENERAL DESCRIPTION

The AD7822, AD7825, and AD7829 are high speed, 1-, 4-, and 8-channel, microprocessor-compatible, 8-bit analog-to-digital converters with a maximum throughput of 2 MSPS. The AD7822, AD7825, and AD7829 contain an on-chip reference of 2.5 V (2% tolerance), a track/hold amplifier, a 420 ns 8-bit half-flash ADC and a high speed parallel interface. The converters can operate from a single 3 V ± 10% and 5 V ± 10% supply.

The AD7822, AD7825, and AD7829 combine the convert start and power-down functions at one pin, i.e., the $\overline{\text{CONVST}}$ pin. This allows a unique automatic power-down at the end of a conversion to be implemented. The logic level on the $\overline{\text{CONVST}}$ pin is sampled after the end of a conversion when an $\overline{\text{EOC}}$ (End of Conversion) signal goes high, and if it is logic low at that point, the ADC is powered down. The AD7822 and AD7825 also have a separate power-down pin. (See Operating Modes section of the data sheet.)

The parallel interface is designed to allow easy interfacing to microprocessors and DSPs. Using only address decoding logic, the parts are easily mapped into the microprocessor address space. The $\overline{\text{EOC}}$ pulse allows the ADCs to be used in a stand-alone manner. (See Parallel Interface section of the data sheet.)

The AD7822 and AD7825 are available in a 20-/24-lead 0.3" wide, plastic dual-in-line package (DIP), a 20-/24-lead small outline IC (SOIC) and a 20-/24-lead thin shrink small outline package (TSSOP). The AD7829 is available in a 28-lead 0.6" wide, plastic dual-in-line package (DIP), a 28-lead small outline IC (SOIC) and in a 28-lead thin shrink small outline package (TSSOP).

PRODUCT HIGHLIGHTS

1. Fast Conversion Time
   The AD7822, AD7825, and AD7829 have a conversion time of 420 ns. Faster conversion times maximize the DSP processing time in a real-time system.
2. Analog Input Span Adjustment
   The V_{MID} pin allows the user to offset the input span. This feature can reduce the requirements of single-supply op amps and take into account any system offsets.
3. FPBW (Full Power Bandwidth) of Track-and-Hold
   The track-and-hold amplifier has an excellent high-frequency performance. The AD7822, AD7825, and AD7829 are capable of converting full-scale input signals up to a frequency of 10 MHz. This makes the parts ideally suited to subsampling applications.
4. Channel Selection
   Channel selection is made without the necessity of writing to the part.

REV. B

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 www.analog.com
Fax: 781/326-8703
© Analog Devices, Inc., 2001

ANALOG DEVICES

Low-Cost 270 MHz Differential Receiver Amplifiers

AD8129/AD8130

FEATURES
High Speed
AD8130: 270 MHz, 1090 V/μs @ G = 1
AD8129: 200 MHz, 1060 V/μs @ G = 10
High CMRR
94 dB Min, DC to 100 kHz
80 dB Min @ 2 MHz
70 dB @ 10 MHz
High-Input Impedance: 1 MΩ Differential
Input Common-Mode Range ±10.5 V
Low Noise
AD8130: 12.5 nV/√Hz
AD8129: 4.5 nV/√Hz
Low Distortion, 1 V p-p @ 5 MHz:
AD8130, –79 dBc Worst Harmonic @ 5 MHz
AD8129, –74 dBc Worst Harmonic @ 5 MHz
User-Adjustable Gain
No External Components for G = 1
Power Supply Range +4.5 V to ±12.6 V
Power-Down

APPLICATIONS
High-Speed Differential Line Receiver
Differential-to-Single-Ended Converter
High-Speed Instrumentation Amp
Level-Shifting

GENERAL DESCRIPTION

The AD8129 and AD8130 are designed as receivers for the transmission of high-speed signals over twisted-pair cables to work with the AD8131 or AD8132 drivers. Either can be used for analog or digital video signals and for high-speed data transmission. The AD8129 and AD8130 are differential-to-single-ended amplifiers with extremely high CMRR at high frequency. Therefore, they can also be effectively used as high-speed instrumentation amps or for converting differential signals to single-ended signals.

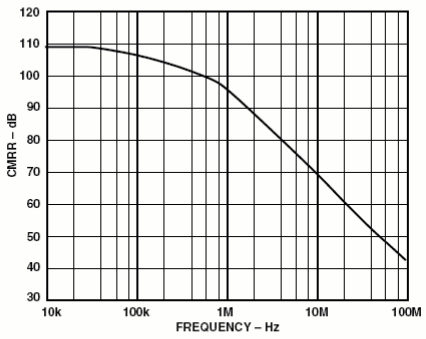
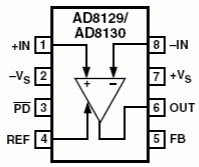


Figure 1. AD8129 CMRR vs. Frequency

CONNECTION DIAGRAM
(Top View)
SO-8 (R) and Micro_SO-8 (RM)



The AD8129 is a low-noise high-gain (10 or greater) version intended for applications over very long cables where signal attenuation is significant. The AD8130 is stable at a gain of one and can be used for those applications where lower gains are required. Both have user adjustable gain to help compensate for losses in the transmission line. The gain is set by the ratio of two resistor values. The AD8129 and AD8130 have very high input impedance on both inputs regardless of the gain setting.

The AD8129 and AD8130 have excellent common-mode rejection (70 dB @ 10 MHz) allowing the use of low cost unshielded twisted-pair cables without fear of corruption by external noise sources or crosstalk.

The AD8129 and AD8130 have a wide power supply range from single 5 V supply to ±12 V, allowing wide common-mode and differential-mode voltage ranges while maintaining signal integrity. The wide common-mode voltage range will enable the driver receiver pair to operate without isolation transformers in many systems where the ground potential difference between drive and receive locations is many volts. The AD8129 and AD8130 have considerable cost and performance improvements over op amps and other multi-amplifier receiving solutions.

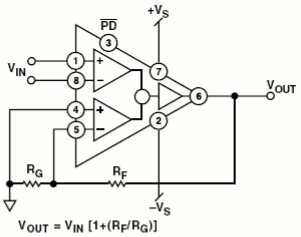


Figure 2. Typical Connection Configuration

REV. 0

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 www.analog.com
Fax: 781/326-8703 © Analog Devices, Inc., 2001

ANALOG DEVICES

50 MHz–1000 MHz Quadrature Demodulator

AD8348

FEATURES

Integrated I/Q demodulator with IF VGA amplifier
Operating IF frequency 50 MHz to 1000 MHz
(3 dB IF BW of 500 MHz driven from R_S = 200 Ω)
Demodulation bandwidth 75 MHz
Linear-in-dB AGC range 44 dB
Third order intercept
IIP3 +28 dBm @ min gain (F_{IF} = 380 MHz)
IIP3 –8 dBm @ max gain (F_{IF} = 380 MHz)
Quadrature demodulation accuracy
Phase accuracy 0.5°
Amplitude balance 0.25 dB
Noise figure 11 dB @ max gain (F_{IF} = 380 MHz)
LO input –10 dBm
Single supply 2.7 V to 5.5 V
Power-down mode
Compact 28-lead TSSOP package

APPLICATIONS

QAM/QPSK demodulator
W-CDMA/CDMA/GSM/NADC
Wireless local loop
LMDS

PRODUCT OVERVIEW

The AD8348 is a broadband quadrature demodulator with an integrated intermediate frequency (IF), variable gain amplifier (VGA), and integrated baseband amplifiers. It is suitable for use in communications receivers, performing quadrature demodulation from IF directly to baseband frequencies. The baseband amplifiers have been designed to directly interface with dual-channel ADCs, such as the AD9201, AD9283, and AD9218, for digitizing and postprocessing.

The IF input signal is fed into two Gilbert cell mixers through an X-AMP® VGA. The IF VGA provides 44 dB of gain control. A precision gain control circuit sets a linear-in-dB gain characteristic for the VGA and provides temperature compensation. The LO quadrature phase splitter employs a divide-by-two frequency divider to achieve high quadrature accuracy and amplitude balance over the entire operating frequency range.

FUNCTIONAL BLOCK DIAGRAM

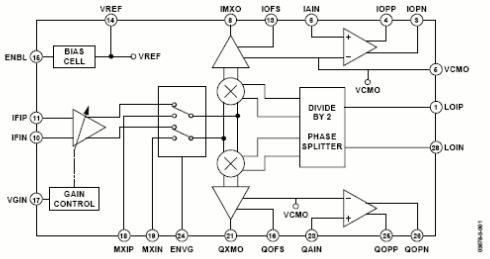


Figure 1.

Optionally, the IF VGA can be disabled and bypassed. In this mode, the IF signal is applied directly to the quadrature mixer inputs via the MXIP and MXIN pins.

Separate I and Q channel baseband amplifiers follow the baseband outputs of the mixers. The voltage applied to the VCMO pin sets the dc common-mode voltage level at the baseband outputs. Typically VCMO is connected to the internal VREF voltage, but it can also be connected to an external voltage. This flexibility allows the user to maximize the input dynamic range to the ADC. Connecting a bypass capacitor at each offset compensation input (IOFS and QOFS) nulls dc offsets produced in the mixer. Offset compensation can be overridden by applying an external voltage at the offset compensation inputs.

The mixers' outputs are brought off-chip for optional filtering before final amplification. Inserting a channel selection filter before each baseband amplifier increases the baseband amplifiers' signal handling range by reducing the amplitude of high level, out-of-channel interferers before the baseband signal is fed into the I/Q baseband amplifiers. The single-ended mixer output is amplified and converted to a differential signal for driving ADCs.

Rev. 0
Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective companies.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703 **© 2003 Analog Devices, Inc. All rights reserved.**

ANALOG DEVICES

500 MHz, Linear-in-dB VGA with AGC Detector

AD8367

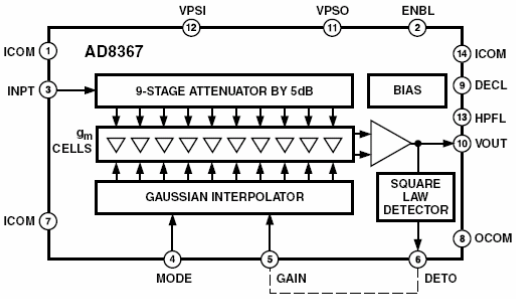
FEATURES

Broad Range Analog Variable Gain
–2.5 dB to +42.5 dB
3 dB Cutoff Frequency of 500 MHz
Gain Up and Gain Down Modes
Linear-in-dB, Scaled 20 mV/dB
Resistive Ground Referenced Input
Nominal Z_{IN} = 200 Ω
On-Chip Square-Law Detector
Single-Supply Operation: 2.7 V to 5.5 V

APPLICATIONS

Cellular Base Station
Broadband Access
Power Amplifier Control Loops
Complete, Linear IF AGC Amplifiers
High-Speed Data I/O

FUNCTIONAL BLOCK DIAGRAM



GENERAL DESCRIPTION

The AD8367 is a high-performance 45 dB variable gain amplifier with linear-in-dB gain control for use from low frequencies up to several hundred megahertz. The range, flatness, and accuracy of the gain response are achieved using Analog Devices' X-AMP® architecture, the most recent in a series of powerful proprietary concepts for variable gain applications, which far surpasses what can be achieved using competing techniques.

The input is applied to a 200 Ω resistive ladder network, having nine sections each of 5 dB loss, for a total attenuation of 45 dB. At maximum gain, the first tap is selected; at progressively lower gains, the tap moves smoothly and continuously toward higher attenuation values. The attenuator is followed by a 42.5 dB fixed gain feedback amplifier—essentially an operational amplifier with a gain bandwidth product of 100 GHz—and is very linear, even at high frequencies. The output third order intercept is +20 dBV at 100 MHz (+27 dBm re 200 Ω), measured at an output level of 1 V p-p with V_S = 5 V.

The analog gain-control interface is very simple to use. It is scaled at 20 mV/dB, and the control voltage, V_{GAIN}, runs from 50 mV at –2.5 dB to 950 mV at +42.5 dB. In the inverse-gain mode of operation, selected by a simple pin-strap, the gain decreases from +42.5 dB at V_{GAIN} = 50 mV to –2.5 dB at V_{GAIN} = 950 mV. This inverse mode is needed in AGC applications, which are supported by the integrated square-law detector, whose set point is chosen to level the output to 354 mV rms, regardless of the waveshape. A single external capacitor sets up the loop averaging time.

The AD8367 may be powered on or off by a voltage applied to the ENBL pin. When this voltage is at a logic LO, the total power dissipation drops to the milliwatt range. For a logic HI, the chip powers-up rapidly to its normal quiescent current of 26 mA at 25°C. The AD8367 is available in a 14-lead TSSOP package for the industrial temperature range of –40°C to +85°C.

X-AMP is a registered trademark of Analog Devices, Inc.

REV. 0

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781/329-4700 www.analog.com
Fax: 781/326-8703
© Analog Devices, Inc., 2001

ANALOG DEVICES

Integrated Synthesizer and VCO

ADF4360-8

FEATURES

Output frequency range: 65 MHz to 400 MHz
3.0 V to 3.6 V power supply
1.8 V logic compatibility
Integer-N synthesizer
Programmable output power level
3-wire serial interface
Digital lock detect
Hardware and software power-down mode

APPLICATIONS

System clock generation
Test equipment
Wireless LANs
CATV equipment

GENERAL DESCRIPTION

The ADF4360-8 is an integrated integer-N synthesizer and voltage-controlled oscillator (VCO). The ADF4360-8 center frequency is set by external inductors. This allows a frequency range of between 65 MHz to 400 MHz.

Control of all the on-chip registers is through a simple 3-wire interface. The device operates with a power supply ranging from 3.0 V to 3.6 V and can be powered down when not in use.

FUNCTIONAL BLOCK DIAGRAM

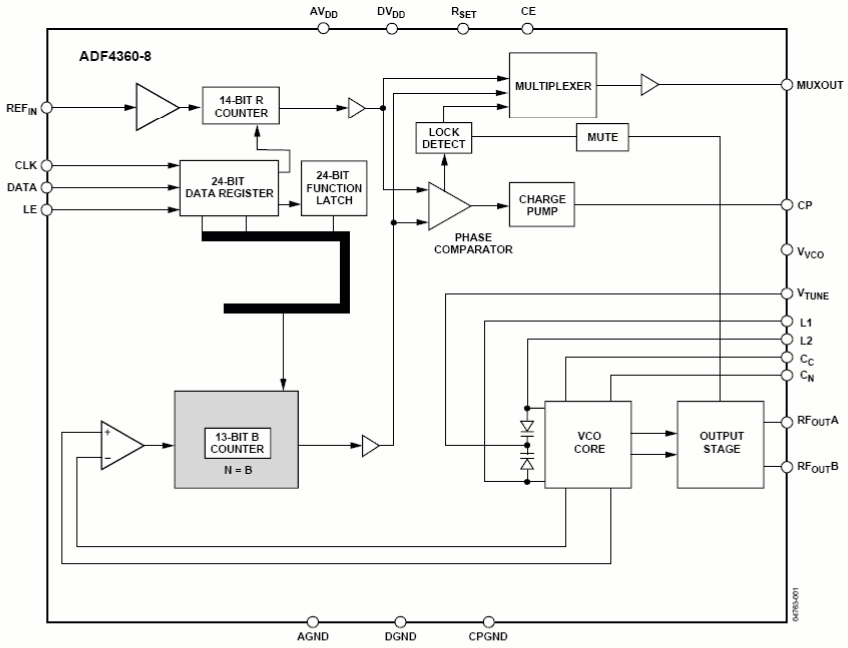


Figure 1.

Rev. A

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703 **© 2005 Analog Devices, Inc. All rights reserved.**

High Accuracy, Ultralow I_Q, 1 A, anyCAP® Low Dropout Regulator

ADP3338

FEATURES

High accuracy over line and load: ±0.8% @ 25°C, ±1.4% over temperature
Ultralow dropout voltage: 190 mV (typ) @ 1 A
Requires only C_O = 1.0 µF for stability
anyCAP = stable with any type of capacitor (including MLCC)
Current and thermal limiting
Low noise
2.7 V to 8 V supply range
−40°C to +85°C ambient temperature range
SOT-223 package

APPLICATIONS

Notebook, palmtop computers
SCSI terminators
Battery-powered systems
Bar code scanners
Camcorders, cameras
Home entertainment systems
Networking systems
DSP/ASIC supplies

GENERAL DESCRIPTION

The ADP3338 is a member of the ADP33xx family of precision, low dropout, anyCAP voltage regulators. The ADP3338 operates with an input voltage range of 2.7 V to 8 V and delivers a load current up to 1 A. The ADP3338 stands out from conventional LDOs with a novel architecture and an enhanced process that allows it to offer performance advantages and higher output current than its competition. Its patented design requires only a 1 µF output capacitor for stability. This device is insensitive to output capacitor equivalent series resistance (ESR), and is stable with any good quality capacitor, including ceramic (MLCC) types for space-restricted applications. The ADP3338 achieves exceptional accuracy of ±0.8% at room temperature and ±1.4% over temperature, line, and load variations. The dropout voltage of the ADP3338 is only 190 mV (typical) at 1 A. The device also includes a safety current limit and thermal overload protection. The ADP3338 has ultralow quiescent current: 110 µA (typical) in light load situations.

FUNCTIONAL BLOCK DIAGRAM

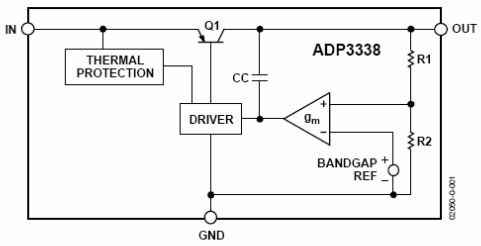


Figure 1.

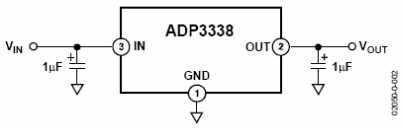


Figure 2. Typical Application Circuit

Rev. A
Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703 **© 2004 Analog Devices, Inc. All rights reserved.**

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- RISC Architecture
  - 130 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
  - 8K Bytes of In-System Self-programmable Flash
    Endurance: 10,000 Write/Erase Cycles
  - Optional Boot Code Section with Independent Lock bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - 512 Bytes EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 512 Bytes Internal SRAM
  - Up to 64K Bytes Optional External Memory Space
  - Programming Lock for Software Security
- Peripheral Features
  - One 8-bit Timer/Counter with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Three PWM Channels
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Three Sleep Modes: Idle, Power-down and Standby
- I/O and Packages
  - 35 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, 44-lead PLCC, and 44-pad MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega8515L
  - 4.5 - 5.5V for ATmega8515
- Speed Grades
  - 0 - 8 MHz for ATmega8515L
  - 0 - 16 MHz for ATmega8515

8-bit AVR® Microcontroller with 8K Bytes In-System Programmable Flash

ATmega8515
ATmega8515L

Rev. 2512F–AVR–12/03

CYPRESS

CY7C419/21/25/29/33

256/512/1K/2K/4K x 9 Asynchronous FIFO

Features

- **Asynchronous first-in first-out (FIFO) buffer memories**
- **256 x 9 (CY7C419)**
- **512 x 9 (CY7C421)**
- **1K x 9 (CY7C425)**
- **2K x 9 (CY7C429)**
- **4K x 9 (CY7C433)**
- **Dual-ported RAM cell**
- **High-speed 50.0-MHz read/write independent of depth/width**
- **Low operating power: I_{CC} = 35 mA**
- **Empty and Full flags (Half Full flag in standalone)**
- **TTL compatible**
- **Retransmit in standalone**
- **Expandable in width**
- **PLCC, 7x7 TQFP, SOJ, 300-mil and 600-mil DIP**
- **Pin compatible and functionally equivalent to IDT7200, IDT7201, IDT7202, IDT7203, IDT7204, AM7200, AM7201, AM7202, AM7203, and AM7204**

Functional Description

The CY7C419, CY7C420/1, CY7C424/5, CY7C428/9, and CY7C432/3 are first-in first-out (FIFO) memories offered in 600-mil wide and 300-mil wide packages. They are, respectively, 256, 512, 1,024, 2,048, and 4,096 words by 9-bits wide.

Each FIFO memory is organized such that the data is read in the same sequential order that it was written. Full and Empty flags are provided to prevent overrun and underrun. Three additional pins are also provided to facilitate unlimited expansion in width, depth, or both. The depth expansion technique steers the control signals from one device to another in parallel, thus eliminating the serial addition of propagation delays, so that throughput is not reduced. Data is steered in a similar manner.

The read and write operations may be asynchronous; each can occur at a rate of 50.0 MHz. The write operation occurs when the write ($\overline{W}$) signal is LOW. Read occurs when read ($\overline{R}$) goes LOW. The nine data outputs go to the high-impedance state when $\overline{R}$ is HIGH.

A Half Full ($\overline{HF}$) output flag is provided that is valid in the standalone and width expansion configurations. In the depth expansion configuration, this pin provides the expansion out ($\overline{XO}$) information that is used to tell the next FIFO that it will be activated.

In the standalone and width expansion configurations, a LOW on the retransmit ($\overline{RT}$) input causes the FIFOs to retransmit the data. Read enable ($\overline{R}$) and write enable ($\overline{W}$) must both be HIGH during retransmit, and then $\overline{R}$ is used to access the data.

The CY7C419, CY7C420, CY7C421, CY7C424, CY7C425, CY7C428, CY7C429, CY7C432, and CY7C433 are fabricated using an advanced 0.65-micron P-well CMOS technology. Input ESD protection is greater than 2000V and latch-up is prevented by careful layout and guard rings.

Cypress Semiconductor Corporation • 3901 North First Street • San Jose • CA 95134 • 408-943-2600
Document #: 38-06001 Rev. ** Revised November 4, 1997

FT245BM USB FIFO (USB - Parallel) I.C.

The FT245BM is the 2nd generation of FTDI's popular USB FIFO I.C. This device not only adds extra functionality to its FT8U245AM predecessor and reduces external component count, but also maintains a high degree of pin compatibility with the original, making it easy to upgrade or cost reduce existing designs as well as increasing the potential for using the device in new application areas.

1.0 Features

HARDWARE FEATURES

- Single Chip USB ⇔ Parallel FIFO bi-directional Data Transfer
- Transfer Data rate to 1M Byte / Sec - D2XX Drivers
- Transfer Data rate to 300 Kilobyte / Sec - VCP Drivers
- Simple to interface to MCU / PLD/ FPGA logic with a 4 wire handshake interface
- Entire USB protocol handled on-chip… no USB-specific firmware programming required
- FTDI's royalty-free VCP and D2XX drivers eliminate the requirement for USB driver development in most cases.
- 384 Byte FIFO Tx buffer / 128 Byte FIFO Rx Buffer for high data throughput.
- New Send Immediate support via SI Pin for optimised data throughput.
- Support for USB Suspend / Resume through PWREN# and WAKEUP pins.
- Support for high power USB Bus powered devices through PWREN# pin
- Adjustable RX buffer timeout
- In-built support for event characters
- Integrated level converter on FIFO and control signals for interfacing to 5V and 3.3V logic
- Integrated 3.3V regulator for USB IO
- Integrated Power-On-Reset circuit
- Integrated 6MHz – 48Mhz clock multiplier PLL
- USB Bulk or Isochronous data transfer modes
- New Bit-Bang Mode allows the data bus to be used as an 8 bit general purpose IO Port without the need for MCU or other support logic.
- 4.35V to 5.25V single supply operation
- UHCI / OHCI / EHCI host controller compatible
- USB 1.1 and USB 2.0 compatible
- USB VID, PID , Serial Number and Product Description strings in external EEPROM
- EEPROM programmable on-board via USB
- Compact 32LD LQFP package

VIRTUAL COM PORT (VCP) DRIVERS for

- Windows 98 and Windows 98 SE
- Windows 2000 / ME / XP
- Windows CE **
- MAC OS-8 and OS-9
- MAC OS-X
- Linux 2.40 and greater

D2XX (USB Direct Drivers + DLL S/W Interface)

- Windows 98 and Windows 98 SE
- Windows 2000 / ME / XP

APPLICATION AREAS

- Easy MCU / PLD / FPGA interface to USB
- Upgrading Legacy Peripheral Designs to USB
- USB Instrumentation
- USB Industrial Control
- USB Audio and Low Bandwidth Video data transfer
- PDA ⇔ USB data transfer
- USB MP3 Player Interface
- USB FLASH Card Reader / Writers
- Set Top Box (S.T.B.) PC - USB interface
- USB Digital Camera Interface
- USB Hardware Modems
- USB Wireless Modems

[** = In planning or under development]

19-1094; Rev 1; 8/03

EVALUATION KIT AVAILABLE

MAXIM

DC-to-Microwave, Low-Noise Amplifier

MAX2611

General Description

The MAX2611 is a low-voltage, low-noise amplifier for use from DC to microwave frequencies. Operating from a single +5V supply, it has a 3dB bandwidth of 1100MHz. The MAX2611's low noise figure and high drive capability make it ideal for a variety of transmit, receive, and buffer applications.

In a typical application, the only external components needed are input and output blocking capacitors and a V_{CC} series resistor. To improve gain and output power, an RF choke can be added in series to the bias resistor.

The MAX2611 comes in a 4-pin SOT143 package, requiring minimal board space.

Features

- **Single +5V Supply Operation**
- **3dB Bandwidth: DC to 1100MHz**
- **High Gain: 18dB at 500MHz**
- **Low Noise Figure: 3.5dB at 500MHz**
- **High Drive Capability: +3dBm at 16mA I_D**
- **Ultra-Small SOT143 Package**

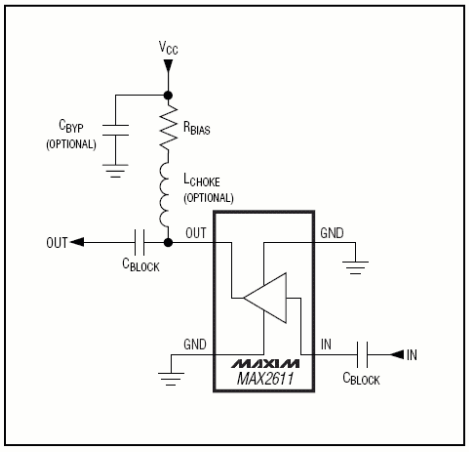
Applications

Satellite Receivers
TV Tuners
Wireless Local Loops
ISM Radios
Global Positioning Systems
Wireless Local-Area Networks
Land Mobile Radios
Set-Top Boxes

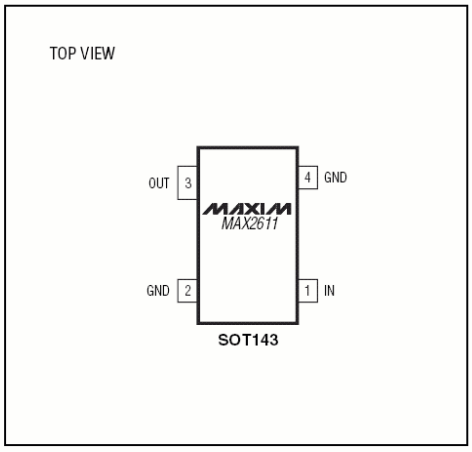
Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX2611EUS	-40°C to +85°C	4 SOT143

Typical Operating Circuit



Pin Configuration



For pricing, delivery, and ordering information, please contact Maxim/Dallas Direct! at 1-888-629-4642, or visit Maxim's website at www.maxim-ic.com.

19-2672; Rev 2; 1/05

MAXIM

+1.2V to +5.5V, ±15kV ESD-Protected, 0.1µA, 35Mbps, 8-Channel Level Translators

General Description

The MAX3000E/MAX3001E/MAX3002–MAX3012 8-channel level translators provide the level shifting necessary to allow data transfer in a multivoltage system. Externally applied voltages, V_{CC} and V_L, set the logic levels on either side of the device. Logic signals present on the V_L side of the device appear as a higher voltage logic signal on the V_{CC} side of the device, and vice-versa.

The MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012 use an architecture specifically designed to be bidirectional without the use of a directional pin.

The MAX3000E/MAX3001E/MAX3002/MAX3004–MAX3012 feature an EN input that, when low, reduces the V_{CC} and V_L supply currents to <2µA. The MAX3000E/MAX3001E also have ±15kV ESD protection on the I/O V_{CC} side for greater protection in applications that route signals externally. The MAX3000E operates at a guaranteed data rate of 230kbps. The MAX3001E operates at a guaranteed data rate of 4Mbps. The MAX3002–MAX3012 operate at a guaranteed data rate of 20Mbps over the entire specified operating voltage range.

The MAX3000E/MAX3001E/MAX3002–MAX3012 accept V_L voltages from +1.2V to +5.5V and V_{CC} voltages from +1.65V to +5.5V, making them ideal for data transfer between low-voltage ASICs/PLDs and higher voltage systems. The MAX3000E/MAX3001E/MAX3002–MAX3012 are available in 20-pin UCSP™ and 20-pin TSSOP packages.

Applications

CMOS Logic-Level Translation
Cellphones
SPI™ and MICROWIRE™ Level Translation
Low-Voltage ASIC Level Translation
Smart Card Readers
Cellphone Cradles
Portable POS Systems
Portable Communication Devices
Low-Cost Serial Interfaces
GPS
Telecommunications Equipment

UCSP is a trademark of Maxim Integrated Products, Inc.

SPI is a trademark of Motorola, Inc.

MICROWIRE is a trademark of National Semiconductor.

Features

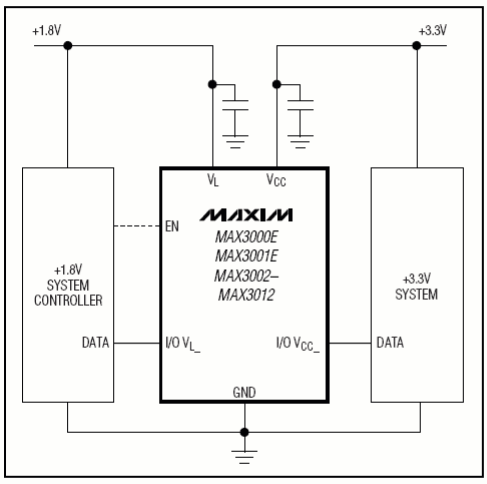
- **Guaranteed Data Rate Options**
  230kbps (MAX3000E)
  4Mbps (MAX3001E)
  20Mbps (MAX3002–MAX3012)
- **Bidirectional Level Translation Without Using a Direction Pin (MAX3000E/MAX3001E/MAX3002/MAX3003)**
- **Unidirectional Level Translation (MAX3004–MAX3012)**
- **Operation Down to +1.2V on V_L**
- **±15kV ESD Protection on I/O V_{CC} Lines (MAX3000E/MAX3001E)**
- **Ultra-Low 0.1µA Supply Current in Shutdown**
- **Low Quiescent Current (<10µA)**
- **UCSP and TSSOP Packages**

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX3000EEUP	-40°C to +85°C	20 TSSOP
MAX3000EEBP-T	-40°C to +85°C	4 x 5 UCSP

Ordering Information continued at end of data sheet.

Typical Operating Circuit



Pin Configurations and Functional Diagrams appear at end of data sheet.

For pricing, delivery, and ordering information, please contact Maxim/Dallas Direct! at 1-888-629-4642, or visit Maxim's website at www.maxim-ic.com.

SOD323 SILICON HYPERABRUPT VARIABLE CAPACITANCE DIODES

ZMV930 SERIES

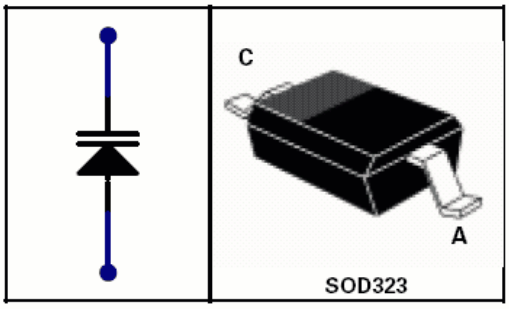
ISSUE 1 - NOVEMBER 1998

FEATURES

* VHF Tuning
* Octave Tuning from 0 TO 6 Volts
* High Reliability and Low Parasitics
* Low Leakage (Typically <200pA at 10V)
* Miniature Surface Mount Package

APPLICATIONS

* Mobile Radios and Pagers
* Cellular Telephones



ABSOLUTE MAXIMUM RATINGS

PARAMETER	SYMBOL	VALUE	UNIT
Reverse Voltage	V_R	12	V
Forward Current	I_F	100	mA
Power Dissipation at T_{amb}=25°C	P_{tot}	330	mW
Junction Temperature	T_j	125	°C
Storage Temperature	T_{stg}	-55 to 150	°C

*Maximum power dissipation is calculated assuming that the device is mounted on a ceramic substrate measuring 10 x 8 x 0.6mm

ELECTRICAL CHARACTERISTICS (T_{amb}=25°C)

PARAMETER	SYMBOL	MIN	MAX	UNIT	CONDITIONS
Breakdown Voltage	V_{BR}	12		Volts	I_R=10μA
Reverse Leakage	I_R		100	nA	V_R=8V

TUNING CHARACTERISTICS (T_{amb}=25°C)

Type	Capacitance @V_R=1V	Capacitance @V_R=2.5 V		Capacitance @V_R=4 V	Figure of merit minimum Q V_R=4 V, f=50MHz	Part-mark
	Min. pF	Min. pF	Max. pF	Max. pF		
ZMV930	8.70	4.30	5.50	2.90	200	AH
ZMV931	14.50	6.50	7.80	4.00	300	AJ
ZMV932	17.00	8.50	10.50	5.50	200	AK
ZMV933	42.00	18.00	27.00	12.00	150	AL
ZMV933A	42.00	20.25	24.75	12.00	150	AM
ZMV934	95.00	40.00	65.00	25.00	80	AN
ZMV934A	95.00	47.25	57.75	25.00	80	AO

