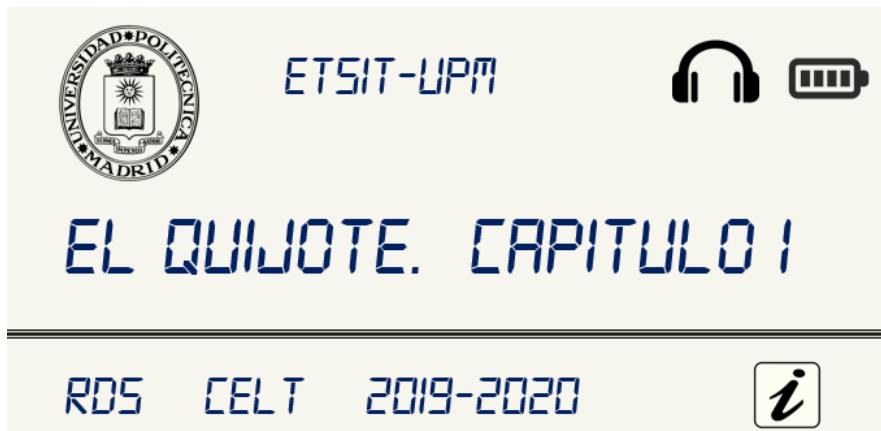


CELT



DECODIFICADOR DE LA SEÑAL RDS

JAVIER LÓPEZ INIESTA DÍAZ DEL CAMPO
ANDRÉS CASTILLEJO AMESCUA

JT02

MEMORIA PROYECTO:
DECODIFICADOR DE LA SEÑAL RDS



CIRCUITOS ELECTRÓNICOS – CELT
GITST

ÍNDICE:

1. ¿QUÉ ES LA SEÑAL RDS?.....	4
2. MODULACIÓN ASK	6
3. CIRCUITO ANALÓGICO.....	8
3.1 ACONDICIONADOR DE LA SEÑAL	10
3.1.1 IMPEDANCIA DE ENTRADA:.....	10
3.1.2 FILTRO PASO ALTO:	10
3.1.3 AMPLIFICADOR DE SEÑAL:	11
3.1.4 ESQUEMA DEL CIRCUITO.....	12
3.1.5. SEÑAL DE SALIDA EN EL OSCILOSCOPIO.....	13
3.2 FILTRO PASO BAJO	14
3.2.1 DISEÑO DEL FILTRO PASO BAJO	14
3.2.2 ESQUEMA DEL FILTRO PASO BAJO	18
3.2.3 MEDIDAS DEL FILTRO PASO BAJO	19
3.2.4 DIAGRAMAS DE BODE DEL FILTRO PASO BAJO	20
3.2.5 SEÑAL DE SALIDA EN EL OSCILOSCOPIO.....	21
3.2.6 ESPECTRO A LA SALIDA DEL FILTRO	21
3.3 AMPLIFICADOR DE POTENCIA.....	22
3.3.1 ESQUEMA DEL AMPLIFICADOR DE POTENCIA	22
3.3.2 SEÑAL DE SALIDA DEL OSCILOSCOPIO.....	23
3.4 FILTRO PASO BANDA.....	24
3.4.1 DISEÑO DEL FILTRO PASO BANDA.....	24
3.4.2 ESQUEMA DEL FILTRO PASO BANDA.....	26
3.4.3 MEDIDAS FILTRO PASO BANDA.....	27
3.4.4 SEÑAL DE SALIDA EN EL OSCILOSCOPIO.....	28
3.4.5 DIAGRAMAS DE BODE DEL FILTRO PASO BANDA	29
3.4.6 ESPECTRO A LA SALIDA DEL FILTRO	30
3.5 DETECTOR DE ENVOLVENTE.....	31
3.5.1 RECTIFICADOR DE ONDA.....	31
3.5.2 FILTRO PASO BAJO	31
3.5.3 SEÑAL DE SALIDA EN EL OSCILOSCOPIO.....	32

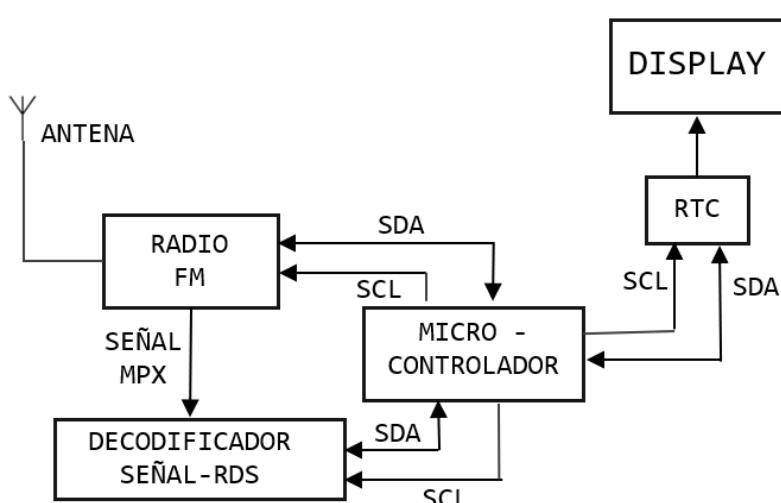
3.6 COMPARADOR	33
3.6.1 ESQUEMA DEL COMPARADOR.....	33
3.6.2 SEÑAL DE SALIDA EN EL OSCILOSCOPIO.....	34
3.7 ESQUEMA ELÉCTRICO DEL CIRCUITO ANALÓGICO COMPLETO	35
3.8 DIFICULTADES PARTE ANALÓGICA.....	36
4. PARTE DIGITAL.....	37
4.1 DIVISOR DE RELOJ	39
4.2 DETECTOR DE BITS	41
4.3 REGISTRO DE DESPLAZAMIENTO DE 9 BITS	45
4.4 AUTÓMATA DE CAPTURA.....	48
4.5 AUTÓMATA DE CONTROL	52
4.6 VISUALIZACIÓN.....	56
4.6.1 REGISTRO DE DESPLAZAMIENTO	57
4.6.2 REFRESCO	60
4.6.3 MULTIPLEXOR.....	61
4.6.4 DESCODIFICADOR ASCII DE 7 SEGMENTOS.....	63
4.7 DIAGRAMA LÓGICO DEL CIRCUITO DIGITAL.....	66
4.8 ASOCIACIONES	67
5. MEJORAS	69
5.1 RESETS	69
5.1.1 RESET MANUAL	71
5.1.2 RESET AUTOMÁTICO	74
5.3 DETECCIÓN DE LA LLEGADA DE CARACTERES ERRÓNEOS	77
5.4 RECTIFICADOR DE PRECISIÓN.....	82
5.5. DISTINCIÓN ENTRE CARACTERES ALFABÉTICOS Y NUMÉRICOS.....	88
6. CONCLUSIÓN	90

1. ¿QUÉ ES LA SEÑAL RDS?

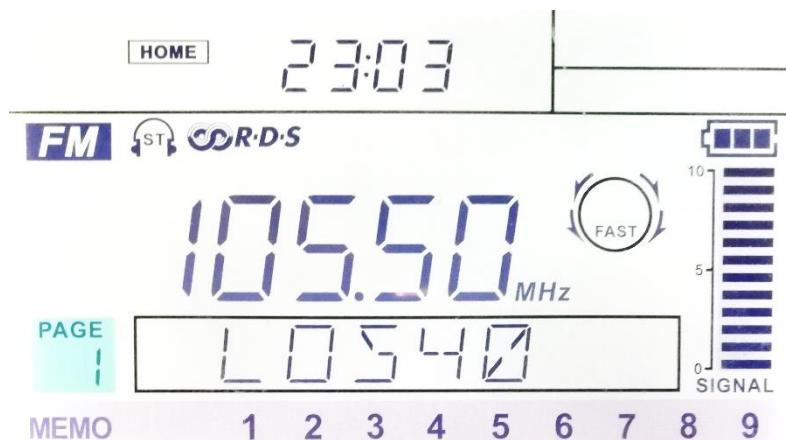
RDS (Radio Data System) es un protocolo de comunicaciones que permite enviar pequeñas cantidades de datos digitales, inaudibles en el receptor, con la señal de una emisora de radio FM; parte de dichos datos se ven presentados en una pantalla del aparato receptor.



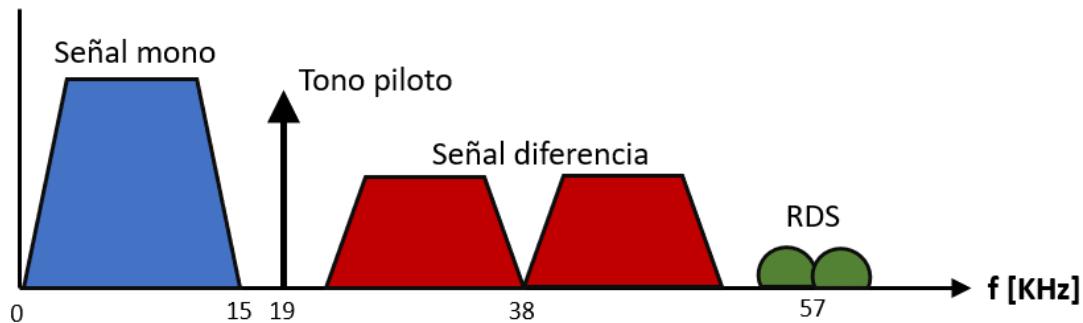
Por tanto, una señal RDS es una señal digital con modulación ASK y multiplexada en frecuencia dentro de la señal que se transmite por la radio FM. Los receptores modernos son capaces de decodificar dicha señal, y mostrar la información en una pantalla.



