Universidad de Zaragoza
Escuela Universitaria de Ingeniería Técnica Industrial
Departamento de Ingeniería Electrónica y de Comunicaciones

Proyecto Fin de Carrera

Diseño de un receptor de señal DCF77 para sincronización horaria por radio

Autor: Jaime Vinués Alconchel
Directores: Aránzazu Otín Acín e Isidro Urriza Parroqué

Ingeniería Técnica Industrial, especialidad en Electrónica
Curso 2014-2015
Septiembre de 2015

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

PROPUESTA y ACEPTACIÓN DEL PROYECTO FIN DE CARRERA DE INGENIERÍA TÉCNICA

DATOS PERSONALES

APELLIDOS, Nombre
VIÑUÉS ALCONCHEL, JAIME

Nº DNI 18050103W Dirección MATEO ESTAUN LLANAS 4 4ºB

C.P. 22005 Localidad HUESCA

Provincia HUESCA Teléfono 630729366 NIA:

Firma:

DATOS DEL PROYECTO FIN DE CARRERA

INGENIERIA TECNICA INDUSTRIAL, Especialidad ELECTRÓNICA

TITULO DISEÑO DE UN RECEPTOR DE SEÑAL DCF77 PARA SINCRONIZACIÓN HORARIA POR RADIO

DEPÓSITO EN: ZAGUAN (Obligatorio) ☐ y CD-ROM ☐ (si PFC es tipo B aplicación informática)

DIRECTOR ISIDRO URRIZA Y ARANTXA OTIN

VERIFICACIÓN EN SECRETARÍA

El alumno reúne los requisitos académicos **(1)** para la adjudicación de Proyecto Fin de Carrera

SELLO DEL CENTRO

EL FUNCIONARIO DE SECRETARIA

Fdo.:

SE ACEPTA LA PROPUESTA DEL PROYECTO **(2)**	SE ACEPTA EL DEPÓSITO DEL PROYECTO
En Zaragoza, a 4 de SEPTIEMBRE de 2.015	En Zaragoza, a 7 de SEPTIEMBRE de 2.015
Fdo.: Isidro Urriza / Arantxa Otin	Fdo.: Isidro Urriza / Arantxa Otin
DIRECTOR DEL PFC	DIRECTOR DEL PFC

(1) Requisitos académicos: tener pendientes un máximo de 24 créditos o dos asignaturas para finalizar la titulación.

(2) Para que la propuesta sea aceptada por el Director, es imprescindible que este impreso esté sellado por la Secretaría de la EINA una vez comprobados los requisitos académicos.

RESUMEN

Este proyecto fin de carrera se ha desarrollado bajo la dirección de Dª. Aránzazu Otín Acín y D. Isidro Urriza Parroqué, profesores titulares del Departamento de Ingeniería Electrónica de la Universidad de Zaragoza.

El objeto de este proyecto es diseñar e implementar un receptor de señal DCF77 para sincronización horaria por radio, de prestaciones similares a los relojes de radiofrecuencia domésticos existentes en el mercado.

La motivación que ha llevado a realizarlo es la curiosidad que el autor siempre ha tenido sobre los dispositivos domésticos que usan este tipo de sincronización.

Quiero dedicar este proyecto a mi familia y amigos.

Agradezco su implicación en el proyecto a Aránzazu Otín Acín e Isidro Urriza Parroqué.

Índice de la memoria

Índice de ilustraciones

1. Introducción

1.1 Marco del trabajo

Este proyecto fin de carrera se ha desarrollado bajo la dirección de Dª. Aránzazu Otín Acín y D. Isidro Urriza Parroqué, profesores titulares del Departamento de Ingeniería Electrónica de la Universidad de Zaragoza.

El proyecto se ha llevado a cabo íntegramente en las instalaciones del campus EINA desde diciembre de 2014 a septiembre de 2015.

1.2 Justificación

1.2.1 Objeto del proyecto

El presente trabajo fin de carrera tiene como objetivo diseñar e implementar un receptor de señal DCF77 para sincronización horaria por radio, de prestaciones similares a los relojes de radiofrecuencia domésticos existentes en el mercado. A continuación se describe al detalle todas las fases del mismo, desde la definición teórica del problema a resolver, hasta su implementación electrónica.

DCF77 es una señal de radiofrecuencia, emitida desde Alemania, para la sincronización horaria. La emisión transporta una señal de datos de 1bit/s codificada en ancho de pulso y modulada en amplitud.

El diseño de este receptor tiene dos partes diferenciadas:

- Bloque de adquisición y procesado analógico de la señal de radio.
- Bloque de tratamiento digital en FPGA.

El bloque de adquisición y procesado analógico está implementado mediante amplificadores operacionales y componentes pasivos.

El bloque digital se realiza mediante lenguaje VHDL, usando la herramienta Xilinx ISE. Se ha simulado el funcionamiento del diseño con la herramienta ModelSIM. A continuación se ha programado sobre la placa de desarrollo Nexys2 de Digilent.

Por último, para validar el correcto funcionamiento del receptor, se han conectado ambos bloques, y se ha observado que el funcionamiento real del receptor implementado, se corresponde con el obtenido en las simulaciones (PSpice y ModelSIM).

1.2.2 Motivación del proyecto

Una de las primeras aplicaciones de las comunicaciones por radio ha sido la transmisión de información a distancia.

Poco después de que se descubriera la posibilidad de enviar información a través de las ondas, los primeros científicos de la radio ya estudiaban las posibilidades de enviar de esta manera información horaria.

La tecnología de los relojes de radiofrecuencia actuales se inventó hace más de un siglo pero no se han hecho populares hasta los años 90, en que el precio ha sido asequible para el consumidor medio.

Con la actual difusión de estos aparatos, hace pensar que a medio plazo, todo reloj irá provisto de un sistema de sincronización.

El autor dispone de varios de estos aparatos y siempre ha sentido curiosidad por su funcionamiento interno.

1.2.3 Objetivos

Los objetivos de este trabajo son:

- Estudio de la señal DCF77.
- Aprendizaje de conceptos relacionados con la radiofrecuencia y el tratamiento de señal.
- Diseño del receptor de señal DCF77 para sincronización horaria vía radio.

Gracias a éste proyecto, el autor ha adquirido muchos conocimientos no tratados en la carrera, sobre todo los relacionados con la electrónica analógica y el tratamiento de señal. Además se ha perfeccionado el manejo del software de simulación electrónica PSpice y el manejo de funciones avanzadas del osciloscopio.

2. Fundamentos

2.1 Difusión de señales horarias

Desde 1905, las señales de tiempo se han retransmitido vía radio.

En la actualidad existen estaciones de radio dedicadas a la emisión de señales horarias por todo el mundo.

Varios ejemplos de ello son la MSF emitida desde Reino Unido, la WWV emitida desde EEUU, la BMP emitida desde China, o la DCF77 emitida desde Alemania.

Todas estas señales presentan entre sí muchas similitudes. Varían normalmente en la transmisión, aunque la mayoría de ellas usan la modulación de amplitud.

2.2 DCF77

DCF77 es una estación de radio alemana, que emite una señal horaria de onda larga generada a partir de relojes atómicos.

Dicha emisora está situada en Mainflingen, a unos 25 Km al sureste de Frankfurt. La empresa privada T-Systems Media Broadcast (subsidiaria de German Telekom) se ocupa de la emisión de la señal, bajo la supervisión del Physikalisch-Technische Bundesanstalt (PTB), el laboratorio nacional de física de Alemania.

DCF77 lleva dando servicio como estación de frecuencia patrón desde 1959. La información de fecha y hora se añadió en 1973.

Emite con una potencia de 50 KW y con una frecuencia portadora de 77,5 KHz una señal digital modulada en amplitud.

La emisión transporta una señal de datos digital de 1bit/s codificada en ancho de pulso.

La codificación de tiempo transmitida por la emisora, se basa en los relojes de referencia atómicos del PTB que se sincronizan permanentemente con los relojes de referencia de otros países para distribuir el tiempo universal coordinado UTC.

El UTC es el principal estándar de tiempo por el cual el mundo regula los relojes y el tiempo. Está estrechamente relacionado con el tiempo medio de Greenwich (GMT), que está basado en la posición media del sol. Para la mayoría de los propósitos comunes, UTC es sinónimo de GMT, pero GMT ya no es el estándar definido con más precisión para la comunidad científica.

UTC se obtiene a partir de la media ponderada de las señales de relojes atómicos de alrededor de 70 laboratorios nacionales de todo el mundo. Se sincroniza con el tiempo medio de Greenwich y se le añade un segundo intercalar cuando es necesario.

Las zonas horarias de todo el mundo se expresan como desviaciones positivas y negativas respecto de UTC, tomando como referencia el meridiano cero (meridiano de Greenwich). Al pasar de un uso horario a otro en dirección este hay que sumar una hora, debido a que la Tierra gira de oeste a este, y viceversa.

UTC no cambia con las estaciones, pero la hora local sí puede cambiar si una jurisdicción contempla el horario de verano.

DCF77 emite en UTC+1 o UTC+2, dependiendo de si es horario estándar o de verano.

Esta señal es la más usada en Europa. Es conocida popularmente por ser la señal que suelen usar para sincronizar los relojes de radiofrecuencia vendidos en el continente europeo.

Las iniciales DCF77 provienen de D=Deutschland (Alemania), C=Señal de gran longitud de onda, F=Frankfurt, 77=Frecuencia: 77.5 kHz. En alemán Deutch Ca Fly 77 times.

La información de la señal se modula dos en amplitud y en fase.

2.2.1 Modulación de amplitud

La amplitud de la oscilación DCF77 se modula con las marcas de segundo. Al inicio de cada segundo se reduce la amplitud un 15% (-16dB) durante 0.1 s o 0.2 s por bit, dando como resultado una velocidad de transmisión de 1 bit/s.

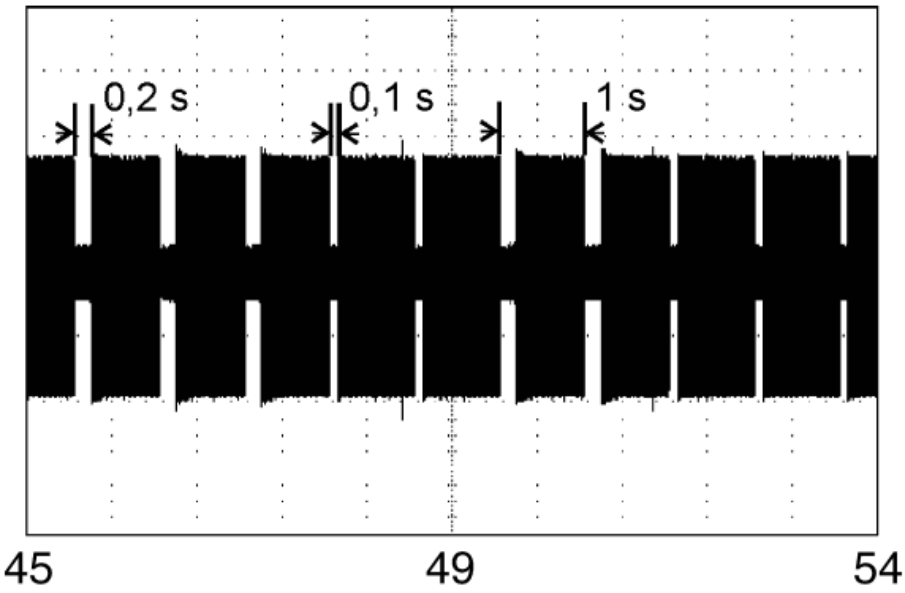

Figura 1. Señal DCF77 modulación AM

La excepción ocurre en el último segundo de cada minuto, en el que la amplitud no se reduce, que se usa como identificador del minuto siguiente.

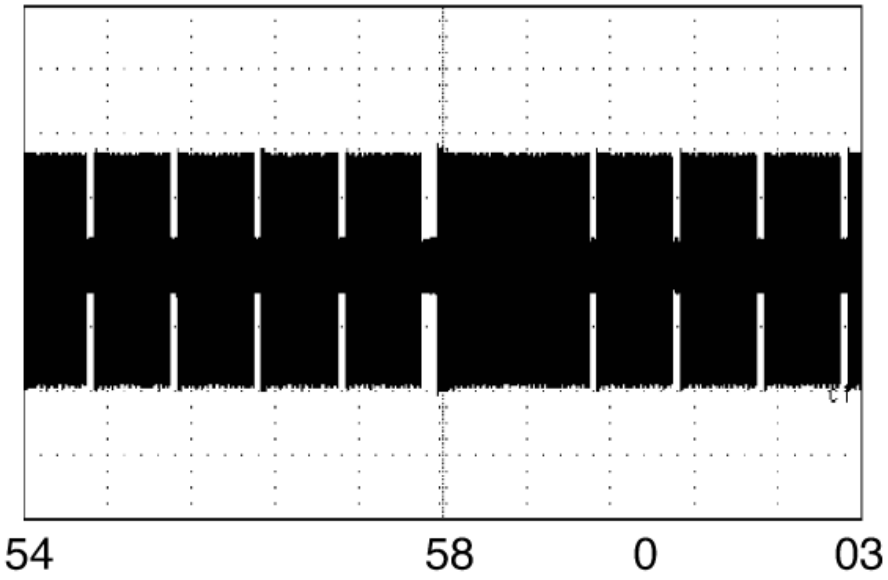

Figura 2. Detalle bit 58 señal DCF77 modulación AM

2.2.2 Modulación de fase

Adicionalmente a la modulación de amplitud, la señal también está modulada en fase.

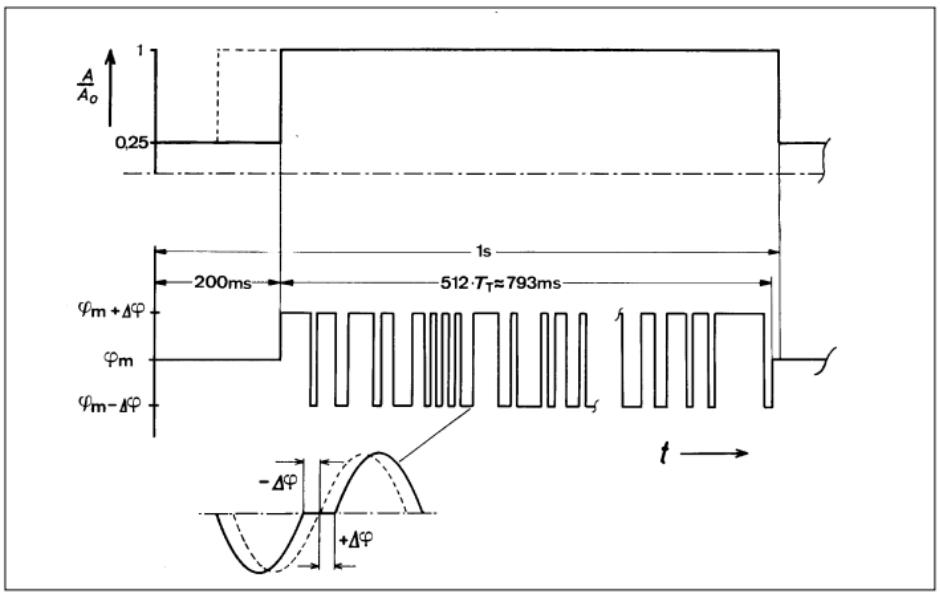

Figura 3. Modulación de fase DCF77

Empezando en 200 ms, y durante 793 ms se codifican los bits usando espectro expandido (DS-SS). El bit se multiplica por usa secuencia pseudoaleatoria de 512 bits usando una codificación de fase de ±13º.

2.2.3 Código de tiempo

La duración variable de las marcas de segundo sirve para la codificación binaria de la información. Marcadores de segundo con una duración de 0.1 s corresponden a un 0 binario, mientras que los de 0.2 s corresponden a un 1 binario.

Se transmite una trama cada minuto, a una velocidad de 1 bit/s. La trama completa comprende 59 bits más 1 segundo adicional para la sincronización.

Cada minuto se transmite la misma información:

- Hora
- Fecha
- Bits de paridad
- Ajuste de tiempo e información de la transmisión
- Información meteorológica

Lo que transmite es la información del siguiente minuto, y en el segundo de sincronización marca el comienzo de éste. Por ejemplo, a las 10:59 transmite el minuto 11:00.