A4. Listados de código fuente

A4.1. MATLAB

```
%-------------------------------------------------------------------------
% Script MATLAB: DibujarRespFiltro.m
%   Descripcion: Representa la respuesta del filtro de preseleccion de RF
%-------------------------------------------------------------------------
close all;
clear all;

f=linspace(log(10e6),log(1e9),200);
f=exp(f);

H_Ideal=resp_filtro_RF(f,63,50,2.2e-9,9.4e-12,280.1e-9,180.2e-12,14.64e-9);
set(semilogx(f/1e6,H_Ideal,'r'),'LineWidth',3);

hold on;

H_real=resp_filtro_RF(f,63,50,2.2e-9,10e-12,270e-9,160e-12,15e-9);
set(semilogx(f/1e6,H_real,'b-.'),'LineWidth',3);

semilogx([85.8 85.8],[-3 -60],'k');
semilogx([118.2 118.2],[-3 -60],'k');

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
ylabel('Respuesta en potencia (dB)');
grid on;


%-------------------------------------------------------------------------
% Funcion MATLAB: resp_filtro_RF.m
%    Descripcion: Calcula la ganancia en potencia del filtro de
%                 preseleccion de RF para los valores de componentes
%                 dados como parametros.
%-------------------------------------------------------------------------
function [ G ] = resp_filtro_RF( f, ZL, ZS, C0, C1,L1,C2,L2)

V1 = 1;

ZC0 = Z_cond(C0,f);
ZC1 = Z_cond(C1,f);
ZC2 = Z_cond(C2,f);
ZL1 = Z_ind(L1,f);
ZL2 = Z_ind(L2,f);

ZF1 = ZC1 + ZL1;
ZF2 = paralelo( ZC2, ZL2 );

I1 = V1 ./ ( ZS + ZF1 + paralelo( ZF2,ZC0+ZL ) );
V2 = I1 .* paralelo( ZF2,ZC0+ZL );
I2 = V2 ./ ( ZC0 + ZL );
V0 = I2 .* ZL;

Pi = abs(V1).^2 ./ ( abs(ZS) );
Po = abs(V0).^2 ./ ( abs(ZL) );

G = 10.*log10( Po ./ (Pi.*0.25) );
```

```
%-------------------------------------------------------------------------
% Script MATLAB: DibujarRespSalidaLNA.m
%   Descripcion: Representa la potencia de salida del LNA en funcion
%                del valor de la inductancia del choque RF.
%-------------------------------------------------------------------------
close all;
clear all;

L=linspace(log(1e-9),log(100e-6),100);
L=exp(L);

H=salida_LNA(98e6,L);

set(semilogx(L,H),'LineWidth',2);

xlabel('Inductancia (H)');
ylabel('Resp. en potencia (dB)');
grid on;



%-------------------------------------------------------------------------
% Funcion MATLAB: salida_LNA.m
%    Descripcion: Calcula la ganancia de potencia resultante de usar
%                 la inductancia pasada como parametro como
%                 choque en el LNA.
%-------------------------------------------------------------------------
function [ G ] = salida_LNA( f, L)

V1 = 1;

ZS = 53+j*4;    % S22=0.05 / 53º
ZL = 53-j*4;
Z2 = 75 + Z_ind( L,f );

I = V1 ./ ( ZS + paralelo( ZL, Z2 ) );
V0 = I.*paralelo( ZL, Z2 );

Pi = abs(V1).^2 ./ ( real(ZS) );
Po = abs(V0).^2 ./ ( real(ZL) );

G = 10.*log10( Po ./ (Pi.*0.25) );


%-------------------------------------------------------------------------
% Funcion MATLAB: DibujarRespRedAdaptadora.m
%    Descripcion: Muestra graficamente resultados de la simulacion
%                 para la primera red adaptadora
%-------------------------------------------------------------------------
close all;
clear all;

f=linspace(log(10e6),log(1e9),500);
f=exp(f);

ZS = 53;
ZL = 193-45*j;

[R_Ideal_1 S11_ideal_1]=resp_red_adaptadora(f,ZL,ZS,17.24e-12,168.84e-9);
set(semilogx(f/1e6,R_Ideal_1,'r-.'),'LineWidth',3);

hold on;

[R_real_1 S11_real_1]=resp_red_adaptadora(f,ZL,ZS,18e-12,180e-9);
set(semilogx(f/1e6,R_real_1,'b'),'LineWidth',3);

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
title('Respuesta en potencia (dB)');
grid on;
```

```
% En otra grafica 1-|S11|^2:
figure;
PotTx_ideal_1 = 1-abs(S11_ideal_1).^2;
PotTx_real_1  = 1-abs(S11_real_1).^2;

set(semilogx(f/1e6,PotTx_ideal_1,'r-.'),'LineWidth',3);hold on;
set(semilogx(f/1e6,PotTx_real_1,'b'),'LineWidth',3);

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
title('1-|S_1_1|^2');
grid on;

% En otra grafica el S11:
figure;

set(semilogx(f/1e6,abs(S11_ideal_1),'r-.'),'LineWidth',3);hold on;
set(semilogx(f/1e6,abs(S11_real_1),'b'),'LineWidth',3);

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
title('|S_1_1|');
grid on;

%-------------------------------------------------------------------------
% Funcion MATLAB: DibujarRespRedAdaptadora.m
%    Descripcion: Muestra graficamente resultados de la simulacion
%                 para la segunda red adaptadora
%-------------------------------------------------------------------------
close all;
clear all;

f=linspace(log(10e6),log(1e9),500);
f=exp(f);

ZS = 53;
ZL = 193-45*j;

[R_Ideal_2 S11_ideal_2]=resp_red_adaptadora2(f,ZL,ZS,11.36e-12,137.36e-9);
set(semilogx(f/1e6,R_Ideal_2,'r-.'),'LineWidth',3);

hold on;

[R_real_2 S11_real_2]=resp_red_adaptadora2(f,ZL,ZS,10e-12,150e-9);
set(semilogx(f/1e6,R_real_2,'b'),'LineWidth',3);

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
title('Respuesta en potencia (dB)');
grid on;

% En otra grafica el 1-|S11|^2
figure;
PotTx_ideal_2 = 1-abs(S11_ideal_2).^2;
PotTx_real_2  = 1-abs(S11_real_2).^2;

set(semilogx(f/1e6,PotTx_ideal_2,'r-.'),'LineWidth',3);hold on;
set(semilogx(f/1e6,PotTx_real_2,'b'),'LineWidth',3);

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
title('1-|S_1_1|^2');
grid on;

% En otra grafica el S11:
figure;

set(semilogx(f/1e6,abs(S11_ideal_2),'r-.'),'LineWidth',3);hold on;
set(semilogx(f/1e6,abs(S11_real_2),'b'),'LineWidth',3);

legend('IDEAL','REAL');
```

```
xlabel('Frecuencia (MHz)');
title('|S_1_1|');
grid on;


%-------------------------------------------------------------------------
% Funcion MATLAB: resp_red_adaptadora.m
%    Descripcion: Simula la respuesta de la primera red adaptadora para
%             los valores de componentes dados.
%-------------------------------------------------------------------------
function [ G, S11 ] = resp_red_adaptadora( f, ZL, ZS, C1,L1)

V1 = 1;

ZC1 = Z_cond(C1,f);
ZL1 = Z_ind(L1,f);

I1 = V1 ./ ( ZS + ZC1 + paralelo( ZL1, ZL ) );
V0 = I1 .* paralelo( ZL1, ZL );

Pi = (abs(V1)/2).^2 ./ ( real(ZS) );        % Pi es en realidad la potencia
disponible del generador
Po = abs(V0).^2 ./ ( real(ZL) );

G = 10.*log10( Po ./ Pi );

% Coeficiente de reflexion:
Zin = ZC1 + paralelo( ZL1, ZL );

S11 = (Zin - ZS)./(Zin + ZS);


%-------------------------------------------------------------------------
% Funcion MATLAB: resp_red_adaptadora2.m
%    Descripcion: Simula la respuesta de la segunda red adaptadora para
%             los valores de componentes dados.
%-------------------------------------------------------------------------
function [ G, S11 ] = resp_red_adaptadora2( f, ZL, ZS, C1,L1 )
% Red paso bajo
V1 = 1;

ZC1 = Z_cond(C1,f);
ZL1 = Z_ind(L1,f);

I1 = V1 ./ ( ZS + ZL1 + paralelo( ZC1, ZL ) );
V0 = I1 .* paralelo( ZC1, ZL );

Pi = (abs(V1)/2).^2 ./ ( real(ZS) );        % Pi es en realidad la potencia
disponible del generador
Po = abs(V0).^2 ./ ( real(ZL) );

G = 10.*log10( Po ./ Pi );

% Coeficiente de reflexion:
Zin = ZL1 + paralelo( ZC1, ZL );

S11 = (Zin - ZS)./(Zin + ZS);


%-------------------------------------------------------------------------
% Funcion MATLAB: DibujarRespRedAdaptadora_1y2.m
%    Descripcion: Muestra graficamente resultados de la simulacion
%                 para la concatenacion en cascada de las dos redes
%                 adaptadoras
%-------------------------------------------------------------------------
close all;
clear all;

f=linspace(log(10e6),log(1e9),200);
f=exp(f);
```

```
ZS = 53;
ZL = 193-45*j;

[R_Ideal_1 S11_ideal_1]=resp_red_adaptadora(f,ZL,ZS,17.24e-12,168.84e-9);
[R_Ideal_2 S11_ideal_2]=resp_red_adaptadora2(f,ZL,ZS,11.36e-12,137.36e-9);

set(semilogx(f/1e6,R_Ideal_1+R_Ideal_2,'r-.'),'LineWidth',3);

hold on;

[R_real_1 S11_real_1]=resp_red_adaptadora(f,ZL,ZS,18e-12,180e-9);
[R_real_2 S11_real_2]=resp_red_adaptadora2(f,ZL,ZS,10e-12,150e-9);

set(semilogx(f/1e6,R_real_1+R_real_2,'b'),'LineWidth',3);

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
title('Respuesta en potencia (dB)');
grid on;

% En otra grafica el 1-|S11|^2
figure;
PotTx_ideal_1 = 1-abs(S11_ideal_1).^2;
PotTx_real_1  = 1-abs(S11_real_1).^2;

PotTx_ideal_2 = 1-abs(S11_ideal_2).^2;
PotTx_real_2  = 1-abs(S11_real_2).^2;

set(semilogx(f/1e6,PotTx_ideal_1 .* PotTx_ideal_2,'r-.'),'LineWidth',3);hold
on;
set(semilogx(f/1e6,PotTx_real_1 .* PotTx_real_2,'b'),'LineWidth',3);

legend('IDEAL','REAL');
xlabel('Frecuencia (MHz)');
title('(1-|S_1_1|^2)·(1-|S''_1_1|^2)');
grid on;


%--------------------------------------------------------------------------
% Funcion MATLAB: paralelo.m
%    Descripcion: Calcula la impedancia resultado de poner dos en paralelo
%--------------------------------------------------------------------------
function [ Z ] = paralelo( z1, z2 )
    Z = (z1.*z2)./(z1+z2);

%--------------------------------------------------------------------------
% Funcion MATLAB: Z_cond.m
%    Descripcion: Calcula la impedancia compleja de un condensador
%--------------------------------------------------------------------------
function [ Z ] = Z_cond( C, f )
    Z = 1 ./ ( j.*2.*pi.*f.*C );

%--------------------------------------------------------------------------
% Funcion MATLAB: Z_ind.m
%    Descripcion: Calcula la impedancia compleja de una inductancia
%--------------------------------------------------------------------------
function [ Z ] = Z_ind( L, f )
    Z = j.*2.*pi.*f.*L;
```

A4.2. Firmware

```
;-------------------------------------------------------------------------
; FICHERO: RadioSW_Main.asm
; DESCRIPCION: Es el fichero del programa principal del receptor.
;   Aqui es donde se inician los perifericos y se procesan las tramas
;   que llegan del PC.
;-------------------------------------------------------------------------

; Declaraciones estandar para este micro ATMEGA8515
.include "m8515def.inc"

; ---------------------------------------------------------
;       Declaraciones  globales
; ---------------------------------------------------------
; La pila estara al final de la RAM
.equ    PILA            = RAMEND

; Nombre que doy a algunos registros muy usados:
.def    temp            =r16
.def    conta           =r17
.def    temp2           =r18
.def    aux                     =r19

; ---------------------------------------------------------
;  Nombres para los pins interfaces de cada chip
;   externo al microcontrolador
; ---------------------------------------------------------
.include "PinDefs.inc"


; ---------------------------------------------------------
;       Declaraciones de variables en RAM
; ---------------------------------------------------------
            .DSEG

; Variables generales
; -----------------------------------------------
            .ORG                0x60

TRAMA_LONG:                     .byte   1
TRAMA_BUFFER:           .byte   260
MODO_MUESTREO:          .byte   1



;-----------------------------------------------------
;          Vectores de interrupcion
;-----------------------------------------------------
            .CSEG
            .ORG     0x0000
        rjmp    RESET                   ; RESET
        reti                            ; EXT_INT0
        reti                            ; EXT_INT1
        reti                            ; TIM1_CAPT
        reti                            ; TIM1_COMPA
```

```
        reti                        ; TIM1_COMPB
        reti                        ; TIM1_OVF
        reti                        ; TIM0_OVF
        reti                        ; SPI_STC
        reti                        ; USART_RXC
        reti                        ; USART_UDRE
        reti                        ; USART_TXC
        reti                        ; ANA_COMP
        reti                        ; EXT_INT2
        reti                        ; TIM0_COMP
        reti                        ; EE_RDY
        reti                        ; SPM_RDY


; ---------------------------------------------------------
;
;                     RUTINA DE RESET
;
; ---------------------------------------------------------
RESET:
        ;---------------------------------------------
        ; Cargar puntero de pila
        ;---------------------------------------------
        ldi     temp,HIGH ( PILA )
        out     SPH,temp
        ldi     r16,LOW (PILA )
        out     SPL,temp


        ;-------------------------------------------------
        ; Inicializacion de modulos, puertos de E/S,...
        ;-------------------------------------------------

        ; Poner todos los subsistemas en modo POWERDOWN
        ; -----------------------------------------------
        sbi     POWERDOWN_DDR, POWERDOWN
        cbi     POWERDOWN_PORT, POWERDOWN

        ; Iniciar los dos buses de datos:
        ldi     temp, 0x00
        out     USB_DATA_DDR, temp
        ldi     temp, 0xFF
        out     USB_DATA_PORT, temp

        ldi     temp, 0x00
        out     FIFO_DATA_DDR, temp
        ldi     temp, 0xFF
        out     FIFO_DATA_PORT, temp

        ; Iniciar FIFO:
        sbi     FIFO_CTRL_DDR,  FIFO_RESET    ; /RESET = 0
        cbi     FIFO_CTRL_PORT, FIFO_RESET

        cbi     FIFO_CTRL_DDR,  FIFO_EMPTY    ; FIFO_EMPTY = Entrada (con pull-up)
        sbi     FIFO_CTRL_PORT, FIFO_EMPTY

        sbi     FIFO_CTRL_DDR,  FIFO_WR              ; /W=1
        sbi     FIFO_CTRL_PORT, FIFO_WR
```

```
        sbi     FIFO_CTRL_DDR,  FIFO_WR_USB_FF       ; /FIFO_WR_USB_FF=1
        sbi     FIFO_CTRL_PORT, FIFO_WR_USB_FF

        ; Iniciar pins de constrol del chip USB FT8U245
        rcall   USB_INICIAR

        ; Iniciar BUS 3-Wires
        rcall   BUS3W_INICIAR

        ; Iniciar DAC
        rcall   DAC_INICIAR

        ; Iniciar PLL
        rcall   PLL_INICIAR

        ; Iniciar señales de control de ADCs:
        ; ------------------------------------
        ; /CONVSTART = 1
        sbi     ADC_CTRL_DDR,  ADC_CONVSTART
        sbi     ADC_CTRL_PORT, ADC_CONVSTART

        ; /RD_I = 1
        sbi     ADC_CTRL_DDR,  ADC_RD_I
        sbi     ADC_CTRL_PORT, ADC_RD_I

        ; /RD_Q = 1
        sbi     ADC_CTRL_DDR,  ADC_RD_Q
        sbi     ADC_CTRL_PORT, ADC_RD_Q


        ; Sacar a la FIFO del estado de RESET:
        sbi     FIFO_CTRL_PORT, FIFO_RESET

        ; Iniciar LCD
        rcall   LCD_INICIAR


        ; Normalmente, no usar interrupciones
        cli

;=====================================================================
;
;               Mostrar pantalla inicial con la version del SW
;
;=====================================================================

        ldi             ZH, high( STR_PANTALLA_INI_1 * 2 )
        ldi             ZL,  low( STR_PANTALLA_INI_1 * 2 )
        rcall   LCD_ESCRIBE_CADENA_Z_FLASH

        ; Segunda linea:
        ldi             r16, 0x40
        rcall   LCD_SET_CURSOR_POS_R16

        ldi             ZH, high( STR_PANTALLA_INI_2 * 2 )
        ldi             ZL,  low( STR_PANTALLA_INI_2 * 2 )
        rcall   LCD_ESCRIBE_CADENA_Z_FLASH
```

```
        ; Esperar un tiempo y borrar la pantalla:
        ldi             r25, 80
        rcall   RETARDO_R25x25ms

        rcall   LCD_BORRAR


        ; Mostrar mensaje de preparado:
        ; ------------------------------
        ldi             ZH, high( STR_PREPARADO * 2 )
        ldi             ZL,  low( STR_PREPARADO * 2 )
        rcall   LCD_ESCRIBE_CADENA_Z_FLASH

        ; Iniciar variables:
        ; ------------------------------
        ldi             r16, 0x00
        sts             MODO_MUESTREO, r16

        ;-------------------------------------------
        ; Bucle central: Esperar comando por USB
        ;-------------------------------------------
MAIN_LOOP:
        ; Esperar a recibir comando:
        ; ------------------------------
        rcall   USB_RX_TRAMA   ; C=0 si trama recibida OK. Sino, timeout en 32ms C=1
        brcc    mainloop_proc_cmd

        ; No hay trama.
        ; Si estamos en modo de muestreo continuo, ir a su rutina:
        lds             r16, MODO_MUESTREO
        sbrc    r16, 0
        rjmp    CMD_40                  ; MODO_MUESTREO = 0x01
        rjmp    MAIN_LOOP               ; C=1, seguir esperando


mainloop_proc_cmd:
        ; Ejecutar comando segun numero de comando:
        lds             r16, TRAMA_BUFFER     ; primer byte = num. de cmd.

        cpi             r16, 0x10                       ; CMD 0x10
        brne    skip1
        rjmp    CMD_10
skip1:
        cpi             r16, 0x70                       ; CMD 0x70
        brne    skip2
        rjmp    bootloader_responder_70
skip2:
        cpi             r16, 0x20                       ; CMD 0x20
        brne    skip3
        rjmp    CMD_20
skip3:
        cpi             r16, 0x21                       ; CMD 0x21
        brne    skip4
        rjmp    CMD_21
skip4:
        cpi             r16, 0x30                       ; CMD 0x30
        brne    skip5
```

```
        rjmp    CMD_30
skip5:
        cpi             r16, 0x31                       ; CMD 0x31
        brne    skip6
        rjmp    CMD_31
skip6:
        cpi             r16, 0x32                       ; CMD 0x32
        brne    skip7
        rjmp    CMD_32
skip7:
        cpi             r16, 0x33                       ; CMD 0x33
        brne    skip8
        rjmp    CMD_33
skip8:
        cpi             r16, 0x34                       ; CMD 0x34
        brne    skip9
        rjmp    CMD_34
skip9:
        cpi             r16, 0x40                       ; CMD 0x40
        brne    skip10
        rjmp    CMD_40
skip10:
        cpi             r16, 0x41                       ; CMD 0x41
        brne    skip11
        rjmp    CMD_41
skip11:

        rjmp    MAIN_LOOP



;-----------------------------------------------------
;
;   COMANDOS ENTRE PC<->MICROCONTROLADOR
; ============================================
;
;-----------------------------------------------------

;--------------------------------------------------------------------------------
;   Identificacion del firmware:
;  ---------------------------------
;   [tx] <FLAG INI> <LON> 0x10 <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0x90 <Cadena de identificacion> 0x00 <FLAG FIN>
;--------------------------------------------------------------------------------
CMD_10:
        ; Cargar cadena de identificacion en BUFFER
        ldi             ZH, high( STR_ID * 2)
        ldi             ZL,  low( STR_ID * 2)
        ldi             XH, high (TRAMA_BUFFER)
        ldi             XL, low  (TRAMA_BUFFER)

        ldi             r16, 0x90
        st              X+, r16

        ldi             r18, 1

cmd10_lp:
        lpm
```

```
        adiw    ZL, 1
        inc             r18

        st              X+, r0

        tst             r0
        brne    cmd10_lp

        ; Enviar respuesta
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    MAIN_LOOP



;-------------------------------------------------------------------------
;   Borrar LCD
;  ---------------------------------
;   [tx] <FLAG INI> <LON> 0x20 <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0xA0 <FLAG FIN>
;-------------------------------------------------------------------------
CMD_20:
        rcall   LCD_BORRAR

        ; Enviar respuesta
        ldi     XH, high (TRAMA_BUFFER)
        ldi     XL, low  (TRAMA_BUFFER)

        ldi             r16, 0xA0
        st              X+, r16
        ldi             r18, 1
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    MAIN_LOOP

;-------------------------------------------------------------------------
;   Escribir en LCD
;  ---------------------------------
;   [tx] <FLAG INI> <LON> 0x21 <Columna> <Fila> <Texto> 0x00 <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0xA1 <FLAG FIN>
;-------------------------------------------------------------------------
CMD_21:
        ; Formar posicion del LCD:
        lds             r16, TRAMA_BUFFER+1
        lds             r17, TRAMA_BUFFER+2

        ; Mover cursor:
        sbrc    r17, 0          ; 2° fila = 0x40
        ori             r16, 0x40

        rcall   LCD_SET_CURSOR_POS_R16

        ; Mostrar texto:
        ldi             XH, high (TRAMA_BUFFER+3)
        ldi             XL, low  (TRAMA_BUFFER+3)

cmd21_lp:
        ld              r16, X+
```

```
        tst             r16                             ; La cadena termina en 0x00
        breq    cmd21_lp_fin

        rcall   LCD_SEND_R16_DATA
        rjmp    cmd21_lp

cmd21_lp_fin:

        ; Enviar respuesta
        ldi             XH, high (TRAMA_BUFFER)
        ldi             XL, low  (TRAMA_BUFFER)

        ldi             r16, 0xA1
        st              X+, r16
        ldi             r18, 1
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    MAIN_LOOP


;-------------------------------------------------------------------------
;   Cambiar estado de PowerDown
;  ---------------------------------
;   [tx] <FLAG INI> <LON> 0x30 <00/01> <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0xB0 <FLAG FIN>
;-------------------------------------------------------------------------
CMD_30:
        ; Leer "PowerDown" deseado:
        lds             r16, TRAMA_BUFFER+1

        tst             r16
        breq    cmd30_no_pwdown

        ; Si, poner en modo PowerDown
cmd30_si_pwdown:
        sbi             POWERDOWN_DDR,  POWERDOWN     ; Para asegurar
        cbi             POWERDOWN_PORT, POWERDOWN
        rjmp    cmd30_resp

        ; No, quitar modo PowerDown
cmd30_no_pwdown:
        sbi             POWERDOWN_DDR,  POWERDOWN     ; Para asegurar
        sbi             POWERDOWN_PORT, POWERDOWN


cmd30_resp:
        ; Enviar respuesta
        ldi     XH, high (TRAMA_BUFFER)
        ldi     XL, low  (TRAMA_BUFFER)

        ldi             r16, 0xB0
        st              X+, r16
        ldi             r18, 1
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    MAIN_LOOP
```

```
;-------------------------------------------------------------------------
;   Leer estado de PowerDown
;  ---------------------------------
;   [tx] <FLAG INI> <LON> 0x31 <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0xB1 <00/01> <FLAG FIN>
;-------------------------------------------------------------------------
CMD_31:
       ; Leer estado "PowerDown" deseado:

       ldi     XH, high (TRAMA_BUFFER)
       ldi     XL, low  (TRAMA_BUFFER)

       ldi             r16, 0xB1
       st              X+, r16

       ; r16 = 0x01 si pin de /PWDOWN = 0
       ldi             r16, 0x00
       sbis    POWERDOWN_PIN,  POWERDOWN
       ldi             r16, 0x01

       st              X+, r16

       ; Enviar:
       ldi             r18, 2
       rcall   USB_TX_TRAMA_LONG_R18

       rjmp    MAIN_LOOP


;-------------------------------------------------------------------------
;   Cambiar un DAC
;  ---------------------------------
;   [tx] <FLAG INI> <LON> 0x32 <# de DAC> <Valor> <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0xB2 <FLAG FIN>
;-------------------------------------------------------------------------
CMD_32:
       ; Leer el DAC
       lds             r16, TRAMA_BUFFER+1
       lds             r17, TRAMA_BUFFER+2

       rcall   DAC_CAMBIAR           ; r16=#DAC, r17=valor

       ; Enviar respuesta
       ldi     XH, high (TRAMA_BUFFER)
       ldi     XL, low  (TRAMA_BUFFER)

       ldi             r16, 0xB2
       st              X+, r16
       ldi             r18, 1
       rcall   USB_TX_TRAMA_LONG_R18

       rjmp    MAIN_LOOP

;-------------------------------------------------------------------------
;   Enviar configuracion al PLL
;  ---------------------------------
;   [tx] <FLAG INI> <LON> 0x33 <BYTE2> <BYTE1> <BYTE0> <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0xB3 <FLAG FIN>
```

```
;-------------------------------------------------------------------------
CMD_33:
       ; Enviar dato al PLL:
       lds             r3, TRAMA_BUFFER+1
       lds             r2, TRAMA_BUFFER+2
       lds             r1, TRAMA_BUFFER+3

       rcall   PLL_ENVIAR

       ; Enviar respuesta
       ldi             XH, high (TRAMA_BUFFER)
       ldi             XL, low  (TRAMA_BUFFER)

       ldi             r16, 0xB3
       st              X+, r16
       ldi             r18, 1
       rcall   USB_TX_TRAMA_LONG_R18

       rjmp    MAIN_LOOP


;-------------------------------------------------------------------------
;   Muestrear salida MUXOUT del PLL y enviarla
;  --------------------------------
;   [tx] <FLAG INI> <LON> 0x34 <FLAG FIN>
;   [rx] <FLAG INI> <LON> 0xB4 255x<BYTEs> <FLAG FIN>
;-------------------------------------------------------------------------
CMD_34:
       ; Formar respuesta
       ldi     XH, high (TRAMA_BUFFER)
       ldi     XL, low  (TRAMA_BUFFER)

       ldi             r16, 0xB4
       st              X+, r16

       ; Y ahora muestrear MUXOUT cada 2 us:
       ; ------------------------------------
       ldi             r18, 255
cmd34_lp:
       ldi             r16, 0x01             ; Acumulador de bits/muestras
cmd34_lp2:
       clc
       sbic    PLLMUX_PIN, PLLMUX
       sec

       rol             r16
       brcs    cmd34_sig

       nop
       nop
       nop
       nop
       nop

       rjmp    cmd34_lp2

cmd34_sig:
       st              X+, r16
```

```
        dec             r18
        brne    cmd34_lp

        ; Enviar
        ldi             r18, 255
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    MAIN_LOOP



; ---------------------------------------------------------
;   Iniciar modo de muestreo continuo
;  ------------------------------------
;      [tx] <FLAG INI> <LON> 0x40 <FLAG FIN>
;   Nx[rx] <FLAG INI> <0x81>0xC0 128x<Bytes> <FLAG FIN>
; ---------------------------------------------------------
CMD_40:
        ; Señalar indicador:
        ldi             r16, 0x01
        sts             MODO_MUESTREO, r16

; ---------------------------------------------------------
;    Bucle de envio de tramas de muestras I/Q
;  Se ejecuta indefinidamente hasta que llegue un
;   comando 0x41
; ---------------------------------------------------------
        ; Estos registros tendran unos valores de forma fija,
        ;   para poder acceder rapidamente a estas constantes.
        ldi             r16, 0x00
        mov             r5, r16                 ; r5 = 0x00
        ldi             r16, 0xFF
        mov             r6, r16                 ; r6 = 0xFF
        ldi             r16, 0b01101101
        mov             r7, r16                 ; r7 = PORTD para "/ADC_CONST"=0
        ldi             r16, 0b01011101
        mov             r8, r16                 ; r8 = PORTD para "/ADC_RD_I"=0
        ldi             r16, 0b00111101
        mov             r9, r16                 ; r9 = PORTD para "/ADC_RD_Q"=0
        ldi             r16, 0b01111101
        mov             r10, r16                ; r10 = PORTD sin ninguna señal activa
        ldi             r16, 0b00111111
        mov             r11, r16                ; r11 = PORTB sin ninguna señal activa
        ldi             r16, 0b00101111
        mov             r12, r16                ; r12 = PORTB para "/FIFO_WR"=0
        ldi             r16, 0x96
        mov             r13, r16                ; r13 = 0x96 (Flag inicio)
        ldi             r16, 0x69
        mov             r14, r16                ; r14 = 0x69 (Flag final)
        ldi             r16, 0x81
        mov             r15, r16                ; r15 = 0x81 (LON)


        out             FIFO_DATA_DDR,  r6      ; FIFO_DATA como salida

        out             USB_DATA_DDR, r5        ; USB_BUS como HI-Z
        out             USB_DATA_PORT, r5
```

```
        ; Enviar a la FIFO la cabecera de la trama:
cmd40_forma_cabecera: ; Hasta el bucle son 14 ciclos
        out             FIFO_DATA_PORT, r13   ; <FLAG INICIO>
        out             FIFO_CTRL_PORT, r12   ; /FIFO_WR=0
        out             FIFO_CTRL_PORT, r11   ; /FIFO_WR=1

        out             FIFO_DATA_PORT, r15   ; <LON=0x81>
        out             FIFO_CTRL_PORT, r12   ; /FIFO_WR=0
        out             FIFO_CTRL_PORT, r11   ; /FIFO_WR=1

        ldi             r16, 0xC0
        out             FIFO_DATA_PORT, r16   ; <CMD=0xE0>
        out             FIFO_CTRL_PORT, r12   ; /FIFO_WR=0
        out             FIFO_CTRL_PORT, r11   ; /FIFO_WR=1

        ; FIFO_DATA como HI-ZI
        out             FIFO_DATA_DDR,  r5
        out             FIFO_DATA_PORT, r5

        ; Este bucle se debe ejecutar exactamente a la frecuencia
        ;  de muestreo: 250KHz -> Cada 40 ciclos a 10 Mhz

        ldi             r18, 64        ; Se ejecuta 64 veces -> 128 muestras (1 de cada
canal:I y Q)
cmd40_bucle_muestras:
        out             PORTD, r7                     ; /ADC_CONVST = 0
        nop             ; 2 ciclos
        out             PORTD, r10                    ; /ADC_CONVST = 1

        nop             ; 4 ciclos
        nop             ; 5 ciclos
        nop             ; 6 ciclos
        nop             ; 7 ciclos
        nop             ; 8 ciclos


        ; Enviar muestra "I"
        out             PORTD, r8                     ; /ADC_RD_I = 0
        out             FIFO_CTRL_PORT, r12   ; /FIFO_WR = 0
        out             FIFO_CTRL_PORT, r11   ; /FIFO_WR = 1
        out             PORTD, r10                    ; /ADC_RD_I = 1


        ; Enviar muestra "Q"
        out             PORTD, r9                     ; /ADC_RD_Q = 0
        out             FIFO_CTRL_PORT, r12   ; /FIFO_WR = 0
        out             FIFO_CTRL_PORT, r11   ; /FIFO_WR = 1
        out             PORTD, r10                    ; /ADC_RD_Q = 1

        ; Ahora "sobran" 21 ciclos hasta siguiente muestra:
        ; ------------------------------------------------
        ;  Si es la ultima muestra, no realizar los "nops" de relleno:
        dec             r18
        breq    cmd40_env_flag_final

        nop
        nop
```

```
        nop
        nop
        nop

        nop
        nop
        nop
        nop
        nop

        nop
        nop
        nop
        nop
        nop

        nop
        nop
        nop
        nop
        nop


        rjmp    cmd40_bucle_muestras


cmd40_env_flag_final:
        ; Enviar Flag de final
        out             FIFO_DATA_DDR,  r6    ; FIFO_DATA como salida
        out             FIFO_DATA_PORT, r14   ; <FLAG FINAL>
        out             FIFO_CTRL_PORT, r12   ; /FIFO_WR=0
        nop             ; Este NOP es de relleno para alcanzar los 2.5us en total
        out             FIFO_CTRL_PORT, r11   ; /FIFO_WR=1

        ; Ya se ha enviado una trama completa de muestras.
        ;  Si no se ha recibido nada por USB, seguir:
        sbic    USB_PIN, USB_PIN_RXF
        rjmp    cmd40_forma_cabecera  ; Volver a formar cabecera de trama

        rjmp    cmd40_procesatrama; Si /RXF = 0, parar de enviar muestras para procesar
trama entrante:


cmd40_procesatrama:
        ; Esperar a que la FIFO este vacia para poder hacer que el
        ;  chip USB saque su contenido al bus sin colisiones:
        sbic    FIFO_CTRL_PIN, FIFO_EMPTY
        rjmp    cmd40_procesatrama    ; /FIFO_EMPTY = 1, esperar

        ; Ahora procesar trama desde el bucle central, que tras
        ;  ello volvera a esta rutina.
        rjmp    MAIN_LOOP


;-------------------------------------------------------------------------
;   Finalizar modo muestreo:
;  --------------------------------
;   [tx] <FLAG INI> <LON> 0x41 <FLAG FIN>
```

```
;   [rx] <FLAG INI> <LON> 0xC1 <FLAG FIN>
;-------------------------------------------------------------------------
CMD_41:
        ldi             r16, 0x00
        sts             MODO_MUESTREO, r16

        ; Enviar respuesta
        ldi     XH, high (TRAMA_BUFFER)
        ldi     XL, low  (TRAMA_BUFFER)

        ldi             r16, 0xC1
        st              X+, r16
        ldi             r18, 1
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    MAIN_LOOP

fin:    rjmp    fin


; ---------------------------------------------------------
;   Constantes, tablas y cadenas que irán en FLASH:
; ---------------------------------------------------------
STR_ID:
        .db     "RadioSW v1.0 (JLBC)-BUILD058",0,0
STR_PANTALLA_INI_1:
        .db     "Radio SW v1.0",0
STR_PANTALLA_INI_2:
        .db     "J.L.Blanco 2004",0

STR_PREPARADO:
        .db "-- Preparado --",0

; ---------------------------------------------------------
;       Inclusión de modulos para cada periferico
; ---------------------------------------------------------

;  Rutinas para manejo del bus 3-Wire (DAC y PLL)
; ================================================
.include "BUS3W.asm"

;  Rutinas para manejo del DAC
; =====================================
.include "DAC.asm"

;  Rutinas para manejo del PLL
; =====================================
.include "PLL_ADF4111.ASM"

;  El BOOT LOADER que permite actualizar
;    el firmware. Desde alli se incluyen
;    las rutinas de LCD,USB y Retardos