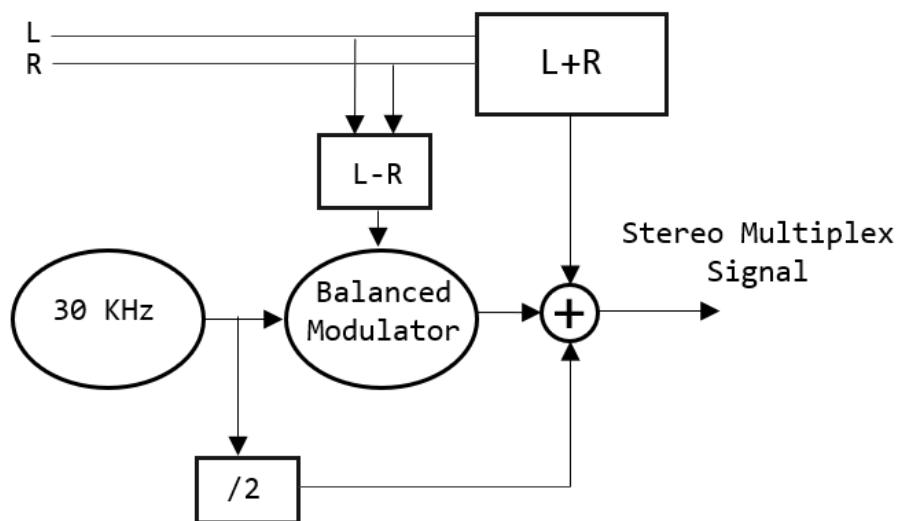
En una señal RDS podemos incluir datos como: identificación de la red de emisoras, nombre de la red de emisoras, frecuencias alternativas identificación de red con programas de tráfico, tipo de programa, información sobre otras redes de emisoras, identificación de información sobre el tráfico, identificación para el decodificador, número relacionado con la fecha y hora de emisión de un programa determinado, fecha y hora (CT), radio búsqueda, canal de mensajes de tráfico codificado, sistema de aviso de emergencia, etc.



El espectro en frecuencia de la señal RDS transmitida por las emisoras FM contiene varias componentes multiplexadas en frecuencia. El audio MONO (suma de los canales izquierdo y derecho) se encuentra en banda base, el audio STEREO (diferencia de los canales L-R) se encuentra modulada en doble banda lateral sobre una portadora de 38 [kHz]. Por otro lado, la señal digital RDS se transmite en ASK (Amplitude Shift Keying) sobre una portadora de 57 [kHz]. Por último, también se transmite un tono piloto sinusoidal de 19 [kHz] que puede utilizarse para recuperar las subportadoras sobre las que van moduladas las señales L-R y RDS. Por tanto, la señal FM estéreo múltiples tiene un ancho de banda de unos 60 [kHz].

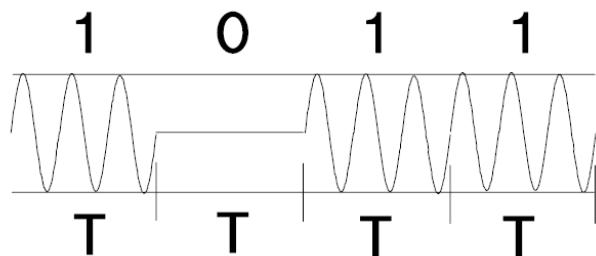


La razón de esta señal tan compleja se debe a los receptores monoaurales, que deben ser capaces de reproducir la señal MONO. Estos receptores filtran todas las componentes por encima de 15 [kHz] y reproducen exclusivamente la señal L+R. En cambio, los receptores estéreo demodulan también la señal diferencia L-R, y mediante las dos obtienen las señales L y R separadas. Por último, los receptores más modernos, son capaces de reproducir la señal RDS, la demodulan y además la componente digital a 57 [kHz] y decodifican los bits para producir una secuencia de caracteres que aparecen en una pantalla. La señal RDS, es muy útil en la actualidad porque permite por ejemplo el seguimiento de las emisoras a lo largo de un trayecto largo donde sus frecuencias van cambiando.

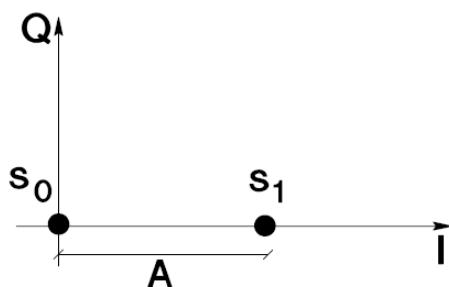


2. MODULACIÓN ASK

Consiste en enviar, cada T segundos, una de entre M sinusoides, todas ellas de la misma frecuencia, pero de diferentes amplitudes, incluyendo entre estas la amplitud nula (ausencia de señal). A la modulación ASK binaria se le llama también OOK (On-Off Keying).



La constelación de este esquema de modulación está formada por dos puntos, uno en el origen, asociado al dato "0" y uno en una sinusoida de amplitud A asociado al dato "1".



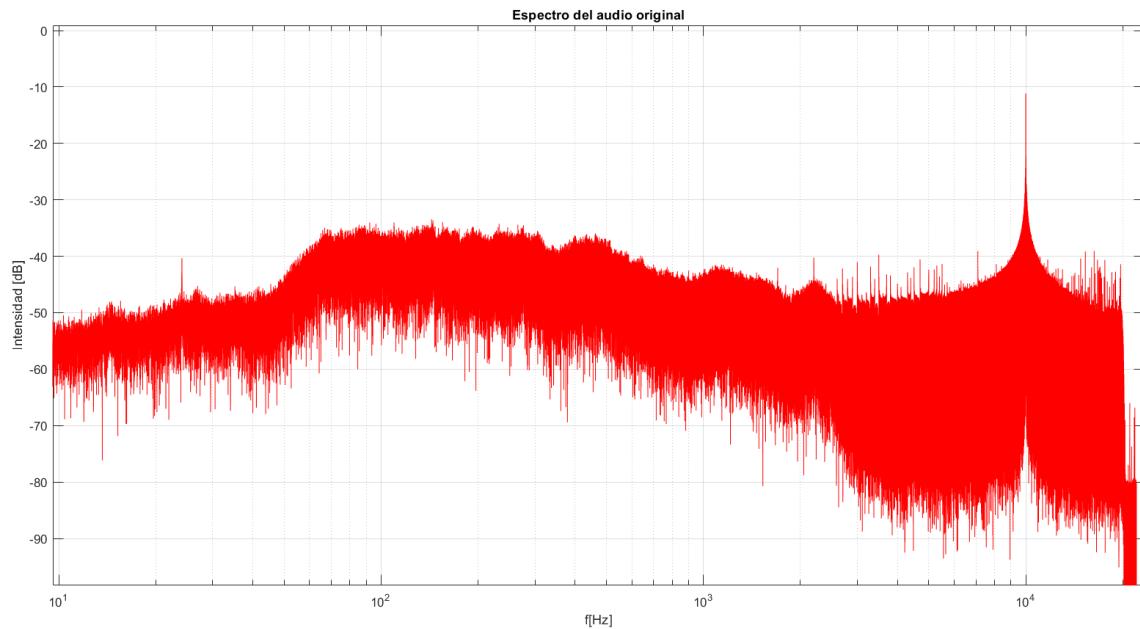
En la demodulación de la señal ASK puede utilizarse un simple diodo detector (demodulación incoherente) o la multiplicación por un coseno (demodulación coherente).

PRESTACIONES:

La modulación ASK no tiene buenas prestaciones, siendo su única baza la sencillez. Desde el punto de vista de la constelación, se observa que no es óptima frente al ruido, al no ser una constelación centrada. Desde el punto de vista del espectro, se observa que hay potencia desperdiciada en portadora. En concreto un 50%, que es la eficiencia de potencia de una modulación AM de estas características. En cuanto a otras propiedades, ASK tampoco es una modulación de envolvente constante. Su mayor (única) ventaja es la sencillez, tanto en su generación como en la recepción. De hecho, ASK es una modulación que apenas se usa en los sistemas comerciales ya que existen alternativas que, siendo poco más complejas, ofrecen prestaciones superiores.

ESPECTRO:

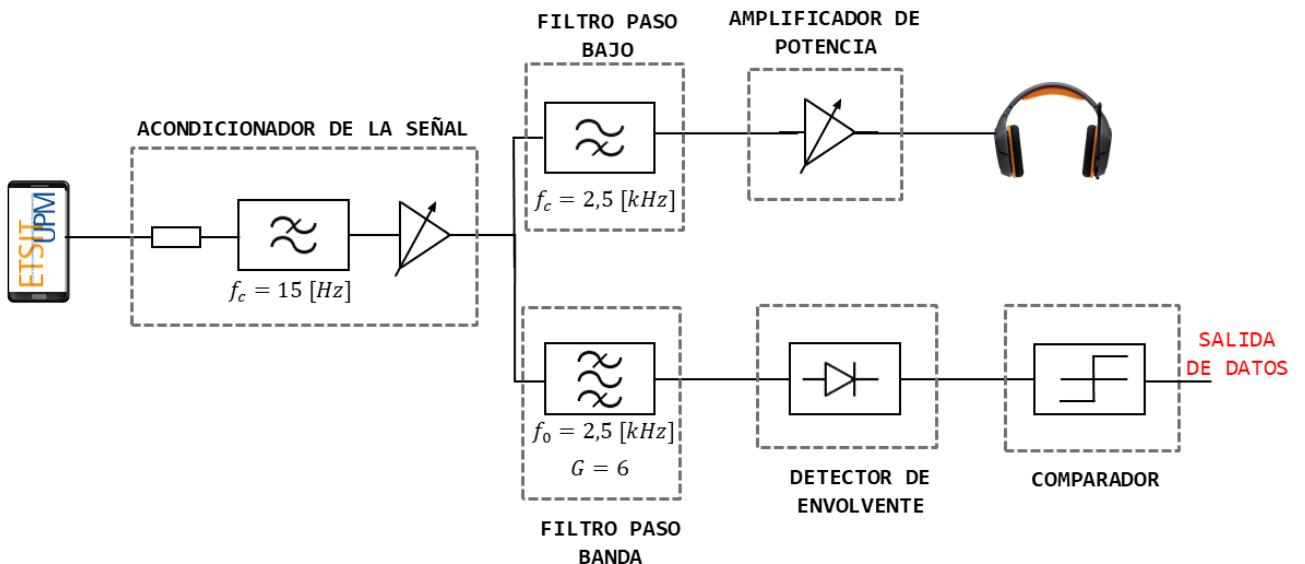
En este proyecto, la señal de datos tiene un espectro centrado en 10 [kHz], siendo las bandas laterales las que contienen los bits en baja frecuencia ($f=10$ [Hz] y pulsos de $T=100$ [ms]) que pretendemos extraer.