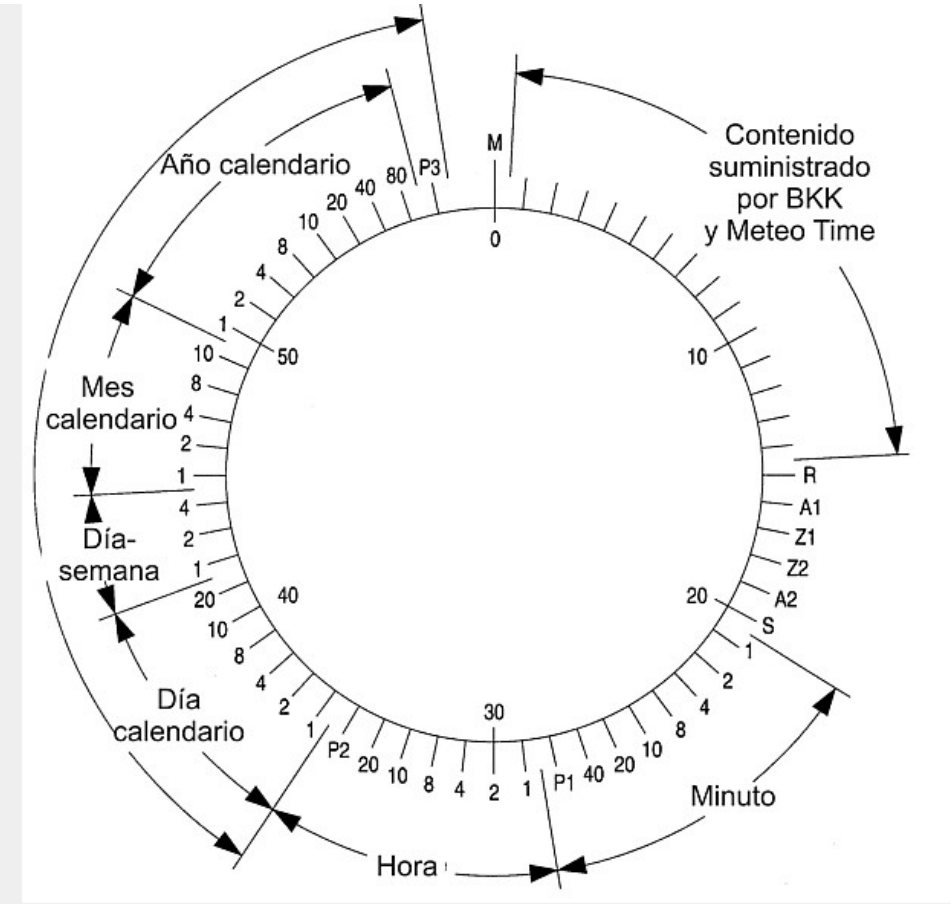

Figura 4. Codificación de la información de tiempo transmitida con DCF77

La secuencia de bits de la trama completa se muestra en la figura 4.

- Bits 1-20. Indicadores especiales
- Bits 21-27. Minutos
- Bits 28. Paridad asociada a los minutos
- Bits 29-34. Horas
- Bit 35. Paridad asociada a las horas
- Bits 36-58. Fecha
- Bit 59. Paridad asociada a la fecha

La información correspondiente a los bits 1-14 no se usaba hasta 2004. A partir de entonces, se usa para transmitir información meteorológica y alertas poblacionales. Es la única información provista por terceros. Actualmente la suministra la compañía suiza Meteo Time GmbH. Este servicio sigue siendo experimental, y sólo se transmite en modulación por amplitud. Pretende reemplazar en el futuro la red civil de sirenas de defensa. El PTB especifica que los datos relativos a esos segundos son de su responsabilidad.

Los bits 15-27 indican el código de la zona horaria, inclusión de segundo intercalar, e información relativa a la transmisión de la señal. El código de zona horaria se puede considerar como un desplazamiento del UTC+1 a UTC+2, y sirve para indicar el cambio al horario de verano. Un segundo intercalar es un ajuste que se añade cada cierto tiempo para sincronizar el tiempo a la rotación de la Tierra. La inclusión de segundo intercalar, indica que el siguiente minuto será de 61 segundos. Los bits que activan estos cambios se activan durante la hora previa al evento.

La información de fecha y hora está codificada en BCD (binary coded decimal), de modo que cada posición numérica se codifica de manera separada.

Los bits 35-58, correspondientes a la fecha, indican día del mes, día de la semana, mes y año. Para el día de la semana se codifica según el estándar ISO 8601 (se fija el lunes como primer día de la semana y es codificado como 1, martes el 2, y así sucesivamente hasta domingo el 7). Del año únicamente se transmiten decenas y unidades. Por ejemplo, el 2015 se transmite como 15.

Ignorando los bits especiales 1-20, un ejemplo de información contenida en una trama completa de un minuto podría ser:

14:45 Lunes 31 Agosto 15

2.2.4 Disponibilidad y alcance de la señal

DCF77 opera las 24 horas del día. Se garantiza anualmente una disponibilidad de emisión del 99,7%.

Desde 1977 se dispone de un emisor sustituto y una antena de reserva, de modo que no se realizan desconexiones largas para realizar trabajos de mantenimiento. Únicamente deben esperarse interrupciones cortas de pocos minutos en casos en que haya que cambiarse al emisor sustituto y antena de reserva debido a averías inesperadas y trabajos de mantenimiento.

La figura muestra la disponibilidad anual de la señal, sin considerar las interrupciones menores de dos minutos de duración.

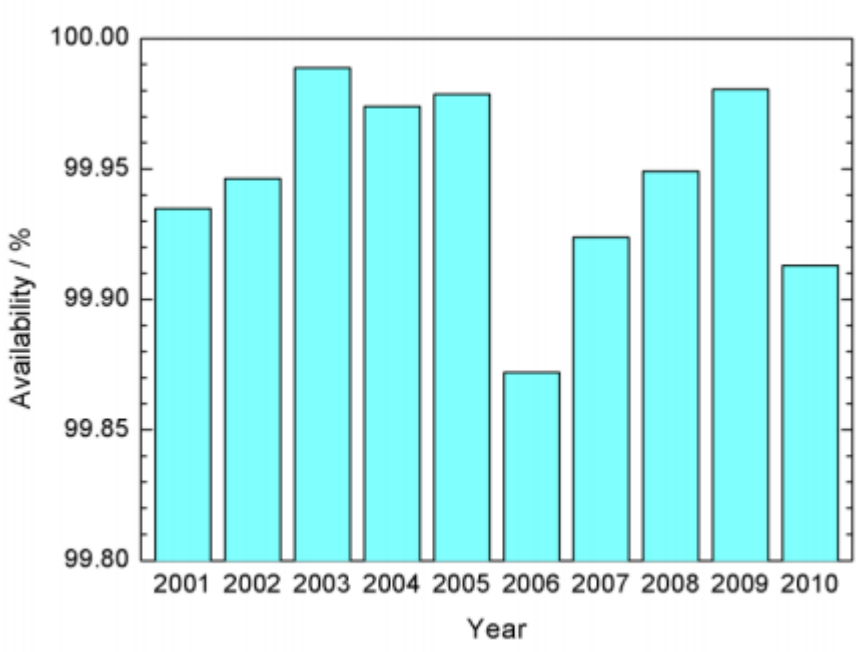

Figura 5. Disponibilidad anual de la señal DCF77. No se consideran los cortes inferiores a 2 minutos

DCF77 opera con una frecuencia de 77,5 KHz. La ventaja que ofrece esta frecuencia en la banda de LF, es que las ondas se propagan a baja altitud, y su elevada longitud de onda favorece que la onda electromagnética siga la curvatura terrestre, en lugar de seguir una línea recta desde el emisor y no llegar hasta lo que hay detrás del horizonte.

La señal de DCF77 irradiada por la antena emisora alcanza al receptor por dos caminos: se disemina como onda terrestre a lo largo de la superficie terrestre, y por otro lado como onda espacial tras la reflexión en la ionosfera.

Para distancias menores de 500 Km la onda terrestre es muy estable.

Para distancias entre 600 y 1100 Km la onda terrestre es aproximadamente del mismo tamaño que la onda espacial. Si las ondas se encuentran en oposición de fase, se anulan mutuamente. En caso de estar en fase se tiene un incremento de la fuerza de campo. La pulsación entre las ondas terrestres y espaciales varía lentamente (al menos un cuarto de hora) de modo que hay suficiente tiempo para recibir la señal con información de tiempo.

A distancias mayores de 1100 Km predomina la onda espacial.

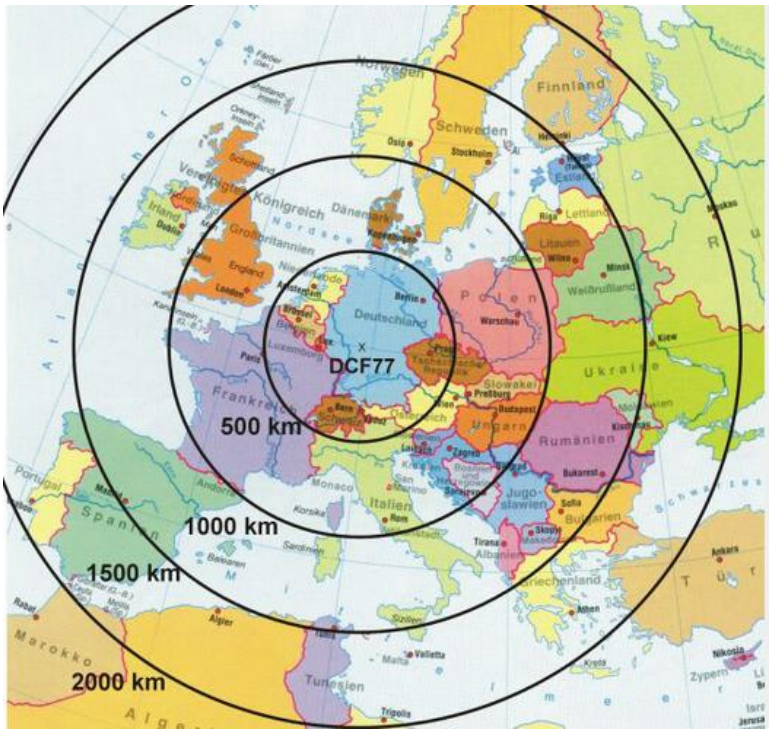

Figura 6. Alcance de la señal DCF77

El alcance durante el día es de 1900 Km y durante la noche 2100 Km. Para receptores más alejados la señal los alcanza después de varias reflexiones, y se ha demostrado que la cobertura sólo es fiable en casos aislados.

El orden de sincronismo de la señal depende con la distancia. Así, un receptor a 1000 Km recibe la señal con uno 3 ms de retraso. Esto no supone un problema, salvo en los casos de que se requiera una precisión alta.

2.2.5 Uso aplicado

Existe una tendencia incremental en el uso de relojes radiocontrolados (relojes de pulsera, despertadores y de pared).

El uso más habitual es el de personas privadas, pero también se usa en el control de servicio de tiempo en estaciones de tren, en el campo de la telecomunicación, en estaciones de radio y televisión, en relojes de contactos de tarifas de compañías de energía eléctrica y relojes en las instalaciones de semáforos.

Los sistemas de difusión del tiempo vía satélite son el gran competidor de esta tecnología, puesto que ofrecen una mayor precisión. La ventaja principal de las señales de onda larga es que la recepción apenas se ve afectada por obstáculos o edificios, y penetra en ellos fácilmente. La señal se puede recibir fácilmente sin antena externa. Con una antena interna de ferrita es suficiente. Mientras que los sistemas vía satélite requieren una antena externa con una vista libre al cielo.

2.3 Modulación AM

La modulación de amplitud, más conocida como AM, consiste en cambiar la amplitud de una portadora de frecuencia alta de acuerdo a la amplitud de la señal de información (señal moduladora). El cambio de frecuencia se realiza para mejorar la efectividad de la onda para radiarse a través de una antena. La onda resultante de la suma de ambas se conoce como onda modulada.

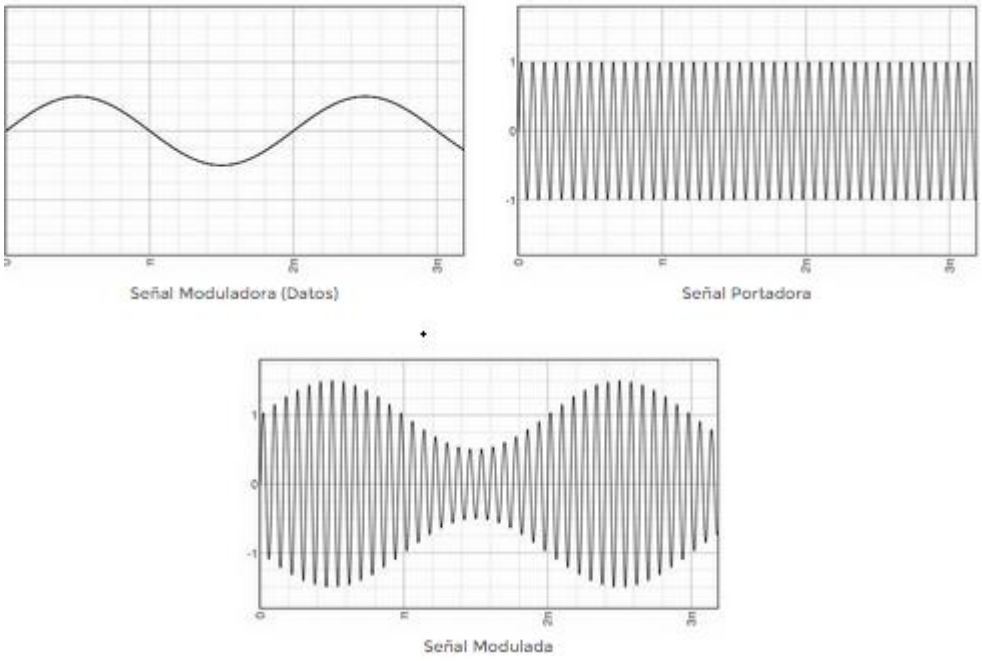

Figura 7. Señales de modulación AM

En la envolvente de la señal modulada se encuentra la información de la señal de datos.

El espectro de una señal modulada está compuesto por tres señales:

- La portadora.
- Señal mitad de amplitud de la moduladora y frecuencia la de la portadora menos la de la moduladora.
- Señal mitad de amplitud de la moduladora y frecuencia la de la portadora más la de la moduladora.

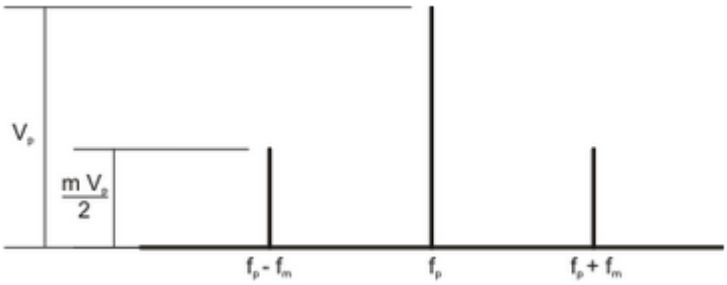

Figura 8. Espectro de una señal modulada en amplitud

A las señales situadas a ambos lados de la frecuencia de la portadora se las conoce como bandas laterales, y son las que contienen la información de la señal.

2.3.1 Recepción de AM

La recepción AM es el proceso inverso a la transmisión AM. El receptor de AM convierte la señal modulada en la señal de información. A este proceso se le llama demodulación.

Al demodular una señal AM, la portadora y la envolvente de la señal (las bandas laterales) se trasladan del espectro de radiofrecuencia a la fuente original de información.

Un receptor de AM debe recibir, amplificar y demodular una señal de radiofrecuencia.

El método más sencillo de demodulación es el detector de envolvente. Consta de un rectificador y un filtro paso bajo.

El rectificador más simple es un diodo. Debido a que el diodo es un dispositivo no lineal, cuando se aplican dos o más señales en su entrada, se produce una mezcla no lineal. Por tanto la salida contiene las frecuencias de entrada originales y una serie de componentes frecuenciales adicionales.

3. Bloques principales

3.1 Subsistema analógico

El circuito analógico recepciona la señal de radiofrecuencia de 77,5 KHz, la acondiciona, y demodula la señal AM.

La finalidad de este bloque es proporcionar al subsistema digital una señal con valores de tensión de 0 y 5V, de acuerdo con la amplitud de la señal portadora en cada instante de tiempo.

3.1.1 Recepción de señal

Una antena es un conductor metálico que tiene como objetivo emitir o capturar ondas electromagnéticas hacia el espacio libre.

En caso de transformar energía eléctrica en ondas electromagnéticas, hablamos de antena transmisora. En caso contrario, hablamos de antena receptora.

Es el elemento de enlace que conecta las señales de radiofrecuencia a los circuitos eléctricos.

El funcionamiento de una antena se basa en la radiación producida al circular una corriente eléctrica por un conductor. Dicha corriente crea un campo magnético alrededor del conductor.

La señal a captar es de 77,5 KHz. Se puede calcular la longitud de onda con la siguiente ecuación:

$$\lambda = \frac{c}{f}$$

Donde λ representa la longitud de onda, c la velocidad de la luz (300.000.000 m/s) y f la frecuencia.

$$\lambda = \frac{c}{f} = \frac{300.000.000\,\frac{m}{s}}{77.500\ Hz} = 3871\ m$$

Utilizando una antena Marconi simple, sería necesario que su longitud fuese de:

$$\frac{\lambda}{4} = \frac{3871}{4} = 968\ m$$

Dicho valor impide su uso práctico, de manera que se utilizará una antena de ferrita con una bobina y un condensador.