; ========================================
.include "BootLoader.asm"
```

```
;-------------------------------------------------------------------------
; FICHERO: Bootloader.asm
; DESCRIPCION:
;  Este programa se ejecutará siempre al
;    reiniciar el sistema, y esperará una
;    conexion de actualizacion por dos
;    segundos, tras lo cual ejecutara el
;    programa principal.
;
; ----------------------------------------

; Comenzar en la zona de 1KW de BOOTLOADER
.org   LARGEBOOTSTART

BOOTLOADER:
       cli

       ;-----------------------------------------
       ; Cargar puntero de pila
       ;-----------------------------------------
       ldi     temp,HIGH ( PILA )
       out     SPH,temp
       ldi     r16,LOW (PILA )
       out     SPL,temp

       ; Inicializar los perifericos basicos
       ;   que se usarán aqui: LCD y USB.
       ; Poner todos los subsistemas en modo POWERDOWN
       ; -------------------------------------------------
       sbi     POWERDOWN_DDR, POWERDOWN
       cbi     POWERDOWN_PORT, POWERDOWN

       ; Iniciar los dos buses de datos:
       ldi     temp, 0x00
       out     USB_DATA_DDR, temp
       ldi     temp, 0xFF
       out     USB_DATA_PORT, temp

       ldi     temp, 0x00
       out     FIFO_DATA_DDR, temp
       ldi     temp, 0xFF
       out     FIFO_DATA_PORT, temp

       ; Iniciar FIFO:
       sbi     FIFO_CTRL_DDR,  FIFO_RESET    ; /RESET = 0
       cbi     FIFO_CTRL_PORT, FIFO_RESET

       cbi     FIFO_CTRL_DDR,  FIFO_EMPTY    ; FIFO_EMPTY = Entrada (con pull-up)
       sbi     FIFO_CTRL_PORT, FIFO_EMPTY

       sbi     FIFO_CTRL_DDR,  FIFO_WR              ; /W=1
       sbi     FIFO_CTRL_PORT, FIFO_WR

       sbi     FIFO_CTRL_DDR,  FIFO_WR_USB_FF       ; /FIFO_WR_USB_FF=1
       sbi     FIFO_CTRL_PORT, FIFO_WR_USB_FF
```

```
        ; Iniciar pins de constrol del chip USB FT8U245
        rcall   USB_INICIAR

        ; Sacar a la FIFO del estado de RESET:
        sbi             FIFO_CTRL_PORT, FIFO_RESET

        ; Iniciar señales de control de ADCs:
        ; ------------------------------------
        ; /CONVSTART = 1
        sbi     ADC_CTRL_DDR,  ADC_CONVSTART
        sbi     ADC_CTRL_PORT, ADC_CONVSTART

        ; /RD_I = 1
        sbi     ADC_CTRL_DDR,  ADC_RD_I
        sbi     ADC_CTRL_PORT, ADC_RD_I

        ; /RD_Q = 1
        sbi     ADC_CTRL_DDR,  ADC_RD_Q
        sbi     ADC_CTRL_PORT, ADC_RD_Q


        ; Iniciar LCD
        rcall   LCD_INICIAR


        ; Segunda linea:
        rcall   LCD_BORRAR

        ldi             ZH, high( STR_BOOTLDR_TXT * 2 )
        ldi             ZL,  low( STR_BOOTLDR_TXT * 2 )
        rcall   LCD_ESCRIBE_CADENA_Z_FLASH

        ldi             r16, 0x40
        rcall   LCD_SET_CURSOR_POS_R16

        ; Poner un "." cada 125ms ( 125ms x 16 = 2 seg )
        ldi             conta, 16
bootloader_wait_lp:

        ; Enviar un "." al LCD:
        ldi             r16, '.'
        rcall   LCD_SEND_R16_DATA

        ; Intentar recibir trama desde el USB
        rcall   USB_RX_TRAMA   ; C=0 si trama recibida OK. Sino, timeout en 32ms C=1
        brcc    bootloader_trama_recibida
bootloader_byteusb_seguir:

        ; Esperar 125ms - 32ms (de la rutina de recepcion USB)
        ldi             r25, 4
        rcall   RETARDO_R25x25ms

        ; Limite de espera?
        dec             conta
        brne    bootloader_wait_lp

        ; Ir al programa principal ya:
        ; --------------------------------
```

```
bootloader_ir_prog_ppal:
	ldi		r16, 0x40
	rcall	LCD_SET_CURSOR_POS_R16

	; Mostrar un breve mensaje avisando de la ejecucion:
	ldi		ZH, high( STR_BOOTLDR_TXT2 * 2 )
	ldi		ZL,  low( STR_BOOTLDR_TXT2 * 2 )
	rcall	LCD_ESCRIBE_CADENA_Z_FLASH

	ldi		r25, 16
	rcall	RETARDO_R25x25ms

	rjmp	RESET


; Se ha recibido una trama.
; ------------------------------------------------------------
bootloader_trama_recibida:
	; Es el comando 0x70 "Iniciar modo BootLoader" ?
	lds		r16, TRAMA_BUFFER
	cpi		r16, 0x70
	brne	bootloader_byteusb_seguir

bootloader_responder_70:
	; Si:Responder e iniciar el modo de BootLoader.
	ldi		r16, 0xF0
	sts		TRAMA_BUFFER, r16
	ldi		r18, 1
	rcall	USB_TX_TRAMA_LONG_R18

	; Mostrar mensaje y entrar en bucle principal del BootLoader.
	rcall	LCD_BORRAR

	ldi		ZH, high( STR_BOOTLDR_TXT3 * 2 )
	ldi		ZL,  low( STR_BOOTLDR_TXT3 * 2 )
	rcall	LCD_ESCRIBE_CADENA_Z_FLASH

; ------------------------------------------------------------
;		BUCLE PRINCIPAL DEL PROGRAMA DE BOOTLOADER
; ------------------------------------------------------------
BOOTLOADER_LOOP:
	; Esperar una trama por USB:
	rcall	USB_RX_TRAMA	; C=0 si trama recibida OK. Sino, timeout en 32ms C=1
	brcs	BOOTLOADER_LOOP

	ldi		r16, 0x40
	rcall	LCD_SET_CURSOR_POS_R16

	; Ejecutar segun comando:
	lds		r16, TRAMA_BUFFER

	; 0x70 = Entrar en modo BootLoader
	cpi		r16, 0x70
	breq	bootloader_responder_70

	; 0x71 = Reiniciar en el programa principal:
	cpi		r16, 0x71
	breq	bootloader_responder_71
```

```
        ; 0x72 = Leer una pagina de flash
        cpi             r16, 0x72
        breq    go_bootloader_responder_72

        ; 0x73 = Escribir una pagina de flash
        cpi             r16, 0x73
        breq    go_bootloader_responder_73

        rjmp    BOOTLOADER_LOOP


go_bootloader_responder_72:
        rjmp    bootloader_responder_72
go_bootloader_responder_73:
        rjmp    bootloader_responder_73


; ------------------------------------------------------------
; Comando 0x71
;   Reiniciar en el programa principal
; ------------------------------------------------------------
bootloader_responder_71:
        ldi             r16, 0xF1
        sts             TRAMA_BUFFER, r16
        ldi             r18, 1
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    RESET


; ------------------------------------------------------------
; Comando 0x72
;   Leer una pagina de flash
; ------------------------------------------------------------
bootloader_responder_72:
        ; Poner en el LCD lo que se está haciendo:
        ldi             ZH, high( STR_BOOTLDR_TXT4 * 2 )
        ldi             ZL,  low( STR_BOOTLDR_TXT4 * 2 )
        rcall   LCD_ESCRIBE_CADENA_Z_FLASH
        lds             r16, TRAMA_BUFFER+1    ; Y la direccion en hexadecimal:
        rcall   LCD_ESCRIBE_R16_HEX
        lds             r16, TRAMA_BUFFER+2
        rcall   LCD_ESCRIBE_R16_HEX

        ; Cargar puntero a flash:
        lds             ZH, TRAMA_BUFFER+1
        lds             ZL, TRAMA_BUFFER+2

        ; Puntero a datos de salida:
        ldi             XH, high(TRAMA_BUFFER)
        ldi             XL,  low(TRAMA_BUFFER)

        ; Byte de respuesta al comando 0x72:
        ldi             r16, 0xF2
        st              X+, r16

        ; Leer la pagina de flash:
```

```
        ldi             r18, 64
bootldr_rdflsh_lp:
        lpm             r0, Z+
        st              X+, r0

        dec             r18
        brne    bootldr_rdflsh_lp

        ; Enviar la trama al PC:
        ldi             r18, 65
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    BOOTLOADER_LOOP


; ------------------------------------------------------------
; Comando 0x73
;   Grabar una pagina de flash
; ------------------------------------------------------------
bootloader_responder_73:
        ; Poner en el LCD lo que se está haciendo:
        ldi             ZH, high( STR_BOOTLDR_TXT5 * 2 )
        ldi             ZL,  low( STR_BOOTLDR_TXT5 * 2 )
        rcall   LCD_ESCRIBE_CADENA_Z_FLASH
        lds             r16, TRAMA_BUFFER+1   ; Y la direccion en hexadecimal:
        rcall   LCD_ESCRIBE_R16_HEX
        lds             r16, TRAMA_BUFFER+2
        rcall   LCD_ESCRIBE_R16_HEX

        ; Cargar puntero a flash:
        lds             ZH, TRAMA_BUFFER+1
        lds             ZL, TRAMA_BUFFER+2

        ; Puntero a datos:
        ldi             XH, high(TRAMA_BUFFER+3)
        ldi             XL,  low(TRAMA_BUFFER+3)

        ; Grabar
        rcall   FLASH_PROG            ; Z=Dir Flash, X=Dir RAM

        ; Enviar la trama de respuesta al PC:
        ldi             r16, 0xF3
        sts             TRAMA_BUFFER, r16
        ldi             r18, 1
        rcall   USB_TX_TRAMA_LONG_R18

        rjmp    BOOTLOADER_LOOP


STR_BOOTLDR_TXT:        .db     "== BootLoader ==",0,0
STR_BOOTLDR_TXT2:       .db     "   Iniciando... ",0,0
STR_BOOTLDR_TXT3:       .db     " Modo BootLoader",0,0
STR_BOOTLDR_TXT4:       .db     "--> flash 0x",0,0
STR_BOOTLDR_TXT5:       .db     "<-- flash 0x",0,0


;  Rutinas para manejo del LCD
; ====================================
```

```
.include "LCD.asm"

;  Rutinas para manejo del chip USB
; ===================================
.include "FT8U245.asm"

;  Rutinas de retardos
; ===================================
.include "Retardos.asm"

;  Rutina para grabar la flash
;   desde el mismo microcontrolador
; ===================================
.include "FlashProg.asm"
```

```
;-----------------------------------------------------
; FICHERO: Retardos.asm
; DESCRIPCION: Implementa distintas funciones para
;               generar retardos de distinta longitud
;
; NOTA: Calibrado para funcionar a 10 MHz
;-----------------------------------------------------

;-----------------------------------------------------
;  RETARDO_R25_MAX100us  (r25: 1,2,...,255,0)
;
;-----------------------------------------------------
RETARDO_R25_MAX100us:
        dec             r25
        nop
        brne    RETARDO_R25_MAX100us
        ret

;-----------------------------------------------------
;  RETARDO_R25_MAX25ms   (r25: 1,2,...,255,0)
;
;-----------------------------------------------------
RETARDO_R25_MAX25ms:
        push    r25
        ldi             r25,0
        rcall   RETARDO_R25_MAX100us
        pop             r25

        dec             r25
        brne    RETARDO_R25_MAX25ms
        ret

;-----------------------------------------------------
;  RETARDO_R25x25ms
;
;    Retarda r25 veces 25 ms
;-----------------------------------------------------
RETARDO_R25x25ms:
        push    r25
```

```
ret_25x_bucl:
       push    r25
       ldi     r25,0
       rcall   RETARDO_R25_MAX25ms
       pop     r25
       dec     r25
       brne    ret_25x_bucl

       pop     r25
       ret
```

```
;-----------------------------------------------------
; FICHERO: BUS3W.ASM
; DESCRIPCION:
;   Proporciona una rutina para enviar una palabra
;    de un numero variable de bits por un bus de
;    3 hilos sincrono usado en multitud de chips
;    en el mercado (PLLs,ADCs,DACs,...)
;-----------------------------------------------------

;-----------------------------------------------------
; BUS3W_INICIAR
;
;  Debe llamarse al iniciar la ejecucion
;-----------------------------------------------------
BUS3W_INICIAR:
       ; 3W_SCLK = 0
       sbi            BUS3W_DDR,  BUS3W_SCLK
       cbi            BUS3W_PORT, BUS3W_SCLK

       ; 3W_DOUT = 0
       sbi            BUS3W_DDR,  BUS3W_DOUT
       cbi            BUS3W_PORT, BUS3W_DOUT

       ret


;-----------------------------------------------------------
;  BUS3W_ENVIAR
;
;   r18               : Numero de bits
;   r3:r2:r1:r0 : Datos.
;                            MSB=bit 7 de r3, LSB=bit 0 de r0
;
;  Se envia primero el MSB.
;  NOTA: Hay que dar los datos alineados al MSB de r3!!
;-----------------------------------------------------------
BUS3W_ENVIAR:
       ; Asegurar que 3W_SCLK = 0
       sbi            BUS3W_DDR,  BUS3W_SCLK
       cbi            BUS3W_PORT, BUS3W_SCLK

       ; Asegurar 3W_DOUT = salida
       sbi            BUS3W_DDR,  BUS3W_DOUT

       ; Bucle de envio de bits:
       ; -------------------------------
```

```
bus3wenv_lp:
        ; Sacar siguiente bit:
        cbi             BUS3W_PORT, BUS3W_DOUT
        sbrc    r3, 7
        sbi             BUS3W_PORT, BUS3W_DOUT

        ; Desplazar bits en registros:
        lsl             r0
        rol             r1
        rol             r2
        rol             r3


        ; Dar un pulso a SCLK:
        sbi             BUS3W_PORT, BUS3W_SCLK
        nop
        nop
        cbi             BUS3W_PORT, BUS3W_SCLK

        ; Seguir si quedan bits:
        dec             r18
        brne    bus3wenv_lp

        ret
```

```
;-----------------------------------------------------
; FICHERO: DAC.ASM
; DESCRIPCION: Permite manejar los conversores DAC.
;-----------------------------------------------------

;-----------------------------------------------------
; Inicializa los puertos
;-----------------------------------------------------
DAC_INICIAR:
        ; SYNC = /CS = 1
        sbi     BUS3W_DAC_SYNC_DDR,  BUS3W_DAC_SYNC
        sbi     BUS3W_DAC_SYNC_PORT, BUS3W_DAC_SYNC

        ret


;-----------------------------------------------------
;  Cambia el DAC numero "r16", al valor "r17"
;-----------------------------------------------------
DAC_CAMBIAR:
        ; Formar comando en r21:r20:r19:r18
        ldi             r21, 0xC0
        andi    r16, 0x03
        swap    r16
        or              r21, r16

        mov             r16, r17
        swap    r16
        andi    r16, 0x0F
        or              r21, r16

        mov             r20, r17
```

```
        andi    r20, 0x0F
        swap    r20


        ; Mover a r3:r2:r1:r0
        mov             r3, r21
        mov             r2, r20
        mov             r1, r19
        mov             r0, r18

        ; Dirigir el comando al DAC:
        cbi     BUS3W_DAC_SYNC_PORT, BUS3W_DAC_SYNC

        ; Enviar
        ldi             r18, 12                 ; Son 12 bits
        rcall   BUS3W_ENVIAR   ; r18: #de bits; datos en r3:r2:r1:r0

        ; Quitar señal:
        sbi     BUS3W_DAC_SYNC_PORT, BUS3W_DAC_SYNC

        ret
```

```
;-------------------------------------------
; FICHERO: FlashProg.asm
; DESCRIPCION: Implementacion de una rutina
;    que permite la autoprogramación de la
;    memoria flash de programa del micro.
; ------------------------------------------

; ------------------------------------------
;       FLASH_PROG
;
;  Graba una pagina de 64 bytes en la direccion
;   de flash indicada por ZH:L (en bytes).
;  Los datos los coge de RAM, de XH:L
; ------------------------------------------
FLASH_PROG:
        ; Y = Z
        mov             YH, ZH
        mov             YL, ZL

        ; Borrar pagina de flash:
        ; -------------------------
        ldi             r17, 0x03
        rcall   HAZ_SPM_R17

        ; Re-enable the RWW section
        ; ----------------------------
        ldi     r17, (1<<RWWSRE) | (1<<SPMEN)
        rcall   HAZ_SPM_R17

        ; Cargar datos en buffer temporal:
        ; ---------------------------------
        ldi             r18, 32         ; 32 words
flashprog_lp1:
        ; Cargar palabra:
        ld              r0, X+
```

```
        ld              r1, X+

        ldi             r17, 0x01
        rcall   HAZ_SPM_R17

        ; Incrementar direccion
        adiw    ZL, 2

        dec             r18
        brne    flashprog_lp1


        ; Realizar grabacion:
        ; ---------------------------------
        mov             ZH, YH
        mov             ZL, YL

        ldi             r17, 0x05
        rcall   HAZ_SPM_R17


        ; Esperar final de grabacion:
flashprog_lp2:
        in              r16, SPMCR
        sbrc    r16, SPMEN
        rjmp    flashprog_lp2

        ; Re-enable the RWW section:
        ;  Con esto se permite volver a LEER la flash
        ; ----------------------------------------------------
        ldi     r17, (1<<RWWSRE) | (1<<SPMEN)
        rcall   HAZ_SPM_R17

        ret



HAZ_SPM_R17:
        ; Ha terminado un SPM anterior?
        in              r16, SPMCR
        sbrc    r16, SPMEN
        rjmp    HAZ_SPM_R17


        ; SPM timed sequence
        out     SPMCR, r17
        spm

        ret
```

```
;-------------------------------------------------------------------------
; FICHERO: FT8U245.asm
; DESCRIPCION: Incluye funciones para manejar la recepcion y envio de
;   datos desde y hacia el PC a traves del interfaz USB. Ademas se
;   implementan funciones para manejar directamente tramas completas.
;-------------------------------------------------------------------------
```

```
;-------------------------------------------------------------------------
;  Inicia los pins de control del chip USB
;-------------------------------------------------------------------------
USB_INICIAR:
        ; /RD: Salida, 1
        sbi     USB_PORT, USB_PIN_RD
        sbi     USB_DDR,  USB_PIN_RD
        ; WR: Entrada, ya que se pone desde un FlipFlop externo
        cbi     USB_PORT, USB_PIN_WR
        cbi     USB_DDR,  USB_PIN_WR
        ; /TXE: Entrada con pull-up
        sbi     USB_PORT, USB_PIN_TXE
        cbi     USB_DDR,  USB_PIN_TXE
        ; /RFX: Entrada con pull-up
        sbi     USB_PORT, USB_PIN_RXF
        cbi     USB_DDR,  USB_PIN_RXF

        ret


;-------------------------------------------------------------------------
;  Recibe un byte (en r16)
;
;   Carry = 0 si todo OK
;   Carry = 1 si Timeout esperando
;-------------------------------------------------------------------------
USB_RX_BYTE:
        ; TIMEOUT:
        clr     ZH
        clr     ZL
usbrx_wait:
        adiw    ZL, 1
        tst     ZL
        brne    usbrx_tst_rxf
        tst     ZH
        brne    usbrx_tst_rxf

        ; Ha dado 0x10000 ciclos esperando
        sec
        ret

usbrx_tst_rxf:
        ; Asegurar que RXF = 0
        sbic    USB_PIN, USB_PIN_RXF
        rjmp    usbrx_wait

        ; Entrada:
        ldi     r16, 0x00
        out     USB_DATA_PORT,r16
        out     USB_DATA_DDR,r16

        nop
        nop
        nop
        nop

        ; Ciclo de lectura:
        cbi     USB_PORT, USB_PIN_RD
```

```
        nop
        nop
        nop
        nop

        nop
        nop
        nop
        nop

        in      r16, USB_DATA_PIN

        sbi     USB_PORT, USB_PIN_RD

        clc
        ret


;-----------------------------------------------------
; USB_TX_BYTE
;
; Escribir un byte al chip USB, a traves de la FIFO:
;  Dato de entrada en r16
;-----------------------------------------------------
USB_TX_BYTE:
        push    r17

        ; /WR de la FIFO = 0
        sbi FIFO_CTRL_DDR, FIFO_WR
        cbi     FIFO_CTRL_PORT, FIFO_WR

        ; Poner puerto de la FIFO como salida y sacar dato
        ser             r17
        out             FIFO_DATA_DDR, r17
        out             FIFO_DATA_PORT, r16

        nop
        nop

        ; /WR de la FIFO = 1
        sbi     FIFO_CTRL_PORT, FIFO_WR

        nop

        ; Volver a poner puerto como HI-Z
        clr             r17
        out             FIFO_DATA_DDR,  r17
        out             FIFO_DATA_PORT, r17

        ; El sistema de lógica externa se encargará
        ;   de sacar este byte al chip USB automaticamente.
        pop             r17
        ret


;-----------------------------------------------------
; USB_TX_TRAMA_LONG_R18
```

```
;
;  Envia una trama con "r18" bytes desde "TRAMA_BUFFER"
;  Incluye automaticamente el flag inicial,
;    campo de longitud y flag final.
;-----------------------------------------------------
USB_TX_TRAMA_LONG_R18:
       ; <FLAG.INI>
       ldi            r16, 0x96
       rcall  USB_TX_BYTE

       ; <LON>
       mov            r16, r18
       rcall  USB_TX_BYTE

       ldi    XH, high(TRAMA_BUFFER)
       ldi    XL, low (TRAMA_BUFFER)
usbtxbuf_lp:
       ld     r16, X+
       rcall  USB_TX_BYTE

       dec    r18
       brne   usbtxbuf_lp

       ; <FLAG.FIN>
       ldi            r16, 0x69
       rcall  USB_TX_BYTE

       ret


; ------------------------------------------------------------
;                                          USB_RX_TRAMA
;
;   Recibe una trama por USB. C=0 si recibida OK.
;     C=1 si trama erronea o no se recibe nada en 32ms
;   Almacena la trama en RAM en "TRAMA_BUFFER", y
;    la longitud en "TRAMA_LONG"
; ------------------------------------------------------------
USB_RX_TRAMA:
       rcall  USB_RX_BYTE           ; Devuelve dato en r16, o C=1 si timeout
       brcs   USBRXTRAMA_FIN_NO_OK  ; Timeout!

       ; Si ha recibido un primer byte: Comprobar si es el flag de inicio:
       cpi            r16, 0x96                        ; Flag inicio
       brne   USBRXTRAMA_FIN_NO_OK  ; Erroneo, terminar

       ; Si, es el flag de inicio.

       ; Ahora esperar campo de longitud:
       rcall  USB_RX_BYTE           ; Devuelve dato en r16, o C=1 si timeout
       brcs   USBRXTRAMA_FIN_NO_OK  ; Timeout

       ; Guardar longitud en r18 y en RAM en "TRAMA_LONG"
       mov            r18, r16
       sts            TRAMA_LONG, r18

       ; Si no hay datos, ir ya al flag final:
       tst            r18
```

```
        breq    USBRXTRAMA_RX_FLAG_FIN

        ; Recibir todos los datos:
        ; -------------------------
        ldi             XH, high( TRAMA_BUFFER )
        ldi             XL,  low( TRAMA_BUFFER )

usbrxtrama_rxdatos_lp:
        rcall   USB_RX_BYTE                         ; Devuelve dato en r16, o C=1 si
timeout
        brcs    USBRXTRAMA_FIN_NO_OK  ; Timeout

        ; Almacenar
        st              X+, r16

        ; Descontar y seguir:
        dec             r18
        brne    usbrxtrama_rxdatos_lp

        ; Todos los datos recibidos.

        ; Ahora recibir el flag de final:
USBRXTRAMA_RX_FLAG_FIN:
        rcall   USB_RX_BYTE                         ; Devuelve dato en r16, o C=1 si
timeout
        brcs    USBRXTRAMA_FIN_NO_OK  ; Timeout

        cpi             r16, 0x69                           ; Flag final
        brne    USBRXTRAMA_FIN_NO_OK  ; Erroneo, terminar

        ; La trama se ha recibido correctamente.

USBRXTRAMA_FIN_OK:
        ret

USBRXTRAMA_FIN_NO_OK:
        sec
        ret
```

```
;-------------------------------------------------------------------------
; FICHERO: LCD.asm
; DESCRIPCION: Incluye funciones para controlar el display LCD
;-------------------------------------------------------------------------

;-----------------------------------------------------
;  Inicializar los puertos y activar LCD en modo
;    de dos lineas de texto.
;-----------------------------------------------------
LCD_INICIAR:
        cbi     LCD_CTRL_PORT,LCD_CTRL_E ; E=0 inicialmente:
        sbi     LCD_CTRL_DDR,LCD_CTRL_E

        sbi     LCD_CTRL_PORT,LCD_CTRL_RS ; RS=1 inicialmente:
        sbi     LCD_CTRL_DDR,LCD_CTRL_RS

        sbi             LCD_DATA_DDR, 0
        sbi             LCD_DATA_DDR, 1
```

```
        sbi             LCD_DATA_DDR, 2
        sbi             LCD_DATA_DDR, 3

        ; Esperar 1 segundo al menos:
        ldi             r25,40
        rcall   RETARDO_R25x25ms


        ; Iniciar LCD:
        ;  En datasheets recomiendan reiniciar el LCD 3 veces, con comando $3X
        ;   para asegurar correcta inicializacion en cualquier condicion de
        ;   subida de tension de alimentacion:

        cbi     LCD_CTRL_PORT,LCD_CTRL_RS     ; rs=0

        ldi             conta,3
lcd_init_bucle:
        ; Enviar comando de inicializacion:
        ldi             r16,0x30
        rcall   LCD_R16_SACA_NIBBLE_HIGH

        ; Un tiempo hasta dar pulso E (Aprox. 10us)
        ldi             r25,35
        rcall   RETARDO_R25_MAX100us

        ; Dar un pulso de E para que acepte el dato:
        rcall   LCD_PULSO_E

        ; Retardo de proceso de comando: 3ms
        ldi             r25,1
        rcall   RETARDO_R25x25ms

        dec             conta
        brne    lcd_init_bucle

        ; Fin de inicializacion. Mandar comandos de configuracion:
        ; ---------------------------------------------------------------

        ; Poner en modo de 4 bits: CMD $20
        ; Activar modo de 2 lineas y caracteres 5x7 estandar: CMD $28
        ldi             r16,0x20
        rcall   LCD_SEND_R16_INST

        ; Activar display y no mostrar cursor : CMD 0x0C
        ldi             r16,0x0C
        rcall   LCD_SEND_R16_INST

        ; Para evitar problemas ocasionales, activar, desactivar y volver a activar el
LCD:

        ; Desactivar
        ldi             r16,0x08
        rcall   LCD_SEND_R16_INST

        ; Activar display y no mostrar cursor : CMD 0x0C
        ldi             r16,0x0C
        rcall   LCD_SEND_R16_INST
```

```
        ; Borrar pantalla:
        rcall   LCD_BORRAR

        ret


;-----------------------------------------------------
;  LCD_BORRAR
;-----------------------------------------------------
LCD_BORRAR:
        ; Comando: $01 = Clear display
        ldi     r16,0x01
        rjmp    LCD_SEND_R16_INST


;-----------------------------------------------------
;  LCD_CURSOR_NO_BLINK
;-----------------------------------------------------
LCD_CURSOR_NO_BLINK:
        ldi     r16,0x0E
        rjmp    LCD_SEND_R16_INST

;-----------------------------------------------------
;  LCD_CURSOR_BLINK
;-----------------------------------------------------
LCD_CURSOR_BLINK:
        ldi     r16,0x0F
        rjmp    LCD_SEND_R16_INST


;-----------------------------------------------------
; LCD_CURSOR_OCULTAR
;-----------------------------------------------------
LCD_CURSOR_OCULTAR:
        ldi     r16,0x0C
        rjmp    LCD_SEND_R16_INST

;-----------------------------------------------------
;  LCD_R16_SACA_NIBBLE_LOW
;    low_nibble(r16) -> Nibble bajo de LCD_PORT
;-----------------------------------------------------
LCD_R16_SACA_NIBBLE_LOW:
        push    r16
        push    r17

        andi    r16,0x0F

        in              r17,LCD_DATA_PORT
        andi    r17,0xF0
        or              r17,r16

        out             LCD_DATA_PORT,r17

        pop             r17
        pop             r16
        ret

;-----------------------------------------------------
```

```
;  LCD_R16_SACA_NIBBLE_HIGH
;    high_nibble(r16) -> Nibble bajo de LCD_PORT
;-----------------------------------------------------
LCD_R16_SACA_NIBBLE_HIGH:
       push    r16
       push    r17

       swap    r16
       andi    r16,0x0F

       in              r17,LCD_DATA_PORT
       andi    r17,0xF0
       or              r17,r16

       out             LCD_DATA_PORT,r17

       pop             r17
       pop             r16
       ret

;-----------------------------------------------------
;  LCD_PULSO_E
;
;   Genera un pulso 0->1->0 en linea E con
;     la temporizacion adecuada a 8Mhz de XTAL
;-----------------------------------------------------
LCD_PULSO_E:
       ldi             r25,5
       rcall   RETARDO_R25_MAX100us          ; aprox. 1.5us

       sbi             LCD_CTRL_DDR ,LCD_CTRL_E      ; (Para asegurarse)
       sbi             LCD_CTRL_PORT,LCD_CTRL_E      ; E=1:min 450ns

       ldi             r25,5
       rcall   RETARDO_R25_MAX100us          ; aprox. 1.5us

       cbi             LCD_CTRL_PORT,LCD_CTRL_E      ; E=0.min 500ns

       ldi             r25,5
       rcall   RETARDO_R25_MAX100us          ; aprox. 1.5us

       ret


;-----------------------------------------------------
;  LCD_SEND_R16_DATA
;
;       Envia el contenido de r16 al LCD como dato
;-----------------------------------------------------
LCD_SEND_R16_DATA:
       sbi             LCD_CTRL_PORT,LCD_CTRL_RS     ; rs=1

       ; Enviar
       rcall   LCD_SEND_R16

       ; Esperar a que el LCD procese el dato: 2ms
       ldi             r25,12
       rcall   RETARDO_R25_MAX25ms
```

```
        ret


LCD_SEND_R16:

        ; Antes de usar estas 4 lineas, asegurar que no van
        ;   a ser usadas como salida por la FIFO:
lcd_esperar_fifo_empty:
        sbic    FIFO_CTRL_PIN, FIFO_EMPTY
        rjmp    lcd_esperar_fifo_empty


        sbi             LCD_DATA_DDR, 0
        sbi             LCD_DATA_DDR, 1
        sbi             LCD_DATA_DDR, 2
        sbi             LCD_DATA_DDR, 3

        rcall   LCD_R16_SACA_NIBBLE_HIGH

        ; Un tiempo hasta dar pulso E (Aprox. 10us)
        ldi             r25,35
        rcall   RETARDO_R25_MAX100us

        rcall   LCD_PULSO_E

        ; 2° nibble bajo:
        rcall   LCD_R16_SACA_NIBBLE_LOW

        ; Un tiempo hasta dar pulso E (Aprox. 10us)
        ldi             r25,35
        rcall   RETARDO_R25_MAX100us

        rcall   LCD_PULSO_E

        cbi             LCD_DATA_DDR, 0
        cbi             LCD_DATA_DDR, 1
        cbi             LCD_DATA_DDR, 2
        cbi             LCD_DATA_DDR, 3

        ret


;-----------------------------------------------------
;  LCD_SEND_R16_INST
;
;       Envia el contenido de r16 al LCD como instruccion
;-----------------------------------------------------
LCD_SEND_R16_INST:
        cbi             LCD_CTRL_PORT,LCD_CTRL_RS     ; RS=0

        ; Enviar
        rcall   LCD_SEND_R16

        ; Esperar a que el LCD procese el dato: 2ms
        ldi             r25,25
        rcall   RETARDO_R25_MAX25ms
```

```
        ret



;-----------------------------------------------------
;  LCD_SET_CURSOR_POS_R16
;
;       Mueve el cursor a la posicion del LCD "r16"
;
; 1° linea: $00 a $0F
; 2° linea: $40 a $4F
;-----------------------------------------------------
LCD_SET_CURSOR_POS_R16:
        ori             r16,$80         ;instruccion : 0b1XXXXXX
        rjmp    LCD_SEND_R16_INST


;-----------------------------------------------------
; LCD_ESCRIBE_CADENA_Z_FLASH
;
;  Escribe una cadena de bytes terminada en 0,
;   almacenada en flash en direccion Z
;-----------------------------------------------------
LCD_ESCRIBE_CADENA_Z_FLASH:

lcd_esc_cad_bucl:
        ; Cargar un byte de flash:
        lpm
        mov     r16,r0
        tst     r16                     ; 0=fin de cadena
        breq    lcd_esc_cad_fin

        rcall   LCD_SEND_R16_DATA       ; Escribir caracter
        ; Incrementar contador Z:
        ld      r16,Z+
        rjmp    lcd_esc_cad_bucl


lcd_esc_cad_fin:

        ret



;-----------------------------------------------------
;           LCD_ESCRIBE_R16_HEX
;
;  Escribe dos digitos en hexadecimal de valor (r16)
;-----------------------------------------------------
LCD_ESCRIBE_R16_HEX:
        push    r17
        push    r16

        mov     r17,r16 ; Guardar byte a mostrar en r16

        ; Nibble superior:
        swap    r17
        rcall   LCD_LOW_NIBBLE_R17_2_R16_ASCII
```

```
        rcall   LCD_SEND_R16_DATA

        ; Nibble inferior:
        pop     r16
        push    r16
        mov     r17,r16 ; Guardar byte a mostrar en r16

        rcall   LCD_LOW_NIBBLE_R17_2_R16_ASCII
        rcall   LCD_SEND_R16_DATA

        pop     r16
        pop     r17
        ret

;-----------------------------------------------------
;           LCD_ESCRIBE_R16_NIBB
;
;  Escribe un digitos en hexadecimal del valor
;    del nibble inferior de (r16)
;-----------------------------------------------------
LCD_ESCRIBE_R16_NIBB:
        push    r17
        push    r16

        mov     r17,r16 ; Guardar byte a mostrar en r16
        rcall   LCD_LOW_NIBBLE_R17_2_R16_ASCII
        rcall   LCD_SEND_R16_DATA

        pop     r16
        pop     r17
        ret


;-----------------------------------------------------------
; Rutina auxiliar: Modifica el valor en r17
;-----------------------------------------------------------
LCD_LOW_NIBBLE_R17_2_R16_ASCII:
        push    r18

        andi    r17,0x0F
        ; Tratar como si fuese digito 0-9:
        ldi     r16,'0'
        add     r16,r17

        ; Ver si se trata de A-F:
        ldi     r18,10
        cp      r17,r18
        brmi    fin_r17_2_r16_asc     ; Saltar si NUM-10 < 0 -> NUM < 10

        ; Es una letra:
        subi    r17,10
        ldi     r16,'A'
        add     r16,r17


fin_r17_2_r16_asc:
        pop     r18
        ret
```

```
;-----------------------------------------------------
; FICHERO: PLL.asm
; DESCRIPCION: Implementa las funciones necesarias
;  para comunicarse a traves del BUS-3W (SPI) con
;   el PLL
;-----------------------------------------------------

; ------------------------------------------------
;  Inicializacion de los puertos
; ------------------------------------------------
PLL_INICIAR:
       ; SYNC = LE = 1
       sbi            BUS3W_PLL_SYNC_DDR,  BUS3W_PLL_SYNC
       sbi            BUS3W_PLL_SYNC_PORT, BUS3W_PLL_SYNC

       ; MUXOUT como entrada HI-Z:
       cbi            PLLMUX_DDR, PLLMUX
       cbi            PLLMUX_PORT, PLLMUX

       ret


; ------------------------------------------------
; Los 3 bytes que se mandan deben estar
;   en  r3:r2:r1  , siendo r3 el MSB
; ------------------------------------------------
PLL_ENVIAR:
       ; Comando ya esta en r3:r2:r1:r0

       ; Dirigir el comando al PLL:
       cbi            BUS3W_PLL_SYNC_PORT, BUS3W_PLL_SYNC

       ; Enviar
       ldi            r18, 24               ; Son 24 bits
       rcall   BUS3W_ENVIAR   ; r18: #de bits; datos en r3:r2:r1:r0

       ; Quitar señal:
       sbi            BUS3W_PLL_SYNC_PORT, BUS3W_PLL_SYNC

       ret
```