Para la demodulación ASK usaremos un detector de envolvente, con un rectificador (diodo) y un filtro paso bajo. Finalmente, se usará un comparador, obteniendo niveles de tensión estables entre 0 y 5 [V].

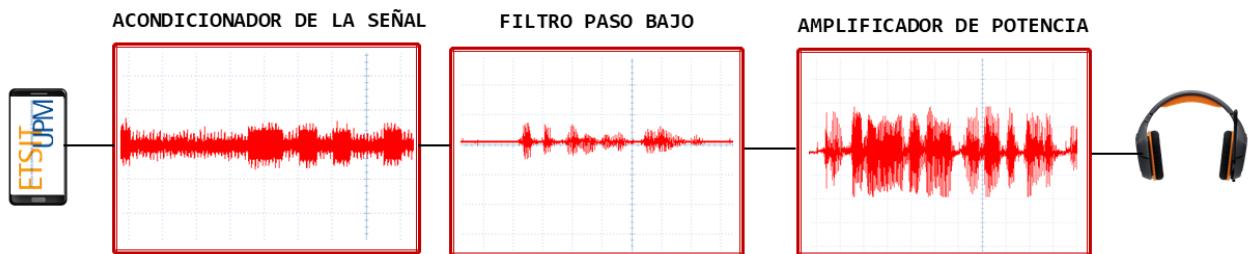
3. CIRCUITO ANALÓGICO

La función del circuito analógico es separar la señal de voz (situada en la banda base, con ancho de banda 2,5 [kHz]) de la señal de datos ASK (modulada sobre una portadora de 10 [kHz]). Una vez separadas, la señal de voz deberá ser amplificada, y la señal de datos demodulada. Todo esto se realizará con el circuito que se describe a continuación:

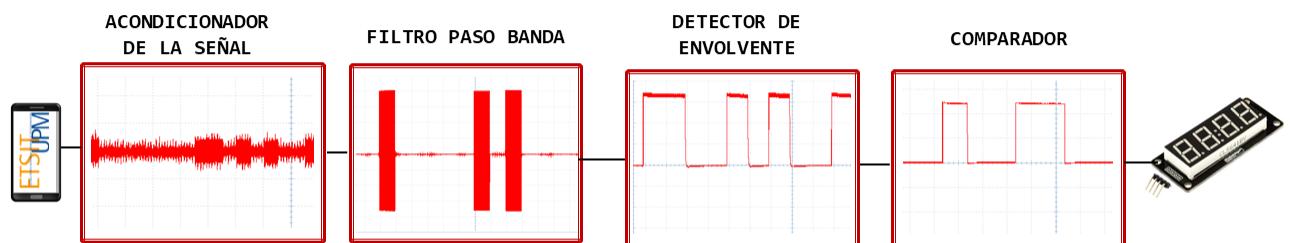


- **Acondicionador de señal:** Se encarga de adaptar la entrada del circuito a la salida de nuestro reproductor MP3, en nuestro caso un móvil, eliminando la componente continua y amplificando la señal.
- **Filtro paso bajo:** Se diseña este filtro con 4 polos (pendiente de -80 [dB/década]) y con frecuencia de corte $2,5$ [kHz], de forma que deje pasar la señal de voz y atenúe la señal de datos localizada a 10 [kHz] unos 50 [dB].
- **Amplificador de potencia:** Se utiliza para amplificar la señal de salida del amplificador operacional del filtro paso bajo, porque por sí sola no es capaz de excitar un auricular.
- **Filtro paso banda:** Este filtro se diseña con frecuencia de paso de 10 [kHz] de forma que tome la señal de datos y atenúe la señal de voz.
- **Demodulador ASK:** Toma la señal y la transforma en una señal de bits. Se compone de:
 - **Detector de envolvente:** Se encarga de obtener la envolvente positiva de la señal de datos mediante un diodo que actúa como rectificador de media onda y un filtro paso bajo de orden 1.
 - **Comparador:** Compara los niveles de tensión de la salida del detector con un umbral, de forma que, si es mayor, entrega una tensión de 5 [V] y en caso contrario de 0 [V]. De esta forma se obtiene la señal con flancos definidos entre 0 y 5 [V], compatibles con la entrada de la FPGA.

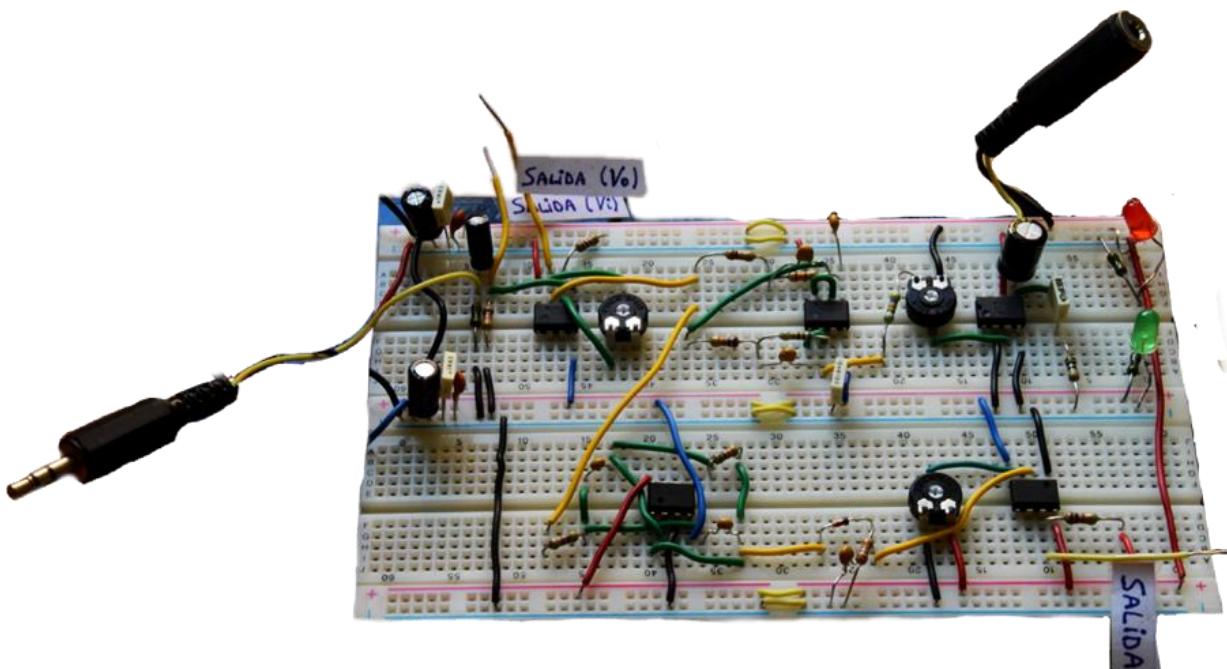
A continuación, podemos observar la evolución de la señal, desde el dispositivo MP3 (móvil) hasta llegar a los auriculares, para escuchar la primera parte del primer capítulo del Quijote:



Por otra parte, tenemos la evolución de la señal, hasta llegar a la entrada de la tarjeta BASYS2:



La implementación del circuito completo analógico en la placa de inserción queda de la siguiente forma:



3.1 ACONDICIONADOR DE LA SEÑAL

La función del acondicionador de la señal es eliminar la componente continua que pueda presentar la señal en la salida del móvil, además amplifica la señal para obtener 1 Vpp a su salida.

Este módulo está compuesto por tres elementos: una impedancia de entrada, un filtro paso alto y un amplificador.

3.1.1 IMPEDANCIA DE ENTRADA:

Una resistencia de valor igual a 1 [kΩ], su función es actuar como carga para el amplificador de salida del dispositivo de reproducción. A la entrada del circuito, se ha conectado un puerto Jack de 3,5 mm estéreo, soldado a la entrada para poder conectar el dispositivo MP3.

3.1.2 FILTRO PASO ALTO:

Elimina la componente continua procedente del reproductor MP3 (el móvil en nuestro caso). En un primer momento, elegimos la frecuencia de corte del filtro en 15 [Hz]. Con dicha frecuencia podemos obtener el valor de R y C:

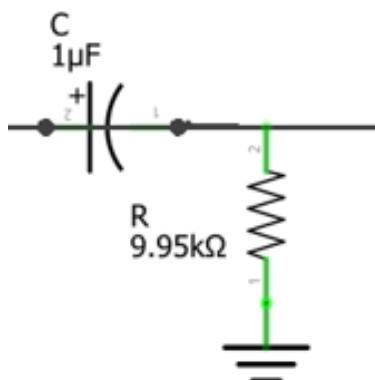
$$f_c = \frac{1}{2 \cdot \pi \cdot R \cdot C}$$

Para un valor de C= 1 [μF], la resistencia será igual a:

$$R = \frac{1}{2 \cdot \pi \cdot f_c \cdot C} = \frac{1}{2 \cdot \pi \cdot 15 \text{ [Hz]} \cdot 10^{-6} \text{ [F]}} = 10610,33 \text{ [\Omega]}$$

Sin embargo, para obtener un valor más exacto de la resistencia del filtro paso alto, la redondeamos a 10 [kΩ], finalmente la frecuencia de corte de nuestro filtro nos quedará en 15,91 [Hz].

$$f_c = 15,91 \text{ [Hz]}$$



➤ **Comprobación de la frecuencia de corte del filtro paso alto:**

Conectamos el generador de señal para obtener una salida senoidal de 1 [Vpp] y 1 [kHz] de frecuencia, al bajar la frecuencia del generador, el nivel de salida del filtro paso alto es 3 dB menor que la amplitud, es decir, $v_{OUT} = \frac{1 \text{ Vpp}}{\sqrt{2}} = 0,707 \text{ [Vpp]}$, en nuestro caso $v_{OUT} = 688 \text{ [mVpp]}$, por lo tanto, la frecuencia de corte coincide con la esperada.

3.1.3 AMPLIFICADOR DE SEÑAL:

Finalmente, la última parte del módulo es amplificar la señal mediante un amplificador no inversor (con realimentación negativa y alimentado entre ± 5 [V]) implementado mediante un amplificador operacional “TL082”, con ganancia variable, controlado mediante un potenciómetro de 10 [$k\Omega$].