La antena de ferrita sintonizada consiste en un núcleo de ferrita sobre el que se arrolla una bobina. En paralelo a dicha bobina, se conecta un condensador en paralelo para lograr la resonancia.

Esta antena de radiofrecuencia, aprovecha su alta permeabilidad para concentrar las componentes magnéticas de las ondas de radio, lo que significa que es directiva.

La operación óptima sucede cuando las líneas de fuerza del campo magnético se alinean con la antena. Esto ocurre cuando la antena se coloca en dirección perpendicular al emisor de la señal a sintonizar. Por el contrario, cuando la antena está en línea con el emisor, la señal captada es mínima.

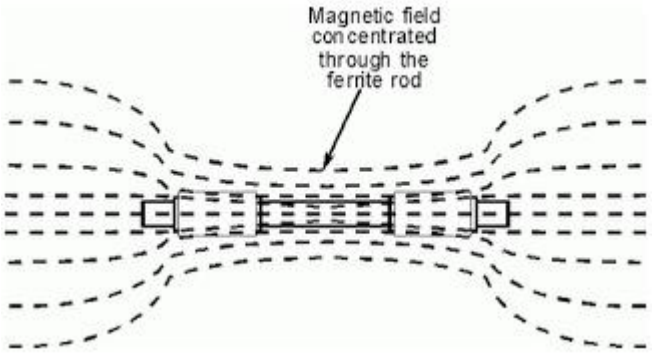

Figura 9. Líneas de campo en una antena de ferrita

La principal ventaja de este tipo de antenas reside en su portabilidad, debido a su pequeño tamaño.

Se usa principalmente para recepción de onda media y larga.

3.1.2 Acondicionamiento de señal

Para amplificar la señal se puede recurrir a etapas amplificadoras básicas basadas en amplificadores operacionales (etapa inversora y no inversora), o pueden usarse los filtros para realizar la amplificación.

En un principio se diseñó este bloque mediante una etapa amplificadora no inversora, para adaptar la impedancia de la antena y amplificar al mismo tiempo.

Pero hubo que sustituirlo por una etapa seguidora, que adaptara la impedancia de la antena, y se aprovechó la necesidad de implementar un filtro notch para realizar la amplificación con el mismo.

El filtro pasa banda limita en banda la señal que pasa al demodulador, reduciendo la posible interferencia de señales diferentes a la DCF77.

3.1.3 Demodulador

En la señal DCF77, la señal de datos está modulada sobre una portadora de 77,5 KHz.

La información de la señal se encuentra en las bandas laterales. Para extraer dicha información hay que devolver la señal a la banda base original.

Debido a que el diodo es un dispositivo no lineal, en él sucede una mezcla no lineal cuando se aplican dos o más señales en su entrada. Por tanto, la salida contiene las frecuencias de entrada originales, y una serie de componentes frecuenciales adicionales.

El rectificador devuelve la señal de datos a su banda base original, además de la portadora modulada y los armónicos de la misma.

El filtro paso bajo se encarga de eliminar las componentes de alta frecuencia, quedándose sólo con la parte baja del espectro. De este modo, se extrae la señal de datos en banda base, que es la que contiene la información.

El comparador obtiene una señal digital a partir de la señal del paso bajo.

3.2 Subsistema digital

El circuito digital recibe como entrada la señal de salida del circuito analógico.

La señal analógica se conectará a la parte digital a través de uno de los conectores PMOD de la tarjeta Nexys 2.

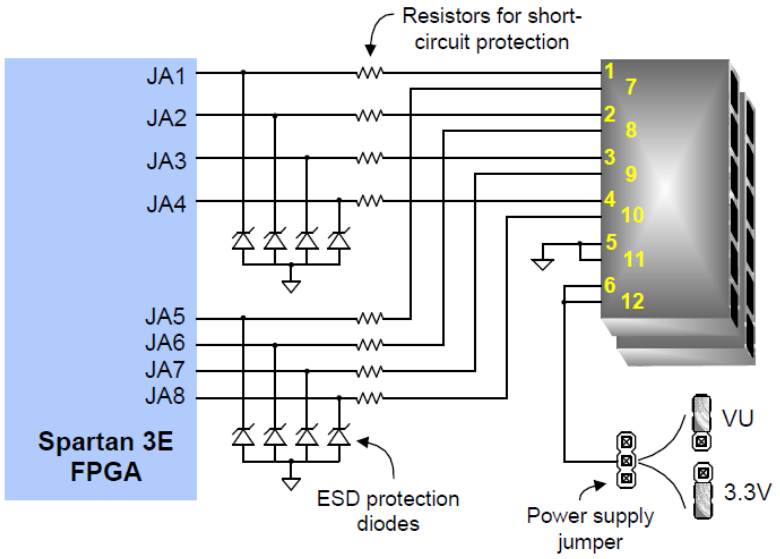

Figura 10. Conexión de la FPGA Spartan 3E con los conectores PMOD de entrada/salida

Hay que extraer la información contenida en cada bit de un segundo: “0”, “1” o SYNC, y construir con ella la trama completa de información, para su posterior visualización.

El proceso para la extracción de datos es el siguiente:

- Muestrear la señal.
- Sumar los valores de las muestras.
- Comparar con umbrales para realizar la decisión sobre la información contenida.
- Almacenar los datos.
- Validar la cadena de datos.
- Visualización

El bloque de entradas/salidas del diseño digital es el de la figura 11.

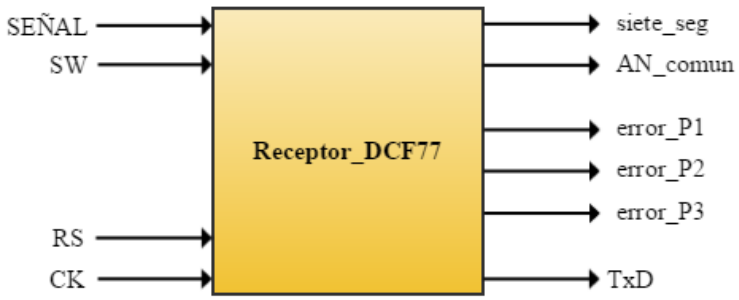

Figura 11. Bloque entradas/salidas del diseño digital

4. Desarrollo de la solución

4.1 Subsistema analógico

En la parte analógica se emplean amplificadores operacionales TL082 y alimentación simétrica ±5 V. A la salida del circuito analógico se proporciona una señal de datos con una tensión entre 0 y 3,3 V compatible con las entradas de la tarjeta de desarrollo usada para implementar la parte digital.

Se utilizan condensadores de desacoplo para reducir el ruido que puede producirse en los circuitos de conmutación. Se utilizan 3 en cada una de las alimentaciones. Se usan valores de 100 pF, 100 nF y 100 µF para cada uno de los intervalos de frecuencias (altas, medias y bajas respectivamente).

4.1.1 Antena de ferrita

El elemento crítico para la recepción es la bobina, de modo que su construcción condicionará el funcionamiento del sistema.

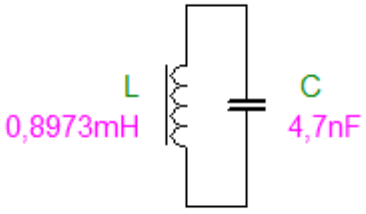

Figura 12. Esquemático antena de ferrita

La frecuencia de resonancia viene dada por:

$$f_r = \frac{1}{2\pi\sqrt{LC}}$$

En primer lugar se escoge un valor comercial de C, por ejemplo 4,7nF, y de ahí se deduce la L necesaria:

$$L = \frac{1}{C(2\pi f_r)^2} = \frac{1}{4{,}7 * 10^{-9}(2 * \pi * 77500)^2} = 0{,}8973mH$$

Será necesario añadir un condensador de capacidad variable en paralelo para el ajuste fino de la frecuencia de resonancia f_r.

El valor de la bobina viene dado por la siguiente ecuación:

$$L = \mu\frac{N^2A}{l}$$

donde:

L: inductancia [H]

μ: permeabilidad magnética del núcleo = $\mu_0\mu_r = 4 * \pi * 10^{-7}\mu_r[\frac{Tesla * metro}{amperio}]$

N: número de vueltas

A: área de la espira [m^2]

l: longitud del solenoide [m]

Escogiendo un cilindro de ferrita ($\mu_r = 10$) de 1cm de diámetro, y fijando l=1cm, el número de vueltas necesario es:

$$N = \sqrt{\frac{Ll}{\mu A}} = \sqrt{\frac{0{,}8973 * 10^{-3} * 1 * 10^{-2}}{4 * \pi * 10^{-7} * 10 * \pi * (5 * 10^{-3})^2}} = 95\ vueltas$$

Para el desempeño de este proyecto no se ha fabricado dicha antena manualmente. Se ha usado una antena comercial ya sintonizada para simplificar la tediosa labor del bobinado, y asegurar la no dispersión de parámetros fundamentales, que afectarán ineludiblemente al resto de las etapas.

La antena es marca HKW, modelo FTD02041R (100mm).

Figura 13. Antena de ferrita usada en el prototipo

Se desconoce el valor de L, R_B y C de la antena sintonizada, ya que el fabricante no proporciona dichos valores.

Para medir la resistencia interna de la bobina, se ha usado un multímetro. $R_B = 2{,}6\Omega$

El valor de L y C se determina a través de la siguiente espresión:

$$w_r^2 = \frac{1}{LC}$$

Usando el generador de funciones para excitar la antena, y variando la frecuencia de excitación, se mide con el osciloscopio la frecuencia de resonancia $f_r = 77{,}5\ KHz$ (por tanto w_r es conocida).

Para determinar los valores de L y C, la solución estriba en introducir un condensador de valor conocido en paralelo para modificar la frecuencia de resonancia. Se usa el valor comercial $4{,}7\ \mu F$. Se usa este valor para que sea dominante sobre el condensador de la antena de ferrita sintonizada al realizar la medida. Se mide el valor del condensador con el multímetro (debido a la gran tolerancia que suelen tener los condensadores electrolíticos). $C_1 = 4{,}3\ \mu F$.

Con éste nuevo condensador, la nueva frecuencia de resonancia del sistema es $f_r' = 2{,}56\ KHz$

$$\begin{cases} w_r^2 = \dfrac{1}{LC} \\ w_r'^2 = \dfrac{1}{LC'} \end{cases}$$

Siendo $C' = C + C_1$

$$w_r^2 LC = w_r'^2 LC' \rightarrow w_r^2 C = w_r'^2 (C + C_1) \rightarrow C(w_r^2 - w_r'^2) = w_r'^2 C_1$$

$$C = \frac{w_r'^2 C_1}{w_r^2 - w_r'^2} = \frac{(2 * \pi * 2560)^2 * 4{,}3 * 10^{-6}}{(2 * \pi * 77500)^2 - (2 * \pi * 2560)^2} = 4{,}697\ nF$$

El equivalente sonda (en posición x10) + osciloscopio se representa por el paralelo de una resistencia equivalente $R^* = 10M\Omega$ y una capacidad equivalente $C^* =$

$13pF$. De modo que dicha capacidad equivalente apenas está afectando a la medida. El condensador real que está en paralelo en la antena sintonizada tiene el siguiente valor:

$$C_{real} = C - C^* = 4{,}697\ nF - 13pF = 4{,}684nF$$

Conocido C_{real}, sólo queda despejar L:

$$w_r^2 = \frac{1}{LC_{real}} \rightarrow L = \frac{1}{C_{real}w_r^2} == \frac{1}{4{,}684 * 10^{-9}(2 * \pi * 77500)^2} = 0{,}9\ mH$$

De modo que el circuito equivalente de la antena es el siguiente:

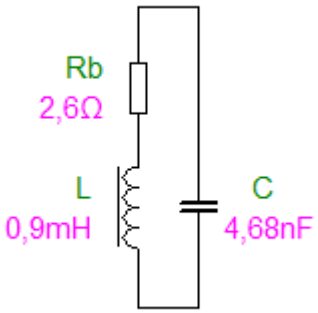

Figura 14. Modelo equivalente de la antena de ferrita

Vistos los valores equivalentes del circuito de la antena de ferrita se hace precisa la inclusión de un elemento para adaptar su impedancia. Por ello, se requiere de un seguidor de tensión.

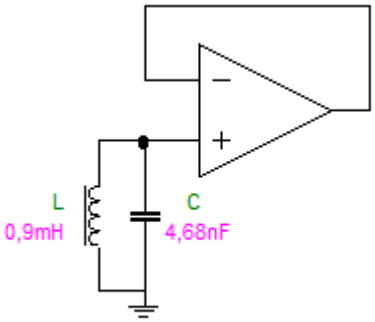

Figura 15. Seguidor de tensión

4.1.2 Filtro notch

Debido a que la señal que suministra la antena es pequeña. Se hizo una prueba con una etapa amplificadora en configuración no inversora, para amplificar la señal y poder visualizar el espectro de frecuencias a la salida. El espectro es el de la figura 16.

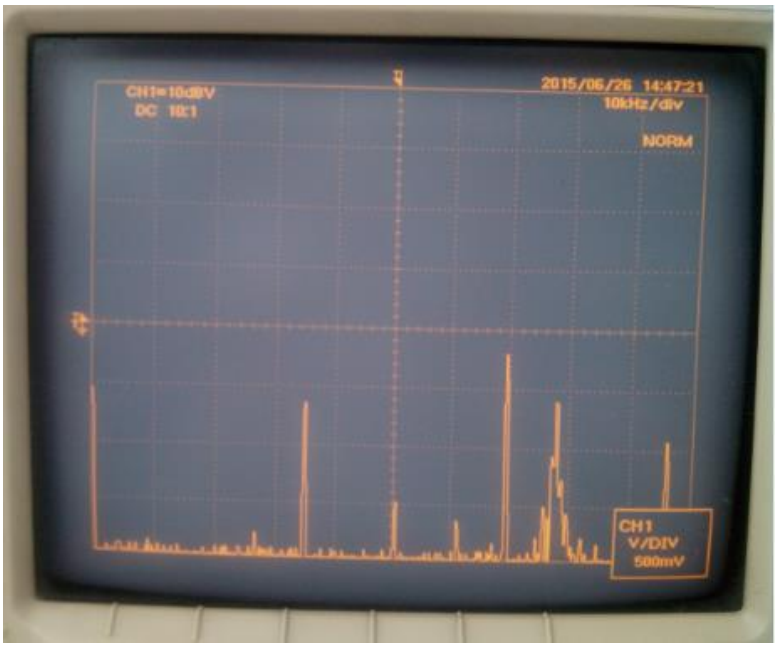

Figura 16. Espectro de señal de la antena amplificada

Como puede apreciarse, hay una componente espectral de 69 KHz muy cerca de la señal DCF77 de 77,5 KHz.

El filtro paso banda no es suficiente para eliminar la componente espectral de 69 KHz. Por este motivo se opta por la inclusión de un filtro notch, que además de atenuar la componente se usa para dar ganancia.

Se opta por un filtro notch de segundo orden basado en la topología de realimentación múltiple.

El filtro notch de segundo orden basado en la topología de realimentación múltiple, se construye a partir de la misma estructura que el filtro paso banda del siguiente capítulo, de modo que va a realizarse un análisis matemático del mismo:

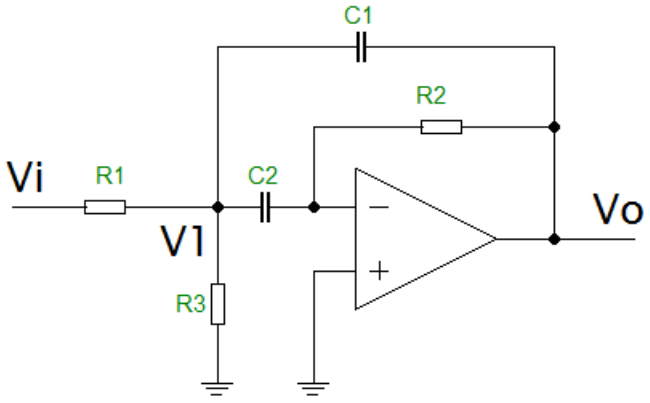

Figura 17. Filtro paso banda de segundo orden de realimentación múltiple (MFB)

El amplificador operacional actúa como un diferenciador con respecto a la tensión V_1, de modo que:

$$V_0 = -sR_2C_2V_1$$

Sumando corrientes en el nodo V_1:

$$\frac{V_i - V_1}{R_1} + \frac{V_0 - V_1}{\frac{1}{sC_1}} + \frac{0 - V_1}{\frac{1}{sC_2}} - \frac{0 - V_1}{R_3} = 0$$

Sustituyendo $V_1 = \frac{-V_0}{sC_2R_2}$, en la ecuación y operando:

$$H(s) = \frac{V_0}{V_i} = \frac{s\frac{C_2R_2R_3}{R_1 + R_3}}{1 + s^2\frac{R_1R_2R_3C_1C_2}{R_1 + R_3} + s\frac{R_1R_3(C_1 + C_2)}{R_1 + R_3}}$$

Se hace $s \to jw$:

$$H(jw) = \frac{V_0}{V_i} = \frac{-jw\frac{C_2R_2R_3}{R_1 + R_3}}{1 - w^2\frac{R_1R_2R_3C_1C_2}{R_1 + R_3} + jw\frac{R_1R_3(C_1 + C_2)}{R_1 + R_3}}$$

La forma estándar cualquier función pasa banda de segundo orden es:

$$H(jw) = H_{0BP}H_{BP}(jw) = H_{0BP}\frac{\frac{(\frac{jw}{w_0})}{Q}}{1 - \left(\frac{w}{w_0}\right)^2 + \frac{(\frac{jw}{w_0})}{Q}}$$

Donde H_{0BP} es la ganancia en resonancia.