A4.3. SWRadioStudio

```
/*************************************************************************
  Fichero: CCargadorDemodDLL.cpp
  USO: Implementa el objeto que carga una DLL de un demodulador software
*************************************************************************/
#include <vcl.h>
#pragma hdrstop

#include "CCargadorDemodDLL.h"


/*************************************************************************
  Constructor

*************************************************************************/
CCargadorDemodDLL::CCargadorDemodDLL( char *path_dll )
{
        try
        {
                // Cargar DLL:
                handle = LoadLibrary( path_dll );

                // Se ha encontrado la DLL?
                if (!handle) throw "No se encuentra la DLL";

                // Pedir direcciones de funciones:
                // ----------------------------------------

                // Si no tiene la funcion "GetInfo", no es una DLL valida:
                if (! (*((FARPROC*)&ptr_GetInfo) = GetProcAddress( handle,
"@SWDEMOD_GetInfo" ) ) ) throw "No se ha encontrado funcion requerida en DLL";
                if (! (*((FARPROC*)&ptr_Demod) = GetProcAddress( handle,
"@SWDEMOD_Demod" ) ) ) throw "No se ha encontrado funcion requerida en DLL";
                if (! (*((FARPROC*)&ptr_GUICallback) = GetProcAddress( handle,
"@SWDEMOD_GUICallback" ) ) ) throw "No se ha encontrado funcion requerida en DLL";
                if (! (*((FARPROC*)&ptr_SetGUI_Functions) = GetProcAddress( handle,
"@SWDEMOD_SetGUI_Functions" ) ) ) throw "No se ha encontrado funcion requerida en DLL";

        } catch (...)
        {
                if (handle) FreeLibrary(handle);
                handle = NULL;  // Indicador de que la DLL no se ha cargado
correctamente.
        }
}

/*************************************************************************
 Destructor
*************************************************************************/
CCargadorDemodDLL::~CCargadorDemodDLL(  )
{
        // Liberar DLL cargada.
        if (handle) FreeLibrary(handle);
}


/*************************************************************************
CargadaCorrectamente()
*************************************************************************/
bool    CCargadorDemodDLL::CargadaCorrectamente()
{
        return handle != NULL;
}

/*************************************************************************
Devuelve la descripcion
*************************************************************************/
bool    CCargadorDemodDLL::DLL_GetDescription( DemodSW::DescripcionDemodulador *info )
{
        try
        {
                if (!handle) return false;      // Si la DLL no esta cargada: Error

                // Llamar a la DLL:
```

```
                ptr_GetInfo( info );

                // Si se llega aqui es que todo ha ido bien.
                return true;
        } catch (...)
        {
                return false;
        }
}

/*************************************************************************
Realiza la demodulacion en si
*************************************************************************/
bool    CCargadorDemodDLL::DLL_Demodulation(float* in,DWORD nSamplesIn,float *out,DWORD
&nSamplesOut,DWORD nMaxSamplesOut,CUsbSwRadioInterfaz *USBSwRadioInterfaz)
{
        try
        {
                if (!handle) return false;      // Si la DLL no esta cargada: Error

                // Llamar a la DLL:

ptr_Demod(in,nSamplesIn,out,nSamplesOut,nMaxSamplesOut,USBSwRadioInterfaz);

                // Si se llega aqui es que todo ha ido bien.
                return true;
        } catch (...)
        {
                return false;
        }
}


/*************************************************************************
Callback de la GUI
*************************************************************************/
bool    CCargadorDemodDLL::DLL_GUICallback(char *name,WORD event,DWORD param1,DWORD
param2 )
{
        try
        {
                if (!handle) return false;      // Si la DLL no esta cargada: Error

                // Llamar a la DLL:
                ptr_GUICallback(name,event,param1,param2);

                // Si se llega aqui es que todo ha ido bien.
                return true;
        } catch (...)
        {
                return false;
        }
}


/*************************************************************************
DLL_SetGUI_Functions
*************************************************************************/
bool    CCargadorDemodDLL::DLL_SetGUI_Functions(DemodSW::GUI_Funciones *functions )
{
        try
        {
                if (!handle) return false;      // Si la DLL no esta cargada: Error

                // Llamar a la DLL:
                ptr_SetGUI_Functions(functions);

                // Si se llega aqui es que todo ha ido bien.
                return true;
        } catch (...)
        {
                return false;
        }
}
```

```
/*********************************************************************
FICHERO: CCargadorDemodDLL.h
*********************************************************************/

#ifndef CCargadorDemodDLLH
#define CCargadorDemodDLLH

#include "CUsbSwRadioInterfaz.h"
#include "comunSWDemods.h"

class CCargadorDemodDLL
{
  public:
        // Constructor:
        CCargadorDemodDLL( char *path_dll );

        // Destructor:
        ~CCargadorDemodDLL();

        // Devuelve "true" si se ha cargado la DLL correctamente:
        bool    CargadaCorrectamente();

        // Metodos que llaman a funciones en la DLL:
        //   Todas devuelven "true" si se llama correctamente a la funcion:
        // ------------------------------------------------------------------
        bool    DLL_GetDescription( DemodSW::DescripcionDemodulador *info );
        bool    DLL_Demodulation(float* in,DWORD nSamplesIn,float *out,DWORD
&nSamplesOut,DWORD nMaxSamplesOut,CUsbSwRadioInterfaz* USBSwRadioInterfaz);
        bool    DLL_GUICallback(char *name,WORD event,DWORD param1,DWORD param2 );
        bool    DLL_SetGUI_Functions(DemodSW::GUI_Funciones *functions );


  private:
        HANDLE          handle;         // handle de la DLL

        // Punteros a funciones dentro de la DLL que implementan la
        //   interfaz "SWRadioStudio-Demodulador"
        // ------------------------------------------------------------------
        void __fastcall (*ptr_GetInfo)(
                        DemodSW::DescripcionDemodulador *info_out );

        void __fastcall (*ptr_Demod)(
                        float *in,
                        DWORD nSamplesIn,
                        float *out,
                        DWORD &nSamplesOut,
                        DWORD nMaxSamplesOut,
                        CUsbSwRadioInterfaz     *USBSwRadioInterfaz );

        void __fastcall (*ptr_GUICallback)(
                        char *name,
                        WORD event,
                        DWORD param1,
                        DWORD param2 );

        void __fastcall (*ptr_SetGUI_Functions) (
                        DemodSW::GUI_Funciones *functions );

};

#endif
```

```
/*********************************************************************
 Fichero: unFormModuloControlManual.cpp
 Uso: Implementa la ventana del modulo de control manual.

 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
*********************************************************************/

#include <vcl.h>
#pragma hdrstop
```

```
#include "unFormModuloControlManual.h"
#include "CUsbSwRadioInterfaz.h"
#include "CManejadorFicherosHEX.h"
#include "FFTs.hpp"
#include "CGoertzel.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "sgr_def"
#pragma link "sgr_data"
#pragma resource "*.dfm"

/*-------------------------------------------------------------------
        Variables globales
  -------------------------------------------------------------------*/
// La ventana principal de la aplicacion
TFormModuloControlManual               *FormModuloControlManual;

// El objeto para acceder via USB a la radio
// ------------------------------------------
extern CUsbSwRadioInterfaz     UsbSwRadioInterfaz;

DWORD           flujoCount=0;
DWORD           flujoInterval = 0;
DWORD           flujoCuentaBytes = 0;


/*-------------------------------------------------------------------
        Constructor
  -------------------------------------------------------------------*/
__fastcall TFormModuloControlManual::TFormModuloControlManual(TComponent* Owner)
        : TForm(Owner)
{
}

/*-------------------------------------------------------------------
   Pedir cadena de identificacion al radioreceptor.
  -------------------------------------------------------------------*/
void __fastcall TFormModuloControlManual::btnPedirFirmwareClick(TObject *Sender)
{
        char    version[300];

        if (! UsbSwRadioInterfaz.PeticionSinc_VersionFirmware(version) )
                edFirmware->Text = "[ERROR al solicitar comando al radioreceptor (¿No
conectado?)]";
        else    edFirmware->Text = version;

}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::Button10Click(TObject *Sender)
{
        if ( !UsbSwRadioInterfaz.PeticionSinc_LCD_Borrar() )
                throw "Error al enviar comando";
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::Button11Click(TObject *Sender)
{
        int     col = StrToInt( edLCDcol->Text );
        int     fil = StrToInt( edLCDfil->Text );

        if ( !UsbSwRadioInterfaz.PeticionSinc_LCD_Escribir(col,fil,edLCDtext-
>Text.c_str() ) )
                throw "Error al enviar comando de escritura en LCD";
}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnCheckPDClick(TObject *Sender)
{
        bool    pwdown;

        if ( !UsbSwRadioInterfaz.PeticionSinc_LeerPowerdown(&pwdown) )
        {
                ShowMessage("Error al enviar comando");
                return;
        }
```

```
        cbPWDOWN->Checked = pwdown;
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::btnEntrarPDClick(TObject *Sender)
{
        if ( !UsbSwRadioInterfaz.PeticionSinc_CambiarPowerdown(true) )
                ShowMessage("Error al enviar comando");
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::btnSalirPDClick(TObject *Sender)
{
        if ( !UsbSwRadioInterfaz.PeticionSinc_CambiarPowerdown(false) )
                ShowMessage("Error al enviar comando");
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::btnCapturarMuestrasClick(TObject *Sender)
{
    int                             i=0;
    int                             nTramas = 0;
    int                             MaxTramas = StrToInt(edCuantasTramas->Text);
    CUsbSwRadioInterfaz::TTrama     *trama;
    DWORD                           timeout;

   try
   {
        // Intentar iniciar modo de muestreo:
        if (! UsbSwRadioInterfaz.EnviarTrama(0x40, 0,NULL) )
                throw "Error al comunicar con receptor (¿Esta conectado?)";

        Screen->Cursor=crHourGlass;

        FILE    *f = fopen(edFicheroCapturar->Text.c_str(),"wt");

        timeout = GetTickCount();
        do
        {
            trama = UsbSwRadioInterfaz.CogerTramaDeRespuesta();
            if (trama)
            {
             if (trama->nCMD==0xC0)
             {
                timeout = GetTickCount();  // Reiniciar timeout
                for (i=0;i<64;i++)
                {
                        float I = trama->datos[2*i+0] * 2.5f/256;
                        float Q = trama->datos[2*i+1] * 2.5f/256;
                        fprintf(f,"%f %f\n",I,Q);
                }
                nTramas++;
             }

             delete trama;
            }
            else Sleep(0);
        } while (nTramas<MaxTramas && (GetTickCount()-timeout)<1000);

        // Cerrar fichero
        fclose(f);

        // Detener modo de muestreo:
        UsbSwRadioInterfaz.EnviarTrama(0x41, 0,NULL);
        Sleep(100);
        Screen->Cursor=crDefault;

   }
   catch (char* s)
   {
                ShowMessage(s);
   }

}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::Button17Click(TObject *Sender)
{
        if (! UsbSwRadioInterfaz.EnviarTrama(0x41, 0,NULL) )
        {
                Label1->Caption="No enchufada??";
```

```
                return;
        }
        else    Label1->Caption="0x41 OK";

}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnConfigPLLClick(TObject *Sender)
{
    try
    {
        CUsbSwRadioInterfaz::PLL_CONFIG_DATA     pllConfig;

        pllConfig.AntiBacklashWidth = cbABSW->ItemIndex;
        pllConfig.LockDetectionPrecision = cbLDP->ItemIndex != 0;
        pllConfig.CorePowerLevel = cbCorePower->ItemIndex;   // 5mA (Lo recomendado)
        pllConfig.CounterReset = false;
        pllConfig.MuxOut =  (CUsbSwRadioInterfaz::MUXOUTVALUE)rgMUXOUT->ItemIndex;
//CUsbSwRadioInterfaz::R_DIVIDER;
        pllConfig.PhasePositive = cbPhasePos->Checked;        // Filtro de lazo pasivo
("non-inverting")
        pllConfig.ChargePumpThreeState = cbCPHIZ->Checked;
        pllConfig.MuteTilLock = cbMTL->Checked;          // No usado.
        pllConfig.OutputPowerLevel = cbOPL->ItemIndex;         // -3dBm
        pllConfig.PowerDown = cbPD->ItemIndex;

        pllConfig.CP_Gain_CurrentSelect = false;        // Seleccionar siempre la
primera configuracion de corriente.
        pllConfig.CP_CurrentSetting1 = cbCPCL->ItemIndex;       // 2.5mA
        pllConfig.CP_CurrentSetting2 = 7;       // 2.5mA

        float xtalref = StrToFloat( edPLL_XTAL->Text ) * 1e6; //2.5e6
        float ref = StrToFloat(edPLL_Af->Text ) * 1e3; // 100e3
        pllConfig.R_14bits = xtalref / ref;
        pllConfig.B_13bits = StrToFloat( edPLL_Frec->Text ) * 1e6 / ref;

        if (! UsbSwRadioInterfaz.PeticionSinc_ConfiguraPLL( &pllConfig ) )
                throw "Error configurando PLL";

        lbPLLStatus->Caption = "PLL configurado correctamente!";

    } catch (char *s)
    {
        lbPLLStatus->Caption = s;
    }

}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnLeerMUXOUTClick(TObject *Sender)
{
        try
        {
                BYTE    array[255];

                pcGraficas->ActivePageIndex = 2;

                if (! UsbSwRadioInterfaz.PeticionSinc_LeerMuxoutPLL( array ) )
                        throw "Error leyendo salida MUXOUT del PLL";

                puntosMUXOUT->LockInvalidate = true;

                puntosMUXOUT->Clear();

                for (int i=0;i<254*8;i++)
                {
                        int nByte = i / 8;
                        int nBit  = i % 8;

                        bool bit = array[nByte] & (1 << nBit) ;

                        puntosMUXOUT->AddXY( i, bit ? 1.0:0.0 );
                }

                puntosMUXOUT->LockInvalidate = false;
                lbPLLStatus->Caption = "Salida MUXOUT del PLL leida correctamente!";
        } catch (char *s )
```

```
        {
                lbPLLStatus->Caption = s;
        }
}

/*---------------------------------------------------------------
   Cuando el usuario mueva la "track bar", poner voltaje
     en el "edit" y enviar al radioreceptor:
  ---------------------------------------------------------------*/
void __fastcall TFormModuloControlManual::tbDAC0Change(TObject *Sender)
{
        AnsiString s;
        s.sprintf("%.03f", 5.0f * tbDAC0->Position / 255 );
        edDAC0->Text = s;

        BYTE    Valor_DAC;
        Valor_DAC = 255.0f*( StrToFloat( edDAC0->Text ) / 5.0f);

        if ( !UsbSwRadioInterfaz.PeticionAsinc_CambiarDAC( 0, Valor_DAC ) )
        {
                lbDACStatus->Caption = "Error enviando valor al DAC";
                return;
        }

        lbDACStatus->Caption = "DAC actualizado correctamente";
}
/*---------------------------------------------------------------
  ---------------------------------------------------------------*/
void __fastcall TFormModuloControlManual::timCheckConectadoTimer(TObject *Sender)
{
        if (Visible)    // Solo si esta ventana esta abierta...
                StatusBar->SimpleText = UsbSwRadioInterfaz.EstaAbierto() ?
                        "[Conexion USB con radioreceptor abierta]" : "[No conectado a
radioreceptor]";
}



void __fastcall TFormModuloControlManual::FormShow(TObject *Sender)
{
        cbPD->ItemIndex=0;
        cbABSW->ItemIndex = 0;
        cbLDP->ItemIndex = 0;
        cbCorePower->ItemIndex = 0;
        cbOPL->ItemIndex = 2;
        cbCPCL->ItemIndex = 7;
}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnGuardarGraficasClick(
      TObject *Sender)
{
        int     i,n;

        if (!SD->Execute()) return;

        n= puntos_I->Count;

        FILE    *f=fopen(SD->FileName.c_str(),"wt");
        if (!f) return;

        for (i=0;i<n;i++)
        {
                fprintf(f,"%f %f %f\n", (float)puntos_I->GetX(i),(float)puntos_I-
>GetY(i),(float)puntos_Q->GetY(i) );
        }
        fclose(f);
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::btnClearLCDClick(TObject *Sender)
{
        if (! UsbSwRadioInterfaz.PeticionSinc_LCD_Borrar() )
                throw "Error enviando comando de borrar LCD";
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::FormCreate(TObject *Sender)
{
```

```
        DecimalSeparator = '.';
}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnIniciarMuestreoClick(
      TObject *Sender)
{
        // Enviar trama de inicio de muestreo:
        if (! UsbSwRadioInterfaz.EnviarTrama(0x40, 0,NULL) )
        {
                ShowMessage("ERROR: El radioreceptor no responde...");
                return;
        }

        // Como referencia de tiempo:
        timInicioMuestreo = GetTickCount();

        // Habilitar timer:
        timMuestreoContinuo->Interval = StrToInt( edTimeActGraf->Text );
        timMuestreoContinuo->Enabled = true;

        // Habilitar botones:
        btnIniciarMuestreo->Enabled = false;
        btnDetenerMuestreo->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnDetenerMuestreoClick(
      TObject *Sender)
{
        // Enviar trama de fin de muestreo:
        UsbSwRadioInterfaz.EnviarTrama(0x41, 0,NULL);  // Da igual si ocurriese algun
error, hacer lo siguiente:

        // Deshabilitar timer:
        timMuestreoContinuo->Enabled = false;

        // Habilitar botones:
        btnIniciarMuestreo->Enabled = true;
        btnDetenerMuestreo->Enabled = false;
}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::tbSampleSizeChange(
      TObject *Sender)
{
        int     n = tbSampleSize->Position;

        edSamples->Text = n;
        edSampleSize->Text   = 64 * n;
        edSampleLength->Text = 64*0.004* n;
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::timMuestreoContinuoTimer(
      TObject *Sender)
{
        CUsbSwRadioInterfaz::TTrama     *trama;         // La trama encolada
        int                             i,j;            // Auxiliares
        int                             n;              // Num. de muestras total
        int                             nTramasTotal;   // Tramas a recibir
        int                             nTramasRx;      // Tramas recibidas
        AnsiString                      str;

        if (!Visible) return;   // La ventana ha sido cerrada

        // No actualizar ventana gráfica hasta el final:
        puntos_I->LockInvalidate = true;
        puntos_Q->LockInvalidate = true;
        puntos_IQ->LockInvalidate = true;

        // Borrar cronograma anterior:
        puntos_I->Clear();
        puntos_Q->Clear();
        puntos_IQ->Clear();

        // Numero total de muestras (por canal)
        n = 0;
```

```
        // Numero de tramas a recibir:
        nTramasTotal=tbSampleSize->Position;

        // Tramas recibidas
        nTramasRx = 0;

      try
      {
        do
        {
            trama = UsbSwRadioInterfaz.CogerTramaDeRespuesta();
            if (trama)
            {
             if (nTramasRx<nTramasTotal)
             {  // Si todavia necesitamos tramas para el cronograma, añadirla
                // a la grafica. De todas formas hay que quitarlas todas de la cola
                    if (trama->nCMD==0xC0)
                    {
                       // Es una trama de datos muestreados:
                       nTramasRx++;
                       // Añadir a las gráficas:
                       for (i=0;i<64;i++,n++)
                       {
                               puntos_I->AddXY( n*4 /*us*/, trama->datos[2*i+0] *
2.5f/256 );
                               puntos_Q->AddXY( n*4 /*us*/, trama->datos[2*i+1] *
2.5f/256 );
                       }
                    }
             }
             delete trama;
            }
        } while (trama );
      } catch (...) {ShowMessage("Excepcion procesando tramas encoladas...");}

        // Guardar el numero de muestras:
        int     N = puntos_I->Count;

        if (N)
        {
                AnsiString              aux;

                // Calcular media de I y Q:
                float   media_I=0,media_Q=0;
                // Canal I:
                for (i=0;i<N;i++) media_I+=puntos_I->GetY(i);
                media_I /= N;
                // Canal Q:
                for (i=0;i<N;i++) media_Q+=puntos_Q->GetY(i);
                media_Q /= N;

                // Estimar potencia de la señal recibida:
                float var_I=0,var_Q=0;
                for (i=0;i<N;i++) var_I+=(puntos_I->GetY(i)-media_I)*(puntos_I->GetY(i)-
media_I);
                var_I/=(N-1);
                for (i=0;i<N;i++) var_Q+=(puntos_Q->GetY(i)-media_Q)*(puntos_Q->GetY(i)-
media_Q);
                var_Q/=(N-1);

                // Estimacion de la amplitud de la señal compleja en "Vrms"
                float potencia = sqrt(var_I+var_Q);
                float potencia_dB;
                if (potencia>0)
                        potencia_dB = 20*log10(potencia);
                else    potencia_dB = -200;

                // Mostrar estos valores por pantalla:
                aux.sprintf("E[ I(n) ] = %.02f",media_I);
                lbMediaI->Caption = aux;

                aux.sprintf("E[ Q(n) ] = %.02f",media_Q);
                lbMediaQ->Caption = aux;

                aux.sprintf("Amplitud banda base= %.01f mVrms = %.02f
dBVrms",1000*potencia,potencia_dB);
```

```
        lbPotencia->Caption = aux;


        puntos_I->Active=true;
        puntos_Q->Active=true;
        puntos_IQ->Active=false;

        plot_iq->LeftAxis->Min = 0;
        plot_iq->LeftAxis->Max = 2.5;
        plot_iq->BottomAxis->AutoMax=true;
        plot_iq->BottomAxis->AutoMin=true;

}

// Dibujar la FFT para poder visualizar el espectro de la señal BB=I+j*Q
if (N)
{
 try
 {
        N = min(N, 8192);
        TComplex   x[9000], X[9000];
        float      window[9000],window_pow_scale;
        float      re_media,im_media;
        float      pot_max = -1000, pot_max_frec = 0;
        // Enventanar señal??
        switch (rgEnventanado->ItemIndex)
        {
                case 0:         // Ninguna (Ventana rectangular)
                        for (i=0;i<N;i++)
                                window[i] = 1;
                        window_pow_scale = 1;
                break;

                case 1:         // Hanning:
                        for (i=0;i<N;i++)
                                window[i]= 0.5+0.5*cos((M_PI*2.0*(i-N/2))/N);
                        window_pow_scale = 0.5;
                break;

                case 2:         // Hamming:
                        for (i=0;i<N;i++)
                                window[i]= 0.54+0.46*cos((M_PI*2.0*(i-N/2))/N);
                        window_pow_scale = 0.54;
                break;
        }

        // Calcular componente DC:
        re_media=im_media=0;
        for (i=0;i<N;i++)
        {
                x[i].Re = puntos_I->GetY(i);
                x[i].Im = puntos_Q->GetY(i);
                re_media+=x[i].Re;
                im_media+=x[i].Im;
        }
        re_media /=N; im_media /=N;

        // Quitar componente DC:
        for (i=0;i<N;i++)
        {
                x[i].Re -= re_media;
                x[i].Im -= im_media;
        }

        // Aplicar ventana:
        for (i=0;i<N;i++)
        {
                x[i].Re *= window[i];
                x[i].Im *= window[i];
        }

        // Hacer la FFT
        ForwardFFT(x,N,X,N,N);

        puntos_I_FFT->LockInvalidate=true;
        puntos_I_FFT->Clear();
```

```
                for (i=0;i<N;i++)
                {
                        if (i<N/2)      j=N/2+i;
                        else            j=i-N/2;

                        float           frec = -125+(250.0*i)/N;

                        float           Re = X[j].Re;
                        float           Im = X[j].Im;

                        // Amplitud RMS = (X · X* ) / N
                        float           pot = 1.414213562*sqrt(Re*Re + Im*Im) / N;

                        // Escalar segun enventanado:
                        pot/=sqrt(window_pow_scale);

                        if (pot>pot_max)
                        {
                                pot_max = pot;
                                pot_max_frec = frec;
                        }

                        if (pot!=0)
                                pot = 20*log10( pot );
                        else    pot = -50;

                        puntos_I_FFT->AddXY(frec,pot);
                }

                plot_FFT->BottomAxis->Max= 125;
                plot_FFT->BottomAxis->Min=-125;

                plot_FFT->LeftAxis->Max= 20 - rgMaxPotGraf->ItemIndex*10;
                plot_FFT->LeftAxis->Min= plot_FFT->LeftAxis->Max - 50;
                str.sprintf("Max. amplitud = %.01f mVrms = %.02f dBVrms
(f=%.03fKHz)",pot_max*1000,20*log10(pot_max),pot_max_frec);

                lbMaxPot->Caption = str;
                puntos_I_FFT->LockInvalidate=false;
         } catch (...) {}
        }

        puntos_I->LockInvalidate = false;
        puntos_Q->LockInvalidate = false;
        puntos_IQ->LockInvalidate = false;

        StatusBar->SimpleText = IntToStr(nTramasRx) + " tramas recibidas en
t="+FloatToStr((GetTickCount() - timInicioMuestreo)/1000.0);
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::tbDAC1Change(TObject *Sender)
{
        try
        {
                AnsiString s;
                s.sprintf("%.03f", 5.0f * tbDAC1->Position / 255 );
                edDAC1->Text = s;

                BYTE    Valor_DAC;
                Valor_DAC = 255.0f*( StrToFloat( edDAC1->Text ) / 5.0f);

                if ( !UsbSwRadioInterfaz.PeticionAsinc_CambiarDAC( 1, Valor_DAC ) )
                        throw "Error enviando valor al DAC";

                lbDACStatus->Caption = "DAC actualizado correctamente";
        }
        catch (char* s)
        {
                lbDACStatus->Caption = s;
        }
}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::tbDAC2Change(TObject *Sender)
{
        try
        {
                AnsiString s;
```

```
                // 0.0V - 1.0V
                float      v = (1.0f * tbDAC2->Position) / 55;
                s.sprintf("%.03f", v );
                edDAC2->Text = s;

                s.sprintf("Ganancia VGA: %.02fdB", -5.3 + 50.25*v );
                lbGananciaRF->Caption = s;


                BYTE    Valor_DAC;
                Valor_DAC = 255.0f*( StrToFloat( edDAC2->Text ) / 5.0f);

                if ( !UsbSwRadioInterfaz.PeticionAsinc_CambiarDAC( 2, Valor_DAC ) )
                        throw "Error enviando valor al DAC";

                lbDACStatus->Caption = "DAC actualizado correctamente";
        }
        catch (char* s)
        {
                lbDACStatus->Caption = s;
        }

}
//---------------------------------------------------------------------------
void __fastcall TFormModuloControlManual::tbDAC3Change(TObject *Sender)
{
        try
        {
                AnsiString s;
                // 0.2V - 1.2V
                float v = 0.2f + (1.0f * tbDAC3->Position)/55;
                v = max(0.2f,v); // 0.2v Minimo

                s.sprintf("%.03f", v );
                edDAC3->Text = s;

                s.sprintf("Ganancia demodulador: %.02fdB",61-50*v);
                lbGananciaDemod->Caption = s;

                BYTE    Valor_DAC;
                Valor_DAC = 255.0f*( StrToFloat( edDAC3->Text ) / 5.0f);

                if ( !UsbSwRadioInterfaz.PeticionAsinc_CambiarDAC( 3, Valor_DAC ) )
                        throw "Error enviando valor al DAC";

                lbDACStatus->Caption = "DAC actualizado correctamente";
        }
        catch (char* s)
        {
                lbDACStatus->Caption = s;
        }

}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnSelecFichClick(
      TObject *Sender)
{
        SD->FileName = edFicheroCapturar->Text;
        if (SD->Execute())
                edFicheroCapturar->Text = SD->FileName;
}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnReadFlashClick(TObject *Sender)
{
   try
   {
        BYTE                    flash[8*1024];
        CManejadorFicherosHEX   fichsHEX;

        int     tam_flash = 8*1024;
        int     tam_pag   = 64;
        int     i;

        // Representar gráficamente el avance:
        pbFlash->Min = 0;
```

```
        pbFlash->Max = tam_flash;
        pbFlash->Position = 0;

        // Preguntar por el fichero donde guardar el HEX:
        if (!SD_Hex->Execute()) return;


        // Entrar en modo boot-loader
        // ----------------------------------------------------------------------
        if (! UsbSwRadioInterfaz.PeticionSinc_ModoBootLoader() )
                throw "Error intentando entrar en modo 'Bootloader'";

        // Leer la flash por páginas:
        // ----------------------------------------------------------------------
        for (i=0;i<tam_flash;i+=tam_pag)
        {
                if (! UsbSwRadioInterfaz.PeticionSinc_BOOTLDR_LeeFlash( i, flash+i ) )
                        throw "Error durante lectura de flash!!";

                pbFlash->Position = i;
                Application->ProcessMessages();
        }

        // Guardarla en el fichero indicado
        // ----------------------------------------------------------------------
        fichsHEX.GuardaFicheroHEX(8*1024,flash, SD_Hex->FileName.c_str());

        // Salir de bootloader reiniciando en la aplicacion normal:
        // ----------------------------------------------------------------------
        UsbSwRadioInterfaz.PeticionSinc_BOOTLDR_Reinicia();

        // Todo OK!
        pbFlash->Position = 0;
        ShowMessage("Flash leida correctamente!");
   }
   catch (char *s)
   {
        ShowMessage(s);
   }
}
//---------------------------------------------------------------------------

void __fastcall TFormModuloControlManual::btnUploadFlashClick(TObject *Sender)
{
   try
   {
        // Solo los primeros 6Kbs (Sin el Bootloader)
        BYTE                    flash[6*1024];
        CManejadorFicherosHEX   fichsHEX;
        int                     tam_flash       = 6*1024;
        int                     tam_pag         = 64;

        // Representar gráficamente el avance:
        pbFlash->Min = 0;
        pbFlash->Max = tam_flash;
        pbFlash->Position = 0;

        if (!OD_Hex->Execute()) return;

        // Cargar flash del fichero indicado
        // ----------------------------------------------------------------------
        fichsHEX.CargaFicheroHEX(6*1024,flash,OD_Hex->FileName.c_str());

        // Entrar en modo boot-loader
        // ----------------------------------------------------------------------
        if (! UsbSwRadioInterfaz.PeticionSinc_ModoBootLoader() )
                throw "Error intentando entrar en modo 'Bootloader'";

        for (int i=0;i<tam_flash;i+=tam_pag)
        {
                if (! UsbSwRadioInterfaz.PeticionSinc_BOOTLDR_GrabaFlash( i, flash+i ) )
                        throw "Error grabando flash!!";
                pbFlash->Position = i;
                Application->ProcessMessages();
        }

        // Salir de bootloader reiniciando en la aplicacion normal:
```

```
        // ------------------------------------------------------------------
        UsbSwRadioInterfaz.PeticionSinc_BOOTLDR_Reinicia();

        ShowMessage("Nueva flash grabada OK!! \nEl receptor esta reinciando con el nuevo
software...");
    }
    catch (char *s)
    {
        ShowMessage(s);
    }
}
//---------------------------------------------------------------------------
```

```
/*************************************************************************
 Fichero: unFormDemoduladores.cpp
 Uso: Implementa la ventana del modulo de demoduladores.

 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
*************************************************************************/
#include <vcl.h>
#pragma hdrstop

#include <typeinfo>
#include <stdio.h>

#include "unFormDemoduladores.h"
#include "unPanelDeControl.h"
#include "FFTs.hpp"
#include "unFormDemodInfo.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

// Ventana
// ------------------------------------------
TFormDemoduladores *FormDemoduladores;

// El objeto para acceder via USB a la radio
// ------------------------------------------
extern CUsbSwRadioInterfaz     UsbSwRadioInterfaz;

// El objeto que permite reproducir sonidos
// ------------------------------------------
extern CWaveOut                waveOut;

// El filtro FIR o IIR usado para delimitar el BW de I/Q a la entrada
//  del demodulador
// ------------------------------------------------------------------
CFiltroIIR           filtro_I(1,1),filtro_Q(1,1);

// Declaraciones de las funciones de interfaz GUI a demoduladores, implementadas abajo.
// ------------------------------------------------------------------------------
// Creacion de componentes:
void GUI_Create_Button(char *name,char *text, int X, int Y, int LX, int LY );
void GUI_Create_Label(char *name,char *text, int X, int Y );
void GUI_Create_Edit(char *name,char *text, int X, int Y, int LX, int LY );
void GUI_Create_CheckBox(char *name,char *text, int X, int Y, bool init_state);
// Destruccion de componentes:
void GUI_Free_Component(char *name);
// Cambio de componentes.
void GUI_Change_Text(char *name,char *new_text);
void GUI_Get_Text(char *name,char *new_text);
bool GUI_Get_CheckBox_State(char *name);


/*************************************************************************
        Constructor de la ventana
*************************************************************************/
__fastcall TFormDemoduladores::TFormDemoduladores(TComponent* Owner)
        : TForm(Owner)
{
        // Inicialmente ningun demodulador cargado
```

```
        demodulador = NULL;
        demodulando = false;

        // Salida a fichero deshabilitado
        fil_demod_out = NULL;

        // Inicialmente las graficas estan vacias:
        nMuestrasGrafica1=nMuestrasGrafica2=nMuestrasGrafica3=0;

        // Crear hilos:
        HiloDemodulador = new CHiloDemodulador(false);  // Iniciar en ejecucion
        HiloProcesador  = new CHiloProcesaSalidaDemodulada(false); // Iniciar en
ejecucion

        // Poner referencia al demodulador cargado en cada momento:
        HiloDemodulador->demodulador = &demodulador;
}

/*****************************************************************
        Evento: Al abrir la ventana de demoduladores
*****************************************************************/
void __fastcall TFormDemoduladores::FormShow(TObject *Sender)
{
        // Activar timers:
        // ----------------------------------
        timEstadisticasTrafico->Enabled=true;

        //  Poner valores por defecto en componentes:
        // ------------------------------------------------------------------
        // Filtro FIR por defecto:
        cbTipoFiltro->ItemIndex=0;
        btnCargarFiltroClick(this);
        btnOsc1->Down=false;btnOsc2->Down=false;btnOsc3->Down=false;
        btnOsc4->Down=false;btnOsc5->Down=false;btnOsc6->Down=false;

        //  Iniciar ventanas de "osciloscopios" y "analizadores de espectros"
        // ------------------------------------------------------------------
        Osciloscopio1= new TOsciloscopio(this);
        Osciloscopio2= new TOsciloscopio(this);
        Osciloscopio3= new TOsciloscopio(this);
        Osciloscopio4= new TOsciloscopio(this);
        Osciloscopio5= new TOsciloscopio(this);
        Osciloscopio6= new TOsciloscopio(this);

        Osciloscopio1->Tag=1; Osciloscopio1->Caption="Osciloscopio 1"; Osciloscopio1-
>ClientWidth=200; Osciloscopio1->ClientHeight=150;
        Osciloscopio2->Tag=2; Osciloscopio2->Caption="Osciloscopio 2"; Osciloscopio2-
>ClientWidth=200; Osciloscopio2->ClientHeight=150;
        Osciloscopio3->Tag=3; Osciloscopio3->Caption="Osciloscopio 3"; Osciloscopio3-
>ClientWidth=200; Osciloscopio3->ClientHeight=150;
                Osciloscopio3->plot->LeftAxis->Min=-2;Osciloscopio3->plot->LeftAxis-
>Max=2;
        Osciloscopio4->Tag=4; Osciloscopio4->Caption="FFT 1"; Osciloscopio4-
>ClientWidth=200; Osciloscopio4->ClientHeight=150;
        Osciloscopio5->Tag=5; Osciloscopio5->Caption="FFT 2"; Osciloscopio5-
>ClientWidth=200; Osciloscopio5->ClientHeight=150;
        Osciloscopio6->Tag=6; Osciloscopio6->Caption="FFT 3"; Osciloscopio6-
>ClientWidth=200; Osciloscopio6->ClientHeight=150;


        // Crear lista de demoduladores "plug-ins" instalados en el directorio
        //   correspondiente:
        // ------------------------------------------------------------------
        CListadorDirectorios    *listador;
        AnsiString              mascara;
        CCargadorDemodDLL       *cargadorDLLDemod;

        mascara = FormPanelDeControl->pathDemoduladores+"\\*.swdem";

        listador = new CListadorDirectorios(
                mascara.c_str(),
                FILE_ATTRIBUTE_ARCHIVE
                );

        // Añadir las DLLs encontradas a la lista para poder seleccionarlas
        int             n = listador->Cuantos();
        AnsiString      pathDLL;
```

```
        lbDLLs->Clear();
        for (int i=0;i<n;i++)
                lbDLLs->Items->Add( listador->CogerDeLista(i)->cFileName );

        delete listador;
}

/*************************************************************************
    Evento: Se ha pulsado el boton de aplicar cambios en el filtro
*************************************************************************/
void __fastcall TFormDemoduladores::btnCargarFiltroClick(TObject *Sender)
{
        int     i;
        // Coeficientes a cargar:
        float   coefs_a[50],coefs_b[50];
        int     orden;

        // Cargar los datos del filtro segun seleccion del usuario:
        // ----------------------------------------------------------
        switch (cbTipoFiltro->ItemIndex)
        {
          case 0: {
                //   [b,a]=ellip(4,0.5,20,110/125);
                float tmp_a[5]={ 1, 3.247981304, 4.136851993, 2.426329692, 0.5582481705
};
                float tmp_b[5]={ 0.6935930662, 2.68283444, 3.980561249,
2.68283444,0.6935930662 };
                orden = 5;
                memcpy(coefs_a,tmp_a,sizeof(float)*orden);
                memcpy(coefs_b,tmp_b,sizeof(float)*orden);
          } break;
          case 1: {
                //   [b,a]=ellip(4,0.5,20,100/125);
                float tmp_a[5]={ 1,2.616616096,2.988986357,1.630639108,0.3910469641 };
                float tmp_b[5]={
0.5596800312,2.032852603,2.959620296,2.032852603,0.5596800312 };
                orden = 5;
                memcpy(coefs_a,tmp_a,sizeof(float)*orden);
                memcpy(coefs_b,tmp_b,sizeof(float)*orden);
          } break;

          case 2: {
                //   [b,a]=ellip(4,0.5,20,75/125)
                float tmp_a[5]={ 1,0.7873020513,1.236075452,0.3062390226,0.2252389972 };
                float tmp_b[5]={
0.3209122055,0.8048125231,1.104550563,0.8048125231,0.3209122055 };
                orden = 5;
                memcpy(coefs_a,tmp_a,sizeof(float)*orden);
                memcpy(coefs_b,tmp_b,sizeof(float)*orden);
          } break;

          case 3: {
                //   [b,a]=ellip(4,0.4,20,50/125)
                float tmp_a[5]={ 1,-1.147980121,1.510679371,-0.6990867065,0.2702666266
};
                float tmp_b[5]={
0.1807685578,0.1047325619,0.3106365482,0.1047325619,0.1807685578 };
                orden = 5;
                memcpy(coefs_a,tmp_a,sizeof(float)*orden);
                memcpy(coefs_b,tmp_b,sizeof(float)*orden);
          } break;

          case 4: {
                //   [b,a]=ellip(4,0.5,20,25/125);
                float tmp_a[5]={ 1,-2.865598771,3.499508049,-2.048740447,0.4948092355 };
                float tmp_b[5]={ 0.1080630431,-0.2287907309,0.3169595389,-
0.2287907309,0.1080630431 };
                orden = 5;
                memcpy(coefs_a,tmp_a,sizeof(float)*orden);
                memcpy(coefs_b,tmp_b,sizeof(float)*orden);
          } break;

          default:      throw new Exception("ERROR: Debe seleccionar un filtro de la
lista");
        }
```

```
        // Cambiar orden de los filtros:
        // ------------------------------------------------
        filtro_I.CambiarOrden(orden,orden);
        filtro_Q.CambiarOrden(orden,orden);

        // Cargar filtros I/Q con los coeficientes
        // ------------------------------------------------
        for (i=0;i<orden;i++)
        {
                filtro_I.Coeficiente_a(i, coefs_a[i]);
                filtro_Q.Coeficiente_a(i, coefs_a[i]);
        }
        for (i=0;i<orden;i++)
        {
                filtro_I.Coeficiente_b(i, coefs_b[i]);
                filtro_Q.Coeficiente_b(i, coefs_b[i]);
        }

        // Reiniciar valores retrasados en los filtros:
        filtro_I.Reset();
        filtro_Q.Reset();
}

/*************************************************************************
    Evento: Se ha pulsado el boton de cargar un demodulador
*************************************************************************/
void __fastcall TFormDemoduladores::btnCargarDemodClick(TObject *Sender)
{
        // Mientras se esta cambiando de demodulador, parar el hilo:
        HiloDemodulador->Suspend();

        // Si ya hay algun demodulador cargado, liberarlo:
        delete demodulador;

        // Cargar nuevo demodulador:
        demodulador = NULL;

        lbTituloGUI->Caption="[Ningun demodulador cargado]";

        if (lbDLLs->ItemIndex==-1)
        {
                ShowMessage("Debe seleccionar un demodulador de la lista!!");
        }
        else
        {
                // Intentar cargar demodulador seleccionado:
                AnsiString              path_dll,str;

                // Formar path hasta DLL:
                path_dll = FormPanelDeControl->pathDemoduladores+"\\"+lbDLLs->Items-
>Strings[lbDLLs->ItemIndex];

                // Cargar la DLL:
                demodulador = new CCargadorDemodDLL( path_dll.c_str() );

                // Es una DLL valida?
                if (!demodulador->CargadaCorrectamente())
                {       // Error:
                        ShowMessage("Error al cargar demodulador! (DLL no
compatible?)");
                        delete demodulador;
                        demodulador=NULL;
                }
                else
                {       // OK:
                        // Llamar a los metodos que iniciazaran el demodulador
                        //   y crearan la GUI de este, si es que la tiene:
                        // ------------------------------------------------
                        DemodSW::GUI_Funciones  GUI_funcs;

                        GUI_funcs.GUI_Create_Button = GUI_Create_Button;
                        GUI_funcs.GUI_Create_Label = GUI_Create_Label;
                        GUI_funcs.GUI_Create_Edit = GUI_Create_Edit;
                        GUI_funcs.GUI_Create_CheckBox = GUI_Create_CheckBox;
                        GUI_funcs.GUI_Free_Component = GUI_Free_Component;
                        GUI_funcs.GUI_Change_Text = GUI_Change_Text;
                        GUI_funcs.GUI_Get_Text = GUI_Get_Text;
```

```
                        GUI_funcs.GUI_Get_CheckBox_State = GUI_Get_CheckBox_State;

                        // Enviar punteros de las funciones. En este momento
                        //   el demodulador creara su interfaz grafica si es que
                        //   la tiene.
                        demodulador->DLL_SetGUI_Functions( &GUI_funcs );

                        // Poner titulo:
                        lbTituloGUI->Caption="[Demodulador '"+UpperCase(lbDLLs->Items-
>Strings[lbDLLs->ItemIndex])+"' cargado]";

                        // Poner KSPS a la salida:
                        DemodSW::DescripcionDemodulador         info;
                        demodulador->DLL_GetDescription( &info );

                        str.sprintf("%.02f KSPS",info.rateSalida/1000.0f);
                        lbRateDemodOut->Caption = str;

                        // Guardar caracteristicas para uso futuro rapido:
                        rateSalida_demodulador = info.rateSalida;
                        demodulador_es_digital = !info.analogica;
                }
        }

        // Ya se puede reanudar el hilo:
        HiloDemodulador->Resume();
}


/*****************************************************************
    Evento: Se ha pulsado el boton de aplicar cambios en el filtro
*****************************************************************/
void __fastcall TFormDemoduladores::btnOnOffMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
        if (btnOnOff->Down)
        {       // Ahora estamos en OFF
                // ------------------------------------------------
                // Enviar trama de fin de muestreo:
                Screen->Cursor = crAppStart;
                demodulando = false;
                UsbSwRadioInterfaz.EnviarTrama(0x41, 0,NULL);

                // Detener volcado a fichero si esta habilitado:
                if (fil_demod_out)
                {
                        fclose(fil_demod_out);
                        fil_demod_out = NULL;
                }

                // Esperar fin de reproduccion audio:
                while ( waveOut.GetCurrentlyEnqueued() )
                {
                        Sleep(10);
                        Application->ProcessMessages();
                }

                waveOut.Close();        // Cerrar dispositivo audio:

                btnCargarDemod->Enabled = true;
                StatusBar->Panels->Items[1]->Text="OFF";
                StatusBar->Panels->Items[0]->Text="[Desconectado]";
                Screen->Cursor = crDefault;
        }
        else
        {       // Estamos en ON
                // ------------------------------------------------
                // Solo activar si hay algun demodulador activo:
                if (!demodulador)
                {
                        ShowMessage("Debe seleccionar y cargar un demodulador software
de la lista!");
                        btnOnOff->Down=true;
                        return;
                }

                demodulando = false;
```

```
                // Pedir version del firmware al radioreceptor, para comprobar
                //  si esta conectado correctamente
                UsbSwRadioInterfaz.EnviarTrama(0x41, 0,NULL);   // Para asegurar que no
esta en modo muestreo continuo
                Sleep(100);

                char    version[300];
                if (! UsbSwRadioInterfaz.PeticionSinc_VersionFirmware(version) )
                {
                        ShowMessage("El radioreceptor no responde (¿Se encuentra
conectado?)");
                        btnOnOff->Down=true;
                        return;
                }
                StatusBar->Panels->Items[0]->Text=version;
                Sleep(100);

                // Salir del estado de "PowerDown":
                if (! UsbSwRadioInterfaz.PeticionSinc_CambiarPowerdown(false) )
                {
                        ShowMessage("Error activando radioreceptor (Powerdown=false)");
                        btnOnOff->Down=true;
                        return;
                }

                // Reiniciar las estadisticas de trafico de tramas:
                __int64         nTramasRxTotal,nTramasRxError;

UsbSwRadioInterfaz.EstadisticasTramas(nTramasRxTotal,nTramasRxError,true);
                timComienzoEjecucion = GetTickCount();

                // Reiniciar los filtros I/Q a la entrada del demodulador:
                filtro_I.Reset();
                filtro_Q.Reset();

                // Iniciar muestreo continuo:
                if (! UsbSwRadioInterfaz.EnviarTrama(0x40, 0,NULL) )
                {
                        ShowMessage("Error activando radioreceptor (Iniciando modo de
muestreo continuo)");
                        btnOnOff->Down=true;
                        return;
                }

                // Iniciar volcado a fichero si el usuario lo habilitó
                if (btnHabSalidaFichero->Down)
                        fil_demod_out = fopen( SD->FileName.c_str(),"wt" );

                // Todo OK, estamos en modo "ON":
                demodulando = true;
                StatusBar->Panels->Items[1]->Text="ON";
                btnCargarDemod->Enabled = false;
        }

}

/*****************************************************************************

*****************************************************************************/
void __fastcall TFormDemoduladores::btnOsc1Click(TObject *Sender)
{
        Osciloscopio1->Visible = btnOsc1->Down;
}
void __fastcall TFormDemoduladores::btnOsc2Click(TObject *Sender)
{
        Osciloscopio2->Visible = btnOsc2->Down;
}
void __fastcall TFormDemoduladores::btnOsc3Click(TObject *Sender)
{
        Osciloscopio3->Visible = btnOsc3->Down;
}
void __fastcall TFormDemoduladores::btnOsc4Click(TObject *Sender)
{
        Osciloscopio4->Visible = btnOsc4->Down;
}
void __fastcall TFormDemoduladores::btnOsc5Click(TObject *Sender)
```

```
{
        Osciloscopio5->Visible = btnOsc5->Down;
}
void __fastcall TFormDemoduladores::btnOsc6Click(TObject *Sender)
{
        Osciloscopio6->Visible = btnOsc6->Down;
}

/*********************************************************************

*********************************************************************/
void __fastcall TFormDemoduladores::FormClose(TObject *Sender,
      TCloseAction &Action)
{
        // Detener timers:
        // ----------------------------------
        timEstadisticasTrafico->Enabled=false;

        // Detener ejecucion si estamos en "ON":
        // ------------------------------------------
        if (btnOnOff->Down)
        {       // Detener ejecucion:
                btnOnOff->Down = false;
                TShiftState Shift;
                btnOnOffMouseDown(this,mbLeft,Shift,1,1);
        }

        // Liberar demodulador cargado:
        // ----------------------------------
      try
      {
        if (demodulador)
        {
                delete demodulador;     // Si ya hay algun demodulador cargado,
liberarlo:
                demodulador = NULL;
                lbTituloGUI->Caption="[Ningun demodulador cargado]";
        }
      } catch (...) {}

        //  Liberar ventanas de "osciloscopios" y "analizadores de espectros"
        // ---------------------------------------------------------------------
      try
      {
        delete Osciloscopio1;
        delete Osciloscopio2;
        delete Osciloscopio3;
        delete Osciloscopio4;
        delete Osciloscopio5;
        delete Osciloscopio6;
      } catch (...) {}

}

/*********************************************************************

*********************************************************************/
void __fastcall TFormDemoduladores::lbDLLsClick(TObject *Sender)
{
    try
    {
        // Intentar cargar demodulador seleccionado:
        AnsiString              path_dll;

        if (lbDLLs->ItemIndex==-1) return;      // Debe haber uno seleccionado!

        // Formar path hasta DLL:
        path_dll = FormPanelDeControl->pathDemoduladores+"\\"+lbDLLs->Items-
>Strings[lbDLLs->ItemIndex];

        // Cargar la DLL:
        CCargadorDemodDLL       dll(path_dll.c_str());

        // Es una DLL valida?
        if (!dll.CargadaCorrectamente()) throw "El fichero no es una DLL valida";

        // Pedir descripcion:
```

```
        DemodSW::DescripcionDemodulador         descr;

        if (!dll.DLL_GetDescription(&descr)) throw "Error al pedir informacion";