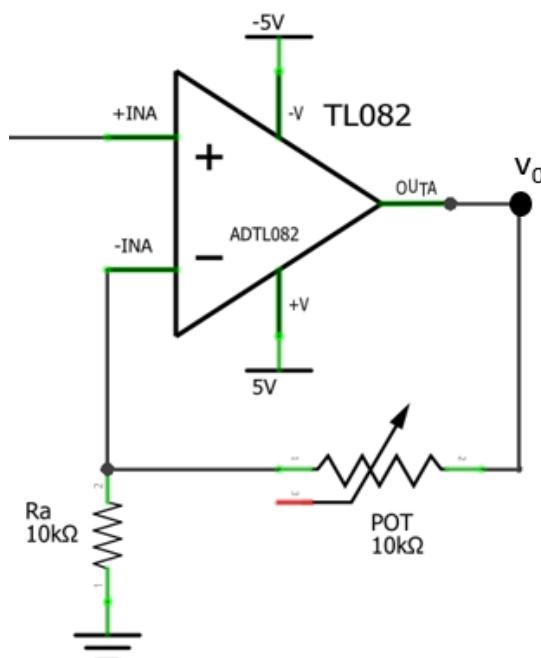
Para calcular la R_a , realizamos los siguientes cálculos:

- Considerando que el amplificador tiene una ganancia de 11 ($G=11$), y que el potenciómetro es de 10 [$k\Omega$] ($POT = 10 \text{ [k}\Omega]$). La resistencia se calcula con esta fórmula:

$$G = 1 + \frac{POT}{R_a}$$

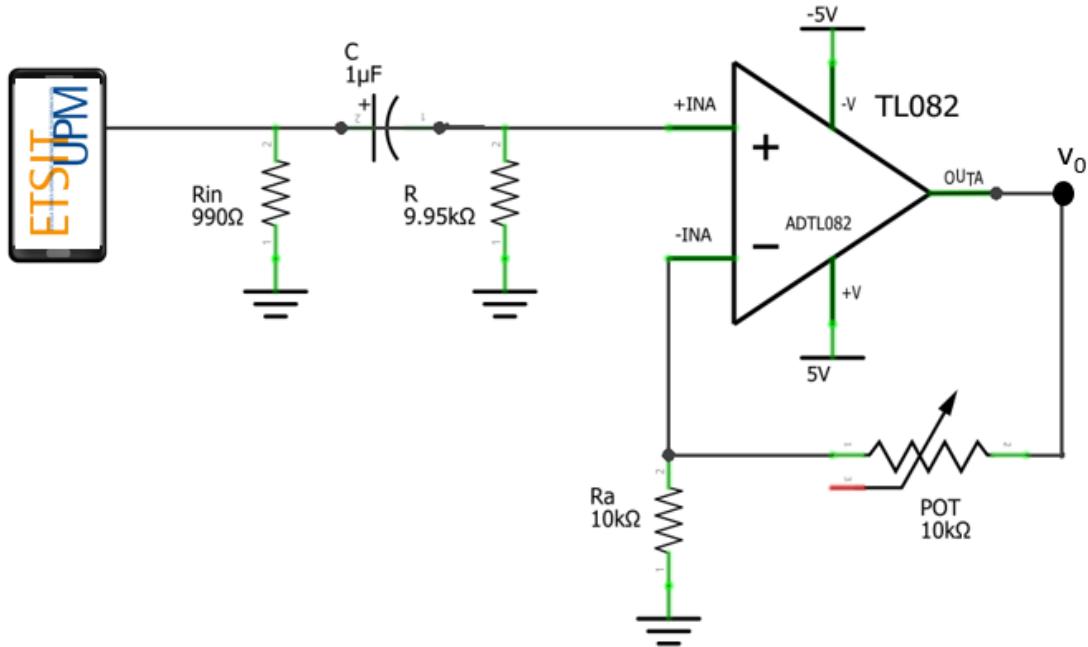
$$G - 1 = \frac{POT}{R_a}$$

$$R_a = \frac{POT}{G - 1} = \frac{10 \text{ [k}\Omega]}{11 - 1} = 1 \text{ [k}\Omega]$$

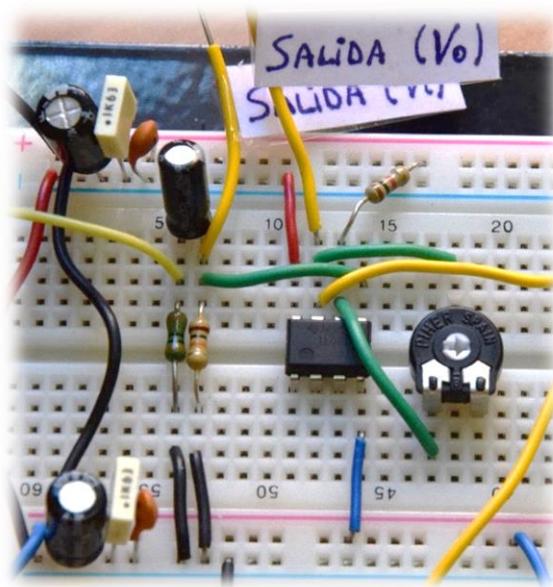


3.1.4 ESQUEMA DEL CIRCUITO

Por lo tanto, el esquema final del acondicionador de señal, con los valores reales será:

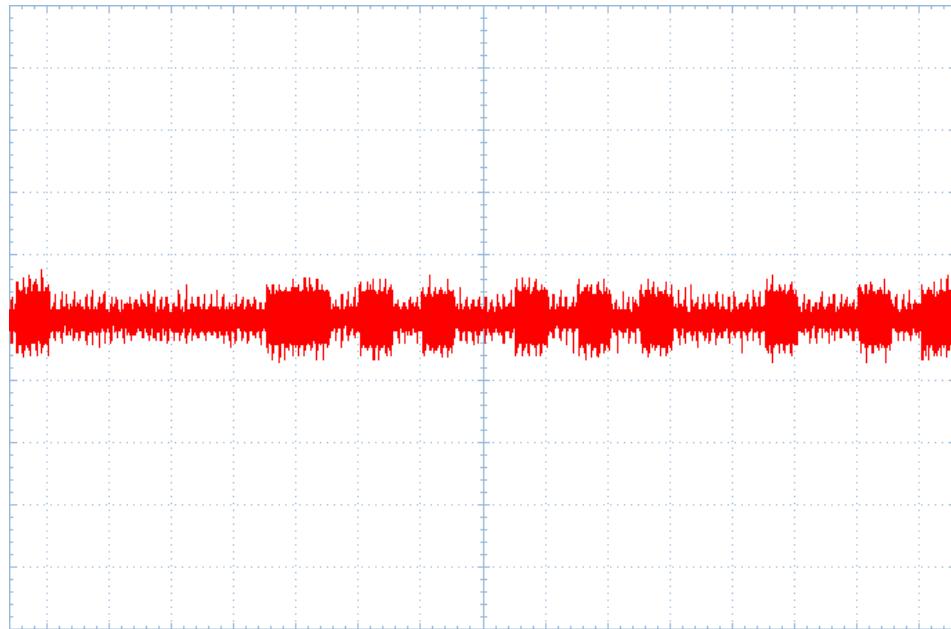


Y la implementación en la placa base con los componentes electrónicos será:



3.1.5. SEÑAL DE SALIDA EN EL OSCILOSCOPIO

Al introducir la señal de audio en la entrada del acondicionador de la señal, a la salida obtenemos la siguiente señal en el osciloscopio (Escala: 1 [V] y 100 [ms]):



3.2 FILTRO PASO BAJO

El filtro paso bajo se encarga de filtrar la señal de audio (con ganancia unidad y 4 polos). La frecuencia de corte de este filtro está situada en 2,5 [kHz], valor que permite atenuar suficientemente la componente de 10 [kHz] dejando pasar el audio sin apenas distorsión.

3.2.1 DISEÑO DEL FILTRO PASO BAJO

La función de transferencia del filtro paso bajo de orden 2 (con etapas Sallen-Key) a diseñar es:

$$H(j\omega) = \frac{\omega_0^2}{-\omega^2 + j\omega \frac{\omega_0}{Q} + \omega_0^2}$$

Donde existen dos parámetros en el filtro a diseñar en cada etapa, el factor de calidad (Q) y la frecuencia característica ($f_0 = \omega_0 / 2\pi$):

$$f_0 = \frac{1}{2\pi \cdot \sqrt{m \cdot n} \cdot RC}$$
$$Q = \frac{\sqrt{m \cdot n}}{m + 1}$$

Los valores de los componentes en cada etapa han sido diseñados siguiendo los siguientes valores:

Etapa	f_0 [Hz]	Q
1	1350	0,7
2	2330	2,94

En la etapa 1:

Elegimos el valor de $n=10$ (década exacta) y utilizando el valor de $Q = 0,7$:

$$Q \cdot (m + 1) = \sqrt{m \cdot n}$$

$$0,7 \cdot (m + 1) = \sqrt{m \cdot n}$$

$$(0,7)^2 \cdot (m^2 + 2m + 1) = 10 \cdot m$$

$$m^2 + 2m + 1 = 20,408 \cdot m$$

$$m^2 - 18,408 \cdot m + 1 = 0$$

Donde los dos posibles valores de m son: 18,353 y 0,054.