Para escribir la función de transferencia en la forma estándar, se hace que $w^2\frac{R_1R_2R_3C_1C_2}{R_1+R_3} = \left(\frac{w}{w_0}\right)^2$, y queda:

$$w_0 = \sqrt{\frac{R_1 + R_3}{R_1R_2R_3C_1C_2}}$$

Se hace que $jw\frac{R_1R_3(C_1+C_2)}{R_1+R_3} = \frac{(\frac{jw}{w_0})}{Q}$, y operando queda:

$$Q = \frac{w_0R_2C_1C_2}{C_1 + C_2}$$

Para terminar, se hace que$-jw\frac{C_2R_2R_3}{R_1+R_3} = H_{0BP} \times \frac{(\frac{jw}{w_0})}{Q}$, y queda:

$$H_{0BP} = \frac{-\frac{R_2}{R_1}}{1+\frac{C_1}{C_2}}$$

Para simplificar estas expresiones se hace $C_1 = C_2 = C$, y queda:

$$w_0 = \frac{\sqrt{R_1+R_3}}{C\sqrt{R_1R_2R_3}} \qquad Q = \frac{\sqrt{R_2(R_1+R_3)}}{2\sqrt{R_1R_3}} \qquad H_{0BP} = \frac{-R_2}{2R_1}$$

Simplificando las expresiones haciendo $R_1 \ll R_3$ queda:

$$w_0 = \frac{1}{C\sqrt{R_2R_3}} \qquad Q = \frac{1}{2}\sqrt{\frac{R_2}{R_3}} \qquad H_{0BP} = \frac{-R_2}{2R_1}$$

Se opta por un filtro notch de segundo orden basado en la topología de realimentación múltiple, con frecuencia central de 69 KHz, factor de calidad Q de 10 y una ganancia de 10.

Se escogen los condensadores C=1 nF.

$$\begin{cases} R_2R_3 = \dfrac{1}{(Cw_0)^2} \\ \dfrac{R_2}{R_3} = (2\mathrm{Q})^2 \end{cases}$$

Si se escoge R_2=46,4 KΩ. Para Q=10, entonces R_3=118 Ω. La ganancia del módulo paso banda será de 1, ya que la ganancia de 100 del filtro de fijará con la resistencia de realimentación R_6 de la siguiente etapa, entonces R_1=23,2 KΩ.

El filtro completo queda:

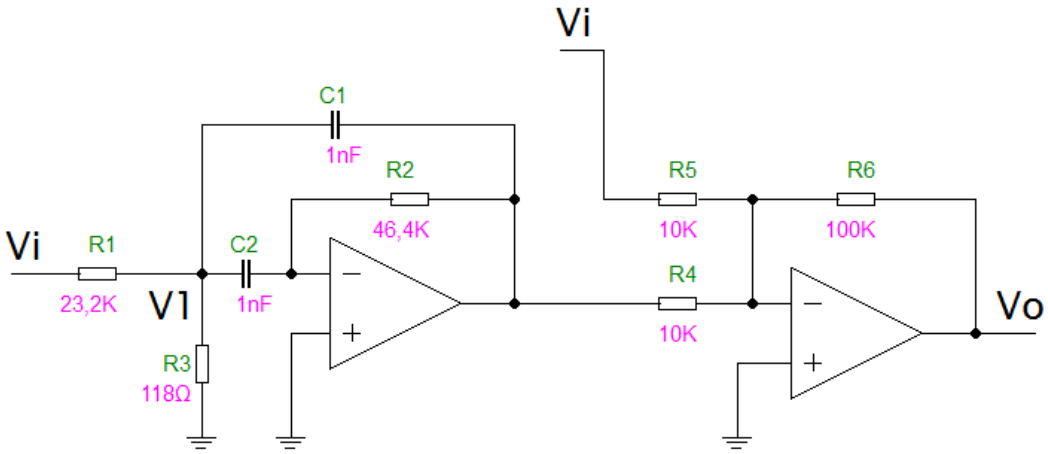

Figura 18. Filtro notch de segundo orden

Las resistencias R1 y R3 se sustituyen por dos potenciómetros para el ajuste fino de la frecuencia y la Q del filtro.

La respuesta del filtro simulada en PSpice es:

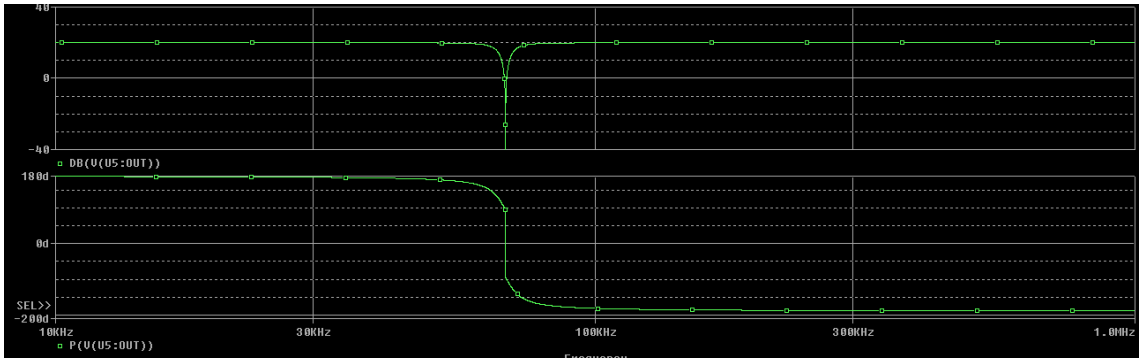

Figura 19. Respuesta simulada del filtro notch

La respuesta en ganancia del filtro medida en el laboratorio es:

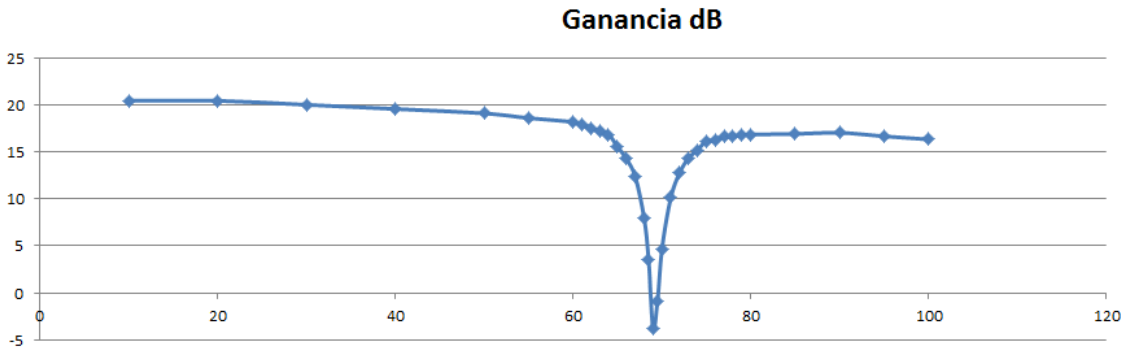

Figura 20. Respuesta real del filtro notch

Tras la inclusión del filtro notch, la componente espectral de 69 KHz se ha atenuado casi por completo.

4.1.3 Filtro pasa banda

Tras simular en PSpice las funciones de transferencia de filtros pasa banda de diferentes órdenes para diferentes valores de Q, se opta por escoger un filtro de cuarto orden basado en la topología de realimentación múltiple, Butterworth, con frecuencia central de 77,5 KHz, factor de calidad Q de 10 y ganancia unidad.

Se ha escogido la topología de realimentación múltiple porque tiene una sensibilidad baja frente a variaciones de los valores de los componentes.

Usando el programa FilterPro, se obtienen los valores de frecuencia, factor de calidad Q y ganancia para cada una de las dos etapas que forman el filtro.

f_{01}= 74,8068 KHz f_{02}= 74,8068 KHz

Q_1= 14,151 Q_2= 14,151

H_{0BP1}= 1 H_{0BP2}= 1

Para el ajuste de los valores de cada una de las dos etapas, se usa el mismo procedimiento descrito en el filtro notch. Al igual que en el caso anterior, se sustituyen las resistencias R1 y R3 por sendos potenciómetros para el ajuste fino de la frecuencia central y el Q del filtro. El filtro completo queda:

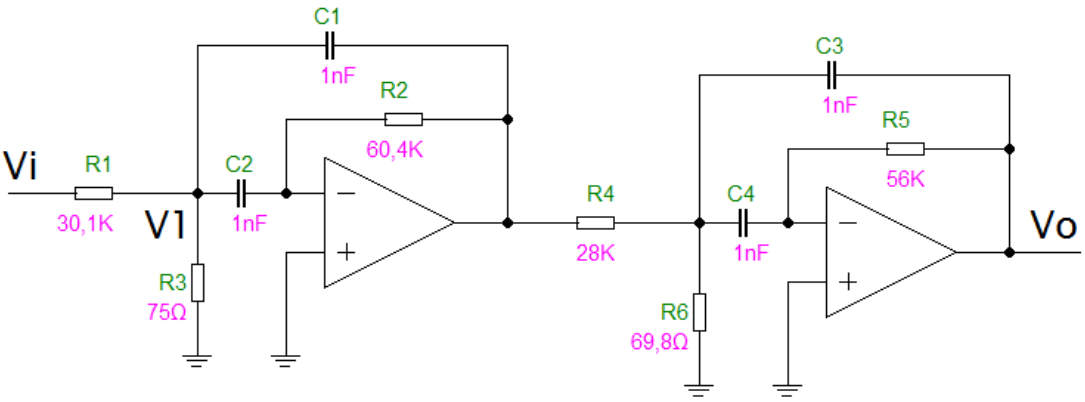

Figura 21. Filtro paso banda de cuarto orden. Estructura MFB

La respuesta del filtro simulada en PSpice es:

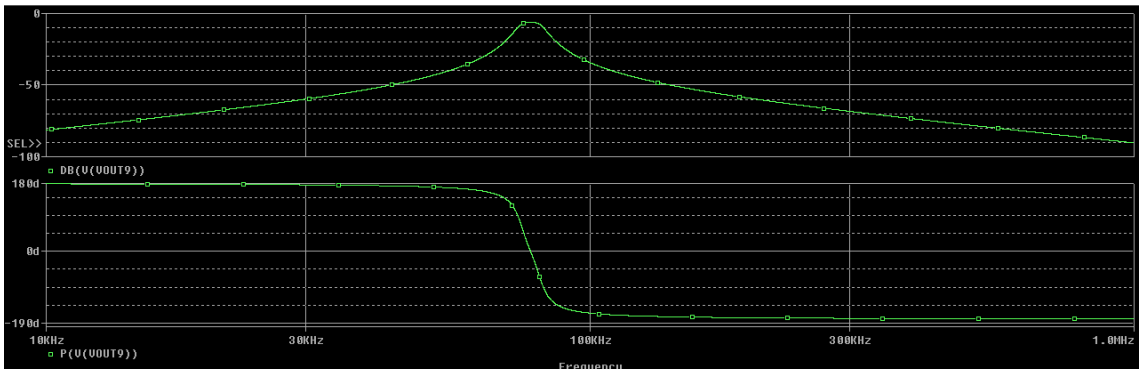

Figura 22. Respuesta simulada del filtro paso banda

La respuesta en ganancia del filtro medida en el laboratorio es:

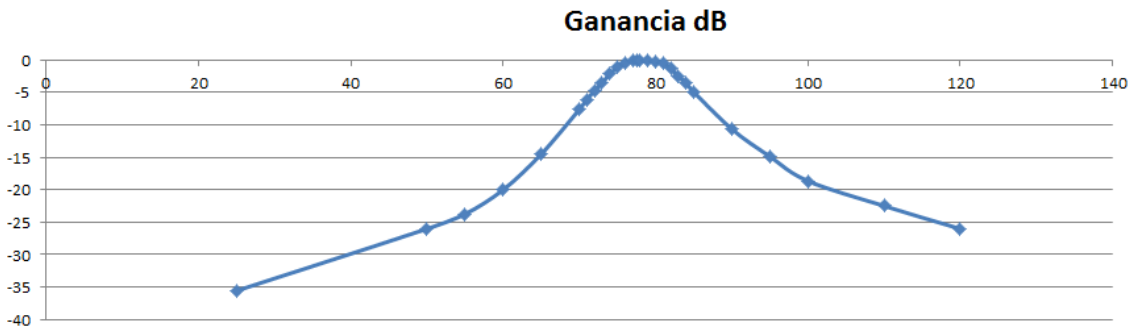

Figura 23. Respuesta real del filtro paso banda

4.1.4 Rectificador de media onda

La estructura del rectificador de media onda formada por un diodo y un amplificador operacional no es válida para esta aplicación debido a que el amplificador entra en modo no lineal en los semiciclos negativos de la onda de entrada, y limita la frecuencia máxima de operación.

Para la frecuencia de operación de 77,5 KHz se requiere la estructura de dos diodos. Se usan diodos Schottky para mejorar las conmutaciones.

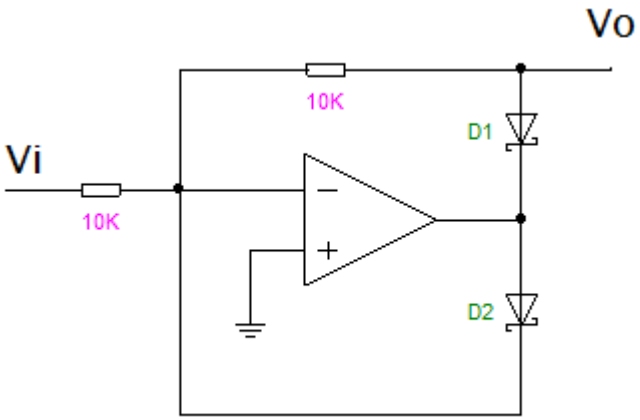

Figura 24. Rectificador de media onda de dos diodos

Debido a que la salida propocionada por el rectificador de media onda está invertida, se conecta un inversor de tensión.

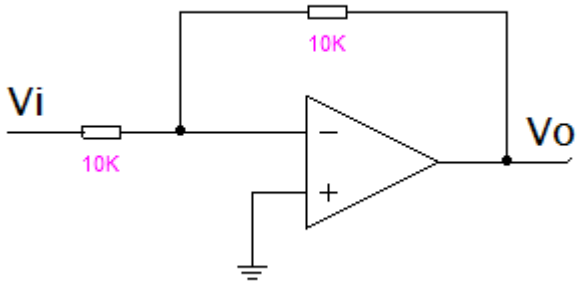

Figura 25. Inversor de tensión

4.1.5 Filtro paso bajo

La señal de datos en banda base tiene un ancho de banda de 100 Hz. El filtro paso bajo debe de eliminar la portadora de 77,5 KHz y los armónicos de la misma (155 KHz, 232,5 KHz, etc).

Para ello se opta por un filtro paso bajo de segundo orden, basado en la estructura Sallen-Key, con frecuencia de corte de 200 Hz, con $Q = \frac{1}{\sqrt{2}}$ (es el mayor valor de Q posible antes de que la respuesta alcance el pico de resonancia. Esta curva se denomina como máximamente plana, respuesta Butterworth) y ganancia unidad.