        demodDescr->Lines->Text = "NOMBRE: " + AnsiString(descr.nombreModulacion)+
                                  "\r\nDESCRIPCION:" + descr.comentarios;
        demodDescr->SelStart=1;

    }
    catch(...)
    {
        demodDescr->Lines->Text="[ERROR al cargar DLL]";
    }
}

/*****************************************************************************

*****************************************************************************/
void __fastcall TFormDemoduladores::FormDestroy(TObject *Sender)
{
        // Detener hilos
        // ----------------------------------------------------------
        HiloDemodulador->Terminate();
        HiloDemodulador->WaitFor();     // Esperar a que termine la ejecucion del hilo:
        delete HiloDemodulador;         // Destruir hilo:

        HiloProcesador->Terminate();
        HiloProcesador->WaitFor();     // Esperar a que termine la ejecucion del hilo:
        delete HiloProcesador;         // Destruir hilo:
}



/*****************************************************************************

         FUNCIONES ENCARGADAS DE IMPLEMENTAR LA INTERFAZ DE GUI
           A LOS MODULOS DE DEMODULADORES SOFTWARE

*****************************************************************************/

/*****************************************************************************
        Creacion de componentes
*****************************************************************************/
void GUI_Create_Button(char *name,char *text, int X, int Y, int LX, int LY )
{
        TButton *c = new TButton(FormDemoduladores);

        c->Name=name;
        c->Left = X + 32;
        c->Top  = Y + 320;
        c->Width  = LX;
        c->Height  = LY;
        c->Caption=text;
        c->OnClick = FormDemoduladores->OnDemodGUIAction;

        FormDemoduladores->InsertControl(c);
}

void GUI_Create_Label(char *name,char *text, int X, int Y )
{
        TLabel *c = new TLabel(FormDemoduladores);

        c->Name=name;
        c->Left = X + 32;
        c->Top  = Y + 320;
        c->AutoSize=true;
        c->Transparent=true;
        c->Caption=text;

        FormDemoduladores->InsertControl(c);
}

void GUI_Create_Edit(char *name,char *text, int X, int Y, int LX, int LY )
{
        TEdit *c = new TEdit(FormDemoduladores);
```

```
        c->Name=name;
        c->Left = X + 32;
        c->Top  = Y + 320;
        c->Width  = LX;
        c->Height  = LY;
        c->Color = clNavy;
        c->Text=text;

        FormDemoduladores->InsertControl(c);
}

void GUI_Create_CheckBox(char *name,char *text, int X, int Y, bool init_state)
{
        TCheckBox *c = new TCheckBox(FormDemoduladores);

        c->Name=name;
        c->Left = X + 32;
        c->Top  = Y + 320;
        c->Checked = init_state;
        c->Caption=text;
        c->Color = clNavy;
        c->Width = 20 + FormDemoduladores->Canvas->TextWidth(text);
        c->OnClick = FormDemoduladores->OnDemodGUIAction;

        FormDemoduladores->InsertControl(c);
}

/*************************************************************************
        Destruccion de componentes
*************************************************************************/
void GUI_Free_Component(char *name)
{
        try
        {
                TComponent      *cmp = FormDemoduladores->FindComponent(name);
                if (cmp)
                {
                        FormDemoduladores->RemoveControl((TControl*)cmp);
                        delete cmp;
                }
        }
        catch (...) { }
}

/*************************************************************************
        Cambio de componentes.
*************************************************************************/
void GUI_Change_Text(char *name,char *new_text)
{
        try
        {
                TComponent      *cmp = FormDemoduladores->FindComponent(name);
                TButton         *btn;
                TLabel          *lb;
                TEdit           *ed;

                if (btn = dynamic_cast<TButton*>(cmp))
                        btn->Caption=new_text;
                else if (lb = dynamic_cast<TLabel*>(cmp))
                        lb->Caption=new_text;
                else if (ed = dynamic_cast<TEdit*>(cmp))
                        ed->Text=new_text;

        }
        catch (...) { }
}

void GUI_Get_Text(char *name,char *new_text)
{
        try
        {
                TComponent      *cmp = FormDemoduladores->FindComponent(name);
                TButton         *btn;
                TLabel          *lb;
                TEdit           *ed;

                if (btn = dynamic_cast<TButton*>(cmp))
```

```
                        strcpy(new_text, btn->Caption.c_str());
                else if (lb = dynamic_cast<TLabel*>(cmp))
                        strcpy(new_text, lb->Caption.c_str());
                else if (ed = dynamic_cast<TEdit*>(cmp))
                        strcpy(new_text, ed->Text.c_str());

        }
        catch (...) { }
}

bool GUI_Get_CheckBox_State(char *name)
{
        try
        {
                TComponent      *cmp = FormDemoduladores->FindComponent(name);
                if (cmp) return ((TCheckBox*)cmp)->Checked;
                else     return false;
        }
        catch (...) { return false; }
}

/*************************************************************************
    Callback de pulsaciones de botones en la GUI del demodulador.
      Incluye botones y checkboxes.
*************************************************************************/
void __fastcall TFormDemoduladores::OnDemodGUIAction(TObject *Sender)
{
        // Utilizar la herramienta RTTI para descubrir si el objeto
        //  pulsado por el usuario es un boton o un check-box:
        TButton         *btn;
        TCheckBox       *cb;

        if (btn = dynamic_cast<TButton*>(Sender))
        {
                // Es un TButton:
                demodulador->DLL_GUICallback( btn->Name.c_str(),        // El nombre
                                              GUIEVENT_BUTTON_PUSHED,   // El evento
                                              0,                        // Param1: Sin
uso
                                              0                         // Param2: Sin
uso
                                              );

        }
        else if (cb = dynamic_cast<TCheckBox*>(Sender))
        {
                // Es un TCheckBox:
                demodulador->DLL_GUICallback( cb->Name.c_str(),         // El nombre
                                              GUIEVENT_CHECKBOX_TOOGLE, // El evento
                                              cb->Checked ? 1:0,        // Param1: El
estado del cb
                                              0                         // Param2: Sin
uso
                                              );
        }
}

void __fastcall TFormDemoduladores::timEstadisticasTraficoTimer(
      TObject *Sender)
{
        __int64         nTramasRxTotal;
        __int64         nTramasRxError;
        AnsiString      str;

        if (!demodulando) return;

        UsbSwRadioInterfaz.EstadisticasTramas(nTramasRxTotal,nTramasRxError,false);

        float t = (GetTickCount() - timComienzoEjecucion)/1000.0f;
        float media;
        if (t!=0) media = nTramasRxTotal*128 / t;
        else      media = 0;
        str.sprintf("%.02f Bytes/s",media);

        StatusBar->Panels->Items[2]->Text="Estadisticas: "+
                                          IntToStr(nTramasRxTotal)+" tramas recibidas /
"+
```

```
                                        IntToStr(nTramasRxError)+" erroneas / "+
                                        str;

}
//---------------------------------------------------------------------------




void __fastcall TFormDemoduladores::timActualizarGraficasTimer(
      TObject *Sender)
{
        // Si hay datos sufientes para la grafica 1:
        if ( nMuestrasGrafica1 == CUANTAS_MUESTRAS_GRAFICAS)
        {
                if ( Osciloscopio1->Visible )   // Cronograma:
                        MostrarCronograma(Osciloscopio1, // La ventana en donde
mostrarlo
                                          grafica1_I,    // La primera grafica
                                          grafica1_Q);   // Y la segunda

                if ( Osciloscopio4->Visible )   // FFT:
                        MostrarFFT(       Osciloscopio4, // La ventana en donde
mostrarlo
                                          250,           // La frecuencia de muestreo
                                          grafica1_I,    // La primera grafica
                                          grafica1_Q);   // Y la segunda

                nMuestrasGrafica1 = 0;  // Indicar que se sigan recogiendo muestras:
        }

        // Si hay datos sufientes para la grafica 2:
        if ( nMuestrasGrafica2 == CUANTAS_MUESTRAS_GRAFICAS)
        {
                if ( Osciloscopio2->Visible )   // Cronograma:
                        MostrarCronograma(Osciloscopio2, // La ventana en donde
mostrarlo
                                          grafica2_I,    // La primera grafica
                                          grafica2_Q);   // Y la segunda

                if ( Osciloscopio5->Visible )   // FFT:
                        MostrarFFT(       Osciloscopio5, // La ventana en donde
mostrarlo
                                          250,           // La frecuencia de muestreo
                                          grafica2_I,    // La primera grafica
                                          grafica2_Q);   // Y la segunda

                nMuestrasGrafica2 = 0;  // Indicar que se sigan recogiendo muestras:
        }

        // Si hay datos sufientes para la grafica 1:
        if ( nMuestrasGrafica3 == CUANTAS_MUESTRAS_GRAFICAS)
        {
                if ( Osciloscopio3->Visible )   // Cronograma:
                        MostrarCronograma(Osciloscopio3, // La ventana en donde
mostrarlo
                                          grafica3,    // La primera grafica
                                          NULL);

                if ( Osciloscopio6->Visible )   // FFT:
                        MostrarFFT(       Osciloscopio6, // La ventana en donde
mostrarlo
                                          50,           // La frecuencia de muestreo
                                          grafica3,    // La primera grafica
                                          NULL);

                nMuestrasGrafica3 = 0;  // Indicar que se sigan recogiendo muestras:
        }

        // Mostrar el nivel del buffer de audio:
        if (!btnHabSalidaSonido->Down)
          StatusBar->Panels->Items[3]->Text="0";
        else
          StatusBar->Panels->Items[3]->Text=IntToStr(waveOut.GetCurrentlyEnqueued());
```

```
}
//---------------------------------------------------------------------------

void __fastcall TFormDemoduladores::tbVolumenChange(TObject *Sender)
{
        // Cambiar el volumen:
        WORD vol = tbVolumen->Position;
        waveOut.SetVolume(vol,vol);
}
//---------------------------------------------------------------------------
void __fastcall TFormDemoduladores::StatusBarDrawPanel(
      TStatusBar *StatusBar, TStatusPanel *Panel, const TRect &Rect)
{
        // Dibujar el panel del buffer de audio:
        StatusBar->Canvas->Brush->Color=clBlack;
        StatusBar->Canvas->FillRect( Rect );

        // Tanto por uno del buffer lleno:
        TRect   rec;
        rec = Rect;

        float f = StrToInt(Panel->Text) / 30.0f;
        rec.Right = rec.Left + (rec.Right - rec.Left)*f;
        StatusBar->Canvas->Brush->Color=clRed;
        StatusBar->Canvas->FillRect( rec );
}

/*************************************************************************
MostrarCronograma
*************************************************************************/
void TFormDemoduladores::MostrarCronograma(TOsciloscopio *osc,float *x,float *y)
{
        int     i,j;

        osc->puntos1->LockInvalidate = true;
        osc->puntos2->LockInvalidate = true;

        osc->puntos1->Clear();
        osc->puntos2->Clear();
        for (i=0;i<CUANTAS_MUESTRAS_GRAFICAS;i++)
        {
                // Si es analogico mostrar cronograma, si es
                //  digital la constelacion I/Q
                if (demodulador_es_digital)
                {       // Digital:
                        if (y!=NULL)    // Es una señal I/Q
                                osc->puntos1->AddXY(x[i]-1.0f,y[i]-1.0f);
                        else            // Es el flujo binario: Dibujar bits:
                        {
                                osc->puntos1->AddXY(i  ,x[i]==0 ? 0:1);
                                osc->puntos1->AddXY(i+1,x[i]==0 ? 0:1);
                        }

                }
                else
                {       // Analogica:
                        osc->puntos1->AddXY(i,x[i]);
                        if (y!=NULL)    osc->puntos2->AddXY(i,y[i]);
                }
        }

     // Ajustar ejes:
     osc->plot->BottomAxis->AutoMax = false;
     osc->plot->BottomAxis->AutoMin = false;
     osc->plot->LeftAxis->AutoMax = false;
     osc->plot->LeftAxis->AutoMin = false;

     if (demodulador_es_digital)
     {  // Digital: Ajustar X,Y en [-0.6,0.6] para señales I/Q
        //   y en [-0.5,1.5] para salida binaria demodulada
        if (osc!=Osciloscopio3)
        {
          osc->plot->BottomAxis->Min=-0.6;
          osc->plot->BottomAxis->Max=+0.6;
          osc->plot->LeftAxis->Min= -0.6;
```

```
          osc->plot->LeftAxis->Max= +0.6;
          // Dibujar solo puntos, sin lineas
          osc->puntos1->LineAttr->Visible = false;
          osc->puntos1->PointAttr->Visible = true;
          osc->puntos1->PointAttr->VSize = 5;
          osc->puntos1->PointAttr->HSize = 5;
        }
        else
        {
          osc->plot->BottomAxis->Min=0;
          osc->plot->BottomAxis->Max=CUANTAS_MUESTRAS_GRAFICAS-1;
          osc->plot->LeftAxis->Min= -0.5;
          osc->plot->LeftAxis->Max=  1.5;
          // Dibujar lineas y puntos
          osc->puntos1->LineAttr->Visible = true;
          osc->puntos1->PointAttr->Visible = true;
          osc->puntos1->PointAttr->VSize = 2;
          osc->puntos1->PointAttr->HSize = 2;
        }

     }
     else
     {  // Analogica: Ajustar Y en [0,2.5] para las señales I/Q
        //   y en [-1,1] para salida demodulada
        if (osc!=Osciloscopio3)
        {
          osc->plot->BottomAxis->Min=0;
          osc->plot->BottomAxis->Max=CUANTAS_MUESTRAS_GRAFICAS-1;
          osc->plot->LeftAxis->Min= 0;
          osc->plot->LeftAxis->Max= 2.5;
        }
        else
        {
          osc->plot->BottomAxis->Min=0;
          osc->plot->BottomAxis->Max=CUANTAS_MUESTRAS_GRAFICAS-1;
          osc->plot->LeftAxis->Min= -1;
          osc->plot->LeftAxis->Max=  1;
        }

        osc->puntos1->LineAttr->Visible = true;
        osc->puntos1->PointAttr->Visible = false;
     }

     osc->puntos1->LockInvalidate=false;
     osc->puntos2->LockInvalidate=false;
}

/*************************************************************************
MostrarFFT
*************************************************************************/
void TFormDemoduladores::MostrarFFT(TOsciloscopio *osc,float fs, float *x,float *y)
{
        TComplex   a[9000], A[9000];
        float      re_media,im_media;
        int        i,j,N;      // Auxiliar

        // Por comodidad:
        N = CUANTAS_MUESTRAS_GRAFICAS;

        // Canal "x":
        re_media=im_media=0;
        for (i=0;i<CUANTAS_MUESTRAS_GRAFICAS;i++)
        {
                a[i].Re=x[i];
                re_media+=a[i].Re;

                if (y)
                {
                        a[i].Im = y[i];
                        im_media += a[i].Im;
                }
                else    a[i].Im = 0;

        }

        re_media /=CUANTAS_MUESTRAS_GRAFICAS;
        im_media /=CUANTAS_MUESTRAS_GRAFICAS;
```

```
        // Quitar componente DC:
        for (i=0;i<CUANTAS_MUESTRAS_GRAFICAS;i++)
        {
                a[i].Re -= re_media;
                a[i].Im -= im_media;
        }

        ForwardFFT(a,N,A,N,N);

        osc->puntos1->LockInvalidate=true;
        osc->puntos1->Clear();

        for (i=0;i<CUANTAS_MUESTRAS_GRAFICAS;i++)
        {
                if (i<CUANTAS_MUESTRAS_GRAFICAS/2)      j=CUANTAS_MUESTRAS_GRAFICAS/2+i;
                else            j=i-CUANTAS_MUESTRAS_GRAFICAS/2;
                float           Re = A[j].Re / N;
                float           Im = A[j].Im / N;
                osc->puntos1->AddXY(-
fs*0.5+(fs*i)/CUANTAS_MUESTRAS_GRAFICAS,sqrt(Re*Re+Im*Im));
        }

        // Canal "y":
        osc->puntos2->Clear();

     osc->plot->BottomAxis->AutoMax = false;
     osc->plot->BottomAxis->AutoMin = false;
     osc->plot->LeftAxis->AutoMax = false;
     osc->plot->LeftAxis->AutoMin = false;

     if (osc->plot->BottomAxis->Min!=-fs/2)
                                        osc->plot->BottomAxis->Min=-fs/2;
     if (osc->plot->BottomAxis->Max!= fs/2)
                                        osc->plot->BottomAxis->Max= fs/2;
     if (osc->plot->LeftAxis->Min!=0)
                                        osc->plot->LeftAxis->Min= 0;
     if (osc->plot->LeftAxis->Max!=0.1)
                                        osc->plot->LeftAxis->Max= 0.1;

        osc->puntos1->LockInvalidate=false;
        osc->puntos2->LockInvalidate=false;
}


void __fastcall TFormDemoduladores::btnHabSalidaFicheroClick(
      TObject *Sender)
{
        if (btnHabSalidaFichero->Down)
        {
                // Iniciar volcado a fichero:
                fil_demod_out = fopen( SD->FileName.c_str(),"wt" );
        }
        else
        {       // Detener volcado a fichero
                fclose(fil_demod_out);
                fil_demod_out = NULL;
        }

}
//---------------------------------------------------------------------------
void __fastcall TFormDemoduladores::btnSeleccionaFichDestClick(
      TObject *Sender)
{
        if ( fil_demod_out )
        {
                ShowMessage("Solo se puede cambiar de fichero cuando no se este en
ejecucion. Detenga \n la ejecucion o la salida por fichero para poder cambiar el fichero
destino");
                return;
        }

        // Abrir cuadro de dialogo
        SD->Execute();
}
//---------------------------------------------------------------------------
void __fastcall TFormDemoduladores::FormCloseQuery(TObject *Sender,
```

```
      bool &CanClose)
{
        // Solo se puede cerrar la ventana cuando la demodulacion
        //  este detenida, lo que se sabe por ejemplo asi:
        CanClose = btnCargarDemod->Enabled;
}
//---------------------------------------------------------------------------
void __fastcall TFormDemoduladores::btnDemodInfoClick(TObject *Sender)
{
        if (lbDLLs->ItemIndex==-1)
        {
                ShowMessage("Debe seleccionar un demodulador de la lista!!");
        }
        else
        {
                // Cargar toda la info y abrir ventana con ella:

                // Formar path hasta DLL:
                AnsiString path_dll = FormPanelDeControl->pathDemoduladores+"\\"+lbDLLs-
>Items->Strings[lbDLLs->ItemIndex];

                // Cargar la DLL:
                CCargadorDemodDLL       dll(path_dll.c_str());

                // Es una DLL valida?
                if (!dll.CargadaCorrectamente()) throw "El fichero no es una DLL
valida";

                // Pedir descripcion:
                DemodSW::DescripcionDemodulador         descr;

                if (!dll.DLL_GetDescription(&descr)) throw "Error al pedir informacion";

                // Rellenar campos de informacion:
                FormDemodInfo->edFich->Text = lbDLLs->Items->Strings[lbDLLs->ItemIndex];
                FormDemodInfo->edMod->Text = descr.nombreModulacion;
                FormDemodInfo->edDesc->Lines->Text = descr.comentarios;
                FormDemodInfo->edRateOut->Text = IntToStr( descr.rateSalida );
                FormDemodInfo->lbAnalog->Caption = descr.analogica ?
"ANALOGICA":"DIGITAL";
                FormDemodInfo->edAutor->Text = descr.autor;



                // Abrir la ventana:
                FormDemodInfo->ShowModal();
        }
}
```

```
/*************************************************************************
 Fichero: CHiloProceasSalidaDemodulada.cpp

 Uso: Implementa el hilo que procesa los datos demodulados del
        demodulador software.

 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
*************************************************************************/

#include <vcl.h>
#pragma hdrstop

#include <stdio.h>
#include "CHiloProceasSalidaDemodulada.h"
#include "CHiloDemodulador.h"
#include "CWaveOut.h"
#include "unFormDemoduladores.h"
#pragma package(smart_init)

// Configuracion del subsistema de buffers de audio
// ------------------------------------------------------
#define AUDIO_BUFFER_SIZE_TO_SEND       5000
#define MAX_SIZE_AUDIO_BUFFER           (AUDIO_BUFFER_SIZE_TO_SEND+100)
#define MAX_AUDIO_BUFFERS               30

// El objeto que permite reproducir sonidos
```

```
// ------------------------------------------
CWaveOut                        waveOut;

// Valor usado para evitar cortes de audio (Ver memoria del PFC)
struct
{
        DWORD    nSoundSamples;
        WORD     soundSamples[MAX_SIZE_AUDIO_BUFFER];
} AudioBuffers[MAX_AUDIO_BUFFERS];
// Indica en cual de los anteriores buffers se esta guardando actualmente.
int                             current_audio_buffer;
// Normalmente sera 1, menos al principio mientras se este "buffering"
int                             audio_buffers_to_play;


__fastcall CHiloProcesaSalidaDemodulada::CHiloProcesaSalidaDemodulada(bool
CreateSuspended)
        : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall CHiloProcesaSalidaDemodulada::Execute()
{
    SetThreadPriority( GetCurrentThread(), THREAD_PRIORITY_HIGHEST);

    while (!Terminated)
    {
        CHiloDemodulador::TBloqueDatos  *bloque_demodulado;

        // Procesar todos los paquetes que haya en cola:
        while (NULL != (bloque_demodulado = FormDemoduladores->HiloDemodulador-
>CogeDatosDeSalida()))
        {
              // Guardar muestras para su visualizacion en la ventana de
              //   los demoduladores:
              // ------------------------------------------------------------------
              for (int i=0;FormDemoduladores-
>nMuestrasGrafica3<CUANTAS_MUESTRAS_GRAFICAS && i<bloque_demodulado->lon;i++)
                    FormDemoduladores->grafica3[FormDemoduladores-
>nMuestrasGrafica3++] = bloque_demodulado->datos[i];


              // Esta habilitada la salida a fichero?
              // ------------------------------------------------------------------
              if (FormDemoduladores->fil_demod_out)
                for (int i=0;i<bloque_demodulado->lon;i++)
                        fprintf(FormDemoduladores->fil_demod_out,
                                "%f ",
                                bloque_demodulado->datos[i]);


              // Esta habilitada la salida por audio?
              // ------------------------------------------------------------------
              if (FormDemoduladores->btnHabSalidaSonido->Down)
              {
                if (!waveOut.IsOpen())
                {
                        // Abrir en modo "mono" y 16 bits
                        waveOut.Open( FormDemoduladores-
>rateSalida_demodulador,true,false );

                        // Poner el volumen indicado por el usuario:
                        FormDemoduladores->tbVolumenChange(this);

                        // Iniciar audio: Buffering de "N" bloques
                        current_audio_buffer = 0;
                        audio_buffers_to_play = (MAX_AUDIO_BUFFERS-1);
                        for (int i=0;i<MAX_AUDIO_BUFFERS;i++)
                                AudioBuffers[i].nSoundSamples=0;
                }

                // Ir rellenando buffers de audio:
                for (int i=0;i<bloque_demodulado->lon;i++)
                        AudioBuffers[current_audio_buffer].

soundSamples[AudioBuffers[current_audio_buffer].nSoundSamples++]
                                 = (int)(30000*bloque_demodulado->datos[i]);
```

```
                // Se ha llenado el buffer actual??
                if
(AudioBuffers[current_audio_buffer].nSoundSamples>=AUDIO_BUFFER_SIZE_TO_SEND)
                {
                        // Pasar al siguiente buffer:
                        current_audio_buffer++;

                        // Si estan todos, enviar a reproducir:
                        if (current_audio_buffer>=audio_buffers_to_play)
                        {
                           // Reproducir los "audio_buffers_to_play" buffers
                           for (int i=0;i<audio_buffers_to_play;i++)
                              waveOut.WriteBlock(
                                   AudioBuffers[i].nSoundSamples*sizeof(WORD),
                                   (BYTE*)AudioBuffers[i].soundSamples);

                           // Reiniciar lista:
                           current_audio_buffer = 0;
                           audio_buffers_to_play = 1;
                        }

                        AudioBuffers[current_audio_buffer].nSoundSamples = 0;
                }
              }

              // Liberar el bloque de datos demodulado:
              // ----------------------------------------------------
              delete[] bloque_demodulado->datos;
              delete[] bloque_demodulado;

        }       // Fin del "while" haya bloques de salida
        Sleep(10);

    }   // Fin del while "!Terminated"


}
```

```
/*************************************************************************
 Fichero: CHiloDemodulador.cpp

 Uso: Implementa el hilo que se encarga de ir pasando los bloques de
        muestras al demodulador software para su demodulacion.
 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
*************************************************************************/
#include <vcl.h>
#pragma hdrstop

#include "CCargadorDemodDLL.h"
#include "CHiloDemodulador.h"
#include "unFormDemoduladores.h"
#pragma package(smart_init)


extern CUsbSwRadioInterfaz UsbSwRadioInterfaz;
// El filtro FIR o IIR usado para delimitar el BW de I/Q a la entrada
//  del demodulador
// ----------------------------------------------------------------
extern CFiltroIIR           filtro_I,filtro_Q;


/*************************************************************************
        Constructor del hilo
*************************************************************************/
__fastcall CHiloDemodulador::CHiloDemodulador(bool CreateSuspended)
        : TThread(CreateSuspended)
{
        // Iniciar objetos y variables --------------------------
        // Crear colas:
        cola_entrada=new TQueue();
        cola_salida=new TQueue();

        // Evento:
```

```
        event_datos_entrada = new TEvent(NULL,false,false,"");

        // Zonas criticas:
        critical_cola_entrada= new TCriticalSection();
        critical_cola_salida = new TCriticalSection();
}

/*************************************************************************
  El codigo a ejecutar en el hilo:
      Mientras no se indique que hay que terminar el hilo, se espera
       un "event" que indica que se han añadido nuevos datos a la cola
       de entrada. Cuando se detecte este "event", se mandaran todos
       los bloques de datos que haya en la cola de entrada al demodulador,
       y se añadiran los bloques de datos de salida a la cola de salida.
       Las colas se acceden mediante zonas críticas.
*************************************************************************/
void __fastcall CHiloDemodulador::Execute()
{
        float           OutSamples[1000];
        unsigned long   nOutSamples;

        SetThreadPriority( GetCurrentThread(), THREAD_PRIORITY_ABOVE_NORMAL);

        while (!Terminated)     // Ejecutar hasta que se señale al hilo que hay que
terminar
        {
        CUsbSwRadioInterfaz::TTrama     *trama;         // La trama encolada
        int                             i;

     if (FormDemoduladores->demodulando)
     {
        // Comprobar si hay tramas de muestras en la cola de entrada:
        do
        {
            trama = UsbSwRadioInterfaz.CogerTramaDeRespuesta();
            if (trama)
            {
                if (trama->nCMD==0xC0)
                {
                        // ----------------------------------------------------------
                        // Hay 64 nuevas muestras por cada canal I/Q de entrada
                        // ----------------------------------------------------------

                        // Copiarlas de la trama y pasarlas a voltios:
                        // ----------------------------------------------------------
                        float   I_entrada[100], Q_entrada[100];

                        // Ir copiando las muestras tambien en el buffer desde
                        //   donde se dibujaran en las gráficas en tiempo real:
                        for (i=0;i<64;i++)
                        {
                                I_entrada[i] = trama->datos[2*i+0] * 2.5f/256;
                                Q_entrada[i] = trama->datos[2*i+1] * 2.5f/256;

                                if (FormDemoduladores-
>nMuestrasGrafica1<CUANTAS_MUESTRAS_GRAFICAS)
                                {
                                        FormDemoduladores->grafica1_I[FormDemoduladores-
>nMuestrasGrafica1] = I_entrada[i];
                                        FormDemoduladores->grafica1_Q[FormDemoduladores-
>nMuestrasGrafica1] = Q_entrada[i];
                                        FormDemoduladores->nMuestrasGrafica1++;
                                }
                        }

                        // Pasarlas por el filtro pre-demodulacion:
                        // ----------------------------------------------------------
                        float   I_filtrado[100], Q_filtrado[100];
                        // Ir copiando las muestras tambien en el buffer desde
                        //   donde se dibujaran en las gráficas en tiempo real:
                        for (i=0;i<64;i++)
                        {
                                I_filtrado[i] = filtro_I.Filtrar(I_entrada[i]);
                                Q_filtrado[i] = filtro_Q.Filtrar(Q_entrada[i]);

                                if (FormDemoduladores-
>nMuestrasGrafica2<CUANTAS_MUESTRAS_GRAFICAS)
```

```
                                {
                                        FormDemoduladores->grafica2_I[FormDemoduladores-
>nMuestrasGrafica2] = I_filtrado[i];
                                        FormDemoduladores->grafica2_Q[FormDemoduladores-
>nMuestrasGrafica2] = Q_filtrado[i];
                                        FormDemoduladores->nMuestrasGrafica2++;
                                }
                        }

                        // Intercalar los datos I/Q filtrados para meterlos
                        //  al demodulador:
                        // ----------------------------------------------------------
                        float IQ_entrada_demod[200];
                        for (i=0;i<64;i++)
                        {
                                IQ_entrada_demod[i*2+0]=I_filtrado[i];
                                IQ_entrada_demod[i*2+1]=Q_filtrado[i];
                        }

                        //  Mandarla al demodulador:
                        // ----------------------------------------------------------
                    try
                    {
                        // Enviar los datos al demodulador:
                        if (!(*demodulador)->DLL_Demodulation(
                                                IQ_entrada_demod,// Datos IN
                                                128,             // Cuantas muestras IN
                                                OutSamples,      // Datos OUT
                                                nOutSamples,     // Cuantos OUT
                                                1000,            // Tamaño del buffer
OUT
                                                &UsbSwRadioInterfaz
                                                ))
                        {       // Ocurrio un error

                        }
                        else
                        {       // Todo OK
                                if (nOutSamples)
                                {
                                        // Formar nuevo bloque de datos
                                        TBloqueDatos    *nuevo_bloque;

                                        nuevo_bloque = new TBloqueDatos[1];

                                        nuevo_bloque->lon = nOutSamples;
                                        nuevo_bloque->datos = new float[nOutSamples];
                                        memcpy(nuevo_bloque-
>datos,OutSamples,sizeof(float)*nOutSamples);

                                        // Añadirlo a la cola de salida:
                                        critical_cola_salida->Acquire();
                                        cola_salida->Push(nuevo_bloque);
                                        critical_cola_salida->Release();
                                }
                        }

                    } catch (...) { MessageBox(NULL,"En
CHiloDemodulador","Excepcion!",0);}

                }  // Fin de "si es una trama de muestras"

             delete trama;      // Ya se puede liberar la memoria de la trama
            }
        } while (trama );

     } // Fin del "if" demodulando

     Sleep(2);
   }       // Fin del "while" not terminated


}

/*****************************************************************
        Destructor
```

```
**********************************************************************/
__fastcall CHiloDemodulador::~CHiloDemodulador()
{
        // Liberar objetos --------------------
        // Colas:
        delete cola_entrada;
        delete cola_salida;

        // Evento:
        delete event_datos_entrada;

        // Zonas criticas:
        delete critical_cola_entrada;
        delete critical_cola_salida;
}


/**********************************************************************
   Metodo que permite añadir un bloque de datos a la cola de entrada
     de muestras I/Q, de una forma segura entre hilos. Los datos
     se copian del puntero dado, asi que despues de llamar a esta funcion
     el puntero original puede ser liberado.
     Las muestras van intercaladas I/Q
**********************************************************************/
void __fastcall CHiloDemodulador::InsertaDatosEnColaEntrada(WORD nMuestras, float *data)
{
        // Crear nuevo bloque de datos
        TBloqueDatos    *bloque = new TBloqueDatos[1];

        bloque->lon = nMuestras;
        bloque->datos = new float[nMuestras];
        memcpy(bloque->datos,data,nMuestras*sizeof(float));

        // Añadir a la cola de forma segura entre hilos:
        critical_cola_entrada->Acquire();
        cola_entrada->Push( (void*)( bloque ) );
        critical_cola_entrada->Release();

        // Avisar al hilo:
        event_datos_entrada->SetEvent();
}


/**********************************************************************
   Metodo que devuelve un bloque de datos de la cola de salida del
    demodulador, de forma segura entre hilos. El usuario debe
    realizar un "delete[] " tanto sobre los datos como sobre
    la estructura en si para liberar la memoria. Devuelve NULL
    cuando no haya mas bloques de datos
**********************************************************************/
CHiloDemodulador::TBloqueDatos *CHiloDemodulador::CogeDatosDeSalida()
{
        CHiloDemodulador::TBloqueDatos *bloque=NULL;

        critical_cola_salida->Acquire();
        if ( cola_salida->AtLeast(1) )
                bloque = (TBloqueDatos*) cola_salida->Pop();

        critical_cola_salida->Release();

        return  bloque;
}
```

```
/**********************************************************************
 Fichero: CWaveOut.cpp

 Uso: Implementa la clase que permite reproducir sonido en un
         dispositivo de onda por defecto de Windows.

 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
**********************************************************************/
#include "CWaveOut.h"
```

```
/*************************************************************************
        Constructor
*************************************************************************/
CWaveOut::CWaveOut()
{
        handle = NULL;  // Indicador de wave-out cerrado

        nPreparedBlocks = 0;    // No hay bloques reproduciendo

        // Crear zona critica para acceder al contador de bloques:
        semaphore = CreateSemaphore(NULL,1,1,NULL);
}

/*************************************************************************
        Destructor
*************************************************************************/
CWaveOut::~CWaveOut()
{
        // Cerrar dispositivo de wave-out
        Close();
        CloseHandle(semaphore);
}


/*************************************************************************
   Intenta abrir el canal de wave-out. Si todo OK, devuelve "true".
        Se le puede indicar la frec. de muestreo y si usar 8/16 bits.
*************************************************************************/
bool CWaveOut::Open(int SamplesPerSec, bool use_16bits, bool use_stereo)
{
        WAVEFORMATEX    format;

        // Esta abierto ya?
        if (handle)
                Close();        // Cerrar dispositivo anterior.

        // El formato requerido:
        format.wFormatTag = WAVE_FORMAT_PCM;
        format.nChannels = use_stereo ? 2:1;
        format.nSamplesPerSec = SamplesPerSec;
        format.wBitsPerSample = use_16bits ? 16:8;
        format.nBlockAlign = (format.nChannels * format.wBitsPerSample)/8;
        format.nAvgBytesPerSec = format.nSamplesPerSec * format.nBlockAlign;
        format.cbSize=0;

        return (MMSYSERR_NOERROR ==
                waveOutOpen(    &handle,        // El handle de salida
                                WAVE_MAPPER,    // El dispositivo por defecto
                                &format,        // El formato a reproducir
                                (DWORD)waveOutProc,   // "event" para "callback"
                                (DWORD)this,           // La referencia al objeto
instanciado
                                CALLBACK_FUNCTION  // El objeto pasado era un "event"
                                ));

}

/*************************************************************************
        Cierra el canal de wave-out. Si todo OK, devuelve "true"
*************************************************************************/
bool CWaveOut::Close()
{
        if (!handle) return false;      // Estaba ya cerrado?

//        while ( GetCurrentlyEnqueued() )
//                Sleep(10);

        waveOutClose(handle);   // Cerrar dispositivo wave-out.
        handle = NULL;
        nPreparedBlocks = 0;
        return true;    // Todo ok.
}

/*************************************************************************
  Encola un bloque de datos para reproducir.
        Los datos apuntados se copian, por lo que el array original
         puede modificarse sin problemas. Devuelve true si ok
```

```
**********************************************************************/
bool CWaveOut::WriteBlock(DWORD len, BYTE* data)
{
        WAVEHDR         *header;         // La cabecera de datos a formar
        BYTE            *new_block;     // Nuevo bloque de memoria
        MMRESULT        res;

        // Reservar memoria:
        header = new WAVEHDR[1];
        new_block = new BYTE[len];
        memcpy(new_block, data,len);    // Copiar

        // Rellenar estructura:
        memset(header,0,sizeof(WAVEHDR));
        header->lpData = new_block;
        header->dwBufferLength = len;

        // Preparar para enviar:
        res = waveOutPrepareHeader( handle,   // wave-out handle
                                    header,  // La cabecera a preparar
                                    sizeof(WAVEHDR) );
        if (MMSYSERR_NOERROR!=res)
        {
                char    buf[500];
                waveOutGetErrorText(res, buf,500);
                return false;
        }

        // Enviar para reproducir:
        res = waveOutWrite( handle,   // Handle al wave-out
                            header,     // La cabecera
                            sizeof(WAVEHDR));
        if (MMSYSERR_NOERROR!=res)
        {
                char    buf[500];
                waveOutGetErrorText(res, buf,500);
                return false;
        }

        // Incrementar la cuenta de bloques reproduciendose:
        WaitForSingleObject(semaphore,INFINITE);
        nPreparedBlocks++;
        ReleaseSemaphore(semaphore, 1, NULL);

        return true;    // Todo OK
}


/**********************************************************************
**********************************************************************/
void CALLBACK CWaveOut::waveOutProc(HWAVEOUT hwo,UINT uMsg,
                DWORD dwInstance,DWORD dwParam1,DWORD dwParam2)
{
        CWaveOut        *obj = (CWaveOut*)dwInstance;

        if (uMsg==WOM_DONE)
        {       // Se ha terminado un paquete de reproduccion:
                //  Liberar la memoria:
                WAVEHDR *header = (WAVEHDR*)dwParam1;

                waveOutUnprepareHeader( hwo,    // Handle a wave-out
                                        header, // La cabecera
                                        sizeof(WAVEHDR));

                // Liberar ahora el buffer dentro de la cabecera del bloque
                //  de datos y la cabecera en si.
                delete[] header->lpData;
                delete[] header;

                // Decrementar la cuenta de bloques reproduciendose:
                WaitForSingleObject(obj->semaphore,INFINITE);
                obj->nPreparedBlocks--;
                ReleaseSemaphore(obj->semaphore, 1, NULL);
        }


}
```

```
/*************************************************************************
        Cambiar el volumen. Los valores van desde 0 hasta 0xFFFF
*************************************************************************/
bool CWaveOut::SetVolume(WORD  volLeft,WORD volRight)
{
        if (!handle) return false;

        // Cambiar volumen:
        DWORD vol = volLeft | (volRight << 16);
        waveOutSetVolume(handle,vol);
        return true;    // Todo OK
}


/*************************************************************************
    Devuelve la cantidad de bloques de audio actualmente encolados
      en reproduccion:
*************************************************************************/
int CWaveOut::GetCurrentlyEnqueued()
{
        int     res;
        WaitForSingleObject(semaphore,INFINITE);
        res = nPreparedBlocks;
        ReleaseSemaphore(semaphore, 1, NULL);
        return res;     // Devolver valor:
}
```

```
/*************************************************************************
 Fichero: CUsbSwRadioInterfaz.h

 Uso: Declara la clase "CUsbSwRadioInterfaz" que hace de interfaz
     al chip USB de la radio SW y permite enviar y recibir datos a este.


 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
*************************************************************************/

#ifndef CUsbSwRadioInterfazH
#define CUsbSwRadioInterfazH

#include <windows.h>
#include <math.h>

#include <Contnrs.hpp>  // Implementacion de TList
#include "ftd2xx.h"


// La funcion que se ejecutara en un hilo paralelo:
DWORD WINAPI HiloRx( LPVOID ptr);


//  Definiciones de constantes
//---------------------------------------------------------
#define CMD_IDENTIFICAR_FIRMWARE        0x10
#define CMD_BORRAR_LCD                  0x20
#define CMD_ESCRIBIR_LCD                0x21
#define CMD_CAMBIAR_MODO_PWDOWN         0x30
#define CMD_LEER_MODO_PWDOWN            0x31

#define FLAG_INICIO                     0x96
#define FLAG_FINAL                      0x69

// Los codigos de las respuestas se obtienen haciendo
//  un "OR" al codigo del comando.
#define CMD_RESPUESTA(cmd)              (cmd | 0x80)


class   CUsbSwRadioInterfaz
{
        public:
        // Estructura que almacena una trama:
        //   Se implementa como clase por comodidad.
```

```
	class  TTrama
	{
		public:
		BYTE    nCMD;
		BYTE    lonDatos;
		BYTE    datos[256];
	};

	// Estructura con datos de configuracion del PLL:
	enum MUXOUTVALUE
	{
		THREE_STATE_OUTPUT = 0,
		DIGITAL_LOCK_DETECT,
		N_DIVIDER,
		DVDD,
		R_DIVIDER,
		NOT_USED1,
		NOT_USED2,
		DGND
	};

	struct PLL_CONFIG_DATA
	{
		// Datos del R-counter latch (OK)
		WORD            R_14bits;               // Reference counter: 1-16383
		BYTE            AntiBacklashWidth;      // 0=3ns,1=1.5ns,2=6ns,3=3ns
		bool            LockDetectionPrecision; // true = 5 ciclos, false = 3
ciclos

		// Datos de N counter latch
		WORD            B_13bits;               // B: 3-8191

		// Datos del control latch (OK)
		BYTE            CorePowerLevel;         // 0:2.5mA, 1:5mA, 2:7.5mA,
3=10mA
		bool            CounterReset;           // false: Lo normal
		MUXOUTVALUE     MuxOut;                 // Selecciona lo que saldra por
MUXOUT
		bool            PhasePositive;          // Polaridad del PD
		bool            ChargePumpThreeState;   // false: Lo normal
		bool            CP_Gain_CurrentSelect;  // false: CurrentSetting1,
true:CurrentSetting2
		bool            MuteTilLock;            // true: no output till lock
		BYTE            OutputPowerLevel;       // 0:-9dBm, 1:-6dBm, 2:-3dBm,
3:-9dBm
		BYTE            CP_CurrentSetting1;     // 0-7
		BYTE            CP_CurrentSetting2;     // 0-7
		BYTE            PowerDown;              // Ver tabla-7 del datasheet del
"ADF4060-8"

	};


	private:
	// Esta funcion puede acceder a los miembros privados de esta clase:
	friend DWORD WINAPI HiloRx( LPVOID ptr);

	// Handle para el chip usb
	FT_HANDLE       handle;

	// Seccion critica para que solo un hilo acceda al chip USB a la vez:
	CRITICAL_SECTION CRITSECT_accesoUSB;

	// Handle para el hilo:
	HANDLE           hHilo;

	// Flags para comunicar al hilo que debe terminar:
	bool            hiloCerrando;
	bool            hiloCerrado;

	// En esta cola se guardan las tramas (TTrama*)
	//   recibidas por el hilo en paralelo.
	TQueue           *colaTramas;
	CRITICAL_SECTION CRITSECT_accesoColaTramas;


	// Momento de la ultima actividad registrada. Utilizado para
```

```
        //  la desconexion automatica tras tiempo sin actividad.
        DWORD           timUltimaActividad;


        // Abre el dispositivo para su uso. Devuelve TRUE si todo OK
        bool    AbrirDispositivo();
        // Cierra el dispositivo tras su uso. Devuelve TRUE si todo OK
        bool    CerrarDispositivo();

        // Para estadisticas de tramas:
        __int64 estad_nTramas_Rx;
        __int64 estad_nTramas_Rx_Err;

        // Funcion para averiguar si hay bytes en cola de lectura
        bool HayBytesEnCola(DWORD &bytesEnCola);


        public:
        // Constructor:
        CUsbSwRadioInterfaz();

        // Destructor:
        ~CUsbSwRadioInterfaz();

        // Devuelve true si el dispositivo esta abierto:
        bool    EstaAbierto();

        // Envia una trama. Devuelve TRUE si todo OK.
        bool    EnviarTrama( BYTE nCMD, BYTE lonDatos, BYTE* datos );

        // Comprueba si hay alguna trama de respuesta recibida:
        bool    HayTramasDeRespuestas();


        // Para acceso a la cola de tramas a traves de Seccion Critica
        //  que evita problemas entre hilos:
        bool    colaTramas_AtLeast(int n);
        TTrama *colaTramas_Pop();
        void    colaTramas_Push(TTrama* trama);

        // Devuelve una trama de las respondidas por la radio.
        //  NOTA: El usuario debe llamar a "delete" sobre este puntero!!
        // (Es igual a la "colaTramas_Pop", pero su nombre es mas claro
        //   para usarlo desde fuera del objeto)
        TTrama* CogerTramaDeRespuesta();


        // Obtencion de estadisticas de tramas recibidas y erroneas:
        void    EstadisticasTramas( __int64 &nTramasRxTotal, __int64 &nTramasRxError,
bool Reset );

        // Peticiones sincronas: Se envia un comando y se bloquea esperando
        //  la respuesta o el cumplimiento del timeout de respuesta:
        //  Todas devuelven TRUE si todo va OK
        // ---------------------------------------------------------------
        bool    PeticionSinc_VersionFirmware( char *sVersion );
        bool    PeticionSinc_LCD_Borrar();
        bool    PeticionSinc_LCD_Escribir(BYTE columna, BYTE fila, char* texto);
        bool    PeticionSinc_LeerPowerdown( bool *estado_actual );
        bool    PeticionSinc_CambiarPowerdown( bool nuevo_estado );
        bool    PeticionSinc_CambiarDAC( BYTE nDAC, BYTE valor );
        bool    PeticionSinc_ConfiguraPLL( PLL_CONFIG_DATA *pll_config  );
        bool    PeticionSinc_LeerMuxoutPLL( BYTE * array255Bytes  );

        bool    PeticionSinc_ModoBootLoader();
        bool    PeticionSinc_BOOTLDR_Reinicia();
        bool    PeticionSinc_BOOTLDR_LeeFlash(WORD flashDir, BYTE *paginaLeida);
        bool    PeticionSinc_BOOTLDR_GrabaFlash(WORD flashDir, BYTE *paginaAGrabar);

        // Peticiones asincronas: Se envia un comando y no espera la
        //   respuesta.
        //  Todas devuelven TRUE si transmision va OK
        // ---------------------------------------------------------------
        bool    PeticionAsinc_CambiarDAC( BYTE nDAC, BYTE valor );
        bool    PeticionAsinc_ConfiguraPLL( CUsbSwRadioInterfaz::PLL_CONFIG_DATA
*pll_config  );
        bool    PeticionAsinc_LCD_Escribir(BYTE columna, BYTE fila, char* texto);
```

```
};

#endif
```

```
/*************************************************************************
 Fichero: CUsbSwRadioInterfaz.cpp

 Uso: Implementa la clase "CUsbSwRadioInterfaz" que hace de interfaz
     al chip USB de la radio SW y permite enviar y recibir datos a este.

      Utiliza un hilo que en paralelo recibe los datos del chip USB.

 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
*************************************************************************/
#include "CUsbSwRadioInterfaz.h"

// "Defines" que configuran el comportamiento del objeto:
// ----------------------------------------------------------
#define TIMEOUT_RECIBIENDO_TRAMA        250     // ms para timeout entre bytes de una
trama
#define TIMEOUT_DESCONEXION_AUTOMATICA  5000    // ms para cerrar handle automaticamente
si no hay actividad
#define TIMEOUT_ESPERAR_RESPUESTA       1000     // ms de espera a respuesta de trama
(funciones sincronas)
#define TAM_BUFFER_TEMPORAL             32*1024 // Buffer para el hilo de recepcion USB
#define TAM_MAX_COLA_TRAMAS             19000   // Suficiente para unos 3 segundos de
muestreo

/*-------------------------------------------------------------------
        Constructor

  -------------------------------------------------------------------*/
CUsbSwRadioInterfaz::CUsbSwRadioInterfaz()
{
        // Iniciar handle para saber que no esta abierto:
        handle = NULL;

        // Crear cola de tramas entrantes:
        colaTramas = new TQueue();

        // Iniciar la seccion critica para acceso a la cola de tramas:
        InitializeCriticalSection(&CRITSECT_accesoColaTramas);

        // Iniciar la seccion critica para E/S del chip USB:
        InitializeCriticalSection(&CRITSECT_accesoUSB);


        // Crear hilo de recepcion:
        hiloCerrando=false;
        hiloCerrado=false;

        hHilo = CreateThread(
                NULL,                   // Security Attrbs.
                0,                      // Stack size
                HiloRx,                 // La funcion que se ejecuta en el hilo
                this,                   // El parametro a la funcion
                0,                      // Flags
                (unsigned long*)hHilo
                );

        if (!hHilo)
                MessageBox(NULL, "Error al crear
'thread'!!","CUsbSwRadioInterfaz::CUsbSwRadioInterfaz",MB_OK | MB_ICONSTOP);

        // Iniciar variables de estadisticas:
        estad_nTramas_Rx=0;
        estad_nTramas_Rx_Err=0;

}

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