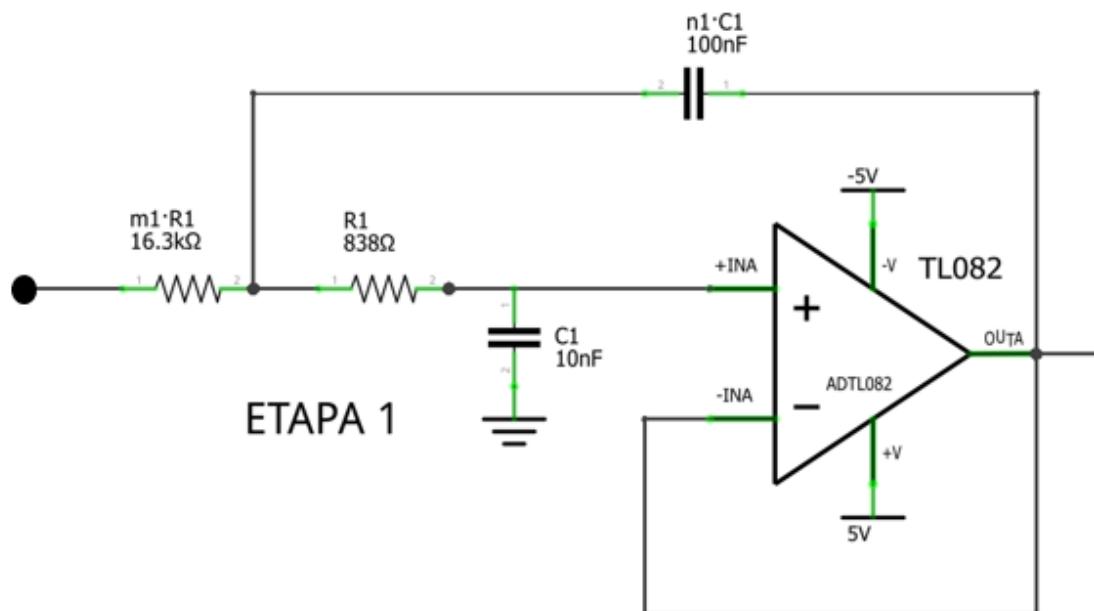
A continuación, elegimos el valor del condensador C , en nuestro caso $C_1 = 10 \text{ [nF]}$ (década exacta), y teniendo en cuenta que $f_0 = 1350 \text{ [Hz]}$, calculamos el valor de R :

$$f_0 = \frac{1}{2\pi\sqrt{m \cdot n} \cdot R_1 \cdot C_1}$$

$$R_1 = \frac{1}{2\pi\sqrt{m \cdot n} \cdot f_0 \cdot C_1}$$

$$R_1 = \frac{1}{2\pi\sqrt{18,353 \cdot 10} \cdot 1350 \cdot 10 \cdot 10^{-9}}$$

$$R_1 = \frac{1}{1,149 \cdot 10^{-3}} = 870,21 \text{ [\Omega]}$$



Por lo tanto, en la etapa 1, los valores elegidos serán:

$$R_1 = 870,21 \text{ [\Omega]}$$

$$m_1 \cdot R_1 = 15971,5 \text{ [\Omega]}$$

$$C_1 = 10 \text{ [nF]}$$

$$n_1 \cdot C_1 = 100 \text{ [nF]}$$

Pero al no ser componentes ideales, poseen cierta tolerancia (más o menos del 5%), y el valor real de los componentes utilizados en la implementación del circuito obtenidos con un multímetro serán:

$$R_1 = 838 \text{ } [\Omega]$$

$$m_1 \cdot R_1 = 16300 \text{ } [\Omega]$$

$$C_1 = 10 \text{ } [nF]$$

$$n_1 \cdot C_1 = 100 \text{ } [nF]$$

Análogamente, en la **etapa 2**:

Elegimos el valor de n=100 y utilizando el valor de Q = 2,94:

$$Q \cdot (m + 1) = \sqrt{m \cdot n}$$

$$2,94 \cdot (m + 1) = \sqrt{m \cdot n}$$

$$(2,94)^2 \cdot (m^2 + 2m + 1) = 100 \cdot m$$

$$m^2 - 9,569 \cdot m + 1 = 0$$

Donde los dos posibles valores de m son: 9,463 y 0,105.

Elegimos el valor del condensador C, en nuestro caso $C_2 = 1 \text{ } [nF]$ y teniendo en cuenta que $f_0 = 2330 \text{ } [\text{Hz}]$, calculamos el valor de R:

$$f_0 = \frac{1}{2\pi\sqrt{m \cdot n} \cdot R_2 \cdot C_2}$$

$$R_2 = \frac{1}{2\pi\sqrt{m \cdot n} \cdot f_0 \cdot C_2}$$

$$R_2 = \frac{1}{2\pi\sqrt{9,463 \cdot 100} \cdot 2330 \cdot 1 \cdot 10^{-9}}$$

$$R_1 = \frac{1}{4,503 \cdot 10^{-4}} = 2220,63 \text{ } [\Omega]$$

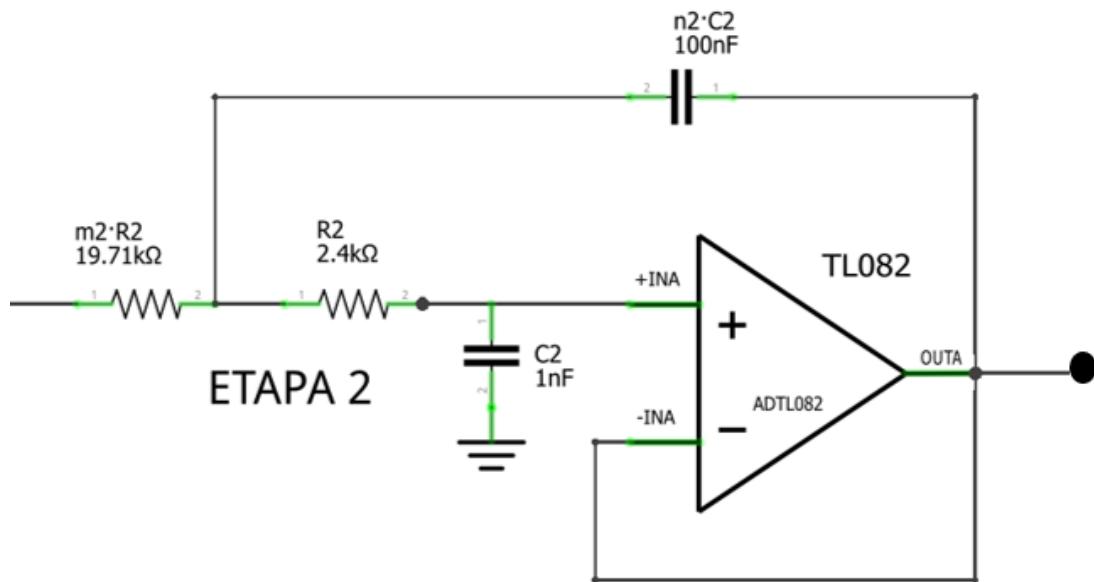
Finalmente, en la etapa 2, los valores elegidos serán:

$$R_2 = 2220,63[\Omega]$$

$$m_2 \cdot R_2 = 21014,57 [\Omega]$$

$$C_2 = 1 [nF]$$

$$n_2 \cdot C_2 = 100 [nF]$$



Como en el caso anterior, al no ser componentes ideales, poseen cierta tolerancia (más o menos del 5%), y el valor real de los componentes utilizados en la implementación del circuito obtenidos con un multímetro serán:

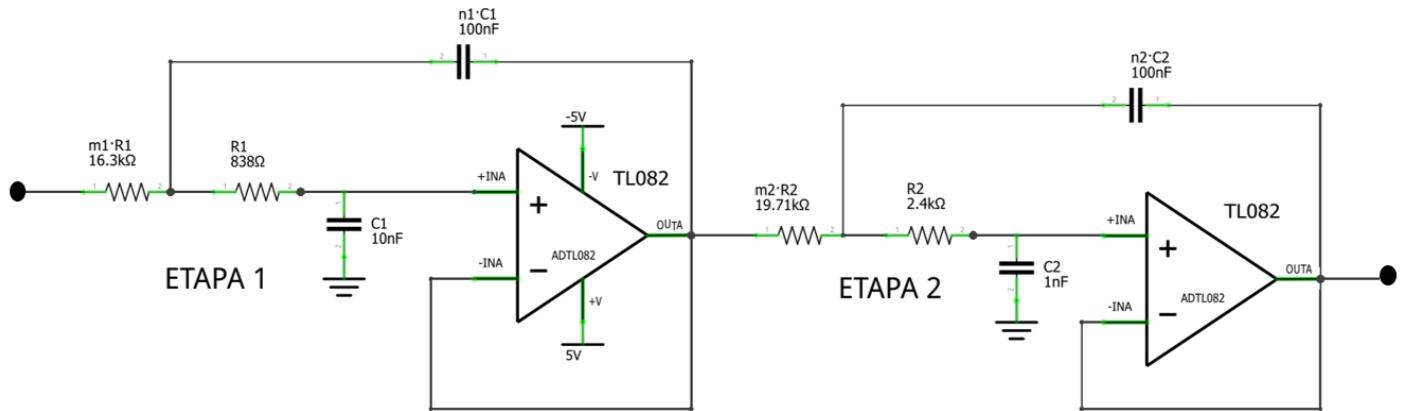
$$R_2 = 2400 [\Omega]$$

$$m_2 \cdot R_2 = 19,71 [\text{k}\Omega]$$

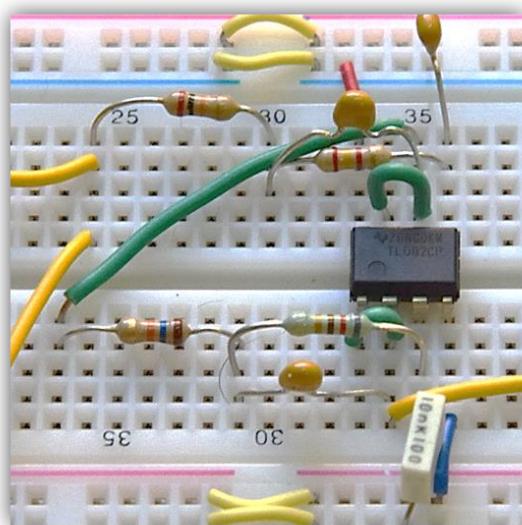
$$C_2 = 1 [nF]$$

$$n_2 \cdot C_2 = 100 [nF]$$

3.2.2 ESQUEMA DEL FILTRO PASO BAJO



La implementación en la placa base quedará:



3.2.3 MEDIDAS DEL FILTRO PASO BAJO

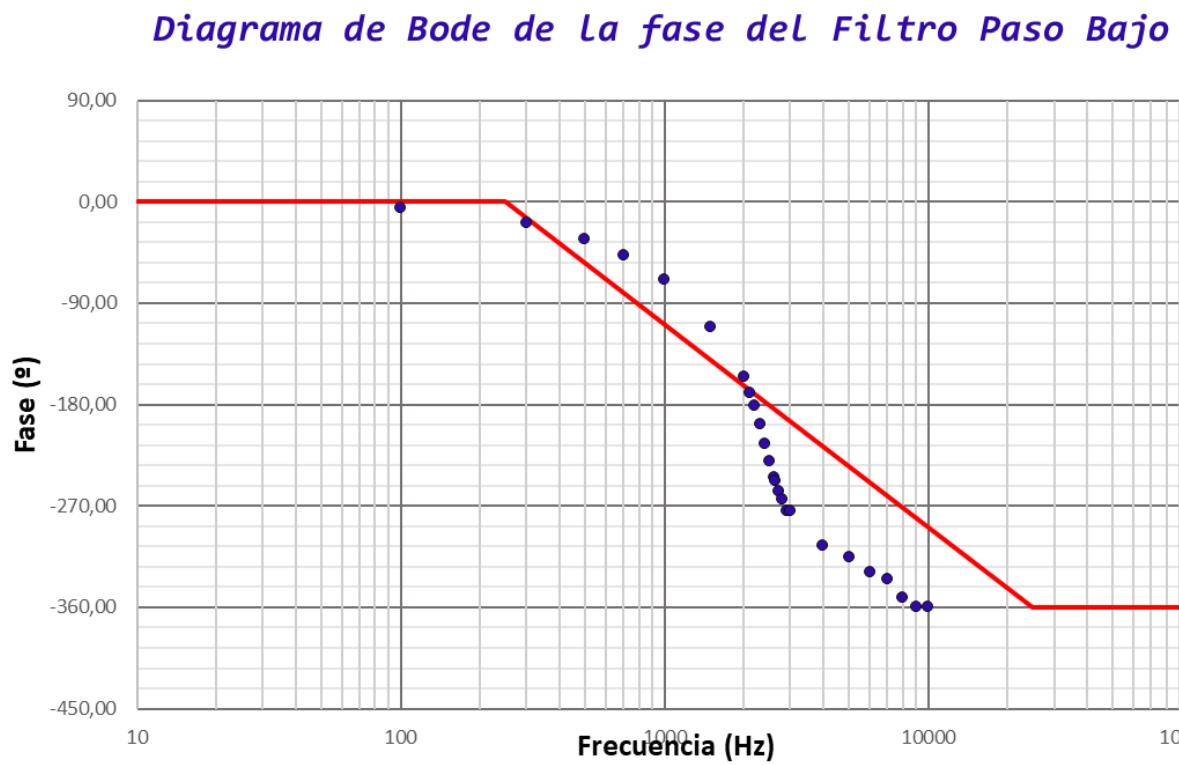
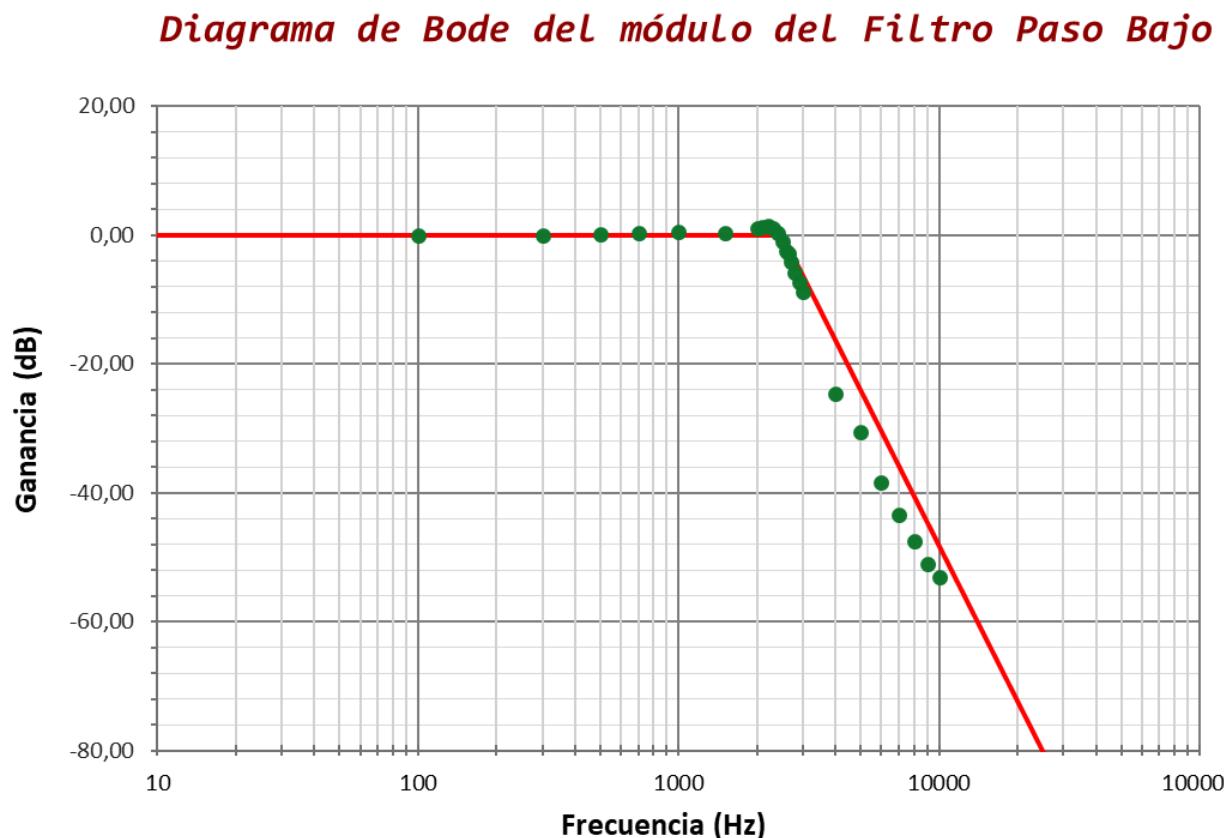
Tabla de valores *medidos*

Frecuencia (Hz)	T(s)	Entrada (Vpp)	Salida (Vpp)	Ganancia	Ganancia (dB)	Δt (s)	Fase (º)
100	0,01000	1	1	1,00	0,00	0,000136	-4,90
300	0,00333	1	1	1,00	0,00	0,00017	-18,36
500	0,00200	1	1,016	1,02	0,14	0,000184	-33,12
700	0,00143	1	1,032	1,03	0,27	0,000187	-47,12
1000	0,00100	1	1,048	1,05	0,41	0,000193	-69,48
1500	0,00067	1	1,04	1,04	0,34	0,000205	-110,70
2000	0,00050	1	1,136	1,14	1,11	0,000216	-155,52
2100	0,00048	1	1,16	1,16	1,29	0,000225	-170,10
2200	0,00045	1	1,168	1,17	1,35	0,000228	-180,58
2300	0,00043	1	1,128	1,13	1,05	0,000238	-197,06
2400	0,00042	1	1,032	1,03	0,27	0,000249	-215,14
2500	0,00040	1	0,896	0,90	-0,95	0,000256	-230,40
2600	0,00038	1	0,752	0,75	-2,48	0,000261	-244,30
2637	0,00038	1	0,712	0,71	-2,95	0,000261	-247,77
2700	0,00037	1	0,624	0,62	-4,10	0,000264	-256,61
2800	0,00036	1	0,512	0,51	-5,81	0,000262	-264,10
2900	0,00034	1	0,432	0,43	-7,29	0,000263	-274,57
3000	0,00033	1	0,36	0,36	-8,87	0,000254	-274,32
4000	0,00025	20	1,18	0,06	-24,58	0,000212	-305,28
5000	0,00020	20	0,59	0,03	-30,60	0,000175	-315,00
6000	0,00017	20	0,24	0,012	-38,42	0,000152	-328,32
7000	0,00014	20	0,136	0,007	-43,35	0,000133	-335,16
8000	0,00013	20	0,084	0,004	-47,54	0,000122	-351,36
9000	0,00011	20	0,056	0,003	-51,06	0,000111	-359,64
10000	0,00010	20	0,044	0,002	-53,15	0,0001	-360,00

Valor de la frecuencia de corte:

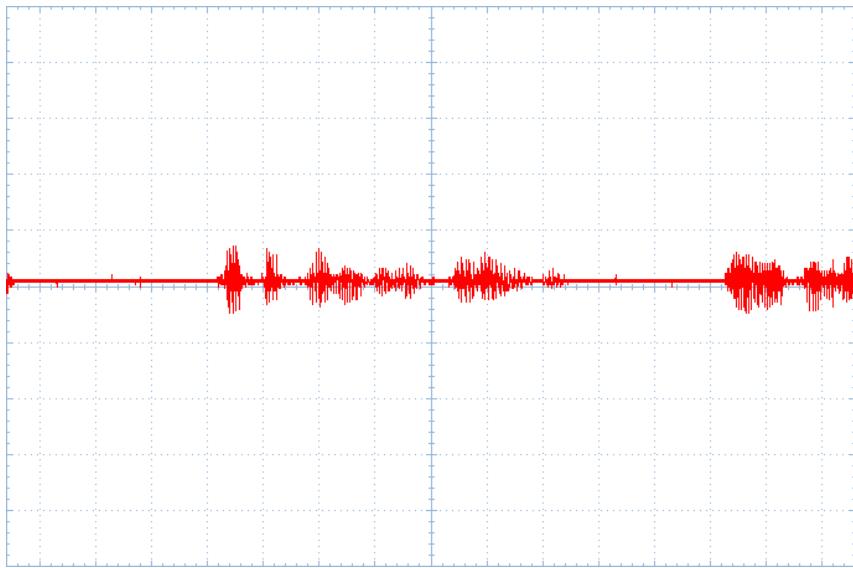
f (Hz)	2637	Ganancia (dB)	-2,95
--------	------	---------------	-------