Se procede a analizar matemáticamente la estructura del filtro paso bajo Sallen-Key de segundo orden:

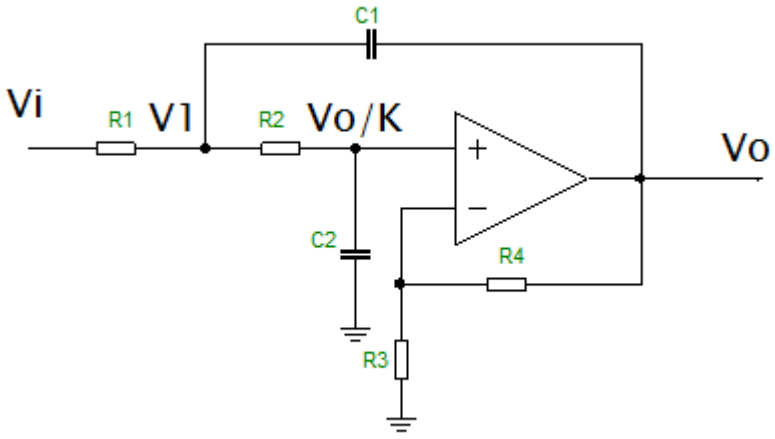

Figura 26. Filtro paso bajo de segundo orden Sallen-Key

El bloque de ganancia es un amplificador operacional configurado como no inversor, de modo que:

$$K = 1 + \frac{R_4}{R_3}$$

La tensión V_1 y V_0 se relacionan de la siguiente forma:

$$V_0 = \frac{KV_1}{sR_2C_2 + 1}$$

Sumando corrientes en el nodo V_1:

$$\frac{V_i - V_1}{R_1} + \frac{\frac{V_0}{K} - V_1}{R_2} + \frac{V_0 - V_1}{\frac{1}{sC_1}} = 0$$

Sustituyendo $V_1 = \frac{(sR_2C_2+1)V_0}{K}$, en la ecuación y operando:

$$H(s) = \frac{V_0}{V_i} = \frac{K}{1 + s^2R_1R_2C_1C_2 + s[(1-K)R_1C_1 + R_1C_2 + R_2C_2]}$$

Se hace $s \rightarrow jw$:

$$H(jw) = \frac{V_0}{V_i} = \frac{K}{1 - w^2R_1R_2C_1C_2 + jw[(1-K)R_1C_1 + R_1C_2 + R_2C_2]}$$

La forma estándar cualquier función pasa bajo de segundo orden es:

$$H(jw) = H_{0LP}H_{LP}(jw) = H_{0LP}\frac{1}{1 - \left(\frac{w}{w_0}\right)^2 + \frac{(\frac{jw}{w_0})}{Q}}$$

Donde H_{0LP} es la ganancia en continua.

Para escribir la función de transferencia en la forma estándar, se hace que:

$$H_{0LP} = K$$

Se hace que: $w^2R_1R_2C_1C_2 = \left(\frac{w}{w_0}\right)^2$, y queda:

$$w_0 = \frac{1}{\sqrt{R_1R_2C_1C_2}}$$

(puede verse cómo w_0 no es más que la media geométrica de cada etapa individual $w_1 = \frac{1}{R_1C_1}$ y $w_2 = \frac{1}{R_2C_2}$)

Para terminar se hace que $jw[(1-K)R_1C_1 + R_1C_2 + R_2C_2] = \frac{(\frac{jw}{w_0})}{Q}$, y queda:

$$Q = \frac{1}{(1-K)\sqrt{\frac{R_1C_1}{R_2C_2}} + \sqrt{\frac{R_1C_2}{R_2C_1}} + \sqrt{\frac{R_2C_2}{R_1C_1}}}$$

En el caso particular de ganancia unitaria (K=1), la función de transferencia se simplifica:

$$H(jw) = \frac{V_0}{V_i} = \frac{1}{1 - w^2 R_1 R_2 C_1 C_2 + jwC_2(R_1 + R_2)}$$

Como antes, al igualar la función de transferencia a la forma estándar, queda:

$$H_{0LP} = 1 \qquad w_0 = \frac{1}{\sqrt{R_1 R_2 C_1 C_2}} \qquad Q = \frac{\sqrt{R_1 R_2 C_1 C_2}}{C_2(R_1 + R_2)}$$

El procedimiento que se sigue para el diseño requerido es el siguiente:

- Para simplificar las fórmulas anteriores se hace $R_1 = R_2 = R$, y queda:

$$H_{0LP} = 1 \qquad w_0 = \frac{1}{R\sqrt{C_1 C_2}} \qquad Q = \frac{\sqrt{C_1}}{2\sqrt{C_2}}$$

- Se calcula la relación entre los condensadores para el valor de Q deseado:

$$Q = \frac{\sqrt{C_1}}{2\sqrt{C_2}} \rightarrow \frac{1}{\sqrt{2}} = \frac{\sqrt{C_1}}{2\sqrt{C_2}} \rightarrow C_1 = 2C_2$$

- Se escogen dos condensadores que cumplan la relación anterior:

$$C_2 = 100nF \Rightarrow C_1 = 2C_2 = 200nF$$

- Se calcula el valor de la resistencia para la frecuencia de corte deseada:

$$w_0 = \frac{1}{R\sqrt{C_1 C_2}} \rightarrow R = \frac{1}{w_0\sqrt{C_1 C_2}} = 5.63K\Omega$$

El filtro paso bajo resultante es el siguiente:

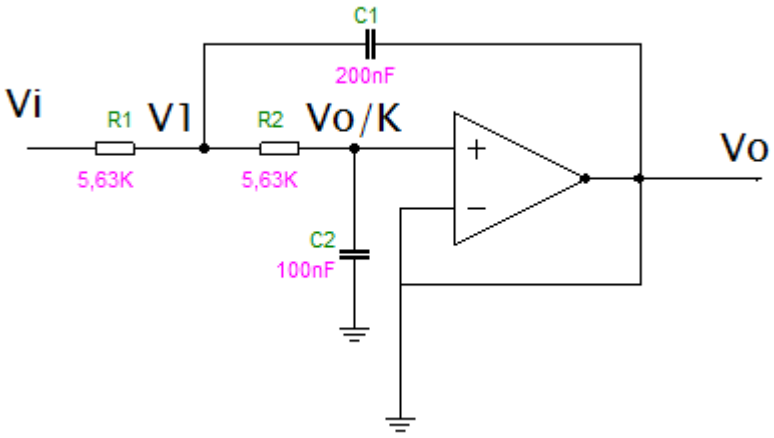

Figura 27. Filtro paso bajo Sallen Key de segundo orden

La respuesta del filtro simulada en PSpice es:

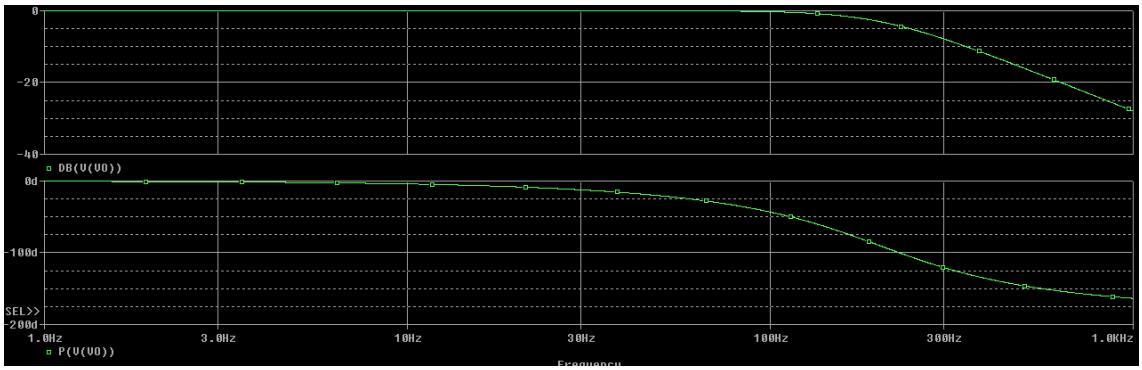

Figura 28. Respuesta simulada del filtro paso bajo

La respuesta en ganancia del filtro medida en el laboratorio es:

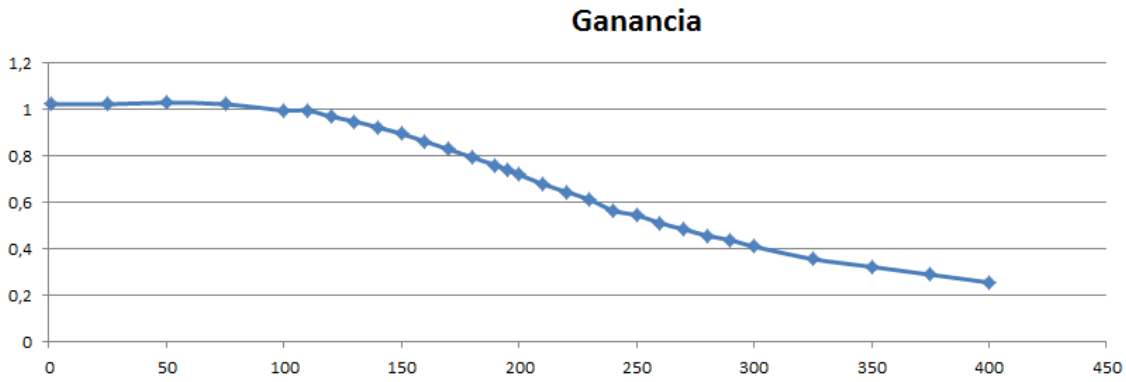

Figura 29. Respuesta real del filtro paso bajo

4.1.6 Comparador

La señal que se obtiene tras el filtro no tiene amplitud suficiente para ser utilizada en el circuito digital. De modo que se usa el comparador para obtener una señal digital cuadrada entre 0 y 5V para conectarla a la entrada de la FPGA.

El comparador usado es un LM111. Este comparador actúa como un operacional en bucle abierto. Cuando la tensión V+ es superior a la V- proporciona +Vcc en la salida. En caso contario proporciona 0V. El fabricante indica que la resistencia de pull-up de 1K es necesaria para su correcto funcionamiento.

La tensión umbral de comparación se fija con un potenciómetro de 10K conectado entra Vcc y masa.

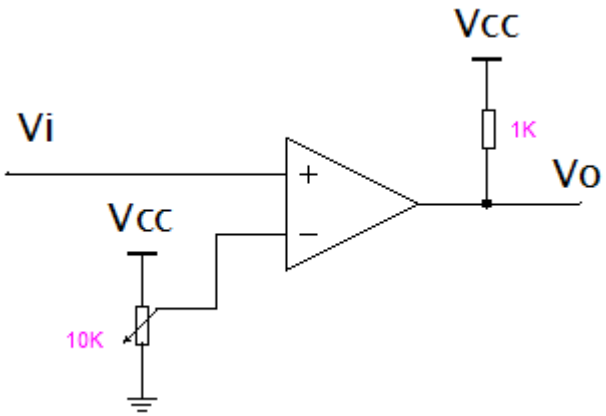

Figura 30. Comparador de tensión

4.2 Subsistema digital

Todo el hardware digital se describe en lenguaje VHDL usando el entorno ISE Design Suite 14.7 de Xilinx, y se sintetiza en la FPGA Spartan 3E de la tarjeta Nexys2 de Digilent.

La estructura de bloques del subsistema digital se muestra con más detalle en la figura.

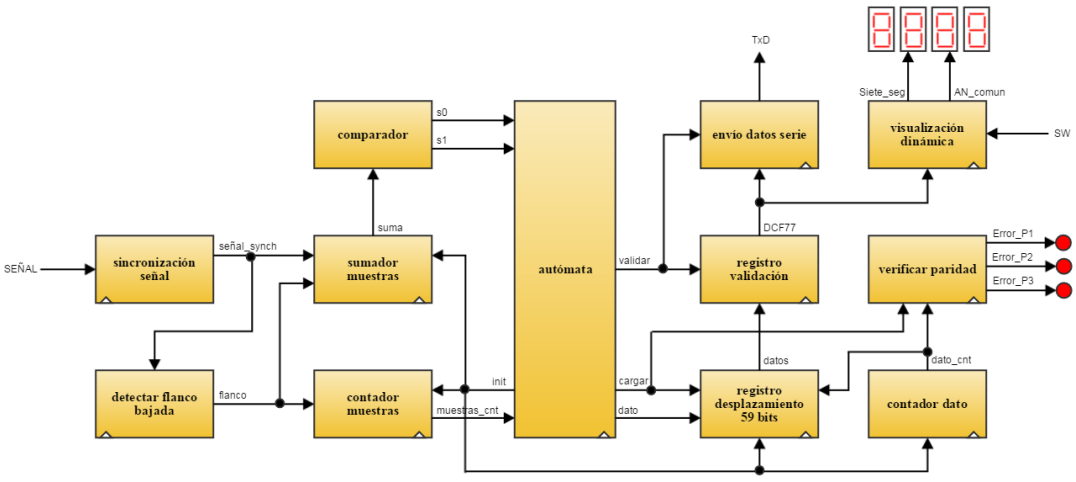

Figura 31. Estructura de bloques del subsistema digital

En el diagrama no se muestran las señales de reloj ni reset para que el flujo de datos quede más claro. Todos los módulos que llevan el triángulo que indica que funcionan por flanco, están conectados a la señal de reloj y a la de reset.

Como puede verse, la señal procedente del subsistema analógico se conecta al módulo "sincronización señal" para obtener una señal sincronizada con el reloj de subsistema digital.

El módulo "detectar flanco bajada" indica la referencia de inicio de cada bit de la trama DCF77.

El "sumador muestras" captura muestras de la señal de entrada sincronizada a una frecuencia de 50 MHz (reloj interno de la FPGA). De esta forma se obtienen 50 millones de muestras en cada tiempo de un bit. A su vez calcula el valor de la suma binaria acumulada de las 50 millones de muestras, y obtiene un número binario de 27 bits (que representa los valores decimales comprendidos entre 0 y 50.000.000).

El "comparador" toma el valor de la suma de las muestras y lo compara con dos umbrales para discernir si la información transmitida en ese bit se corresponde con un SYNC, un "0" o un "1".

Los valores de la suma ideales para los tres casos son:

- 50.000.000 para un SYNC
- 45.000.000 para un "0"
- 40.000.000 para un "1"

No obstante, los valores obtenidos son ligeramente diferentes teniendo en cuenta el ruido y la no idealidad del circuito analógico. Para hacer el sistema más robusto se definen dos umbrales en los valores intermedios a los valores ideales:

- Umbral 2 = 47.500.000
- Umbral 1 = 42.500.000

La condición para la decisión acerca de la información transmitida por ese bit se determina de la siguiente manera:

- Suma > Umbral 2. La información transmitida es un SYNC.
- Umbral 1 < Suma ≤ Umbral 2. La información transmitida es un "0".
- Suma ≤ Umbral 1. La información transmitida es un "1".

El "autómata" espera durante la toma de las 50.000.000 muestras (valor proporcionado por el "contador muestras"), y una vez recibidas decide acerca de la información contenida en el bit.

Si el bit se corresponde con un "0" o un "1", se almacena el valor de la información en el "registro desplazamiento 59 bits" activando su habilitación y poniendo en el valor de captura el dato correspondiente.

Si el bit se corresponde con un SYNC, se valida la trama completa, activando la habilitación del "registro validación", capturando de esta forma la trama completa presente en la salida del "registro desplazamiento 59 bits".

Para que este procedimiento funcione, el autómata deber realizar una sincronización inicial detectando un SYNC de la trama, para luego entrar en un bucle de muestreo de la señal.

La fiabilidad del diseño se ha asegurado ante posibles capturas erróneas de datos. Para ello se ha condicionado que la recepción de un SYNC establezca un punto inicial de sincronización y reseteo de los módulos, desde el cual se empieza la captura de cada trama. De esta forma el sistema nunca pierde el control.

Si se recibe un bit "0" o "1" de forma errónea sólo afecta a la información de ese bit en la posición que ocupa dentro de la trama. Si por el contrario, se recibe un bit SYNC de forma errónea sólo afecta a la información de esa trama, truncándola hasta la recepción de dicho SYNC. En cualquiera de los dos casos, puede verificarse que la recepción de un bit de información errónea sólo afecta a la trama en la que se ha recibido, y no a la siguiente, puesto que al recibir el siguiente SYNC el sistema se reinicia.

El módulo "visualización dinámica" muestra en los displays los valores BCD presentes en la salida del "registro validación". Se visualiza el valor de fecha u hora, en función de la selección del conmutador SW. Los valores se actualizan cada vez que se recibe un SYNC, cada 60 segundos.

El módulo "verificar paridad" indica en tres diodos LED si se ha detectado un error en las tres paridades que proporciona la señal DCF77: minutos, horas y fecha. Para ello requiere de la información del número de dato que se está recibiendo (valor proporcionado por el "contador dato"). Si existe error en la paridad el LED permanecerá encendido durante la visualización de la trama. En caso contrario permanecerá apagado.

Como mejora del proyecto, se ha diseñado el módulo "envío datos serie" que se encarga del envío de los datos a través del puerto serie. Dicho modulo captura los datos de cada trama DCF77 y los transmite por el puerto serie a 9600 bps.

A continuación se describe en detalle la operación de cada uno de los módulos.

4.2.1 Sincronización señal

La señal procedente del subsistema analógico, SEÑAL, no está sincronizada con el reloj del subsistema digital. Es necesario sincronizar dicha señal para evitar problemas de metaestabilidad.

La solución propuesta consiste en conectar en serie dos biestables tipo D síncronos por flanco. A pesar de que esta solución no evita la metaestabilidad, sí la reduce suficientemente.

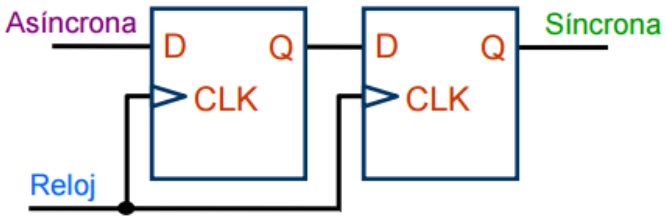

Figura 32. Biestables D para evitar problemas de metaestabilidad

Este bloque genera la señal señal_synch, una señal sincronizada con la señal de entrada, y retrasada un ciclo de reloj.