```
  ------------------------------------------------------------------*/
CUsbSwRadioInterfaz::~CUsbSwRadioInterfaz()
{
        // Terminar hilo:
        hiloCerrando=true;
        while (!hiloCerrado) Sleep(1);
        CloseHandle(hHilo);

        // Limpiar cola de tramas:
        while ( colaTramas_AtLeast(1) )
                delete colaTramas_Pop();
        delete colaTramas;


        // Si el dispositivo USB esta abierto, cerrarlo:
        if (handle)
                CerrarDispositivo();

        // Liberar la seccion critica para acceso a la cola de tramas:
        DeleteCriticalSection(&CRITSECT_accesoColaTramas);

        // y la del chip USB:
        DeleteCriticalSection(&CRITSECT_accesoUSB);

}

/*------------------------------------------------------------------
        AbrirDispositivo

    Es la primera funcion que hay que llamar para comunicarse
      con el dispositivo.

    Devuelve: TRUE si todo OK, FALSE si hay errores.
  ------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::AbrirDispositivo()
{
        // Por si ya estuviera abierto:
        CerrarDispositivo();

        // Limpiar cola de tramas de entrada:
        while ( colaTramas_AtLeast(1) )
                delete (TTrama*) colaTramas_Pop();


        // Para timeout de desconexion automatica:
        timUltimaActividad= GetTickCount();

        // Abrir por el nombre del descriptor:

        if (FT_OK!=
//            FT_Open(0, &handle)
           FT_OpenEx(
                "RadioSoftware @ JLBC'04",
                FT_OPEN_BY_DESCRIPTION,
                &handle )
                     ) return false;   // Error

        // Iniciar:
        if (FT_OK!= FT_ResetDevice( handle ) ) return false;

        if (FT_OK!= FT_Purge( handle, FT_PURGE_RX | FT_PURGE_TX ) ) return false;

        // 5 ms para lectura, 300ms para escritura.
        if (FT_OK!= FT_SetTimeouts( handle,5,300 ) ) return false;

        // Latencia
        if (FT_OK!= FT_SetLatencyTimer( handle, 4 ) ) return false;

        // Tamaño de paquetes USB:
        if (FT_OK!= FT_SetUSBParameters( handle, 4096 , 0 ) ) return false;



        return true;
}
```

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

    Es la primera funcion que hay que llamar para comunicarse
      con el dispositivo.

    Devuelve: TRUE si todo OK, FALSE si hay errores.
  -----------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::CerrarDispositivo()
{
        EnterCriticalSection(&CRITSECT_accesoUSB);
        FT_STATUS res=FT_Close( handle );
        LeaveCriticalSection(&CRITSECT_accesoUSB);

        if (FT_OK!=res)
        {
                handle = NULL;  // El handle es invalido, asi que anularlo
                return false;
        }

        // Indicar que esta cerrado:
        handle = NULL;

        return true;
}

/*-----------------------------------------------------------------
Devuelve true si el dispositivo esta abierto:
  -----------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::EstaAbierto()
{
        return handle != 0;
}

/*-----------------------------------------------------------------
        Envia una trama. Devuelve TRUE si todo OK.
  -----------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::EnviarTrama( BYTE nCMD, BYTE lonDatos, BYTE* datos )
{
        BYTE    buffer[300];
        DWORD   nBytesAEnviar;
        DWORD   nBytesEnviados;

        // Si no esta abierto, intentar abrir ahora:
        if (!handle)
                if (! AbrirDispositivo() )
                        return false;


        // Rellenar buffer:
        nBytesAEnviar=0;

        // Flag de inicio:
        buffer[nBytesAEnviar++]= FLAG_INICIO;

        // Longitud del resto de datos:
        buffer[nBytesAEnviar++]= lonDatos+1;

        // Codigo de comando:
        buffer[nBytesAEnviar++]= nCMD;

        // Datos:
        if (lonDatos && datos)
                memcpy( buffer+nBytesAEnviar, datos, lonDatos );

        nBytesAEnviar+=lonDatos;


        // Flag de fin:
        buffer[nBytesAEnviar++]= FLAG_FINAL;


        // Enviar datos:
        EnterCriticalSection(&CRITSECT_accesoUSB);
        FT_STATUS res=
                FT_Write(
```

```
                        handle,                 // Handle al disp. USB
                        buffer,                 // Datos
                        nBytesAEnviar,          // Bytes a enviar
                        &nBytesEnviados         // Bytes enviados con exito
                );

        // Para timeout de desconexion automatica:
        timUltimaActividad= GetTickCount();

        LeaveCriticalSection(&CRITSECT_accesoUSB);

        if (FT_OK!= res)
                  return false;

        // Asegurar que se ha enviado correctamente:
        if ( nBytesEnviados!=nBytesAEnviar )
                return false;


        // Todo OK:
        return true;
}

/*-------------------------------------------------------------------
        Devuelve "true" si hay hay bytes en cola
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::HayBytesEnCola(DWORD &bytesEnCola)
{
        bool res;

        EnterCriticalSection(&CRITSECT_accesoUSB);

        if (FT_OK!=FT_GetQueueStatus(handle, &bytesEnCola )) res = false;
        else res= bytesEnCola!=0;

        LeaveCriticalSection(&CRITSECT_accesoUSB);

        return res;
}

/*-------------------------------------------------------------------
   La funcion que se ejecuta en un hilo paralelo y se encarga
     de la entrada de datos
  -------------------------------------------------------------------*/
DWORD WINAPI HiloRx( LPVOID ptr)
{
        CUsbSwRadioInterfaz     *obj;

        // En este buffer se guardara desde el flag inicial hasta el final:
        BYTE                    bufferTemporal[TAM_BUFFER_TEMPORAL];
        BYTE                    tramaTemporal[300];
        WORD                    lonBuffer;

        // Para las operaciones de lectura:
        DWORD                   bytesEnCola;
        DWORD                   bytesLeidos;

        // Para posibles timeouts:
        DWORD                   timComienzoRxTrama;
        DWORD                   timActual;

        // Iniciar:
        lonBuffer=0;

        // Recuperar puntero al objeto:
        obj = (CUsbSwRadioInterfaz*)ptr;

        // Asignarnos prioridad:
        SetThreadPriority( GetCurrentThread(), THREAD_PRIORITY_TIME_CRITICAL );

        FT_STATUS ft_res;

        // Bucle central:
        while ( ! obj->hiloCerrando )
        {
         try
         {
```

```
		// Si el dispositivo esta abierto, mirar si hay
		//  algo en la cola de entrada:
		if (obj->handle)
		{
		   // Mientras haya datos que leer, leerlos:
		   while ( obj->HayBytesEnCola(bytesEnCola) )
		   {
			DWORD     nBytesARecibir;

			// Para timeout de desconexion automatica:
			obj->timUltimaActividad= GetTickCount();

			EnterCriticalSection(&obj->CRITSECT_accesoUSB);

			// Ver cuantos bytes hay:
			FT_GetQueueStatus(obj->handle, &bytesEnCola );
			nBytesARecibir = min( bytesEnCola, (DWORD)TAM_BUFFER_TEMPORAL );

			ft_res=FT_Read(
				obj->handle,		// Handle
				bufferTemporal,		// Destino de los datos
				nBytesARecibir,		// Espacio disponible
				&bytesLeidos );
			LeaveCriticalSection(&obj->CRITSECT_accesoUSB);

			// En este caso salimos del while: (Algun error?)
			if ( nBytesARecibir != bytesLeidos )
				bytesEnCola=0;

			if (FT_OK==ft_res && bytesLeidos)
				{

					// Ahora procesar los bytes de uno en uno o en
grupos:
					// ----------------------------------------------
--------
					BYTE    *rx_bytes;      // Apuntara a la
posicion en analisis en cada momento

					DWORD     buffer_indx = 0;

					while ( buffer_indx < bytesLeidos )
					{
						int nBytesAProcesar;

						switch (lonBuffer)
						{
							case 0:
							case 1:
								nBytesAProcesar= 1;
// Solo uno, porque hay que comprobar si es el flag
								break;
							default:
								// Lo que indique <LON>
								nBytesAProcesar= min(
									(int)(
bytesLeidos - buffer_indx ),  // Bytes en el buffer

(int)(tramaTemporal[1]+3-lonBuffer)        // Bytes que se deberian recibir en <datos>
									);
								if (nBytesAProcesar<0 ||
nBytesAProcesar>TAM_BUFFER_TEMPORAL)
								{

MessageBox(NULL,"ASSERT!!","nBytesAProcesar en hilo de recepcion!",0);
									return -1;
								}
								break;

						}

						// Si esperabamos flag inicial, no hacer
caso a otra cosa:
						if (lonBuffer>0 || (!lonBuffer &&
bufferTemporal[buffer_indx]==0x96))
						{
```

```
                                                        // Procesar esos bytes:
Guardarlos en un buffer de trama temporal:
                                                        memcpy( tramaTemporal+lonBuffer,
&bufferTemporal[buffer_indx] ,nBytesAProcesar );
                                                        // Incrementar punteros y
contadores:
                                                        lonBuffer += nBytesAProcesar;
                                                        // Si acabamos de empezar,
guardar "timestamp" para
                                                        //  un posible timeout
                                                        if (lonBuffer==1)
                                                                timComienzoRxTrama =
GetTickCount();
                                                        // Trama completa??
                                                        if (  lonBuffer ==
(tramaTemporal[1]+3) )
                                                        {
                                                                // Actualizar
estadisticas:
                                                                obj->estad_nTramas_Rx++;
                                                              // Se ha recibido el flag
de final correctamente?
                                                              if (
FLAG_FINAL==tramaTemporal[ tramaTemporal[1]+2 ] )
                                                              {
                                                                // Copiar la trama
completa a la cola de entrada
                                                                //  del objeto:
CUsbSwRadioInterfaz::TTrama  *trama = new CUsbSwRadioInterfaz::TTrama();
                                                                // Rellenar con los
datos:
                                                                trama->nCMD     =
tramaTemporal[2];
                                                                trama->lonDatos =
tramaTemporal[1]-1;
                                                                memcpy( trama->datos,
&tramaTemporal[3],
                                                                        trama->lonDatos
);
                                                                // Evitar sobrecarga de
la cola:
                                                                if ( ! obj-
>colaTramas_AtLeast( TAM_MAX_COLA_TRAMAS ) )
                                                                        obj-
>colaTramas_Push( trama );
                                                                else    delete trama;
                                                                // Limpiar buffer y
esperar nuevas tramas:
                                                                lonBuffer=0;
                                                                }
                                                                else
                                                                {
                                                                        // Trama mal
formada:
                                                                        // Actualizar
estadisticas:
                                                                        obj-
>estad_nTramas_Rx_Err++;
                                                                        // Limpiar
buffer y esperar nuevas tramas:
                                                                        lonBuffer=0;
                                                                }
                                                        }
                                                }
                                                buffer_indx += nBytesAProcesar;
```

```
                                        }       // Fin del "while" procesando bytes
leidos
                                }

                }       // Fin del "while" haya bytes en cola

                // Si no se ha completado una trama en un tiempo dado, descartarla:
                timActual=GetTickCount();
                if (lonBuffer && (timActual-timComienzoRxTrama)>
TIMEOUT_RECIBIENDO_TRAMA )
                        lonBuffer=0;

                // Se ha cumplido timeout de desconexion por no actividad?
                if ( ( GetTickCount() - obj->timUltimaActividad ) >
TIMEOUT_DESCONEXION_AUTOMATICA )
                        obj->CerrarDispositivo();

            }
            Sleep(1);
         } // Fin del try, por si hay alguna excepcion
         catch (...) { }
        }       // Fin del bucle principal:

        obj->hiloCerrado=true;

        return 0;
}


/*-------------------------------------------------------------------
   Devuelve true si hay tramas de respuestas en la cola de entrada
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::HayTramasDeRespuestas()
{
        return colaTramas_AtLeast(1);;
}

/*-------------------------------------------------------------------
       Devuelve una trama de respuesta
  -------------------------------------------------------------------*/
CUsbSwRadioInterfaz::TTrama* CUsbSwRadioInterfaz::CogerTramaDeRespuesta()
{
        return colaTramas_Pop();
}

/*-------------------------------------------------------------------
   Pide version a la radio y espera respuesta.
       Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_VersionFirmware( char *sVersion )
{
        // Enviar CMD 0x10
        BYTE    cmd = 0x10;
        if (! EnviarTrama(cmd, 0,NULL) )
                return NULL;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) < TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
```

```
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta
        memcpy( sVersion, trama->datos, trama->lonDatos);
        sVersion[trama->lonDatos] = (char)0;

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}

/*-------------------------------------------------------------------
   Pone la radio en modo BootLoader y espera respuesta.
       Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_ModoBootLoader()
{
        // Enviar CMD 0x70
        BYTE    cmd = 0x70;
        if (! EnviarTrama(cmd, 0,NULL) )
                return NULL;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) < TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}

/*-------------------------------------------------------------------
   Ejecuta el programa principal y espera respuesta.
   Sirve para salir del modo BootLoader
       Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_BOOTLDR_Reinicia()
{
        // Enviar CMD 0x71
        BYTE    cmd = 0x71;
        if (! EnviarTrama(cmd, 0,NULL) )
                return NULL;

        // Esperar respuesta:
```

```
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) < TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}


/*-----------------------------------------------------------------
   Lee una pagina (64bytes) de flash
        Devuelve true si todo OK
  -----------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_BOOTLDR_LeeFlash(WORD flashDir, BYTE
*paginaLeida)
{
        // Enviar CMD 0x72
        BYTE    cmd = 0x72;

        BYTE    _flashDir[2];
        _flashDir[0]= HIBYTE( flashDir );
        _flashDir[1]= LOBYTE( flashDir );

        if (! EnviarTrama(cmd, 2,_flashDir) )
                return NULL;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) < TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }
```

```
        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta
        memcpy( paginaLeida, trama->datos, 64);

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}


/*-----------------------------------------------------------------
   Graba una pagina (64bytes) de flash
       Devuelve true si todo OK
  -----------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_BOOTLDR_GrabaFlash(WORD flashDir, BYTE
*paginaAGrabar)
{
        // Enviar CMD 0x73
        BYTE    cmd = 0x73;

        BYTE    buffer[64+2];
        buffer[0]= HIBYTE( flashDir );
        buffer[1]= LOBYTE( flashDir );
        memcpy( buffer+2,paginaAGrabar, 64 );

        if (! EnviarTrama(cmd,64+2,buffer) )
                return false;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) <   2 * TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}

/*-----------------------------------------------------------------
   Borra el display LCD de la radio
       Devuelve true si todo OK
  -----------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_LCD_Borrar()
{
        // Enviar CMD 0x20
        BYTE    cmd = 0x20;
```

```
        if (! EnviarTrama(cmd,0,NULL) )
                return NULL;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) <   2 * TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}

/*-------------------------------------------------------------------
   Escribe un mensaje en el display LCD de la radio
        Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionAsinc_LCD_Escribir(BYTE columna, BYTE fila, char*
texto)
{
        // Enviar CMD 0x21
        BYTE    cmd = 0x21;

        BYTE    buffer[300];
        buffer[0]= columna;
        buffer[1]= fila;
        strcpy( buffer+2, texto );
        int     lonBuf = 2 + strlen(texto) + 1; // 1 del '\0'

        return EnviarTrama(cmd,lonBuf,buffer);
}


/*-------------------------------------------------------------------
   Escribe un mensaje en el display LCD de la radio
        Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_LCD_Escribir(BYTE columna, BYTE fila, char*
texto)
{
        // Enviar CMD 0x21
        BYTE    cmd = 0x21;

        // Enviar peticion:
        if (!PeticionAsinc_LCD_Escribir(columna,fila,texto))
                return false;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta
```

```
        // Esperar
        while ( (GetTickCount() - timInicio) <   2 * TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}

/*-------------------------------------------------------------------
   Lee el estado de PowerDown de la radio.
    TRUE = en estado powerdown.
       Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_LeerPowerdown( bool *estado_actual )
{
        // Enviar CMD 0x31
        BYTE    cmd = 0x31;

        if (! EnviarTrama(cmd,0,NULL) )
                return NULL;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) <   2 * TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta
        *estado_actual = (trama->datos[0] != 0x00);

        // Liberar:
        delete trama;
```

```
        // Todo OK:
        return true;
}

/*-------------------------------------------------------------------
   Cambia el estado de PowerDown de la radio.
    TRUE = en estado powerdown.
       Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_CambiarPowerdown( bool nuevo_estado )
{
        // Enviar CMD 0x30
        BYTE    cmd = 0x30;

        BYTE    buf[1];
        buf[0] = nuevo_estado ? 0x01:0x00;

        if (! EnviarTrama(cmd,1,buf) )
                return NULL;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) <   2 * TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}

/*-------------------------------------------------------------------
   Cambia el valor de un DAC
     nDAC  = num. de DAC (0,1,2,3)
     valor = nuevo valor
       Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionAsinc_CambiarDAC( BYTE nDAC, BYTE valor )
{
        // Enviar CMD 0x32
        BYTE    cmd = 0x32;

        BYTE    buf[2];
        buf[0] = nDAC;
        buf[1] = valor;

        return EnviarTrama(cmd,2,buf);
}

/*-------------------------------------------------------------------
   Cambia el valor de un DAC
```

```
     nDAC  = num. de DAC (0,1,2,3)
     valor = nuevo valor
       Devuelve true si todo OK
  ------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_CambiarDAC( BYTE nDAC, BYTE valor )
{
        // Enviar CMD 0x32
        BYTE    cmd = 0x32;

        // Enviar peticion:
        if (!PeticionAsinc_CambiarDAC(nDAC, valor))
                return false;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) <   2 * TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}

/*------------------------------------------------------------------
   Cambia la configuracion del PLL
     La configuracion deseada va en la estructura de entrada.
       Devuelve true si todo OK
  ------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionAsinc_ConfiguraPLL(
CUsbSwRadioInterfaz::PLL_CONFIG_DATA *pll_config  )
{
        // ------------------------------------------------------------------
        // Segun el Datasheet del ADF4060-8, hay que usar siempre esta secuencia
        //  a escribir los registros del PLL:
        //    1º: R Counter
        //    2º: Control
        //    3º: N Counter (Contador "B")
        // ------------------------------------------------------------------


        // Enviar 3 comanos 0x33, uno por cada registro del PLL
        // ------------------------------------------------------
        BYTE    cmd = 0x33;
        BYTE    buf[3];


        // Reference register: OK!!!
        // ----------------------------
        buf[0] = buf[1] = buf[2] = 0;

        buf[2] |= 0x01;         // Selecciono registro
```

```
        buf[2] |= ((pll_config->R_14bits & 0x003F) << 2);
        buf[1]  = pll_config->R_14bits  >> 6;
        buf[0] |= ( (pll_config->AntiBacklashWidth & 0x03) << 0 );
        if (pll_config->LockDetectionPrecision) buf[0] |= ( 1 << 2 );

        if (! EnviarTrama(cmd,3,buf) ) return NULL;

        // Control latch: OK!!
        // ----------------------------
        buf[0] = buf[1] = buf[2] = 0;

        buf[2] |= 0x00;         // Selecciono registro

        buf[2] |= ((pll_config->CorePowerLevel & 0x03) << 2);
        if (pll_config->CounterReset)   buf[2] |= (1 << 4);
        buf[2] |= ((pll_config->MuxOut & 0x07) << 5);

        if (pll_config->PhasePositive) buf[1] |= (1 << 0);
        if (pll_config->ChargePumpThreeState ) buf[1] |= (1 << 1);
        if (pll_config->CP_Gain_CurrentSelect ) buf[1] |= (1 << 2);
        if (pll_config->MuteTilLock ) buf[1] |= (1 << 3);
        buf[1] |= ((pll_config->OutputPowerLevel  & 0x03) << 4);

        buf[1] |= (((pll_config->CP_CurrentSetting1 >> 0 ) & 0x03) << 6);
        buf[0] |= (((pll_config->CP_CurrentSetting1 >> 2 ) & 0x01) << 0);
        buf[0] |= (((pll_config->CP_CurrentSetting2  ) & 0x07) << 1);

        buf[0] |= (((pll_config->PowerDown ) & 0x03) << 4);

        if (! EnviarTrama(cmd,3,buf) ) return NULL;

        // Revisar este retardo!!!
        Sleep(20);

        // N register:
        // ----------------------------
        buf[0] = buf[1] = buf[2] = 0;

        buf[2] |= 0x02;         // Selecciono registro

        buf[1] |= ((pll_config->B_13bits & 0x00FF) << 0);
        buf[0] |= (((pll_config->B_13bits >> 8) & 0x01F) << 0);

        if (pll_config->CP_Gain_CurrentSelect ) buf[0] |= (1 << 5);

        if (! EnviarTrama(cmd,3,buf) ) return NULL;

        return true;    // Todo OK!
}

/*-------------------------------------------------------------------
   Cambia la configuracion del PLL
     La configuracion deseada va en la estructura de entrada.
       Devuelve true si todo OK
  -------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_ConfiguraPLL(
CUsbSwRadioInterfaz::PLL_CONFIG_DATA *pll_config  )
{
        // Enviar CMD 0x33
        BYTE    cmd = 0x33;

        // Enviar peticion:
        if (!PeticionAsinc_ConfiguraPLL( pll_config ))
                return false;


      // Esperar las 3 respuestas:
      for (int i=0;i<3;i++)
      {
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) <   TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
```

```
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta

        // Liberar:
        delete trama;
      }


        // Todo OK:
        return true;
}



/*-----------------------------------------------------------------------
   Realiza un muestreo de 255*8 = 2040 muestras de 1 bit, cada 2 us
     del pin MUXOUT del PLL.

     (4 ms para muestrear)

       Devuelve true si todo OK
  -----------------------------------------------------------------------*/
bool CUsbSwRadioInterfaz::PeticionSinc_LeerMuxoutPLL( BYTE * array255Bytes  )
{
        // Enviar CMD 0x34
        BYTE    cmd = 0x34;

        if (! EnviarTrama(cmd,0,NULL) )
                return NULL;

        // Esperar respuesta:
        DWORD   timInicio = GetTickCount();
        bool    recibido  = false;
        TTrama  *trama    = NULL;       // Para la respuesta

        // Esperar
        while ( (GetTickCount() - timInicio) <   2 * TIMEOUT_ESPERAR_RESPUESTA &&
                !recibido )
        {
                if ( colaTramas_AtLeast(1) )
                {
                        // Ver si es la trama de respuesta a nuestro comando:
                        trama = (TTrama*) colaTramas_Pop();

                        recibido = trama->nCMD == (cmd | 0x80);
                        if (!recibido)
                        {
                                delete trama;
                                trama=NULL;
                        }
                }
                Sleep(1);
        }

        // Ha sido timeout?
        if (!trama) return false;

        // En "trama" esta la respuesta
        memcpy( array255Bytes,
                trama->datos,
```

```
                255
               );

        // Liberar:
        delete trama;

        // Todo OK:
        return true;
}


/*-------------------------------------------------------------------
        Dice si hay al menos "n" tramas en la cola.

    Es "thread-safe" al usar una "Critical Zone"
  -------------------------------------------------------------------*/
bool    CUsbSwRadioInterfaz::colaTramas_AtLeast(int n)
{
        EnterCriticalSection(&CRITSECT_accesoColaTramas);
        bool    res=colaTramas->AtLeast(n);
        LeaveCriticalSection(&CRITSECT_accesoColaTramas);
        return res;
}

/*-------------------------------------------------------------------
        Devuelve la siguiente trama de la cola o NULL si no hay

    Es "thread-safe" al usar una "Critical Zone"
  -------------------------------------------------------------------*/
CUsbSwRadioInterfaz::TTrama *CUsbSwRadioInterfaz::colaTramas_Pop()
{
        TTrama* trama;

        EnterCriticalSection(&CRITSECT_accesoColaTramas);
        if (colaTramas->AtLeast(1))
                trama = (TTrama*) colaTramas->Pop();
        else    trama = NULL;
        LeaveCriticalSection(&CRITSECT_accesoColaTramas);

        return trama;
}

/*-------------------------------------------------------------------
        Permite añadir una trama a la cola

    Es "thread-safe" al usar una "Critical Zone"
  -------------------------------------------------------------------*/
void CUsbSwRadioInterfaz::colaTramas_Push(CUsbSwRadioInterfaz::TTrama* trama)
{
        EnterCriticalSection(&CRITSECT_accesoColaTramas);
        colaTramas->Push(trama);
        LeaveCriticalSection(&CRITSECT_accesoColaTramas);
}


/*-------------------------------------------------------------------
    Devuelve las estadisticas de tramas.

  -------------------------------------------------------------------*/
void    CUsbSwRadioInterfaz::EstadisticasTramas( __int64 &nTramasRxTotal, __int64
&nTramasRxError, bool Reset )
{
        nTramasRxTotal = estad_nTramas_Rx;
        nTramasRxError = estad_nTramas_Rx_Err;

        if (Reset)
        {
                estad_nTramas_Rx=0;
                estad_nTramas_Rx_Err=0;
        }

}
```

```
/*****************************************************************
FICHERO: comunSWDemods.h
```

```
USO: Declara las estructuras de datos usadas entre la DLL de demodulacion
   y la aplicacion SWRadioStudio
*********************************************************************/

#ifndef COMUNSWDEMODS_H
#define COMUNSWDEMODS_H


// Definiciones de posibles mensajes de callback para componentes GUI:
// ---------------------------------------------------------------------
// El boton indicado se ha pulsado. No se usan los parametros.
#define GUIEVENT_BUTTON_PUSHED          1
// El checkbox indicado se ha pulsado. "Param1" contiene 1/0 segun el nuevo
//  estado del checkbox.
#define GUIEVENT_CHECKBOX_TOOGLE        2

namespace DemodSW
{

// Estructura usada para describir al demodulador.
// ---------------------------------------------------------------------
struct DescripcionDemodulador
{
        char    nombreModulacion[100];  // Texto para el nombre del modulo
        char    version[10];            // Texto para la version
        char    autor[100];             // Texto para el autor
        char    comentarios[2000];      // Comentarios varios.

        bool    analogica;              // Si se trata de una mod. analogica o digital
        int     rateSalida;             // Muestras por segundo a la salida.
};

// Estructura usada para pasarle al demodulador todas las funciones
//   que necesita para interaccionar con el entorno gráfico.
// ---------------------------------------------------------------------
struct  GUI_Funciones
{
        // Creacion de componentes:
        void (*GUI_Create_Button)(char *name,char *text, int X, int Y, int LX, int LY );
        void (*GUI_Create_Label)(char *name,char *text, int X, int Y );
        void (*GUI_Create_Edit)(char *name,char *text, int X, int Y, int LX, int LY );
        void (*GUI_Create_CheckBox)(char *name,char *text, int X, int Y, bool
init_state);

        // Destruccion de componentes:
        void (*GUI_Free_Component)(char *name);

        // Cambio de componentes.
        void (*GUI_Change_Text)(char *name,char *new_text);
        void (*GUI_Get_Text)(char *name,char *new_text);
        bool (*GUI_Get_CheckBox_State)(char *name);
};



};      // Fin del namespace


#endif
```

```
/*********************************************************************
 Fichero: CManejadorFicherosHEX.h

 Uso: Declara la clase "CManejadorFicherosHEX" que se usa para leer
     y escribir ficheros con el formato Intel HEX

 Jose Luis Blanco Claraco @ AGO-2004
 PFC ETSIT Malaga
*********************************************************************/
#ifndef CManejadorFicherosHEXH
#define CManejadorFicherosHEXH

#include <stdio.h>
```

```
#include <windows.h>


class CManejadorFicherosHEX
{
  public:
        // Guarda un array de bytes como fichero HEX:
        //  Devuelve FALSE si hay algun error al crear fichero,etc..
        bool    GuardaFicheroHEX( DWORD nBytes, BYTE* bytes, char *fichero );

        // Lee un fichero HEX y guarda los bytes en un array
        //  Devuelve FALSE si hay algun error al leer el fichero.
        bool    CargaFicheroHEX( DWORD tamBuffer, BYTE* buffer, char *fichero );
};


#endif
```

```
/*************************************************************************
FICHERO: CListadorDirectorios.cpp
USO: Implementa una clase que permite listar el contenido de un
   directorio.
*************************************************************************/

#include <vcl.h>
#pragma hdrstop

#include "CListadorDirectorios.h"

/*************************************************************************
                        Constructor

   Se le pasa en "directorio" una ruta y una mascara (ej: "c:\test\*.txt")
    y en "tipos" una mascara a base de OR's lógicos de los atributos de
    ficheros a buscar. Los mas normales:
     FILE_ATTRIBUTE_ARCHIVE
     FILE_ATTRIBUTE_DIRECTORY

*************************************************************************/
CListadorDirectorios::CListadorDirectorios( char *directorio, DWORD tipos )
{
        // Inicializar:
        ficheroEncontrado=false;
        nLista = 0;
        listaEncontrados=NULL;

        // Guardar mascara:
        mascaraTipos = tipos;

        // Iniciar una busqueda:
        handle = FindFirstFile(
                directorio,
                &w32fd
                );

        // Puede que no haya ningun fichero:
        if (handle == INVALID_HANDLE_VALUE)
        {
                handle = NULL;
                return;
        }

        ficheroEncontrado=true;

        // Seguir buscando hasta encontrar uno que cumpla la mascara:
        while ( ficheroEncontrado )
        {
                // Si el fichero cumple el filtro de mascara, añadirlo
                //  a la lista:
                if ( mascaraTipos & w32fd.dwFileAttributes )
                {
                        nLista++;

                        // Reservar mas memoria o reservar si es la primera vez
                        if (listaEncontrados)
                                // Incrementar tamaño de la reserva:
                                listaEncontrados = (WIN32_FIND_DATA*)
```

```
                                        realloc( listaEncontrados,
sizeof(WIN32_FIND_DATA)*nLista );
                        else    // Por primera vez:
                                listaEncontrados = (WIN32_FIND_DATA*)
                                        malloc( sizeof(WIN32_FIND_DATA) );

                        // Copiar datos en array:
                        memcpy( &listaEncontrados[nLista-1],
                                &w32fd,
                                sizeof(WIN32_FIND_DATA) );


                }


                // Buscar siguiente:
                ficheroEncontrado=FindNextFile( handle, &w32fd );
        }

        // Cerrar handle:
        if (handle) FindClose( handle );

}


/*************************************************************************
                        Destructor

*************************************************************************/
CListadorDirectorios::~CListadorDirectorios(  )
{

        // Liberar memoria:
        if (nLista)
                free(listaEncontrados);
}



/*************************************************************************
  Cuantos()
*************************************************************************/
DWORD CListadorDirectorios::Cuantos( )
{
        return nLista;
}

/*************************************************************************
  CogerDeLista()
*************************************************************************/
WIN32_FIND_DATA* CListadorDirectorios::CogerDeLista( DWORD indice )
{
        if (indice<0 || indice>=nLista) return NULL;
        return &listaEncontrados[indice];
}
```

```
/*************************************************************************
FICHERO: CGoertzel.cpp
USO:
                        Algoritmo de Goertzel

   Este algoritmo es muy util para detectar tonos sin tener
    que calcular la FFT completa. Solamente la calcula para un "bin"
    dado. Eso si, tambien necesita un bloque de N datos para generar
    un solo valor de salida, compuesto de partes real e imaginaria.
   El uso basico es a traves del metodo "ProcessSample", llamandolo
    para cada una de las N muestras. A la N'esima muestra devolvera
    el valor de la FFT en el "bin" solicitado en el constructor.

   Referencia:
     http://www.embedded.com/story/OEG20020819S0057
 *************************************************************************/

#include <stdlib.h>
#include <mem.h>
```

```
#include <math.h>

#include "CGoertzel.h"


/*************************************************************************
        Constructor
*************************************************************************/
CGoertzel::CGoertzel(float freq_sample,float freq_target,int block_size)
{
        // Copiar parametros:
        f_sample = freq_sample;
        f_target = freq_target;
        N = block_size;

        // Iniciar estado:
        n = 0;
        Q2=Q1=Q0=0;

        // Calcular valores pre-calculados:
        int     k = (int)(0.5 + N * f_target / f_sample);
        float   w = ((float)(2*M_PI*k))/N;
        ccos = cos(w);
        ssin = sin(w);
        GAMMA = 2*ccos;
}


/*************************************************************************
  Procesador de muestras: Devuelve true si las salidas contienen
   realmente un valor de salida (1 de cada N veces)
*************************************************************************/
bool CGoertzel::ProcessSample(float sample_in, float &re_out,float &im_out)
{
        Q0 = GAMMA * Q1 - Q2 + sample_in;
        Q2 = Q1;
        Q1 = Q0;

        if (++n==N)
        {       // Se ha procesado un bloque completo:
                re_out = Q1 - Q2 * ccos;
                im_out = Q2 * ssin;
                n=0;
                Q2 = Q1 = Q0 = 0;
        }

        return (n==0);
}
```

```
/*************************************************************************
FICHERO: CFiltroIIR.cpp
USO: Implementa un filtro digital del tipo IIR
NOTA: Se suponen los coeficientes normalizados: a[0]=1
*************************************************************************/

#include <stdlib.h>
#include <mem.h>
#include <math.h>

#include "CFiltroIIR.h"

/*************************************************************************
    Construye un filtro IIR de orden dado, es decir su respuesta
      al impulso b[k] tendra "orden" elementos.
*************************************************************************/
CFiltroIIR::CFiltroIIR(int Na,int Nb)
{
        a = b = NULL;
        x_d = y_d = NULL;
        CambiarOrden(Na,Nb);
}

/*************************************************************************
        Destructor
*************************************************************************/
```

```
CFiltroIIR::~CFiltroIIR()
{
	if (a) delete[] a;
	if (b) delete[] b;
	if (x_d) delete[] x_d;
	if (y_d) delete[] y_d;
}

/*************************************************************************
	Reiniciar el filtro: Pone a cero todos los elementos "x[n-k]"
*************************************************************************/
void CFiltroIIR::Reset()
{
	memset(x_d,0,sizeof(float)*Nb);
	memset(y_d,0,sizeof(float)*Na);
}

/*************************************************************************
  Filtrar: Entra una muestra x[n] y sale una muestra y[n].
	y[n] = Sum_k[0,Nb-1]{ x[n-k] * b[k] } -
	       Sum_k[1,Na-1]{ y[n-k] * a[k] }
*************************************************************************/
float CFiltroIIR::Filtrar(float x)
{
	int i;
	// Iniciar y a cero:
	double y=0;

	// Introducir la nueva muestra en el array de x[n-k]:
	x_d[0]=x;

	// Realizar el sumatorio:
	for (i=0;i<Nb;i++) y+=b[i]*x_d[i];
	for (i=1;i<Na;i++) y-=a[i]*y_d[i];

	// Desplazar las muestras para la siguiente iteracion:
	for (i=Nb-1;i>0;i--) x_d[i]=x_d[i-1];

	// Poner nuevo valor de y:
	y_d[0]=y;

	// y desplazar:
	for (i=Na-1;i>0;i--) y_d[i]=y_d[i-1];

	return y;	// Devolver la muestra de salida actual:
}

/*************************************************************************
	Establece el nuevo orden del filtro:
*************************************************************************/
void CFiltroIIR::CambiarOrden(int Na,int Nb)
{
	// Guardar los ordenes:
	this->Na = Na;
	this->Nb = Nb;

	// Liberar memoria anterior:
	if (a) delete[] a;
	if (b) delete[] b;
	if (x_d) delete[] x_d;
	if (y_d) delete[] x_d;

	// Reservar memoria:
	a=new float[Na];
	b=new float[Nb];
	x_d=new float[Nb];
	y_d=new float[Na];

	// Iniciar a cero:
	memset(a,0,sizeof(float)*Na);
	memset(b,0,sizeof(float)*Nb);
	memset(x_d,0,sizeof(float)*Nb);
	memset(y_d,0,sizeof(float)*Na);
}
```