3.2.4 DIAGRAMAS DE BODE DEL FILTRO PASO BAJO



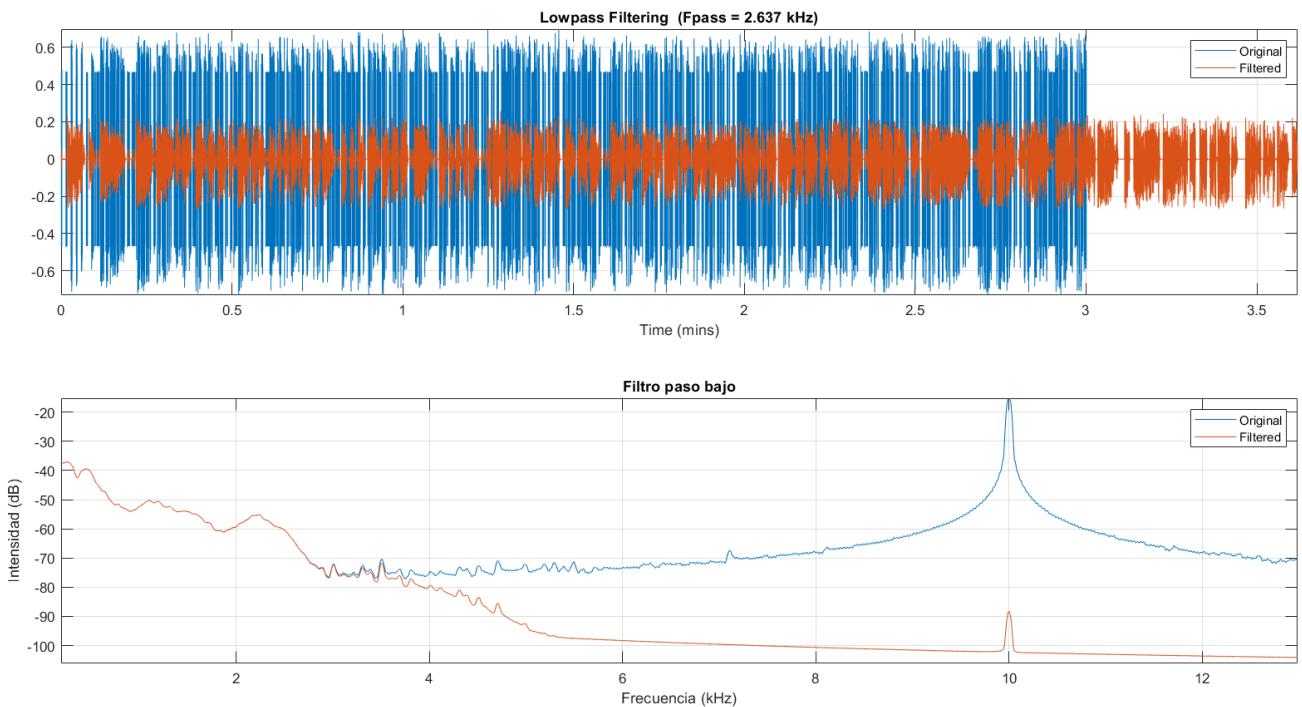
3.2.5 SEÑAL DE SALIDA EN EL OSCILOSCOPIO

Al introducir la señal de audio en la entrada del circuito, a la salida del filtro paso bajo obtenemos la siguiente señal en el osciloscopio (1 [V] y 100 [ms]):



3.2.6 ESPECTRO A LA SALIDA DEL FILTRO

Tras filtrar el audio por el filtro paso bajo, conseguimos atenuar la señal de datos modulados en 10 [kHz], pero no se consigue eliminar por completo. Como consecuencia de ello, al escuchar el audio se escuchan un leve sonido correspondiente a los datos modulados en ASK.

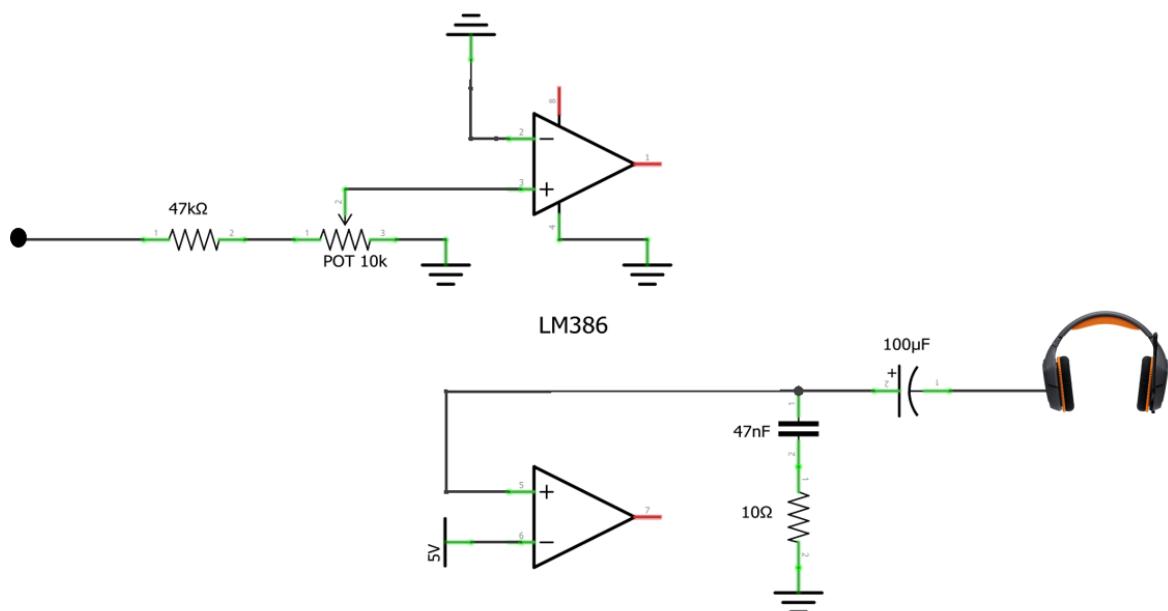


3.3 AMPLIFICADOR DE POTENCIA

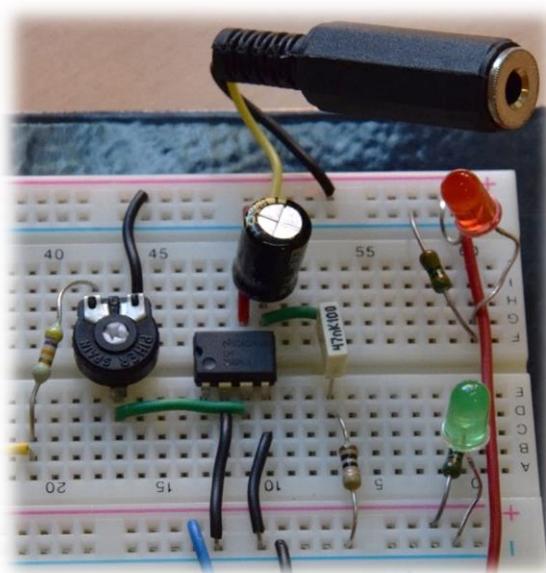
El amplificador de potencia amplifica la señal procedente del filtro paso bajo, para escucharse con mayor intensidad. Este módulo se ha realizado con un circuito integrado LM386, que consta de un amplificador de audio de ganancia 20, junto a una etapa de potencia que suministra la corriente suficiente para excitar los auriculares.

Además, se conectan un potenciómetro de $10\text{ k}\Omega$, que se emplea para regular el volumen del audio. Además, se ha soldado una hembrilla de "jack" en uno de los canales, para conectar los auriculares.

3.3.1 ESQUEMA DEL AMPLIFICADOR DE POTENCIA

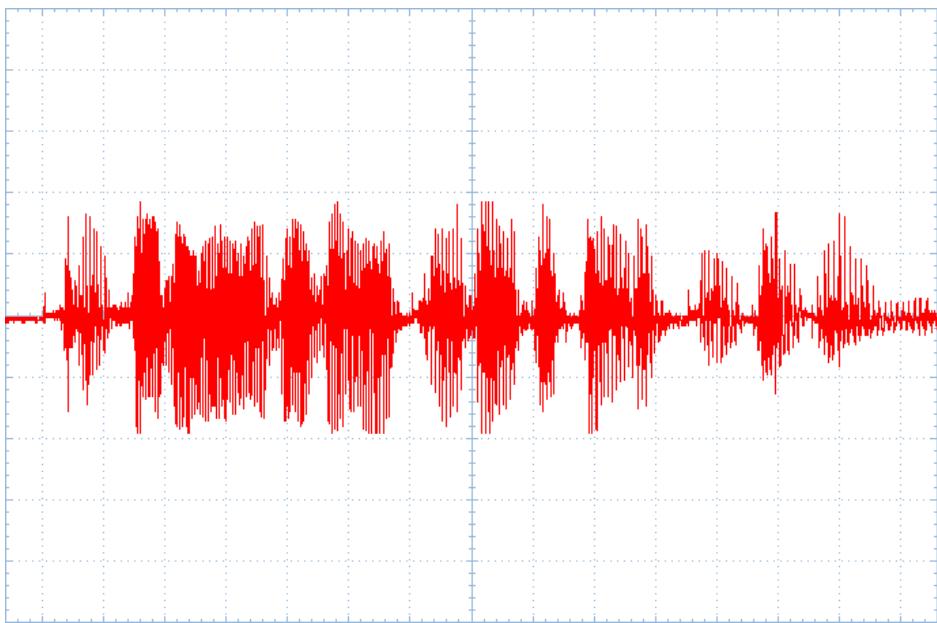


Y la implementación en la placa base quedará:



3.3.2 SEÑAL DE SALIDA DEL OSCILOSCOPIO

Tras introducir la señal de audio en la entrada del circuito, a la salida del amplificador de potencia, es decir, la señal que llega a los auriculares, obtenemos la siguiente señal en el osciloscopio (1 [V] y 100 [ms]):



3.4 FILTRO PASO BANDA

La función del filtro paso banda es extraer los datos (señal centrada en torno a 10 [kHz]), atenuando adecuadamente la señal de voz. Este filtro paso banda posee realimentación múltiple negativa, con un polo en 10 [kHz].