4.2.2 Detector flanco bajada

Este módulo indica la referencia de inicio de cada bit de información. Detecta el flanco de bajada de la señal señal_synch y genera la señal flanco, un pulso a nivel alto de un ciclo de reloj de duración.

4.2.3 Sumador muestras

Realiza una suma acumulada de 50.000.000 de muestras correspondientes al tiempo de un bit. En cada flanco de reloj suma al valor acumulado el valor presente en la entrada señal_synch. Se trata de un sumador de 1 bit de entrada y 27 bits de salida (representa los valores decimales comprendidos entre 0 y 50.000.000).

Se resetea cada vez que se detecta el inicio de un nuevo bit (flanco = “1”), o cuando se recibe un SYNC, señal que indica el inicio de una nueva trama (init = “1”).

4.2.4 Comparador

Toma el valor suma, y lo compara con los umbrales:

- Umbral 2 = 47.500.000
- Umbral 1 = 42.500.000

Las tres posibilidades son:

- Suma > Umbral 2. La información transmitida es un SYNC.
- Umbral 1 < Suma ≤ Umbral 2. La información transmitida es un “0”. Señal s0 = “1”.
- Suma ≤ Umbral 1. La información transmitida es un “1”. Señal s1 = “1”.

Únicamente proporciona dos salidas, s0 y s1. La tercera sería redundante, debido a que SYNC se da cuando s0 = “0” y s1 = “0”.

4.2.5 Contador muestras

Se incrementa con cada flanco de reloj y se resetea cada vez que se detecta el inicio de un nuevo bit (flanco = “1”), o cuando se recibe un SYNC, señal que indica el inicio de una nueva trama (init = “1”).

La señal de salida muestras_cnt de 27 bits (para contar de 0 a 99.999.998) sirve como referencia al "autómata".

4.2.6 Autómata

El autómata necesita esperar durante 50.000.000 muestras para decidir si el dato contenido en un tiempo de bit es un "0" o un "1".

En caso de que el dato sea un SYNC, el autómata deberá esperar durante 100.000.000 de muestras, puesto que durante la transmisión de un bit SYNC la señal permanece en estado alto durante todo el tiempo de bit. En este caso, la señal flanco sólo proporciona la referencia del bit anterior recibido, por lo que se hace necesario que el "contador muestras" proporcione dicha referencia contando el tiempo de dos bits.

Su grafo de estados es el de la figura 33.

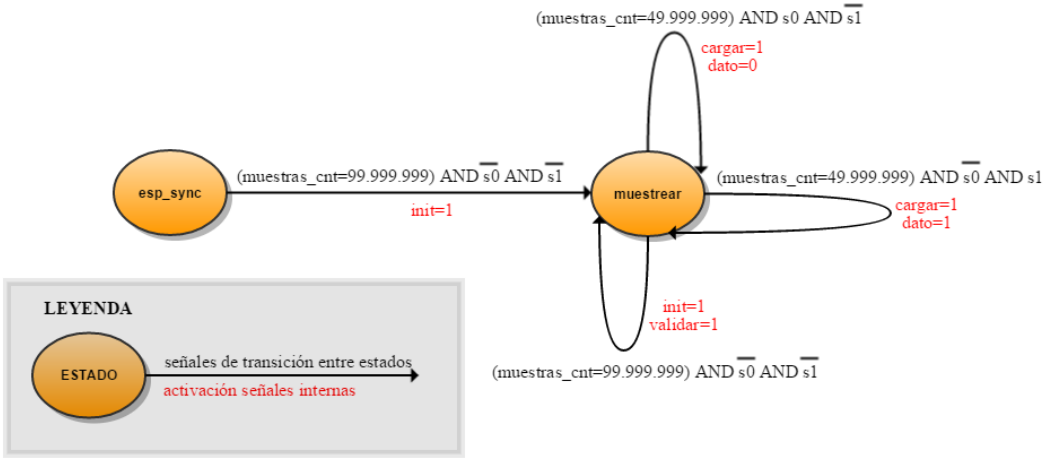

Figura 33. Grafo de estados del subsistema digital

Se ha preferido el uso de un autómata de Mealy porque la estructura resulta mucho más simple que el de Moore, y así el estado puede codificarse con una sola variable.

Permanece en el estado esp_sync hasta que se realiza el sincronismo inicial de la señal. Para detectar el primer SYNC, hay que esperar hasta que la información capturada por el "sumador muestras" se interprete como un SYNC (s0 = "0", s1 = "0" y muestras_cnt = 99.999.999).

En este momento el "autómata" inicializa el "sumador muestras" y el "contador muestras", y entra en bucle en el estado muestrear siguiendo el siguiente proceso:

- Espera durante 49.999.999 muestras.
- Decide la información contenida en el tiempo de bit.
- Si la información contenida es un "0" o un "1" lo almacena en el "registro desplazamiento 59 bits" (cargar = "1" y dato = "0" o "1" en función del dato que se haya recibido).

- Si se trata de un SYNC, deberá un tiempo de un bit adicional (99.999.999 muestras en total) para validar el "registro validación" (validar = "1").
- Vuelve al primer punto.

4.2.7 Registro desplazamiento 59 bits

Se encarga de almacenar los valores binarios que componen la trama. Tiene entrada serie y salida paralelo de 38 bits (señal datos). Las entradas de habilitación y entrada serie son controladas por el "autómata", de modo que su salida (datos) sólo se desplaza cuando la señal cargar = "1".

La trama DCF77 tiene una longitud de 58 bits, más un segundo de sincronización. Los primeros 20 bits, los correspondientes a los indicadores especiales no van a usarse en el proyecto, por lo tanto no se capturan. Por este motivo la longitud de la trama capturada es de 58 – 20 = 38 bits.

Debido a esta selección de los datos a capturar, el "registro de desplazamiento 59 bits" precisa de una referencia del número de dato de la trama DCF77 que se está transmitiendo. El módulo "contador dato" es el encargado de suministrar dicha información a través de la señal dato_cnt. De modo que la habilitación final del "registro de desplazamiento 59 bits" no sólo estará condicionada por la señal cargar = "1" sino que deberá darse al mismo tiempo la condición dato_cnt > 20.

4.2.8 Registro validación

Si el bit de información se corresponde con un SYNC, el "autómata" valida la trama completa, activando la habilitación del "registro validación" (validar = "1"), capturando de esta forma la trama presente en la salida del "registro desplazamiento 59 bits".

4.2.9 Verificar paridad

Indica en sendos diodos led de la placa Nexys2 si se ha detectado un error en alguna de las tres paridades que proporciona la señal DCF77: minutos, horas y fecha.

El "contador dato" es el que proporciona la información acerca del número de dato que se está recibiendo (señal dato_cnt).

El error de paridad se detecta aplicando la operación o-exclusiva (ser diferentes) acumulada a los bits del intervalo al que se refiere la paridad proporcionada por DCF77, y la paridad misma.

El error de paridad de los minutos requiere de la operación o-exclusiva acumulada de los bits 21-27 y la paridad P1 facilitada por DCF77 (bit 28).

En el caso de las horas requiere de la operación o-exclusiva acumulada de los bits 29-34 y la paridad P2 (bit 35).

En el caso de la fecha requiere de la operación o-exclusiva acumulada de los bits 36-57 y la paridad P3 (bit 58).

Una vez finalizada la trama (validar = "1"), un registro de validación captura el valor de los errores de paridad.

Si existe error en alguna de las paridades, el LED permanecerá encendido durante la visualización de la trama.

4.2.10 Visualización dinámica

Para la visualización se usarán los cuatro displays de 7 segmentos disponibles en la tarjeta Nexys2.

Los displays de la Nexys2 son de ánodo común, de modo que la estructura digital activa los segmentos con "0".

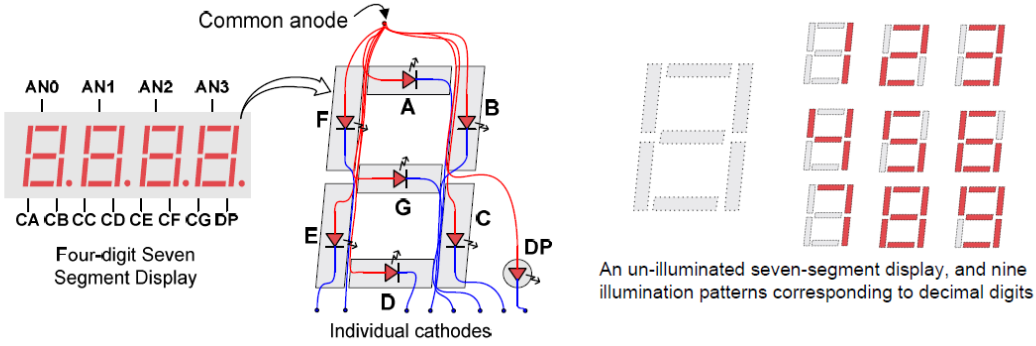

Figura 34. Conexión de los displays de 7 segmentos en la tarjeta Nexys2

Los cuatro displays de 7 segmentos presentes en la Nexys2 comparten las mismas líneas de los segmentos, con activaciones diferentes. Para que el ojo humano no perciba los displays parpadeando es necesario que la frecuencia de refresco de cada display sea mayor de 25 Hz. Como son 4 los displays, la frecuencia deberá ser mayor de 25 * 4 = 100 Hz. Se escoge la frecuencia de 1000 Hz para realizar el diseño.

Toda la información a mostrar no cabe en cuatro displays de 7 segmentos, de modo que se describirá un multiplexor de 2 entradas de 8 bits cada una, que se controlará externamente con un conmutador (SW) de la placa. De este modo se seleccionarán los datos a visualizar:

- SW = "0". Visualización de hora y minutos.
- SW = "1". Visualización de día del mes y mes.

A fin de facilitar la comprensión del bloque "visualización dinámica" se facilita un esquema del mismo.

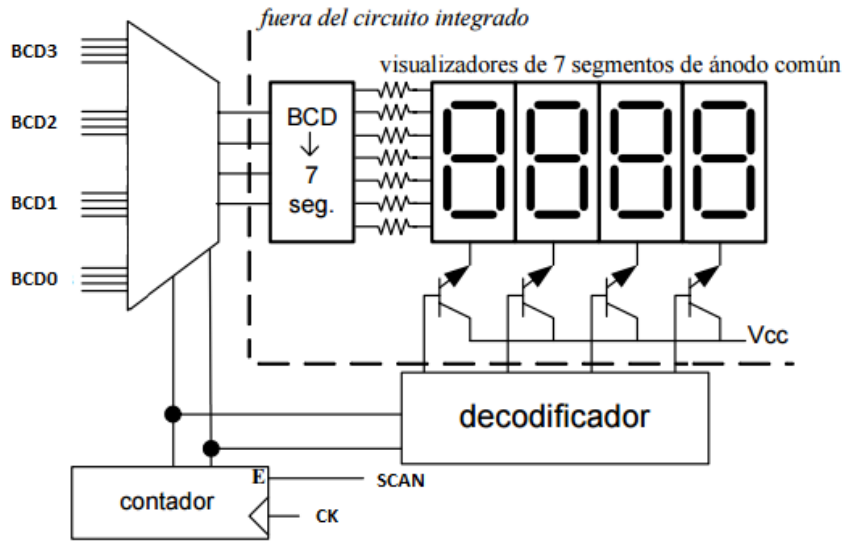

Figura 35. Esquema de bloques de la visualización dinámica

Este módulo está formado a su vez por cuatro bloques:

- Generación de la señal de refresco de los displays.
  Mediante un contador, se genera una señal de salida de 1000 Hz para la activación de los display.
- Barrido de los display.
  Un decodificador se encarga de activar cada uno de los 4 displays.
- Decodificador BCD a 7 segmentos.
  El decodificador transforma los valores BCD a la entrada del multiplexor para adaptarlos a los displays de 7 segmentos.
- Multiplexor de 4 entradas de 4 bits.
  Gracias al multiplexor el decodificador BCD a 7 segmentos se comparte por los 4 valores BCD a visualizar.

4.2.11 Envío datos serie

Como mejora del proyecto, se ha diseñado el módulo "envío datos serie" que se encarga del envío de los datos a través del puerto serie. Dicho modulo captura los datos de cada trama DCF77 y los transmite por el puerto serie a 9600 bps.

El bloque de entrada/salidas es el siguiente:

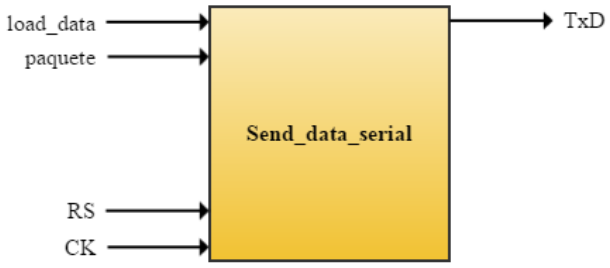

Figura 36. Figura 11. Bloque entradas/salidas del envío de datos serie

Debido a la complejidad y extensión de este módulo se ha diseñado como otro fichero .vhd. En la descripción VHDL del proyecto principal aparece instanciado dentro del mismo, aunque la descripción VHDL se adjunta en el apartado.

Se ha diseñado un módulo para la transmisión de los datos de la trama DCF77 a través del puerto serie de la placa de desarrollo Nexys2. El objetivo es visualizar la en el PC la información horaria DCF77 y que se actualice cada 60 segundos con la llegada de una nueva trama.

La transmisión se realiza por medio de código ASCII a través del puerto RS-232, a una velocidad de 9600 bps. Debido a que la transmisión es asíncrona, hay que ajustarse a un formato para realizar la transmisión de información.

El protocolo usado en la transmisión es el siguiente:

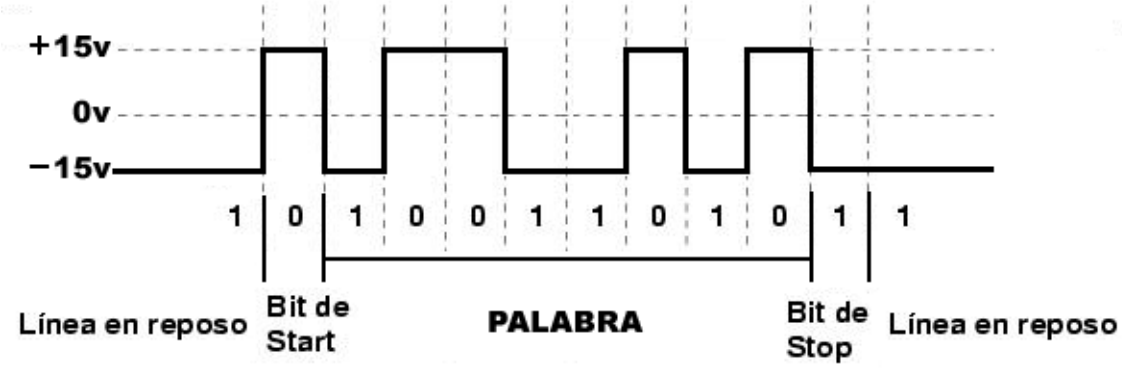

Figura 37. Protocolo usado en la transmisión serie

Un bit de inicio, una palabra de 8 bits y el bit de stop.

Debido a que la transmisión serie usa el estándar ±15V, es necesario usar el conversor de voltaje integrado en la placa Nexys2.