A4.4. Demodulador FM

```
/*************************************************************************
FICHERO: DLLEntry.cpp
USO: Es el punto de entrada de la DLL
*************************************************************************/

#include <windows.h>
#pragma argsused


#include "implementation.h"


/*************************************************************************
                    DllEntryPoint
*************************************************************************/
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{
        switch (reason)
        {
                // La DLL se ha cargado
                // ----------------------------------------
                case DLL_PROCESS_ATTACH:
                        SWDEMOD_Init();
                        break;

                // La DLL esta siendo cerrada
                // ----------------------------------------
                case DLL_PROCESS_DETACH:
                        SWDEMOD_Close();
                        break;
        };

        return 1;
}
```

```
/*************************************************************************
FICHERO: Implementation.cpp
USO: Es donde se encuentra la implementacion del demodulador FM
*************************************************************************/
// Cabeceras estandar:
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>>

#include "implementation.h"

#define ACG_POTENCIA_DESEADA            -5     // dBm
#define AGC_TOLERANCIA_POTENCIA         4       // dB


// Estructura de datos con los punteros a las funciones necesarias para
//   acceder a la GUI que querramos crear:
DemodSW::GUI_Funciones  gui;
bool                    gui_loaded = false;

// ----------------------------------------------------------------------
//                      Variables del demodulador en si
// ----------------------------------------------------------------------
float                   frecuencia_sintonizada;  // La frec. a sintonizar (MHz)
bool                    actualizar_frecuencia;   // "true" cuando haya que actualizar el
PLL

float                   ganancia_RF, ganancia_DEMOD;
float                   ganancia_RF_old, ganancia_DEMOD_old;

CFiltroIIR              filtro_diezmador(1,1);
CFiltroIIR              filtro_audio1(1,1);
CFiltroIIR              filtro_audio2(1,1);
bool                    aplicar_filtro_audio1;
```

```
bool                    aplicar_filtro_audio2;

CGoertzel               detector_portadora_estereo(250000  /* f_sample, 250 KSPS */,
                                                   19000  /* f_target,  19 KHz  */,
                                                    2500  /* bins, res=100 Hz */ );
CGoertzel               detector_portadora_estereo2(250000 /* f_sample, 250 KSPS */,
                                                    18500 /* f_target, 18.5KHz  */,
                                                     2500 /* bins, res=100 Hz */ );
float                   potencia_portadora_estereo;
float                   potencia_portadora_estereo2;
float                   ultimo_nivel_portadora_estereo; // SNR de la portadora

/*************************************************************************
  SWDEMOD_GetInfo
        Devuelve informacion sobre este demodulador.
*************************************************************************/
void __fastcall SWDEMOD_GetInfo(DemodSW::DescripcionDemodulador *info_out)
{
        strcpy( info_out->nombreModulacion, "FM");
        strcpy( info_out->version, "1.0");
        strcpy( info_out->autor, "Jose Luis Blanco Claraco");
        strcpy( info_out->comentarios, "Implementacion de la demodulacion para señales
FM analogicas. Elimina la señal stereo de las recepciones de FM comercial. Tiene un BW
de señal demodulada de 10KHz. Implementa filtros de audio paso alto y de de-énfasis
opcionales.\n\n Proyecto fin de carrera JLBC/2005");
        info_out->analogica = true;     // Es demodulacion analogica
        info_out->rateSalida=50000;     // La tasa de muestras por segundo de salida
}

/*************************************************************************
  SWDEMOD_Demod
        Realiza la demodulacion en si. Entra un bloque de datos y
          sale otro. Los tamaños de los bloques no son fijos.
          A la entrada se alternan muestras de ocho bits para I/Q.
          A la salida hay un flujo de valores analogicos "float",
           o para el caso digital de valores "1.0/0.0"
*************************************************************************/
void __fastcall SWDEMOD_Demod(float *in,DWORD nSamplesIn,float *out,
                        DWORD &nSamplesOut,DWORD nMaxSamplesOut,
                        CUsbSwRadioInterfaz     *USBSwRadioInterfaz )
{
        int             i;              // Auxiliar
        char            str[100];       // Auxiliar

        // Se calculara la potencia de señal para formar un bucle
        //   de control automatico de potencia:
        float           var_I=0,var_Q=0;
        static float    potencia_media=0;
        float           potencia_instantanea;

        // El numero de muestras I/Q es la mitad de las muestras intercaladas:
        int     nSamples = nSamplesIn >> 1;

        // Calcular la media en cada canal:
        float   I_mean = 0;
        float   Q_mean = 0;
        for (i=0;i<nSamples;i++)
        {
                I_mean+=in[2*i+0];
                Q_mean+=in[2*i+1];
        }
        I_mean /= nSamples;
        Q_mean /= nSamples;

        // Calcular la fase de cada (I,Q) y estimar la derivada:
        // -----------------------------------------------------
        float           diff_ang[100];
        static float    last_ang = 0;
        float           ang;
        for (i=0;i<nSamples;i++)
        {
                // Estimar la potencia instantanea:
                float I = in[2*i+0]-I_mean;
                float Q = in[2*i+1]-Q_mean;
                var_I+= I*I;
                var_Q+= Q*Q;
```

```
        // Calcular la fase
        if (I!=0 && Q!=0)
                ang = atan2(Q,I);
        else    ang = 0;

        diff_ang[i] = ang - last_ang;
        if (diff_ang[i]>M_PI) diff_ang[i]= 2*M_PI-diff_ang[i];
        if (diff_ang[i]<-M_PI) diff_ang[i]= -2*M_PI-diff_ang[i];

        last_ang = ang;
}

var_I /= (nSamples-1);
var_Q /= (nSamples-1);

// la potecia estimada de la señal es:
potencia_instantanea = (var_I+var_Q)/50;
potencia_media =  potencia_media * 0.95 + 0.05 * potencia_instantanea;
float   potencia_media_dB = 10*log10(potencia_media*1000); // dBm

// Procesar con el algoritmo de Goertzel para detectar
//  el tono de indicación de estereo:
// ------------------------------------------------------
for (i=0;i<nSamples;i++)
{
        float   re19,im19;
        float   re20,im20;
        bool  hay_muestra;

        hay_muestra =
        detector_portadora_estereo.ProcessSample(diff_ang[i],re19,im19);
        detector_portadora_estereo2.ProcessSample(diff_ang[i],re20,im20);
        if (hay_muestra)
        {
                float pot_19 = sqrt(re19*re19+im19*im19);
                float pot_20 = sqrt(re20*re20+im20*im20);

                potencia_portadora_estereo =
                        0.95*potencia_portadora_estereo + 0.05*pot_19;
                potencia_portadora_estereo2 =
                        0.95*potencia_portadora_estereo2 + 0.05*pot_20;

                if (potencia_portadora_estereo2!=0)
                        ultimo_nivel_portadora_estereo =
                                potencia_portadora_estereo /
                                potencia_portadora_estereo2 ;
        }
}

// Filtrar para poder diezmar
// ------------------------------------------------------
float           senal_a_diezmar[100];
for (i=0;i<nSamples;i++)
         senal_a_diezmar[i] = filtro_diezmador.Filtrar(diff_ang[i]);



// Diezmar: 250KSPS -> 50KSPS
//   Es decir, quedarse con una de cada 5 muestras:
// ------------------------------------------------------
static int      diezmar_index = 0;
nSamplesOut = 0;

for (i=0;i<nSamples;i++)
{
        if ((++diezmar_index) == 5)
        {
                // Quedarse con la muestra "i":
                out[nSamplesOut++] = senal_a_diezmar[i];

                diezmar_index=0;
        }
}


// Filtrar sobre la señal a 50KSPS para eliminar
//   componentes inferiores a 300Hz y filtro de-enfasis
```

```
// ------------------------------------------------------
for (i=0;i<nSamplesOut;i++)
{
	if (aplicar_filtro_audio1) out[i] = filtro_audio1.Filtrar( out[i] );
	if (aplicar_filtro_audio2) out[i] = filtro_audio2.Filtrar( out[i] );
}



// ------------------------------------------------------------------
//   Control automatico de ganancia
//     La potencia media se mantiene entre [1.5,4].
//      Si la señal se sale, se activa la histeresis,
//       hasta que la potencia este en [2.5,3]
// ------------------------------------------------------------------
static int	iteraciones_CAG = 0;
if (iteraciones_CAG) iteraciones_CAG--;

float	umbral_low  = ACG_POTENCIA_DESEADA - AGC_TOLERANCIA_POTENCIA/2;
float	umbral_high = ACG_POTENCIA_DESEADA + AGC_TOLERANCIA_POTENCIA/2;

if (potencia_media_dB<umbral_low)
{
	if (!iteraciones_CAG)
	{
		// Aumentar ganancia:
		if (ganancia_DEMOD>0.21)
		{
			ganancia_DEMOD-=0.02;
			ganancia_RF=0.0;
		}
		else
		{
			ganancia_RF+=0.02;
			ganancia_DEMOD = 0.2;
		}/** /

		if (ganancia_RF<0.99)
		{
			ganancia_RF+=0.02;
			ganancia_DEMOD = 1.2;
		}
		else
		{
			ganancia_RF = 1.0;
			ganancia_DEMOD -= 0.02;
		} /**/

		iteraciones_CAG = 500;
	}

}

if (potencia_media_dB>umbral_high)
{
	if (!iteraciones_CAG)
	{
		/**/// Disminuir ganancia:
		if (ganancia_RF>0.01)
		{
			ganancia_RF-=0.02;
			ganancia_DEMOD=0.2;
		}
		else
		{
			ganancia_RF=0;
			ganancia_DEMOD+=0.02;
		} /** /

		if (ganancia_DEMOD<1.19)
		{
			ganancia_RF = 1.0;
			ganancia_DEMOD += 0.02;
		}
		else
		{
			ganancia_RF -= 0.02;
```

```
                                ganancia_DEMOD = 1.2;
                        } /**/

                        iteraciones_CAG = 500;
                }
        } /**/

        // ------------------------------------------------------------------
        //   Actualizar cada X iteraciones algunos indicadores en la GUI
        // ------------------------------------------------------------------
        static int      cuentaCiclos = 0;
        if ((++cuentaCiclos)>500)
        {
                char    str[100];
                sprintf(str,"Media(I):%.02f",I_mean);
                gui.GUI_Change_Text("txtMediaI",str);

                sprintf(str,"Media(Q):%.02f",Q_mean);
                gui.GUI_Change_Text("txtMediaQ",str);

                sprintf(str,"Potencia BB:%.02f dBm re 50R",potencia_media_dB);
                gui.GUI_Change_Text("txtPotencia",str);

                sprintf(str,"SNR de la portadora estereo:%.02f dB --> Detectada:
%s",10*log10(ultimo_nivel_portadora_estereo),ultimo_nivel_portadora_estereo>2 ?
"SI":"NO");
                gui.GUI_Change_Text("txtPotEstereo",str);

                // Volver a actualizar dentro de unas iteraciones:
                cuentaCiclos = 0;
        }

        // ------------------------------------------------------------------
        //   Cierre del bucle de control: Realimentacion hacia el radio
        //    receptor USB:
        // ------------------------------------------------------------------

        // Hay que actualizar el PLL?
        if (actualizar_frecuencia)
        {
                CUsbSwRadioInterfaz::PLL_CONFIG_DATA     pllConfig;

                pllConfig.AntiBacklashWidth = 0;
                pllConfig.LockDetectionPrecision = 0;
                pllConfig.CorePowerLevel = 0;   // 5mA (Lo recomendado)
                pllConfig.CounterReset = false;
                pllConfig.MuxOut =  CUsbSwRadioInterfaz::DIGITAL_LOCK_DETECT;
                pllConfig.PhasePositive = true;        // Filtro de lazo pasivo ("non-
inverting")
                pllConfig.ChargePumpThreeState = false;
                pllConfig.MuteTilLock = false;
                pllConfig.OutputPowerLevel = 2;         // -3dBm
                pllConfig.PowerDown = 0;

                pllConfig.CP_Gain_CurrentSelect = false;        // Seleccionar siempre
la primera configuracion de corriente.
                pllConfig.CP_CurrentSetting1 = 7;       // 2.5mA
                pllConfig.CP_CurrentSetting2 = 7;       // 2.5mA

                float xtalref = 2.5e6;
                float ref = 100e3;
                pllConfig.R_14bits = xtalref / ref;
                pllConfig.B_13bits = floor(0.5 + (frecuencia_sintonizada * 2 * 1e6 /
ref)); // Redondeo

                // Obtener frecuencia sintonizada realmente (con redondeo):
                frecuencia_sintonizada = (pllConfig.B_13bits * ref) / 2e6;
                sprintf(str,"%.02f",frecuencia_sintonizada);
                gui.GUI_Change_Text("edFrec",str);

                USBSwRadioInterfaz->PeticionAsinc_ConfiguraPLL( &pllConfig );

                // Mostrar la frecuencia en el LCD del receptor:
                // ----------------------------------------------------
                sprintf(str,"Demod: FM   ");
                USBSwRadioInterfaz->PeticionAsinc_LCD_Escribir(0,0,str);
                sprintf(str,"Frec: %.2fMHz",frecuencia_sintonizada);
```

```
                USBSwRadioInterfaz->PeticionAsinc_LCD_Escribir(0,1,str);

                actualizar_frecuencia = false;
        }

        // Asegurar margenes de las ganancias:
        if (ganancia_RF<0) ganancia_RF=0;
        if (ganancia_RF>1.0f) ganancia_RF=1.0f;
//        if (ganancia_RF>0.4) ganancia_RF=0.4f;

        if (ganancia_DEMOD<0.2) ganancia_DEMOD=0.2f;
        if (ganancia_DEMOD>1.2f) ganancia_DEMOD=1.2f;

        // Actualizar ganancias?
        bool hubo_cambios_ganancia = false;
        if (ganancia_RF_old!=ganancia_RF)
        {
                USBSwRadioInterfaz->PeticionAsinc_CambiarDAC(2, 255.0f * ganancia_RF /
5.0f );
                ganancia_RF_old = ganancia_RF;  // Actualizar valor para siguientes
iteraciones
                hubo_cambios_ganancia = true;
                // Mostrar en la interfaz grafica el nuevo valor:
                sprintf(str,"Ganancia RF: %.02fdB (%.02fV, %i%%)", -5.3 +
50.25*ganancia_RF,ganancia_RF,(int)(ganancia_RF*100) );
                gui.GUI_Change_Text("txtGananciaLNA",str);
        }
        if (ganancia_DEMOD_old!=ganancia_DEMOD)
        {
                USBSwRadioInterfaz->PeticionAsinc_CambiarDAC(3, 255.0f * ganancia_DEMOD
/ 5.0f );
                ganancia_DEMOD_old = ganancia_DEMOD;  // Actualizar valor para
siguientes iteraciones
                hubo_cambios_ganancia = true;
                // Mostrar en la interfaz grafica el nuevo valor:
                sprintf(str,"Ganancia Demod: %.02fdB (%.02fV, %i%%)",61-
50*ganancia_DEMOD,ganancia_DEMOD,(int)(100*(1.2-ganancia_DEMOD)));
                gui.GUI_Change_Text("txtGananciaDemod",str);
        }

        if (hubo_cambios_ganancia)
        {
                float ganancia_tot = (-5.3 + 50.25*ganancia_RF) + (61-
50*ganancia_DEMOD);
                sprintf(str,"Ganancia TOTAL: %.02fdB",ganancia_tot);
                gui.GUI_Change_Text("txtGananciaTotal",str);
        }

}

/*************************************************************************
  SWDEMOD_GUICallback
      Se llama a esta funcion cuando el usuario pulsa un boton o cambia
        un componente del GUI que haya creado el demodulador.
*************************************************************************/
void __fastcall SWDEMOD_GUICallback(char *name,WORD event,DWORD param1,DWORD param2)
{
        bool    checked;

        // Procesar segun el tipo de mensaje:
        switch (event)
        {
                // Evento: Un boton ha sido pulsado
                // ------------------------------------------------------
                case GUIEVENT_BUTTON_PUSHED:
                        if (!strcmpi(name,"btnFrec"))
                        {       // El usuario ha cambiado la frecuencia:
                                // ----------------------------------------
                              try
                              {
                                char    str[100];
                                gui.GUI_Get_Text("edFrec",str);
                                float new_frec = atof(str);
                                // Poner como nueva frecuencia:
                                frecuencia_sintonizada = new_frec;  // Mhz
                                actualizar_frecuencia=true;
                              } catch(...) { }
```

```
                        }
                        break;

                // Evento: Un check-box ha sido pulsado
                // ------------------------------------------------------------
                case GUIEVENT_CHECKBOX_TOOGLE:
                        checked = param1;       // "param1" indica si esta o no marcado
                        if (!strcmpi(name,"cbFiltro1")) aplicar_filtro_audio1 = checked;
                        if (!strcmpi(name,"cbFiltro2")) aplicar_filtro_audio2 = checked;
                        break;

        };

}

/*************************************************************************
  SWDEMOD_Init
      Este metodo se llama automaticamente justo al cargar la DLL
*************************************************************************/
void SWDEMOD_Init()
{
        int     i;

        // Cargar valores iniciales para variables:
        frecuencia_sintonizada = 99.2;  // Mhz
        actualizar_frecuencia=true;     // "true" para actualizar el PLL

        // Ganancias de RF y Demod:
        ganancia_RF = 0;
        ganancia_DEMOD = 1.2;
        ganancia_RF_old = ganancia_DEMOD_old = -1; // Para que se actualicen

        // Iniciar filtro IIR: filtro generado en MATLAB usando:
        //[b,a]=ellip(5,0.5,20,15/125);
        // num2str({a,b},10);
       {
//        float coefs_a[6]={ 1, -4.352558532,  7.801982537, -7.168070298,  3.368964783,-
0.6466399062 };
//        float coefs_b[6]={ 0.04972903208,-0.1296550243,0.08176528408,0.08176528408,-
0.1296550243,0.04972903208 };

        //[b,a]=ellip(6,1,40,9/125);
        // num2str({a,b},10);
        float coefs_a[7]={ 1,-5.701865101,  13.6394032,-17.51614893, 12.73488428,-
4.969232152,0.8129839617 };
        float coefs_b[7]={ 0.01018032977,-0.05541992784,  0.1307303861, -0.1709590626,
0.1307303861,-0.05541992784, 0.01018032977 };

        filtro_diezmador.CambiarOrden(7,7);

        for (i=0;i<7;i++) filtro_diezmador.Coeficiente_a(i,coefs_a[i]);
        for (i=0;i<7;i++) filtro_diezmador.Coeficiente_b(i,coefs_b[i]);
       }

        // Iniciar el filtro IIR para audio: DE-ENFASIS
        // [b,a]=butter(1,3/25);
       {
        float coefs_a[2]={ 1,  -0.8816185924 };
        float coefs_b[2]={ 0.05919070382,0.05919070382};

        filtro_audio1.CambiarOrden(2,2);

        for (i=0;i<2;i++) filtro_audio1.Coeficiente_a(i,coefs_a[i]);
        for (i=0;i<2;i++) filtro_audio1.Coeficiente_b(i,coefs_b[i]);
       }

        // Iniciar el filtro IIR para audio:
        // [b,a]=ellip(3,0.5,20,0.3/25,'high');
       {
        float coefs_a[4]={ 1,-2.938328221,2.878590846,-0.9402054773 };
        float coefs_b[4]={ 0.9697785855,-2.908783687,2.908783687,-0.9697785855 };

        filtro_audio2.CambiarOrden(4,4);

        for (i=0;i<4;i++) filtro_audio2.Coeficiente_a(i,coefs_a[i]);
        for (i=0;i<4;i++) filtro_audio2.Coeficiente_b(i,coefs_b[i]);
       }
```

```
	// Hasta obtener datos de Goertzel
	potencia_portadora_estereo = 0;
	potencia_portadora_estereo2 = 0;
	ultimo_nivel_portadora_estereo = 0;

	// Estado inicial de filtros:
	aplicar_filtro_audio1 = true;
	aplicar_filtro_audio2 = false;
}

/*************************************************************************
  SWDEMOD_SetGUI_Functions
      A esta funcion se llamara justo al principio de cargarse un
        demodulador. Si el demodulador quiere presentar una GUI,
        aqui sera cuando debe ser creada con las funciones que se le
        han pasado.
*************************************************************************/
void __fastcall SWDEMOD_SetGUI_Functions(DemodSW::GUI_Funciones *functions )
{
	char	str[100];	// Auxiliar

	gui=*functions; // Copiar punteros a variable local a esta DLL:
	gui_loaded = true;


	// Crear GUI:
	// ----------------------------------------------------------------------
	gui.GUI_Create_Label( "txtFrec","Frecuencia (MHz):",20,20 );
	sprintf(str,"%.01f",frecuencia_sintonizada);
	gui.GUI_Create_Edit( "edFrec",str,20,45,50,24 );
	gui.GUI_Create_Button( "btnFrec","Aplicar",80,45,80,24 );

	gui.GUI_Create_Label( "txtGananciaLNA","Ganancia RF:",200,30 );
	gui.GUI_Create_Label( "txtGananciaDemod","Ganancia Demod:",200,45 );
	gui.GUI_Create_Label( "txtGananciaTotal","Ganancia TOTAL:",450,37 );

	gui.GUI_Create_Label( "txtMediaI","Media(I):",200,70 );
	gui.GUI_Create_Label( "txtMediaQ","Media(I):",200,85 );
	gui.GUI_Create_Label( "txtPotencia","Potencia:",200,100 );
	gui.GUI_Create_Label( "txtPotEstereo","Potencia de la portadora
estereo:",200,120 );

	gui.GUI_Create_CheckBox( "cbFiltro1","Habilitar filtro de de-énfasis",20,150,
aplicar_filtro_audio1);
	gui.GUI_Create_CheckBox( "cbFiltro2","Habilitar filtro paso alto 300Hz",
20,165,aplicar_filtro_audio2);
}


/*************************************************************************
  SWDEMOD_Close
     Este metodo se llama automaticamente antes de descargar la DLL
*************************************************************************/
void SWDEMOD_Close()
{
	// Destruir GUI:
	// ----------------------------------------------
	if (gui_loaded)
	{
		gui.GUI_Free_Component("txtFrec");
		gui.GUI_Free_Component("edFrec");
		gui.GUI_Free_Component("btnFrec");
		gui.GUI_Free_Component("txtGananciaLNA");
		gui.GUI_Free_Component("txtGananciaDemod");
		gui.GUI_Free_Component("txtGananciaTotal");
		gui.GUI_Free_Component("txtMediaI");
		gui.GUI_Free_Component("txtMediaQ");
		gui.GUI_Free_Component("txtPotencia");
		gui.GUI_Free_Component("cbFiltro1");
		gui.GUI_Free_Component("cbFiltro2");
		gui.GUI_Free_Component("txtPotEstereo");
	}

}
```

A4.5. Demodulador ASK (MATLAB)

```
%-----------------------------------------------------------------
%   Demodulador ASK
%
% Este demodulador para MATLAB trabaja de manera offline, y
%  solo se plantea como una prueba de concepto, ya que un
%  demodulador real deberia afrontar muchos problemas no abordados
%-----------------------------------------------------------------
clear all;
close all;

%l=load('ASK_5KBaud_Af=11KHz_pattern01.txt');
l=load('ASK_5KBaud_Af=11KHz_PRBS.txt');
I = l(:,1);
Q = l(:,2);

I = I - mean(I);
Q = Q - mean(Q);

ampl = sqrt(I.^2 + Q.^2);

[b,a]=ellip(4,1,40,6/125);
dd=filter(b,a,ampl);

figure;
plot(dd(1:1000));
title('Amplitud de la señal compleja I/Q (filtrada)');

nBits = 2000;
figure;
plot(dd(1:(nBits*50)));
hold on;
k=6 + 50*(1:nBits);

bits = dd(k);
plot(k,bits,'or');

% Binarizar:
umbral = mean(bits);

figure;
hist(bits,linspace(0,0.1,80));

for i=1:length(bits),
    if (bits(i)>umbral),
        bits(i)=1;
    else
        bits(i)=-1;
    end
end

figure;
corr = xcorr(bits,bits);
plot((1:length(corr))-length(corr)/2,corr);
```

A4.6. Demodulador 2-FSK (MATLAB)

```
%-----------------------------------------------------------------
%   Demodulador 2-FSK
%
% Este demodulador para MATLAB trabaja de manera offline, y
%  solo se plantea como una prueba de concepto, ya que un
%  demodulador real deberia afrontar muchos problemas no abordados
%-----------------------------------------------------------------
clear all;
close all;

%l=load('2FSK_pattern01.txt');
l=load('2FSK_PRBS9bits.txt');

I = l(:,1);
Q = l(:,2);

I = I - mean(I);
Q = Q - mean(Q);

ang = atan2(Q,I);

d = diff(ang);
for i=1:length(d),
       if (d(i)>pi),
           d(i)=2*pi-d(i);
       end;
       if (d(i)<-pi),
           d(i)=-2*pi-d(i);
       end;
end;

[b,a]=ellip(4,1,40,6/125);
dd=filter(b,a,d);

nBits = 2000;
plot(dd(1:(nBits*50)));
hold on;
k=12 + 50*(1:nBits);

bits = dd(k);
plot(k,bits,'o');

figure,
hist(bits,100);

for i=1:length(bits),
    if bits(i)>0,
        bits(i)=1;
    else
        bits(i)=-1;
    end
end

corr = xcorr(bits,bits);

figure,
n= length(corr);
plot((1:n)-round(n/2),corr);
```

A4.7. Demodulador BPSK (MATLAB)

```
%-------------------------------------------------------------------
%   Demodulador BPSK
%
% Este demodulador para MATLAB trabaja de manera offline, y
%  solo se plantea como una prueba de concepto, ya que un
%  demodulador real deberia afrontar muchos problemas no abordados
%-------------------------------------------------------------------
clear all;
close all;

%l=load('BPSK_5KBaud_pattern01.txt');
l=load('BPSK_5KBaud_PRBS.txt');
I = l(:,1);
Q = l(:,2);

I = I - mean(I);
Q = Q - mean(Q);

I = I./max(abs(I));
Q = Q./max(abs(Q));


s_cmplx = complex(I,Q);
ol_cmplx = exp(-i.*(1:length(s_cmplx)).*(11116/125000).*pi);
s_cmplx = s_cmplx' .* ol_cmplx;

[b,a]=ellip(4,1,40,5/125);
s_cmplx= filter(b,a,s_cmplx);

s_cmplx = s_cmplx(10000:length(s_cmplx));

I = real(s_cmplx);
Q = imag(s_cmplx);


close all;
nBits = 1200;
plot(I(1:(nBits*50)),'b'); hold on;
plot(Q(1:(nBits*50)),'r');
k= round(43 + 50.01*(1:nBits));

II = I(k);
QQ = Q(k);

plot(k,II,'o');
plot(k,QQ,'o');

figure;
plot(II,QQ,'.');
hold on;
set(plot([0 0],[-0.4 0.4]),'LineWidth',2);
set(plot([-0.4 0.4],[0 0]),'LineWidth',2);

%
figure;
angs = atan2(QQ, II);
hist(angs,linspace(-pi,pi,50));

for i=1:length(QQ),
    if abs(angs(i))>pi/2,
        bits(i)=1;
    else
        bits(i)=-1;
    end
end


figure;
corr = xcorr(bits,bits);
plot((1:length(corr))-length(corr)/2,corr);
```

A4.8. Código fuente del programador (Delphi)

```
{-------------------------------------------------------------------
 FICHERO: unPrincipal.pas
 VENTANA: TVentanaPrincipal
 USO: Parte principal de la aplicacion. Interfaz grafica de usuario.

AGOSTO - 2004
Jose Luis Blanco Claraco
-------------------------------------------------------------------}
unit unPrincipal;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Tranbtn, ComCtrls,unTProgramadorAtmel;

type
  TVentanaPrincipal = class(TForm)
    GroupBox1: TGroupBox;
    btnIdentificar: TMTranBtn;
    SBar: TStatusBar;
    GroupBox2: TGroupBox;
    Label1: TLabel;
    lbTipoMicro: TLabel;
    GroupBox4: TGroupBox;
    cbFuseH7: TCheckBox;
    cbFuseH6: TCheckBox;
    cbFuseH5: TCheckBox;
    cbFuseH4: TCheckBox;
    cbFuseH3: TCheckBox;
    Label5: TLabel;
    cbBootSize: TComboBox;
    cbFuseH0: TCheckBox;
    cbFuseLO7: TCheckBox;
    cbFuseLO6: TCheckBox;
    Label6: TLabel;
    cbSUT: TComboBox;
    Label7: TLabel;
    cbFuenteReloj: TComboBox;
    btnLeerFuses: TMTranBtn;
    btnGrabarFuses: TMTranBtn;
    GroupBox5: TGroupBox;
    btnLeerFlash: TMTranBtn;
    btnGrabarFlash: TMTranBtn;
    btnBorrar: TMTranBtn;
    GroupBox6: TGroupBox;
    btnLeerEEP: TMTranBtn;
    btnGrabarEEP: TMTranBtn;
    OD: TOpenDialog;
    SD: TSaveDialog;
    Progress: TProgressBar;
    Label8: TLabel;
    lbProgreso: TLabel;
    lbFuses: TLabel;
    lbLocks: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure btnIdentificarClick(Sender: TObject);
    procedure btnBorrarClick(Sender: TObject);
    procedure btnLeerFlashClick(Sender: TObject);
    procedure btnGrabarFlashClick(Sender: TObject);
    procedure btnLeerFusesClick(Sender: TObject);
    procedure OnCambiarFuses(Sender: TObject);
    procedure btnGrabarFusesClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }

    // Usado durante la programacion, para asegurar el modelo del Atmel:
    ID_micro            : DWORD;
```

```
    // El objeto que se usara para programar el microcontrolador.
    //  Definido en "unTProgramadorAtmel.pas"
    ProgramadorAtmel    : TProgramadorAtmel;

    cambiando_fuses     : Boolean;


    procedure ProcesaHint(Sender : TObject);

  end;

var
  VentanaPrincipal: TVentanaPrincipal;

implementation


{$R *.dfm}

{-------------------------------------------------------------
        Evento: Al crear la ventana: Iniciaciones
 -------------------------------------------------------------}
procedure TVentanaPrincipal.FormCreate(Sender: TObject);
begin
        // Preparar el manejador de "Hints" o mensajes de ayuda:
        Application.OnHint:= ProcesaHint;

        // Crear el objeto que controla la programacion:
        ProgramadorAtmel:= TProgramadorAtmel.Create;

        cambiando_fuses:=false;
end;

{-------------------------------------------------------------
   Procesador de "hints" o mensajes de ayuda que saltan cuando
    se pasa el raton sobre un boton, etc...
 -------------------------------------------------------------}
procedure TVentanaPrincipal.ProcesaHint(Sender : TObject);
begin
        // Mostrar el mensaje en la barra de estado:
        SBar.SimpleText:= Application.Hint;
end;

{-------------------------------------------------------------
        Al cerrar la ventana principal:
 -------------------------------------------------------------}
procedure TVentanaPrincipal.FormDestroy(Sender: TObject);
begin
        // Liberar
        ProgramadorAtmel.Destroy;
end;

{-------------------------------------------------------------
        Evento: Al Pulsar el boton de identificar
 -------------------------------------------------------------}
procedure TVentanaPrincipal.btnIdentificarClick(Sender: TObject);
var
        nombre_micro    : AnsiString;
begin
        Screen.Cursor:= crAppStart;
        lbTipoMicro.Caption:='???';
        Application.ProcessMessages;    // actualizar el texto:

        if not ProgramadorAtmel.IdentificaMicro( ID_micro, nombre_micro  ) then
        begin
                Screen.Cursor:= crDefault;
                ShowMessage('Error identificando el microcontrolador...');
                Exit;
        end;

        Screen.Cursor:= crDefault;

        // Poner datos del chip:
        lbTipoMicro.Caption:= '0x'+IntToHex( ID_micro, 8 )+': '+nombre_micro;
end;
```

```
{-------------------------------------------------------------
        Evento: Al Pulsar el boton de borrar chip
 -------------------------------------------------------------}
procedure TVentanaPrincipal.btnBorrarClick(Sender: TObject);
var
        rx      : Byte;
begin
        // Primero entrar en modo programacion
        Screen.Cursor:= crAppStart;
        if not ProgramadorAtmel.EntrarEnModoProgramacion then
        begin
                Screen.Cursor:= crDefault;
                ShowMessage('No se encuentra microcontrolador...');
                exit;
        end;

        // Enviar comandos de borrar chip:
        with ProgramadorAtmel do
        begin
                TXRX( $AC, rx ); DarPulsosCLK( 10 );
                TXRX( $80, rx ); DarPulsosCLK( 10 );
                TXRX( $00, rx ); DarPulsosCLK( 10 );
                TXRX( $00, rx ); DarPulsosCLK( 10 );

                DarPulsosCLK(1000);

                Sleep(100);
                // Quitar alimentacion:
                CambiarPin( pinVCC1, false );
                CambiarPin( pinVCC2, false );
                CambiarPin( pinVCC3, false );
        end;

        // Fin del proceso
        Screen.Cursor:= crDefault;
        ShowMessage('Chip borrado correctamente');
end;

{-------------------------------------------------------------
        Boton: Leer flash:
   Leer flash del micro y guardarla en un fichero HEX
 -------------------------------------------------------------}
procedure TVentanaPrincipal.btnLeerFlashClick(Sender: TObject);
var
        flash_buffer    : array of byte;
        tam_flash       : Integer;
        i,contador      : integer;
        b               : byte;
begin
        // Preguntar al usuario el fichero destino:
        if not SD.Execute then exit;

        // Reconocer el microcontrolador:
        ID_micro:=0;
        btnIdentificarClick( self );
        if ID_micro<>$1E9306 then
        begin
                ShowMessage('Solo se soporta el microcontrolador ATMEGA8515');
                exit;
        end;

        tam_flash:= 8*1024;

        // Buffer para guardar la flash:
        setlength( flash_buffer, tam_flash );
        contador:=0;

        if not ProgramadorAtmel.EntrarEnModoProgramacion then
        begin
                ShowMessage( 'Error reentrando en modo programacion' );
                exit;
        end;


        for i:=0 to tam_flash-1 do
        begin
```

```
                // Leer byte de flash:
                ProgramadorAtmel.LeeByteDeFlash( i, b );
                flash_buffer[i]:=b;

                // No actualizar en todas las iteraciones:
                Inc(contador);
                if Contador=16 then
                begin
                        // Actualizar barra de progreso:
                        Contador:=0;
                        Progress.Position:= Round(100.0 * i / tam_flash);
                        lbProgreso.Caption:='Leyendo flash... 0x'+inttohex(i+1,4)+' /
0x'+inttohex(tam_flash,4);
                        Application.ProcessMessages;
                end;
        end;

        lbProgreso.Caption:='Leyendo flash... Completo';

        // Guardar buffer en HEX:
        ProgramadorAtmel.GuardarFicheroHEX( sd.FileName, flash_buffer, tam_flash );


end;

{-------------------------------------------------------------
        Boton: Grabar flash
   Grabar flash del microcontrolador
 -------------------------------------------------------------}
procedure TVentanaPrincipal.btnGrabarFlashClick(Sender: TObject);
var
        flash_buffer    : array of byte;
        tam_flash       : Integer;
        tam_page        : Integer;
        i,j,contador    : integer;
        todo_FF         : Boolean;
begin
        // Coger fichero a cargar:
        if not OD.Execute then exit;

        // Reconocer el microcontrolador:
        ID_micro:=0;
        btnIdentificarClick( self );
        if ID_micro<> $1E9306 then
        begin
                ShowMessage('Solo se soporta el microcontrolador ATMEGA8515');
                exit;
        end;

        tam_flash:= 8*1024;
        tam_page:= 64;

        // Buffer para guardar la flash:
        setlength( flash_buffer, tam_flash );
        contador:=0;

        // Poner a 0xFF todo para que las paginas que no se usen se queden sin
        //  programar: Mas rapido.
        FillMemory( @flash_buffer[0], tam_flash, $FF );

        // Cargar fichero:
        ProgramadorAtmel.CargarFicheroHEX( od.filename, flash_buffer, tam_flash  );

        // Bucle para cada pagina:
        for i:=0 to (tam_flash div tam_page)-1 do
        begin
                // Ver si toda esta pagina es 0xFF:
                todo_FF:=true;
                for j:=0 to tam_page-1 do
                        if flash_buffer[i*tam_page+j]<>$FF then todo_FF:=false;

                // Si no es todo 0xFF, entonces programar la pagina:
                if not todo_FF then
                begin
                        // Cargar los bytes en la pagina de pre-programacion:
                        for j:=0 to tam_page-1 do
```

```
                                ProgramadorAtmel.CargarBytePagina(
j,flash_buffer[i*tam_page+j] );

                        // Grabar la pagina en su direccion:
                        ProgramadorAtmel.GrabarPaginaFlash( i*tam_page );
                        ProgramadorAtmel.DarPulsosCLK( 1024 );

                        // Esperar tiempo de grabacion:
                        Sleep( 20 );
                end;

                // No actualizar en todas las iteraciones:
                Inc(contador);
                if Contador=2 then
                begin
                        // Actualizar barra de progreso:
                        Contador:=0;
                        Progress.Position:= Round(100.0 * i*tam_page / tam_flash);
                        lbProgreso.Caption:='Grabando flash...
0x'+inttohex((i+1)*tam_page,4)+' / 0x'+inttohex(tam_flash,4);
                        Application.ProcessMessages;
                end;
        end;    // Fin de bucle para cada pagina:

        lbProgreso.Caption:='Grabando flash... Completo';

end;


procedure TVentanaPrincipal.btnLeerFusesClick(Sender: TObject);
var
        fuses, locks    : DWORD;
        temp            : Integer;
begin
        // Asegurar el modelo del microcontrolador:
        ID_micro:=0;
        btnIdentificarClick( self );
        if ID_micro<>$1E9306 then
        begin
                ShowMessage('Solo se soporta el microcontrolador ATMEGA8515');
                exit;
        end;

        // Leer fuses y bits de lock:
        if not ProgramadorAtmel.EntrarEnModoProgramacion then
        begin
                ShowMessage('Error al entrar en modo programacion!');
                exit;
        end;

        ProgramadorAtmel.LeerFuses_y_Lock( fuses, locks );

        // Poner en texto:
        lbFuses.Caption:= '$'+inttohex(fuses,4);
        lbLocks.Caption:= '$'+inttohex(locks,2);

        // Poner en controles:
        // Deshabilitar evento en cambios de esos controles:
        cambiando_fuses:=true;

        cbFuseH7.Checked:= (fuses and (1 shl (7+8)))=0;
        cbFuseH6.Checked:= (fuses and (1 shl (6+8)))=0;
        cbFuseH5.Checked:= (fuses and (1 shl (5+8)))=0;
        cbFuseH4.Checked:= (fuses and (1 shl (4+8)))=0;
        cbFuseH3.Checked:= (fuses and (1 shl (3+8)))=0;
        cbFuseH0.Checked:= (fuses and (1 shl (0+8)))=0;

        cbFuseLO7.Checked:= (fuses and (1 shl 7 ))=0;
        cbFuseLO6.Checked:= (fuses and (1 shl 6 ))=0;

        cbFuenteReloj.ItemIndex:= (fuses and $0F);

        temp:= (fuses shr 4) and $03;
        cbSUT.ItemIndex:= temp;

        temp:= (fuses shr (1+8)) and $03;
        cbBootSize.ItemIndex:=temp;
```

```
        // Volver a habilitar evento en cambios de esos controles:
        cambiando_fuses:=false;


end;

{-------------------------------------------------------------
        Eventos: Cada vez que se cambia una configuracion de
         los bits de "fuses"
   Reconstruye el valor correcto en hexadecimal para "fuses"
 -------------------------------------------------------------}
procedure TVentanaPrincipal.OnCambiarFuses(Sender: TObject);
var
        fuses   : DWORD;
begin
        // Por si no es un evento generado por el usuario,
        //  sino programatico:
        if cambiando_fuses then exit;

        //
        fuses := 0;

        // Ir colocando cada bits de cada control en su sitio, segun
        //  el datasheet.
        if not cbFuseH7.Checked then fuses:=fuses or (1 shl (7+8));
        if not cbFuseH6.Checked then fuses:=fuses or (1 shl (6+8));
        if not cbFuseH4.Checked then fuses:=fuses or (1 shl (4+8));
        if not cbFuseH3.Checked then fuses:=fuses or (1 shl (3+8));
        if not cbFuseH0.Checked then fuses:=fuses or (1 shl (0+8));

        if not cbFuseLO7.Checked then fuses:=fuses or (1 shl 7);
        if not cbFuseLO6.Checked then fuses:=fuses or (1 shl 6);

        fuses := fuses or (cbSUT.ItemIndex shl (4));
        fuses := fuses or (cbBootSize.ItemIndex shl (1+8));
        fuses := fuses or (cbFuenteReloj.ItemIndex);


        lbFuses.Caption:='$'+inttohex(fuses, 4);
end;

procedure TVentanaPrincipal.btnGrabarFusesClick(Sender: TObject);
var
        fuses, lock     : DWORD;
begin
        // Preguntar si esta seguro:
        // ...

        // Asegurar el modelo del microcontrolador:
        ID_micro:=0;
        btnIdentificarClick( self );
        if ID_micro<>$1E9306 then
        begin
                ShowMessage('Solo se soporta el microcontrolador ATMEGA8515');
                exit;
        end;

        fuses := StrToInt(lbFuses.Caption );
        ProgramadorAtmel.GrabarFuses_y_Lock( fuses, lock );

end;

end.
```

```
{-------------------------------------------------------------------
 FICHERO: unTProgramadorAtmel.pas
 VENTANA: Ninguna
 USO: Rutinas que tratan el puerto paralelo y la programacion
   de microcontroladores Atmel en modo "serial downloading".
   Se implementa a traves de la clase "TProgramadorAtmel".

AGOSTO - 2004
Jose Luis Blanco Claraco
-------------------------------------------------------------------}
unit unTProgramadorAtmel;

interface

uses Windows,IniFiles,Dialogs,SysUtils;

type
 TProgramadorAtmel = class
  private
        // Variables
        estado_pins     : BYTE;         // Estado actual de las salidas.

        function        MISO:Boolean;   // Funcion de bajo nivel usada para comms.

  public
        // Variables:

        // Pines del puerto paralelo para cada señal a controlar:
        pinRST          : Integer;
        pinMOSI         : Integer;
        pinMISO         : Integer;
        pinSCK          : Integer;
        pinXTAL         : Integer;
        pinVCC1         : Integer;
        pinVCC2         : Integer;
        pinVCC3         : Integer;

        // Definiciones de funciones, ver cada una para explicacion:
        constructor Create;
        destructor  Destroy; override;

        // Funciones de programacion:
        function    EntrarEnModoProgramacion:Boolean;
        function    IdentificaMicro( var ID : DWORD; var nombre : AnsiString ):Boolean;
        function    LeeByteDeFlash( dir : DWORD; var destino : byte) : Boolean;
        function    CargarBytePagina( dir_pagina : WORD; dato : byte) : Boolean;
        function    GrabarPaginaFlash( dir : DWORD ) : Boolean;
        function    LeerFuses_y_Lock( var fuses: DWORD; var lockbits : DWORD ) :
Boolean;
        function    GrabarFuses_y_Lock( fuses: DWORD; lockbits : DWORD ) : Boolean;

        // Funciones de ficheros HEX:
        procedure   GuardarFicheroHEX( fichero : AnsiString; var memory :array of Byte ;
nBytes : DWORD );
        procedure   CargarFicheroHEX ( fichero : AnsiString; var memory :array of Byte;
MaxDir : DWORD );

        // Funciones de "bajo nivel" para comunicaciones con el micro:
        procedure   TXRX( tx : Byte; var rx : Byte);
        procedure   CambiarPin( pin: Integer ; valor : Boolean );
        procedure   DarPulsosCLK( n : Integer );

 end;

implementation

{
        Declaraciones de DLPORTIO.DLL
}
function DlPortReadPortUchar( Port:longInt ): Integer; stdcall; external 'dlportio.dll';
procedure DlPortWritePortUchar( Port:longInt;Value:byte ); stdcall; external
'dlportio.dll';


{-------------------------------------------------------------
           Constructor
 -------------------------------------------------------------}
```

```
constructor TProgramadorAtmel.Create;
var
        fi      : TIniFile;
begin
        // Cargar configuracion del puerto paralelo:
        fi:= TIniFile.Create('.\atmegaprog.ini');

        pinVCC1:=fi.ReadInteger('Configuracion','pinVCC1', 0 );
        pinVCC2:=fi.ReadInteger('Configuracion','pinVCC2', 1 );
        pinVCC3:=fi.ReadInteger('Configuracion','pinVCC3', 2 );
        pinXTAL:=fi.ReadInteger('Configuracion','pinXTAL', 3 );
        pinRST:=fi.ReadInteger('Configuracion','pinRST',   4 );
        pinMOSI:=fi.ReadInteger('Configuracion','pinMOSI', 5 );
        pinSCK:=fi.ReadInteger('Configuracion','pinSCK',   6 );
        pinMISO:=fi.ReadInteger('Configuracion','pinMISO', 6 );


        // Poner programador en estado inicial:
        CambiarPin( pinVCC1, false );   // VCC = 0v
        CambiarPin( pinVCC2, false );
        CambiarPin( pinVCC3, false );
        CambiarPin( pinRST,  false );   // Todo a cero:
        CambiarPin( pinMOSI, false );
        CambiarPin( pinSCK,  false );
        CambiarPin( pinXTAL, false );

        // Liberar
        fi.Destroy;
end;

{-------------------------------------------------------------
           Destructor
 -------------------------------------------------------------}
destructor  TProgramadorAtmel.Destroy;
var
        fi      : TIniFile;
begin
        // Dejar "apagado" el programador:
        CambiarPin( pinVCC1, false );
        CambiarPin( pinVCC2, false );
        CambiarPin( pinVCC3, false );

        // Guardar configuracion de pins:
        fi:= TIniFile.Create('.\atmegaprog.ini');


        fi.Destroy;


        inherited;      // Llama al metodo "overrided" base.
end;


{-------------------------------------------------------------
  Poner al micro en modo de programacion.
   Devuelve: true si todo OK
 -------------------------------------------------------------}
function TProgramadorAtmel.EntrarEnModoProgramacion:Boolean;
var
        cuenta_intentos         : Integer;
        intentos_alineacion     : Integer;
        rx                      : BYTE;
        sincronizado            : Boolean;
begin
        // Bucle para varios intentos:
        for cuenta_intentos:=0 to 2 do
        begin
                // Quitar VCC:
                CambiarPin( pinVCC1, false );
                CambiarPin( pinVCC2, false );
                CambiarPin( pinVCC3, false );
                Sleep(300);
                // Volver a poner VCC:
                CambiarPin( pinVCC1, true );
                CambiarPin( pinVCC2, true );
                CambiarPin( pinVCC3, true );
                Sleep(300);
```

```
                // Dar unos pulsos de reloj para aseguar
                //  que el micro se ha iniciado:
                DarPulsosCLK( 2000 );

                // 32 intentos para corregir posibles fallos de
                //  alineacion de palabra.
                //   (Ver DATASHEET, Memory Programming, Serial Alg.)
                for intentos_alineacion:=1 to 32 do
                begin
                        // Enviar comando de iniciar modo programacion: (4 bytes)
                        TXRX( $AC, rx ); DarPulsosCLK( 10 );
                        TXRX( $53, rx ); DarPulsosCLK( 10 );
                        TXRX( $00, rx ); DarPulsosCLK( 10 );
                        // Si este byte recibido es $53 estara todo OK.
                        sincronizado:= rx = $53;

                        TXRX( $00, rx ); DarPulsosCLK( 10 );

                        if not sincronizado then
                        begin
                                // Dar pulso a SCK:
                                CambiarPin( pinSCK, true );
                                CambiarPin( pinSCK, false );

                                // Volver a intentar:
                        end else
                        begin
                                // Sincronizacion de bit conseguida:
                                result:= true;
                                exit;   // salir de la funcion
                        end;

                end;    // fin de bucle "intentos_alineacion"


        end; // Fin del bucle de "cuenta_intentos"

        // Si no se ha salido antes es porque no se ha encontrado un micro:
        result:=false;
end;

{-------------------------------------------------------------
   Identificar el microcontrolador de Atmel.
    Devuelve: true si todo OK.
      en ID el Identificador.
      en nombre el nombre del modelo en formato texto.
 -------------------------------------------------------------}
function TProgramadorAtmel.IdentificaMicro( var ID : DWORD; var nombre : AnsiString
):Boolean;
var
        id1, id2, id3           : BYTE;
        rx                      : BYTE;
begin
        result:=false;

        // Intentar poner al micro en modo programacion:
        if not EntrarEnModoProgramacion then
        begin
                ShowMessage('El microcontrolador no responde');
                exit;
        end;

        // Enviar comandos de identificacion de chip:
        // Byte 0:
        TXRX( $30, rx ); DarPulsosCLK( 10 );
        TXRX( $00, rx ); DarPulsosCLK( 10 );
        TXRX( $00, rx ); DarPulsosCLK( 10 );
        TXRX( $00, id1 ); DarPulsosCLK( 10 );

        // Byte 1:
        TXRX( $30, rx ); DarPulsosCLK( 10 );
        TXRX( $00, rx ); DarPulsosCLK( 10 );
        TXRX( $01, rx ); DarPulsosCLK( 10 );
        TXRX( $00, id2 ); DarPulsosCLK( 10 );

        // Byte 2:
```

```
        TXRX( $30, rx ); DarPulsosCLK( 10 );
        TXRX( $00, rx ); DarPulsosCLK( 10 );
        TXRX( $02, rx ); DarPulsosCLK( 10 );
        TXRX( $00, id3 ); DarPulsosCLK( 10 );

        // Formar palabra de ID:
        ID := (id1 shl 16) + (id2 shl 8) + id3;

        // Ver si es un modelo conocido:
        case ID of
                $000102:  nombre:='** PROTEGIDO **';
                $1E9306:  nombre:='ATMEGA8515';
                else
                        nombre:='???';
        end;


        result:=true;
end;


{-------------------------------------------------------------
        Cambia un pin del programador
 -------------------------------------------------------------}
procedure TProgramadorAtmel.CambiarPin( pin: Integer ; valor : Boolean );
begin
        // Quitar el bit afectado:
        estado_pins:= estado_pins and not (1 shl pin);

        // Y volver a ponerlo al valor correcto:
        if valor then
                estado_pins:= estado_pins or (1 shl pin);

        // Y sacarlo por el puerto paralelo:
        DlPortWritePortUchar( $378, estado_pins );
end;

{-------------------------------------------------------------
   Lee estado del pin MISO en el programador, puerto paralelo
 -------------------------------------------------------------}
function TProgramadorAtmel.MISO:Boolean;
var
        b       : Byte;
begin
        b:= DlPortReadPortUchar( $379 );
        result:= ( b and (1 shl pinMISO) ) <> 0;
end;


{-------------------------------------------------------------
   Enviar al chip un byte y al mismo tiempo recibir:
 -------------------------------------------------------------}
procedure TProgramadorAtmel.TXRX( tx : Byte; var rx : Byte);
var
        i : Integer;
begin
        rx:=0;
        for i:=7 downto 0 do
        begin
                if MISO then rx:=rx+ (1 shl i);

                // Poner el bit a enviar:
                CambiarPin( pinMOSI, ( tx and (1 shl i)) <> 0);

                // Dar unos pulsos de reloj:
                DarPulsosCLK ( 10 );

                // Poner SCK = 1
                CambiarPin( pinSCK, true );

                // Dar unos pulsos de reloj:
                DarPulsosCLK ( 10 );

                // Poner SCK = 0
                CambiarPin( pinSCK, false );
        end;
```

```
end;

{-------------------------------------------------------------
   Dar "n" pulsos de reloj al microcontrolador
 -------------------------------------------------------------}
procedure TProgramadorAtmel.DarPulsosCLK( n : Integer );
begin
        for n:=n downto 0 do
        begin
                CambiarPin( pinXTAL, true );
                CambiarPin( pinXTAL, false );
        end;
end;


{-------------------------------------------------------------
  Lee un byte de la memoria flash del microcontrolador
   Devuelve: true si todo OK
 -------------------------------------------------------------}
function TProgramadorAtmel.LeeByteDeFlash( dir : DWORD; var destino : byte) : Boolean;
var
        dir_impar       : Boolean;
        rx              : BYTE;
begin
        // Enviar comando de lectura:
        dir_impar:= (dir and 1)<>0;
        dir:= dir shr 1;
        if dir_impar then
                TXRX( $28, rx )
        else    TXRX( $20, rx );

        TXRX( hibyte(dir), rx );
        TXRX( lobyte(dir), rx );
        TXRX( 00, rx );

        destino:=rx;


        result:=true;
end;

{-------------------------------------------------------------
  Escribe un byte en el buffer de pagina pre-grabacion en
   el chip
    Devuelve: true si todo OK
 -------------------------------------------------------------}
function TProgramadorAtmel.CargarBytePagina( dir_pagina : WORD; dato : byte) : Boolean;
var
        dir_impar       : Boolean;
        rx              : BYTE;
begin
        // Enviar comando de carga de pagina:
        dir_impar:= (dir_pagina and 1)<>0;
        if dir_impar then
                TXRX( $48, rx )
        else    TXRX( $40, rx );

        TXRX( hibyte( dir_pagina shr 1 ), rx );
        TXRX( lobyte( dir_pagina shr 1 ), rx );
        TXRX( dato, rx );

        DarPulsosCLK( 32 );

        result:=true;
end;

{-------------------------------------------------------------
  Escribir la pagina previamente cargada en la flash:
    Devuelve: true si todo OK
 -------------------------------------------------------------}
function TProgramadorAtmel.GrabarPaginaFlash( dir : DWORD ) : Boolean;
var
        rx              : BYTE;
begin
        // Enviar comando de escritura de pagina:
        TXRX( $4C, rx );
        TXRX( hibyte( dir shr 1 ), rx );
```

```
        TXRX( lobyte( dir shr 1 ), rx );
        TXRX( $00, rx );

        DarPulsosCLK( 32 );

        result:=true;
end;

{-------------------------------------------------------------
  Lee valor de los bytes de "fuses"
    Devuelve: true si todo OK
 -------------------------------------------------------------}
function TProgramadorAtmel.LeerFuses_y_Lock( var fuses: DWORD; var lockbits : DWORD ) :
Boolean;
var
        rx              : BYTE;
begin
        fuses := 0;
        lockbits:=0;

        // Leer byte bajo de fuses
        TXRX( $50, rx );
        TXRX( $00, rx );
        TXRX( $00, rx );
        TXRX( $00, rx );
        fuses := fuses or (rx);

        DarPulsosCLK( 10 );

        // Leer byte alto de fuses
        TXRX( $58, rx );
        TXRX( $08, rx );
        TXRX( $00, rx );
        TXRX( $00, rx );
        fuses := fuses or (rx shl 8);

        DarPulsosCLK( 10 );

        // Leer byte bajo de locks
        TXRX( $58, rx );
        TXRX( $00, rx );
        TXRX( $00, rx );
        TXRX( $00, rx );
        lockbits := lockbits or (rx);

        DarPulsosCLK( 10 );

        result:=true;
end;


{-------------------------------------------------------------
  Graba fuses y lock bits en el micro:
    Devuelve: true si todo OK
 -------------------------------------------------------------}
function TProgramadorAtmel.GrabarFuses_y_Lock( fuses: DWORD; lockbits : DWORD ) :
Boolean;
var
        rx              : BYTE;
begin
        // Asegurar bit 7 de fuses_hi = 0: Proteger de situacion
        //  en la que no se pueda programar mas el chip:
        if (fuses or (1 shl (7+8)))=0 then
        begin
                ShowMessage('AVISO: Intento de programar fuse RSTDISBL!!');
                result:=false;
                exit;
        end;

        // Grabar byte alto de fuses
        TXRX( $AC, rx );
        TXRX( $A8, rx );
        TXRX( $00, rx );
        TXRX( hibyte(fuses), rx );

        DarPulsosCLK( 10 );
        Sleep(50);
```

```
        // Grabar byte bajo de fuses
        TXRX( $AC, rx );
        TXRX( $A0, rx );
        TXRX( $00, rx );
        TXRX( lobyte(fuses), rx );

        DarPulsosCLK( 10 );
        Sleep(50);

        result:=true;
end;


{-------------------------------------------------------------
  Guarda un buffer de memoria en un fichero HEX
 -------------------------------------------------------------}
procedure TProgramadorAtmel.GuardarFicheroHEX( fichero : AnsiString; var memory :array
of Byte ; nBytes : DWORD );
var
        pos,tam,i    : DWORD;
        fil          : Text;      // Fichero
        chksum       : BYTE;
begin
        // Si el fichero no existe, crearlo:
        if not FileExists(fichero) then fileclose( FileCreate(fichero) );

        // Abrir fichero para escritura:
        AssignFile(fil,fichero);
        Rewrite(fil);

        pos:=0;          // Contador de posicion:
        while (pos<(nBytes-1)) do
        begin
                // Ver cuantos bytes van en esta linea del fichero:
                tam:=nBytes-pos;
                if (tam>16) then tam:=16;

                // Iniciar acumulador para chechsum al final de linea
                chksum:=0;

                // Formar cabecera de linea:
                write(fil,':');
                write(fil,inttohex(tam,2)); inc(chksum,tam);
                write(fil,inttohex(pos,4));
inc(chksum,HIBYTE(pos));inc(chksum,LOBYTE(pos));
                write(fil,'00');  // Indicador de segmento de datos...

                // Escribir datos en esta linea:
                for i:=1 to tam do
                begin
                        write(fil,inttohex( memory[pos+i-1],2 ) );
                        inc(chksum,memory[pos+i-1]);
                end;

                // Escribir chechsum de linea:
                chksum:=-chksum; // complemento a 2
                writeln(fil,inttohex( chksum ,2) );

                inc(pos,tam);
        end;

        // Registro de final de fichero:
        writeln(fil,':00000001FF');

        closefile(fil);

end;


{-------------------------------------------------------------
  Lee en un buffer de memoria un fichero HEX
 -------------------------------------------------------------}
procedure TProgramadorAtmel.CargarFicheroHEX( fichero : AnsiString; var memory :array of
Byte; MaxDir : DWORD );
var
        pos,tam,i    : DWORD;
```

```
        fil         : Text;     // Fichero
        x           : BYTE;
        linea       : String;
        fin         : Boolean;
begin
        // Abrir el fichero para lectura:
        AssignFile(fil,fichero);
        Reset(fil);

        // Bucle de lectura y decodificacion de lineas:
        fin:=false;
        while (not EOF(fil))and (not fin) do
        begin
                // Leer linea:
                readln(fil,linea);
                delete(linea,1,1); // quitar dos puntos iniciales

                // nº bytes en linea:
                x:= StrToInt( '$'+copy(linea,1,2) );
                delete (linea,1,2);
                tam:=x;

                // Posicion en memoria:
                pos:= StrToInt( '$'+copy(linea,1,4) ); delete (linea,1,4);

                if (pos<=MaxDir) then
                begin
                        // tipo linea:
                        x:=StrToInt( '$'+copy(linea,1,2) ); delete (linea,1,2);
                        if (x=0) then
                        begin   // La linea es de datos:
                                for i:=1 to tam do
                                begin
                                        if (length(linea)<2) then
                                        begin
                                        end
                                        else
                                        begin
                                         x:= StrToInt( '$'+copy(linea,1,2) );
                                         delete (linea,1,2);
                                         memory[pos+i-1]:=x;
                                        end;
                                end;
                        end
                        else
                                // Fin de fichero:
                                if x=1 then FIN:=TRUE;
                end;

        end;

        // Cerrar fichero:
        closefile(fil);
end;


end.
```

```
{-------------------------------------------------------------------
 FICHERO: Tranbtn.pas
USO: Implementa un boton grafico transparente
-------------------------------------------------------------------}
unit Tranbtn;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, StdCtrls, ExtCtrls;

type
  BStyle = (BSnone,BsNormal,BsIe);
  TMTranBtn = class(TGraphicControl)
  private
    FBitMap : TBitmap;
    FOver : Boolean;
    Pushed : boolean;
    Fborder : BStyle;
    BRect : Trect;
    procedure SetBitMap(Value : TBitMap);
    procedure WMLButtonDown(var msg: TWMLButtonDown); message WM_LBUTTONDOWN;
    procedure WMLButtonUp(var msg: TWMLButtonUp); message WM_LBUTTONUP;
    procedure MouseMove(Shift: TShiftState; X, Y: Integer); override;
    function OnGlyphP(X, Y: integer): boolean;
    procedure mouseleave(var msg : tmessage); message cm_mouseleave;
    procedure mousein(var msg : tmessage); message cm_mouseenter;
    Procedure setborderstyle(value:Bstyle);
  protected
    procedure Paint; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    procedure DrawTransparentBitmap (ahdc: HDC; Image: TBitmap; xStart, yStart: Word;
TrCol : Tcolor);
    property BitMap : TBitMap read FBitMap write SetBitMap;
    Property OnClick;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property Visible;
    Property Hint;
    Property ShowHint;
    Property Border : BStyle read fborder write SetBorderStyle;
    Property Caption;
    Property Font;
  end;

procedure Register;

implementation

constructor TMTranBtn.Create(AOwner: TComponent);
begin
inherited Create(AOwner);
Width := 30;
Height := 30;
FBitMap := TBitMap.Create;
ControlStyle := ControlStyle - [csOpaque];
Pushed := false;
Font.name := 'Arial';
Font.size := 9;
Fborder := BsNormal;
end;

destructor TMTranBtn.Destroy;
begin
FBitMap.Free;
inherited Destroy;
end;

procedure TMTranBtn.SetBitMap(Value : TBitMap);
begin
 FBitMap.Assign(Value);
 invalidate;
```

```
end;

{this routine come from unit XparBmp of Michael Vincze (vincze@ti.com), I think it can
be
optimized more. Will find time to check it again}
procedure TMTranBtn.DrawTransparentBitmap (ahdc: HDC; Image: TBitmap; xStart, yStart:
Word; TrCol : Tcolor);
var
  TransparentColor: TColor;
  cColor          : TColorRef;
  bmAndBack,
  bmAndObject,
  bmAndMem,
  bmSave,
  bmBackOld,
  bmObjectOld,
  bmMemOld,
  bmSaveOld       : HBitmap;
  hdcMem,
  hdcBack,
  hdcObject,
  hdcTemp,
  hdcSave         : HDC;
  ptSize          : TPoint;
begin
TransparentColor := TrCol;
TransparentColor := TransparentColor or $02000000;

hdcTemp := CreateCompatibleDC (ahdc);
SelectObject (hdcTemp, Image.Handle); { select the bitmap }
ptSize.x := Image.Width;
ptSize.y := Image.Height;
DPtoLP (hdcTemp, ptSize, 1);  { convert from device logical points }
hdcBack   := CreateCompatibleDC(ahdc);
hdcObject := CreateCompatibleDC(ahdc);
hdcMem    := CreateCompatibleDC(ahdc);
hdcSave   := CreateCompatibleDC(ahdc);

bmAndBack   := CreateBitmap (ptSize.x, ptSize.y, 1, 1, nil);
bmAndObject := CreateBitmap (ptSize.x, ptSize.y, 1, 1, nil);

bmAndMem    := CreateCompatibleBitmap (ahdc, ptSize.x, ptSize.y);
bmSave      := CreateCompatibleBitmap (ahdc, ptSize.x, ptSize.y);

bmBackOld   := SelectObject (hdcBack, bmAndBack);
bmObjectOld := SelectObject (hdcObject, bmAndObject);
bmMemOld    := SelectObject (hdcMem, bmAndMem);
bmSaveOld   := SelectObject (hdcSave, bmSave);

SetMapMode (hdcTemp, GetMapMode (ahdc));
BitBlt (hdcSave, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCCOPY);
cColor := SetBkColor (hdcTemp, TransparentColor);
BitBlt (hdcObject, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCCOPY);

SetBkColor (hdcTemp, cColor);
BitBlt (hdcBack, 0, 0, ptSize.x, ptSize.y, hdcObject, 0, 0, NOTSRCCOPY);

BitBlt (hdcMem, 0, 0, ptSize.x, ptSize.y, ahdc, xStart, yStart, SRCCOPY);
BitBlt (hdcMem, 0, 0, ptSize.x, ptSize.y, hdcObject, 0, 0, SRCAND);
BitBlt (hdcTemp, 0, 0, ptSize.x, ptSize.y, hdcBack, 0, 0, SRCAND);
BitBlt (hdcMem, 0, 0, ptSize.x, ptSize.y, hdcTemp, 0, 0, SRCPAINT);
BitBlt (ahdc, xStart, yStart, ptSize.x, ptSize.y, hdcMem, 0, 0, SRCCOPY);
BitBlt (hdcTemp, 0, 0, ptSize.x, ptSize.y, hdcSave, 0, 0, SRCCOPY);
DeleteObject (SelectObject (hdcBack, bmBackOld));
DeleteObject (SelectObject (hdcObject, bmObjectOld));
DeleteObject (SelectObject (hdcMem, bmMemOld));
DeleteObject (SelectObject (hdcSave, bmSaveOld));
DeleteDC (hdcMem);
DeleteDC (hdcBack);
DeleteDC (hdcObject);
DeleteDC (hdcSave);
DeleteDC (hdcTemp);
end;

procedure TMTranBtn.setborderstyle(value:Bstyle);
begin
if Fborder <> value then
```

```
 begin
  Fborder := value;
  Invalidate;
 end;
end;

procedure TMTranBtn.Paint;
var
  ARect: TRect;
  Tmp : TBitMap;
  x,y : integer;
  text : array[0..40] of char;
  Fontheight : integer;
begin
ARect := Rect(0,0,Width,Height);
Canvas.font := font;
FontHeight := Canvas.TextHeight('W');
if not FBitMap.empty then
  begin
  x := (width - FBitMap.width) div 2;
  if caption <> '' then
    y := ((Height - FBitMap.Height- FontHeight) div 2)
  else
   y := ((Height - FBitMap.Height) div 2);
     BRect := rect(x, y, x + FBitMap.width, y + FBitMap.height);
     Tmp := TBitmap.Create;
     Tmp.Height := FBitMap.Height;
     Tmp.Width := FBitMap.Width;
     Tmp.Canvas.CopyRect(ARect, FBitmap.Canvas, ARect);
     if pushed then
      DrawTransparentBitmap( Canvas.Handle, Tmp, x +1, y+1, FBitmap.TransparentColor )
     else
      DrawTransparentBitmap( Canvas.Handle, Tmp, x, y, FBitmap.TransparentColor );
     Tmp.Free;
  end;

  if caption <> '' then
  with Canvas do
  begin
   Brush.Style := bsClear;
   with ARect do
    begin
     if Fbitmap.empty then
       Top := ((Bottom + Top) - FontHeight) shr 1
     else
       top := Brect. bottom;
      Bottom := Top + FontHeight;
      if pushed then
        begin
         top := top + 1;
         left := 2;
        end;
    end;
    StrPCopy(Text, Caption);
    DrawText(Handle, Text, StrLen(Text), ARect, (DT_EXPANDTABS or DT_center));
   end;

 ARect := getclientrect;
 case fborder of
 BsNormal : BEGIN
            if pushed then
                frame3d(canvas, ARect ,clBtnShadow,clBtnHighlight, 1)
            else
                frame3d(canvas, ARect ,clBtnHighlight,clBtnShadow, 1);
            END;
 BsIe :  Begin
         if pushed then
            frame3d(canvas, ARect ,clBtnShadow,clBtnHighlight, 1)
         else
          if Fover then
            frame3d(canvas, ARect ,clBtnHighlight,clBtnShadow, 1);
         end;
 end; { case}
end;


function TMTranBtn.OnGlyphP(X, Y: integer): boolean;
```

```
begin
  Result := PtInRect({ClientRect} BRect, Point(X, Y)) and
            (FBitmap.Canvas.Pixels[X, Y] <> FBitmap.TransparentColor);
end;

procedure TMTranBtn.MouseMove(Shift: TShiftState; X, Y: Integer);

begin
  FOver := (fborder = bsnormal) or (fborder = bsie) or  OnGlyphP(X, Y);
  Inherited MouseMove(Shift, X, Y);
end;

procedure TMTranBtn.mouseleave(var msg : tmessage);
var  rc : Trect;
BEGIN
  FOver := false;
  rc := getclientrect;
  if Fborder = bsie then
    INVALIDATE;
END;

procedure TMTranBtn.mousein(var msg : tmessage);
var  rc : Trect;
BEGIN
  FOver := true;
  rc := getclientrect;
  if Fborder = bsie then
    frame3d(canvas, rc ,clBtnHighlight,clBtnShadow, 1);
END;

procedure TMTranBtn.WMLButtonDown;
begin
 inherited;
  Pushed := (fborder = bsnormal) or (fborder = bsie) OR FOver;
  if pushed then
     invalidate;
end;

procedure TMTranBtn.WMLButtonUp;
begin
 inherited;
 if (fborder = bsnormal) or (fborder = bsie) or FOver then
    Pushed := false;
 if Pushed = false then
   invalidate;
end;

procedure Register;
begin
  RegisterComponents('Mik', [TMTranBtn]);
end;

end.
```