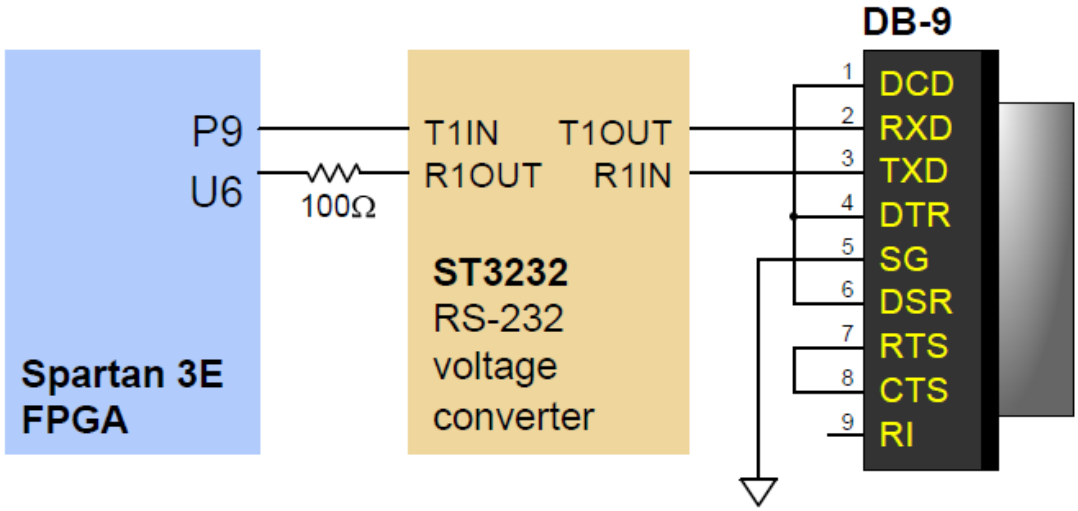

Figura 38. Conversor de voltaje de la Spartan 3E

La estructura de bloques del módulo de transmisión serie es la siguiente:

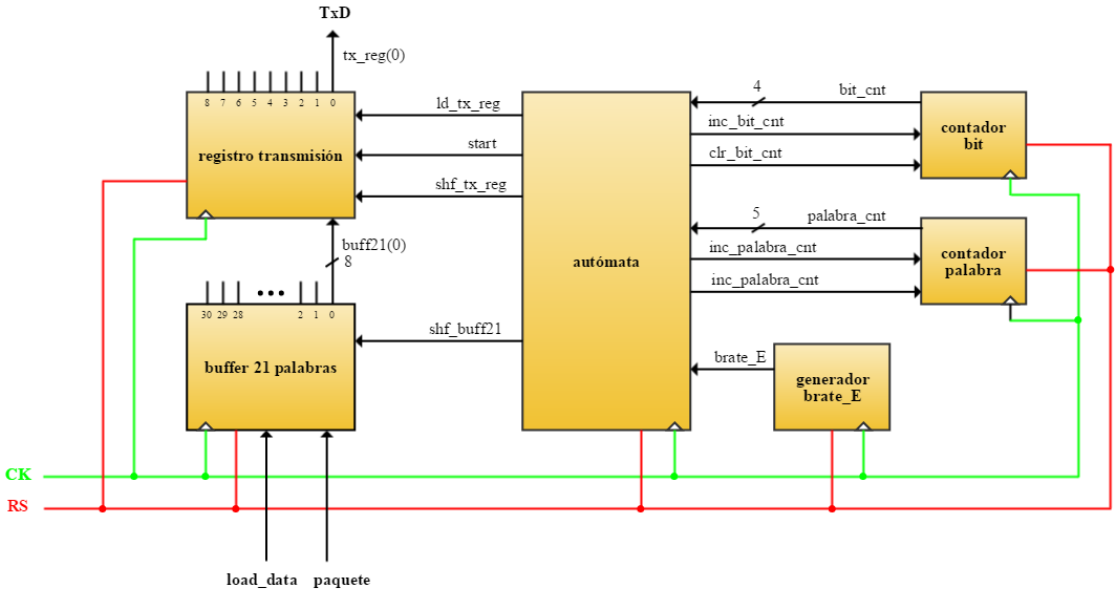

Figura 39. Esquema de bloques de la transmisión de datos serie

En el buffer de 21 palabras, de 8 bits cada una, se cargan los valores de la trama DCF77.

El registro de transmisión es el encargado de realizar el envío de cada paquete de 8 bits. Una vez los ha trasmitido, el autómata carga en el registro de transmisión el siguiente paquete de 8 bits procedente del buffer. La transmisión finaliza cuando se ha transmitido toda la información del buffer de 21 palabras.

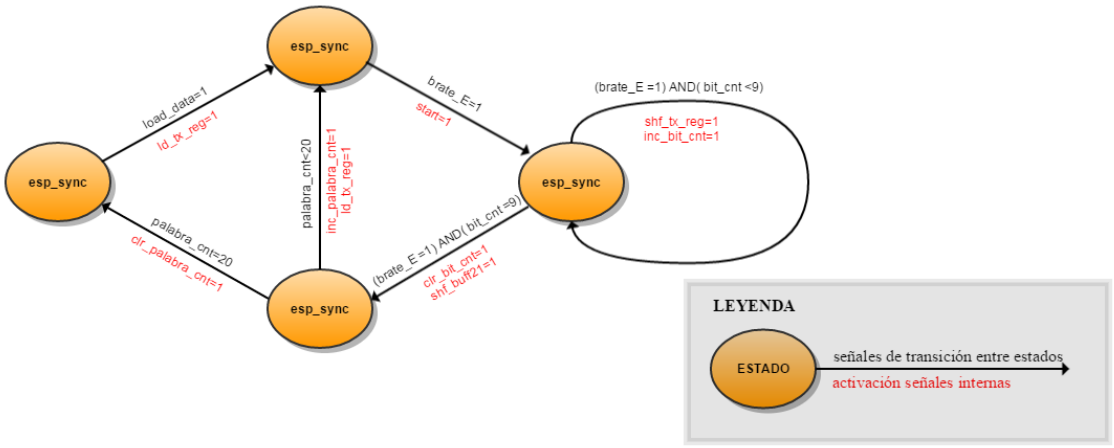

Figura 40. Grafo de estados de la transmisión de datos serie

Cuando se actualice una nueva trama DCF77, el buffer se recargará con los nuevos valores y la transmisión se reiniciará. Los módulos contador bit y contador palabra sirven como referencia al autómata para conocer la cantidad de información transmitida.

Como se ha comentado, la comunicación es asíncrona. Para realizarse de forma exitosa, se ha de cumplir un formato de transmisión, pero también una velocidad. En este caso, 9600 bits por segundo.

Para obtener esa velocidad de transmisión se ha de dividir la frecuencia del reloj de la FPGA (de 50 Mhz). Como los bits a transmitir son 8, pero hay un bit de start y otro de stop, la frecuencia de reloj ha de dividirse por:

$$\frac{50 * 10^6}{10 * 9600} = 520{,}8\hat{3}$$

El número no es exacto. Si se divide por 521, la velocidad de transmisión será de:

$$\frac{50 * 10^6}{10 * 521} = 9596{,}928983$$

Se desvía de la ideal un 0,032%. Es un valor asumible en este tipo de transmisión.

5. Simulaciones.

El diseño digital ha sido íntegramente simulado a través de dos test. Uno para el receptor DCF77 y otro para el envío de datos serie. Mediante la herramienta ModelSIM se ha verificado el correcto funcionamiento. Al final de este documento se proporcionan ambos códigos de simulación.

Debido a las grandes diferencias temporales entre la señal (cada tiempo de bit dura un segundo y cada trama 60 segundos) y la velocidad de toma de muestras del sumador (50 MHz), ha sido necesario escalar el test para la simulación.

Se han dividido todos los tiempos por 1.000.000. De modo que para lanzar las simulaciones, antes deben de ajustarse las variables de ambos ficheros.

6. Conclusiones

Se han cumplido los objetivos principales del proyecto:

- Estudio de la señal DCF77.
- Aprendizaje de conceptos relacionados con la radiofrecuencia y el tratamiento de señal.
- Diseño del receptor de señal DCF77 para sincronización horaria vía radio.

El autor ha adquirido muchos conocimientos no tratados en la carrera, sobre todo los relacionados con la electrónica analógica y el tratamiento de señal. Además se ha perfeccionado el manejo del software de simulación electrónica PSpice y el manejo de funciones avanzadas del osciloscopio.

El principal problema durante el desarrollo del proyecto ha sido la recepción de la señal DCF77 mediante la antena de ferrita. El proyecto se ha llevado a cabo íntegramente en las instalaciones del edificio Ada Byron. Debido al tipo de construcción y materiales empleados en el mismo, conforman una jaula de Faraday, que restringe la calidad de las recepciones de señales de radiofrecuencia en el interior del mismo. En ocasiones, debido a las fluctuaciones en la recepción de la señal, el prototipo no capta suficiente señal para funcionar correctamente.

7. Líneas futuras

Podría repensarse la estructura del diseño para incluir los filtros analógicos dentro del diseño digital. De esta forma se minimizaría el tamaño del receptor, y se mejoraría la posibilidad de ajuste y/o actualización.

Debido a las fluctuaciones en la recepción de la señal, podría incluirse un sistema de ganancia variable controlada digitalmente para asegurar en todo momento una señal estable para funcionamiento de toda la cadena.

8. Referencias bibliográficas

- “Sistemas de comunicaciones electrónicas”, Tomasi (2003).
- “Diseño con amplificadores operacionales y circuitos integrados analógicos”, Sergio Franco (2005).
- “Op. Amps for everyone”, Ron Mancini (2003)
- “VHDL Lenguaje para síntesis y modelado de circuitos”, Fernando Pardo, José A. Boluda (2004).
- “Electrónica digital”, Tomás Pollán Santamaría (2004).
- “Principios de electrónica”, Albert Paul Malvino (2000).

9. Anexos

9.1 Código VHDL del receptor DCF77

```
--------------------------------------------------------------------------------
-- Filename                        : Receptor_DCF77.vhd
-- Author                          : Jaime Vinues
-- Created On                 : 19:25:24 06/10/2015
-- Last Modified              : 17:16:34 07/20/2015
-- Description                : Recibe la señal de datos DCF77 y muestra la hora/fecha
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Receptor_DCF77 is

        port (CK: in std_logic;
                        RS: in std_logic;
                        SEÑAL: in std_logic;
                        SW: in std_logic;
                        Siete_seg: out std_logic_vector(6 downto 0);
                        AN_comun: out std_logic_vector(3 downto 0);
                        Error_P1: out std_logic;
                        Error_P2: out std_logic;
                        Error_P3: out std_logic;
                        TxD: out std_logic);

end Receptor_DCF77;

        architecture Decodificar of Receptor_DCF77 is

        -- Sincronización de la señal
        signal señal_int: std_logic;
        signal señal_synch: std_logic;

        -- Detectar flanco bajada
        signal señal_synch_int: std_logic;
        signal flanco: std_logic;

        -- Sumador muestras
        signal suma: std_logic_vector(26 downto 0);
        signal init: std_logic;

        -- Contador de muestras
        signal muestras_cnt: std_logic_vector(26 downto 0);

        -- Comparador
        signal UM1: std_logic_vector(25 downto 0); -- umbral_inferior
        signal UM2: std_logic_vector(25 downto 0); -- umbral_superior
        signal s1: std_logic; -- suma=<umbral_1
   signal s0: std_logic; -- umbral_1<suma=<umbral_2

        -- Registro de desplazamiento 59 bits
        signal datos: std_logic_vector(58 downto 21);
        signal dato: std_logic;
   signal cargar: std_logic;

        -- Registro de validación
        signal DCF77: std_logic_vector(58 downto 21);
        signal validar: std_logic;

        -- Visualización dinámica
        signal Dec_min: std_logic_vector (3 downto 0);
        signal Uni_min: std_logic_vector (3 downto 0);
        signal Dec_hora: std_logic_vector (3 downto 0);
        signal Uni_hora: std_logic_vector (3 downto 0);
        signal Dec_dia: std_logic_vector (3 downto 0);
        signal Uni_dia: std_logic_vector (3 downto 0);
        signal Dec_mes: std_logic_vector (3 downto 0);
        signal Uni_mes: std_logic_vector (3 downto 0);
        signal gen_scan : std_logic_vector (15 downto 0);
```

```
        signal scan : std_logic;
        signal cont_scan : std_logic_vector (1 downto 0);
        signal BCD0: std_logic_vector (3 downto 0);
        signal BCD1: std_logic_vector (3 downto 0);
        signal BCD2: std_logic_vector (3 downto 0);
        signal BCD3: std_logic_vector (3 downto 0);
        signal BCD : std_logic_vector (3 downto 0);

        -- Contador dato
        signal dato_cnt: std_logic_vector (5 downto 0);

        -- Verificar paridad
        signal p1: std_logic;
        signal p2: std_logic;
        signal p3: std_logic;

        -- Autómata
        type estados is (esp_sync, muestrear);
        signal estado, estado_sig : estados;


begin


        -- Envío de datos serie

U1: entity work.Send_data_serial(uart_transmitter) -- inst. UART transmitter
port map (CK, RS, validar, DCF77, TxD);


        -- Sincronización de la señal

process (CK, RS)
begin
        if (RS='1') then
                señal_int<='0';
                señal_synch<='0';
        elsif (CK'event and CK='1') then
                señal_int<=SEÑAL;
                señal_synch<=señal_int;
        end if;
end process;


        -- Detectar flanco de bajada señal sincronizada

flanco<='1' when ((señal_synch='0') and (señal_synch_int='1')) else '0';

process (CK, RS, señal_synch)
begin
        if (RS='1') then
                señal_synch_int<='0';
        elsif (CK'event and CK='1') then
                señal_synch_int<=señal_synch;
        end if;
end process;


        -- Sumador binario

process (RS, CK, flanco, init)
begin
        if (RS='1') then
                suma<=(others=>'0');
        elsif (CK'event and CK='1') then
                        if ((flanco='1') or (init='1')) then
                                suma<=("00000000000000000000000000000" & señal_synch);
                        else
                                suma<=(suma + señal_synch);
                        end if;
        end if;
end process;


        -- Contador de muestras

process (RS, CK, flanco, init)
```

```
begin
        if (RS='1') then
                muestras_cnt<=(others=>'0');
        elsif (CK'event and CK='1') then
                if ((flanco='1') or (init='1')) then
                        muestras_cnt<=(others=>'0');
                else
                        muestras_cnt<=(muestras_cnt + 1);
                end if;
        end if;
end process;


        -- Comparador

UM1<="10100010000111111110100000"; -- umbral inferior = 42.500.000
UM2<="10110101001100101011100000"; -- umbral superior = 47.500.000
s1<='1' when ((suma<UM1) or (suma=UM1)) else '0';
s0<='1' when ((suma>UM1) and ((suma<UM2) or (suma=UM2))) else '0';


        -- Registro de desplazamiento 59 bits

process (CK, RS, init, dato_cnt, cargar)
begin
        if (RS='1') then
                datos<=(others=>'0');
        elsif (CK'event and CK='1') then
                if (init='1') then
                        datos<=(others=>'0');
                elsif ((dato_cnt>20) and (cargar='1')) then
                        datos<=(dato & datos(58 downto 22));
                end if;
        end if;
end process;


        -- Registro de validación

process (CK, RS, validar)
begin
        if (RS='1') then
                DCF77<=(others=>'0');
        elsif (CK'event and CK='1') then
                if (validar='1') then
                        DCF77<=datos(58 downto 21);
                end if;
        end if;
end process;


        -- Visualización dinámica

REFRESCO: process (RS, CK)
begin
        if (RS='1') then
                gen_scan<=(others =>'0');
                scan<='0';
        elsif (CK'event and CK='1') then
                if (gen_scan=49999) then
                        gen_scan<=(others=>'0');
                        scan<='1'; -- refresco displays 1000Hz
                else
                        gen_scan<=(gen_scan + 1);
                        scan<='0';
                end if;
        end if;
end process;

BARRIDO: process (RS, CK, scan)
begin
        if (RS='1') then
                cont_scan<=(others =>'0');
        elsif (CK'event and CK='1') then
                if (scan='1') then
                        cont_scan<=(cont_scan + 1);
                end if;
```

```
	end if;
end process;

Dec_min<=('0' & DCF77(27 downto 25)); -- Ajuste a 4 bits para visualización dinámica
Uni_min<=DCF77(24 downto 21);
Uni_hora<=DCF77(32 downto 29);
Uni_dia<=DCF77(39 downto 36);
Uni_mes<=DCF77(48 downto 45);

APAGACERO: process (DCF77, Dec_hora, Dec_dia, Dec_mes) -- Apaga el primero dígito del
display si es un cero
begin
	if (DCF77(34 downto 33)="00") then
		Dec_hora<="1111"; -- Asigno un valor no válido en el conversor para que
no se muestre
	else
		Dec_hora<=("00" & DCF77(34 downto 33));
	end if;
	if (DCF77(41 downto 40)="00") then
		Dec_dia<="1111";
	else
		Dec_dia<=("00" & DCF77(41 downto 40));
	end if;
	if (DCF77(49)='0') then
		Dec_mes<="1111";
	else
		Dec_mes<=("000" & DCF77(49));
	end if;
end process;

SWITCH_FECHA_HORA: process (SW, Uni_min, Dec_min, Uni_hora, Dec_hora, Uni_mes, Dec_mes,
Uni_dia, Dec_dia)
begin
	case SW is
		when '0'	=>	BCD0<=Uni_min;
					BCD1<=Dec_min;
					BCD2<=Uni_hora;
					BCD3<=Dec_hora;
		when others => BCD0<=Uni_mes;
					BCD1<=Dec_mes;
					BCD2<=Uni_dia;
					BCD3<=Dec_dia;
	end case;
end process;

MULTIPLEXADO_DINAMICO: process (cont_scan, BCD0, BCD1, BCD2, BCD3)
begin
	case cont_scan is
		when "00"	=> BCD<=BCD0;  AN_comun<="0001";
		when "01"	=> BCD<=BCD1;  AN_comun<="0010";
		when "10"	=> BCD<=BCD2;  AN_comun<="0100";
		when others => BCD<=BCD3;	AN_comun<="1000";
	end case;
end process;

CONVERSOR_BCD_SIETE_SEG: process (BCD) -- Ánodo común: activo el 0
begin
	case BCD is
		when "0000" => Siete_seg<="0000001";
		when "0001" => Siete_seg<="1001111";
		when "0010" => Siete_seg<="0010010";
		when "0011" => Siete_seg<="0000110";
		when "0100" => Siete_seg<="1001100";
		when "0101" => Siete_seg<="0100100";
		when "0110" => Siete_seg<="0100000";
		when "0111" => Siete_seg<="0001111";
		when "1000" => Siete_seg<="0000000";
		when "1001" => Siete_seg<="0000100";
		when others => Siete_seg<="1111111"; -- Display apagado si dato no válido
	end case;
end process;


	-- Contador dato

process (RS, CK, cargar)
begin
```

```
        if (RS='1') then
                dato_cnt<=(others=>'0');
        elsif (CK'event and CK='1') then
                if (init='1') then
                        dato_cnt<=(others=>'0');
                elsif (cargar='1') then
                        dato_cnt<=(dato_cnt + 1);
                end if;
        end if;
end process;


        -- Verificar paridad

PARIDAD_MINUTOS: process (RS, CK, dato_cnt, cargar)
begin
        if (RS='1') then
                p1<='0';
        elsif (CK'event and CK='1') then
                        if (validar='1') then
                                p1<='0';
                        elsif ((dato_cnt>20) and (dato_cnt<29) and (cargar='1')) then
                                p1<=(p1 xor dato);
                        end if;
        end if;
end process;

PARIDAD_HORAS: process (RS, CK, dato_cnt)
begin
        if (RS='1') then
                p2<='0';
        elsif (CK'event and CK='1') then
                        if (validar='1') then
                                p2<='0';
                        elsif ((dato_cnt>28) and (dato_cnt<36) and (cargar='1')) then
                                p2<=(p2 xor dato);
                        end if;
        end if;
end process;

PARIDAD_FECHA: process (RS, CK, dato_cnt)
begin
        if (RS='1') then
                p3<='0';
        elsif (CK'event and CK='1') then
                        if (validar='1') then
                                p3<='0';
                        elsif ((dato_cnt>35) and (cargar='1')) then
                                p3<=(p3 xor dato);
                        end if;
        end if;
end process;

REGISTRO_VALIDAR: process (CK, RS, validar)
begin
        if (RS='1') then
                Error_P1<='1';
                Error_P2<='1';
                Error_P3<='1';
        elsif (CK'event and CK='1') then
                if (validar='1') then
                        Error_P1<=p1;
                        Error_P2<=p2;
                        Error_P3<=p3;
                end if;
        end if;
end process;


        -- Autómata

process (estado, s0, s1, muestras_cnt)
begin
        dato<='0';
        cargar<='0';
        validar<='0';
        init<='0';
```

```
	case estado is
		when esp_sync  => if ((muestras_cnt=99999999) and (s0='0') and (s1='0'))
then
							estado_sig<=muestrear;
							init<='1';
					else
							estado_sig<=esp_sync;
					end if;
		when muestrear =>	if ((muestras_cnt=49999999) and (s0='1') and
(s1='0')) then
							estado_sig<=muestrear;
							cargar<='1';
					elsif
((muestras_cnt=49999999) and (s0='0') and (s1='1')) then
							estado_sig<=muestrear;
							cargar<='1';
							dato<='1';
					elsif
((muestras_cnt=99999999) and (s0='0') and (s1='0')) then
							estado_sig<=muestrear;
							validar<='1';
							init<='1';
					else
							estado_sig<=muestrear;
					end if;
	end case;
end process;

process (RS, CK)
begin
	if (RS='1') then
		estado<=esp_sync;
	elsif (CK'event and CK='1') then
		estado<=estado_sig;
	end if;
end process;


end Decodificar;
```

9.2 Código VHDL del envío de datos serie

```
--------------------------------------------------------------------------------
-- Filename                        : Send_data_serial.vhd
-- Author                          : Jaime Vinues
-- Created On                 : 10:19 07/16/2015
-- Last Modified              : 23:35 08/26/2015
-- Description                : Envía paquetes de datos a través del puerto serie
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Send_data_serial is
	port	(CK: in std_logic;
			RS: in std_logic;
			load_data: in std_logic;
			paquete: in std_logic_vector(58 downto 21);
			TxD: out std_logic);
end Send_data_serial;

architecture UART_transmitter of Send_data_serial is

	-- Buffer de 21 palabras
	subtype slv8 is std_logic_vector(7 downto 0);
	type tx_buffer is array(20 downto 0) of slv8;
	signal buff21: tx_buffer;
	signal shf_buff21: std_logic;

	-- Generador de brate_E (velocidad de transmisión)
	signal brate_cont : std_logic_vector (9 downto 0);
	signal brate_E: std_logic;

	-- Registro de transmisión
	signal tx_reg : std_logic_vector (8 downto 0);
	signal ld_tx_reg: std_logic;
	signal start: std_logic;
	signal shf_tx_reg: std_logic;

	-- Contador bit
	signal bit_cnt: std_logic_vector(3 downto 0);
	signal clr_bit_cnt: std_logic;
	signal inc_bit_cnt: std_logic;

	-- Contador palabra
	signal palabra_cnt: std_logic_vector(4 downto 0);
	signal clr_palabra_cnt: std_logic;
	signal inc_palabra_cnt: std_logic;

	-- Autómata
	type mis_estados is (reposo, sync_brate, transmit_palabra,
cargar_siguiente_palabra);
	signal estado_tx, estado_tx_sig : mis_estados;


begin


	-- buffer de 21 palabras

process (RS, CK, load_data, paquete)
begin
	if (RS='1') then
		buff21<=(others=>(others=>'0'));
	elsif (CK'event and CK='1') then
		if (load_data='1') then
			buff21(0)<=x"3"&"00"&paquete(34 downto 33); -- horas decenas
			buff21(1)<=x"3"&paquete(32 downto 29);              -- horas
unidades
			buff21(2)<=x"3A";
		-- :
			buff21(3)<=x"3"&'0'&paquete(27 downto 25);  -- minutos decenas
```

```
                        buff21(4)<=x"3"&paquete(24 downto 21);              -- minutos
unidades
                        buff21(5)<=x"20";
                        --
                        buff21(6)<=x"3"&'0'&paquete(44 downto 42);  -- día de la semana
                        buff21(7)<=x"20";
                        --
                        buff21(8)<=x"3"&"00"&paquete(41 downto 40); -- día mes decenas
                        buff21(9)<=x"3"&paquete(39 downto 36);                      -- día
mes unidades
                        buff21(10)<=x"2F";
                -- /
                        buff21(11)<=x"3"&"000"&paquete(49);                         -- mes
decenas
                        buff21(12)<=x"3"&paquete(48 downto 45);             -- mes
unidades
                        buff21(13)<=x"2F";
                -- /
                        buff21(14)<=x"3"&paquete(57 downto 54);             -- año decenas
                        buff21(15)<=x"3"&paquete(53 downto 50);             -- año
unidades
                        buff21(16)<=x"20";
                --
                        buff21(17)<=x"3"&"000"&paquete(35);                         --
paridad horas
                        buff21(18)<=x"3"&"000"&paquete(28);                         --
paridad minutos
                        buff21(19)<=x"3"&"000"&paquete(58);                         --
paridad fecha
                        buff21(20)<=x"0D";
                -- carriage return
                end if;
                if (shf_buff21='1') then
                        buff21<="11111111" & buff21(20 downto 1);
                end if;
        end if;
end process;


        -- registro de transmisión

TxD<=tx_reg(0);

process (RS, CK, ld_tx_reg, start, shf_tx_reg)
begin
        if (RS='1') then
                tx_reg<="111111111";
        elsif (CK'event and CK='1') then
                if (ld_tx_reg='1') then
                        tx_reg<=(buff21(0) & '1');
                end if;
                if (start='1') then
                        tx_reg(0)<='0';
                end if;
                if (shf_tx_reg='1') then
                        tx_reg<=('1' & tx_reg(8 downto 1));
                end if;
        end if;
end process;


        -- Generador de brate_E (velocidad de transmisión)

process (RS, CK)
begin
        if (RS='1') then
                brate_cont<=(others=>'0');
                brate_E<='0';
        elsif (CK'event and CK='1') then
                if (brate_cont=520) then
                        brate_cont<=(others=>'0');
                        brate_E<='1'; -- 9600 bps
                else
                        brate_cont<=(brate_cont + 1);
                        brate_E<='0';
                end if;
        end if;
```

```
end process;


        -- contar bit transmitido

process (RS, CK)
begin
        if (RS='1') then
                bit_cnt<=(others=>'0');
        elsif (CK'event and CK='1') then
                if (clr_bit_cnt='1') then
                        bit_cnt<=(others=>'0');
                elsif (inc_bit_cnt='1') then
                        bit_cnt<=(bit_cnt + 1);
                end if;
        end if;
end process;


        -- contar palabra transmitida

process (RS, CK)
begin
        if (RS='1') then
                palabra_cnt<=(others=>'0');
        elsif (CK'event and CK='1') then
                if (clr_palabra_cnt='1') then
                        palabra_cnt<=(others=>'0');
                elsif (inc_palabra_cnt='1') then
                        palabra_cnt<=(palabra_cnt + 1);
                end if;
        end if;
end process;


        -- autómata

process(estado_tx, load_data, brate_E, bit_cnt, palabra_cnt)
begin
        clr_bit_cnt<='0';
        inc_bit_cnt<='0';
        clr_palabra_cnt<='0';
        inc_palabra_cnt<='0';
        ld_tx_reg<='0';
        start<='0';
        shf_tx_reg<='0';
        shf_buff21<='0';

        case estado_tx is
                when reposo                                     => if (load_data='1')
then

                estado_tx_sig<=sync_brate;

                ld_tx_reg<='1';

        else

                estado_tx_sig<=reposo;

        end if;
                when sync_brate                                 => if (brate_E='1')
then

                estado_tx_sig<=transmit_palabra;

                start<='1';


        else

                estado_tx_sig<=sync_brate;

        end if;
                when transmit_palabra               => if (brate_E='1') then

                if (bit_cnt=9) then
```

```
                        estado_tx_sig<=cargar_siguiente_palabra;
                        clr_bit_cnt<='1';
                        shf_buff21<='1';
                else
                        estado_tx_sig<=transmit_palabra;
                        shf_tx_reg<='1';
                        inc_bit_cnt<='1';
                end if;
        else
                estado_tx_sig<=transmit_palabra;
        end if;
                when cargar_siguiente_palabra =>     if (palabra_cnt=20) then
                estado_tx_sig<=reposo;
                clr_palabra_cnt<='1';
        else
                estado_tx_sig<=sync_brate;
                ld_tx_reg<='1';
                inc_palabra_cnt<='1';

        end if;
                end case;
end process;

process (RS, CK)
begin
        if (RS='1') then
                estado_tx<=reposo;
        elsif (CK'event and CK='1') then
                estado_tx<=estado_tx_sig;
        end if;
end process;

end UART_transmitter;
```

9.3 Fichero de test para el receptor DCF77

```
--------------------------------------------------------------------------------
-- Filename                        : TEST Receptor_DCF77
-- Author                          : Jaime Vinues
-- Created On                 : 19:25:24 06/30/2015
-- Last Modified              : 17:16:34 07/20/2015
-- Description                :
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY TEST_Receptor_DCF77 IS
END TEST_Receptor_DCF77;

ARCHITECTURE test OF TEST_Receptor_DCF77 IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Receptor_DCF77
    PORT(
         CK : IN  std_logic;
         RS : IN  std_logic;
         SEÑAL : IN  std_logic;
         SW : IN  std_logic;
         Siete_seg : OUT  std_logic_vector(6 downto 0);
         AN_comun : OUT  std_logic_vector(3 downto 0);
         Error_P1 : OUT  std_logic;
         Error_P2 : OUT  std_logic;
         Error_P3 : OUT  std_logic;
                    TxD : OUT std_logic
        );
    END COMPONENT;


   --Inputs
   signal CK : std_logic := '0';
   signal RS : std_logic := '0';
   signal SEÑAL : std_logic := '0';
   signal SW : std_logic := '0';

       --Outputs
   signal Siete_seg : std_logic_vector(6 downto 0);
   signal AN_comun : std_logic_vector(3 downto 0);
   signal Error_P1 : std_logic;
   signal Error_P2 : std_logic;
   signal Error_P3 : std_logic;
       signal TxD : std_logic;

       signal final : boolean;


BEGIN

       -- Instantiate the Unit Under Test (UUT)
   uut: Receptor_DCF77 PORT MAP (
          CK => CK,
          RS => RS,
          SEÑAL => SEÑAL,
          SW => SW,
          Siete_seg => Siete_seg,
          AN_comun => AN_comun,
          Error_P1 => Error_P1,
          Error_P2 => Error_P2,
          Error_P3 => Error_P3,
                      TxD => TxD
        );

   -- Clock process definitions
   RELOJ: process
```

```
   begin
               if final = not true then
               CK <= '1', '0' after 10 ns;
               wait for 20 ns;
               else CK <= '0';
               end if;
   end process;


   -- Stimulus process
  ESTIMULOS: process
   begin
               final <= false;

               SEÑAL<='1'; RS<='1'; SW<='0'; wait for 100 ns; RS<='0'; wait for 200 ns;
-- reset y señal venía con 1
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 DE 40
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 DE 45
               SEÑAL<='1'; wait for 1000 ns; -- SYNC DE 50

               -- Las 18:23 13 de Junio; Paridades OK
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 0 --
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 10 --
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 20 --
Hasta aqui bits reservados
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Uni min
(LSB first)
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Dec min
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Paridad
min
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Uni
hora
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0    -- 30 -
-
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Dec
hora
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Paridad
hora
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Dia mes
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- 40 --
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Dia
semana
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
               SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Mes
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
               SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
```

```
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 50 --
Año
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1     --
Paridad fecha
                SEÑAL<='1'; wait for 1000 ns; -- SYNC

                -- Las 18:24 13 de Junio; Paridades BAD
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 0 --
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 10 --
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 20 --
Hasta aqui bits reservados
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Uni min
(LSB first)
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Dec min
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Paridad
min
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Uni
hora
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0     -- 30 -
-
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Dec
hora
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Paridad
hora
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Dia mes
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- 40 --
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Dia
semana
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Mes
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 50 --
Año
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
```

```
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0    --
Paridad fecha
                SEÑAL<='1'; wait for 1000 ns; -- SYNC
                -- Las 00:01 31 de Diciembre; Paridades OK
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 0 --
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 10 --
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 20 --
Hasta aqui bits reservados
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Uni min
(LSB first)
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Dec min
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Paridad
min
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Uni
hora
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0    -- 30 -
-
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Dec
hora
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Paridad
hora
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- Dia mes
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1 -- 40 --
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Dia
semana
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- Mes
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0 -- 50 --
Año
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
                SEÑAL<='0'; wait for 100 ns; SEÑAL<='1'; wait for 900 ns; -- 0
```

```
                SEÑAL<='0'; wait for 200 ns; SEÑAL<='1'; wait for 800 ns; -- 1     --
Paridad fecha
                SEÑAL<='1'; wait for 1000 ns; -- SYNC

                final <= true;
        wait;
    end process;

END;
```

9.4 Fichero de test para el envío de datos serie

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   23:25:57 07/23/2015
-- Design Name:
-- Module Name:
C:/Users/user/Documents/PROYECTO/Receptor_DCF77/TEST_Send_data_serial.vhd
-- Project Name:  Receptor_DCF77
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: Send_data_serial
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY TEST_Send_data_serial IS
END TEST_Send_data_serial;

ARCHITECTURE behavior OF TEST_Send_data_serial IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Send_data_serial
    PORT(
         CK : IN  std_logic;
         RS : IN  std_logic;
         load_data : IN  std_logic;
         paquete : IN  std_logic_vector(58 downto 21);
         TxD : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal CK : std_logic := '0';
   signal RS : std_logic := '0';
   signal load_data : std_logic := '0';
   signal paquete : std_logic_vector(58 downto 21) := (others => '0');

       --Outputs
   signal TxD : std_logic;

       signal final: boolean;


BEGIN

       -- Instantiate the Unit Under Test (UUT)
   uut: Send_data_serial PORT MAP (
          CK => CK,
          RS => RS,
          load_data => load_data,
```

```
          paquete => paquete,
          TxD => TxD
        );

   -- Clock process definitions
   RELOJ: process
   begin
               if final = not true then
               CK <= '1', '0' after 10 ns;
               wait for 20 ns;
               else CK <= '0';
               end if;
   end process;


   -- Stimulus process
  ESTIMULOS: process
   begin
               final <= false;

      RS<='1';-- Condiciones iniciales
      wait for 100 ns;
               RS<='0';
               wait for 100 ns;
               paquete <= "11000101000110011001000001100000000001"; -- Las 18:23 13 de
Junio; Paridades OK
               load_data<='1';
               wait for 20 ns;
               load_data<='0';
               wait for 1000000 ns;

      final <= true;
      wait;
   end process;

END;
```