3.4.1 DISEÑO DEL FILTRO PASO BANDA

La función de transferencia del filtro paso banda a diseñar es:

$$H(j\omega) = G \cdot \frac{j\omega \cdot \frac{\omega_0}{Q}}{-\omega^2 + j\omega \frac{\omega_0}{Q} + \omega_0^2}$$

Este filtro paso banda va a tener una ganancia $|G| = 6$, siendo G igual a:

$$G = -\frac{R_2}{2 \cdot R_1}$$

Siendo f_0 la frecuencia donde la ganancia del filtro es máxima ($|G| = 6$), nuestro filtro paso banda ha sido diseñado para que f_0 sea igual a 10 [kHz].

$$f_0 = \frac{1}{2\pi \cdot \sqrt{R_1 \cdot R_2} \cdot C}$$

El factor de calidad del filtro, Q, indica la selectividad del filtro y se define:

$$Q = \frac{1}{2} \cdot \sqrt{\frac{R_2}{R_1}}$$

De tal modo que el ancho de banda a 3 dB cumple necesariamente la expresión:

$$BW = \frac{f_0}{Q}$$

- El valor del condensador, C , elegido es $C = 1 \text{ [nF]}$
- Necesitamos dos ecuaciones para calcular las resistencias R_1 y R_2 :

1) La primera ecuación la obtenemos a partir de la condición de $|G| = 6$:

$$|G| = 6 = \frac{R_2}{2 \cdot R_1}$$

$$12 = \frac{R_2}{R_1}$$

2) En cambio, la segunda ecuación se obtiene a partir de la condición de $f_0 = 10 \text{ [kHz]}$

$$f_0 = 10000 \text{ [Hz]} = \frac{1}{2\pi \cdot \sqrt{R_1 \cdot R_2} \cdot (1 \cdot 10^{-9} \text{ [F]})}$$

$$6,28 \cdot 10^{-5} = \frac{1}{\sqrt{R_1 \cdot R_2}}$$

$$15915,5 = \sqrt{R_1 \cdot R_2}$$

$$253302959,1 = R_1 \cdot R_2$$

➤ Por lo tanto, resolviendo este sistema de ecuaciones, obtenemos los valores de R_1 y R_2

$$\begin{cases} 253302959,1 = R_1 \cdot R_2 \\ R_2 = 12 \cdot R_1 \end{cases}$$

➤ Operando:

$$R_1 = 4594,41 \text{ [\Omega]}$$

$$R_2 = 55132,89 \text{ [\Omega]}$$

➤ El factor de calidad, Q:

$$Q = \frac{1}{2} \cdot \sqrt{\frac{R_2}{R_1}} = 1,73$$

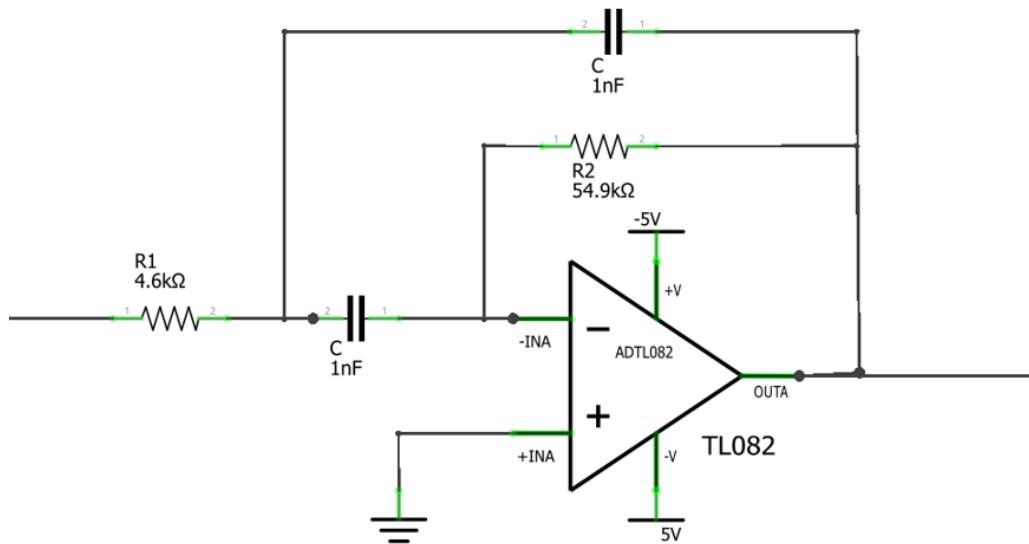
Los valores de las resistencias utilizadas en la implementación del circuito han sido:

$$R_1 = 4610 \text{ [\Omega]}$$

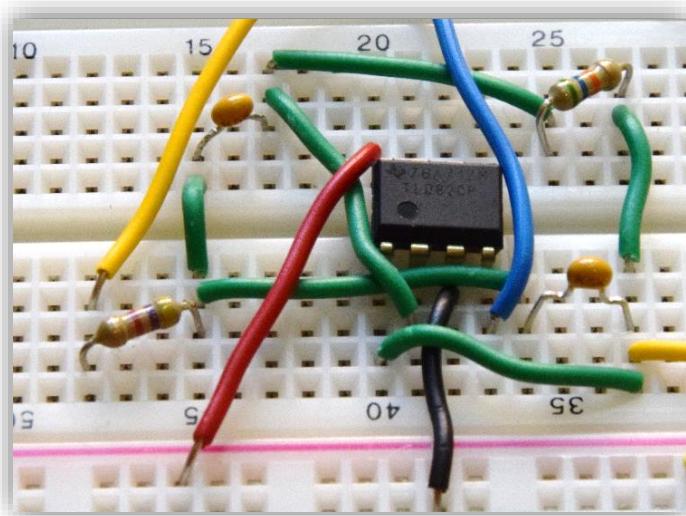
$$R_2 = 54900 \text{ [\Omega]}$$

3.4.2 ESQUEMA DEL FILTRO PASO BANDA

El esquema del circuito con los valores reales de los componentes empleados será:



Y el diseño del circuito en la placa base resulta:



3.4.3 MEDIDAS FILTRO PASO BANDA

Tabla de valores medidos

Frecuencia (Hz)	Entrada (Vpp)	Salida (Vpp)	Ganancia	Ganancia (dB)	Δt (s)	Fase (º)
100	1	0,06	0,06	-24,44	0,0028	-100,80
300	1	0,12	0,12	-18,42	0,00083	-89,64
500	1	0,188	0,19	-14,52	0,000512	-92,16
700	1	0,252	0,25	-11,97	0,000368	-92,74
1000	1	0,34	0,34	-9,37	0,000262	-94,32
3000	1	1,024	1,02	0,21	0,0000944	-101,95
5000	1	1,9	1,90	5,58	0,000063	-113,40
7000	1	3,14	3,14	9,94	0,000052	-131,04
9000	1	4,76	4,76	13,55	0,0000475	-153,90
10000	1	5,2	5,20	14,32	0,000048	-172,80
11000	1	5,2	5,20	14,32	0,0000478	-189,29
13000	1	4,4	4,40	12,87	0,0000458	-214,34
15000	1	3,52	3,52	10,93	0,0000424	-228,96
17000	1	2,88	2,88	9,19	0,0000392	-239,90
20000	1	2,24	2,24	7,00	0,0000342	-246,24
30000	1	1,36	1,36	2,67	0,0000243	-262,44
50000	1	0,8	0,80	-1,94	0,00001512	-272,16
70000	1	0,54	0,54	-5,35	0,00001068	-269,14
100000	1	0,38	0,38	-8,40	0,00000752	-270,72
300000	1	0,136	0,14	-17,33	0,00000254	-274,32
500000	1	0,088	0,09	-21,11	0,00000151	-271,80
700000	1	0,066	0,07	-23,61	0,000001088	-274,18

Valor de la frecuencia central y del ancho de banda

f_0 (Hz)	G(dB)
10430	14,58

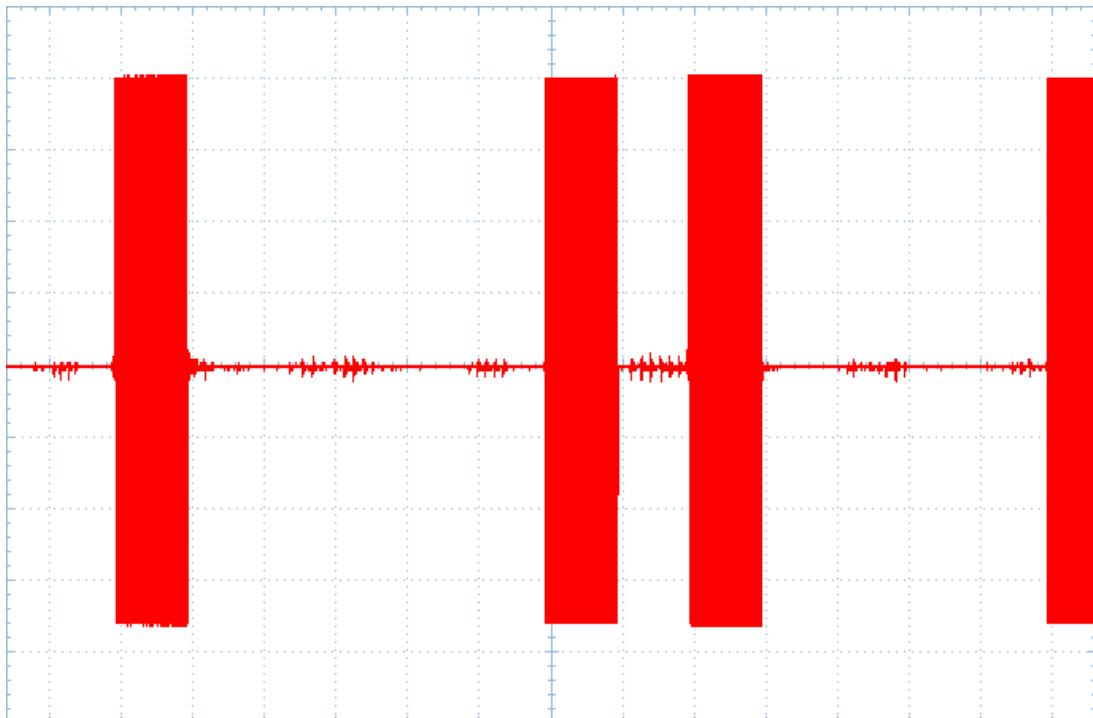
f_L (Hz)	G (dB)
7780	11,50

BW (Hz)
7020

f_H (Hz)	G (dB)
14800	11,55

3.4.4 SEÑAL DE SALIDA EN EL OSCILOSCOPIO

Al introducir la señal de audio en la entrada del circuito, a la salida del filtro paso banda obtenemos la siguiente señal en el osciloscopio (1 [V] y 100 [ms]):



3.4.5 DIAGRAMAS DE BODE DEL FILTRO PASO BANDA

Diagrama de bode del modulo del filtro paso banda

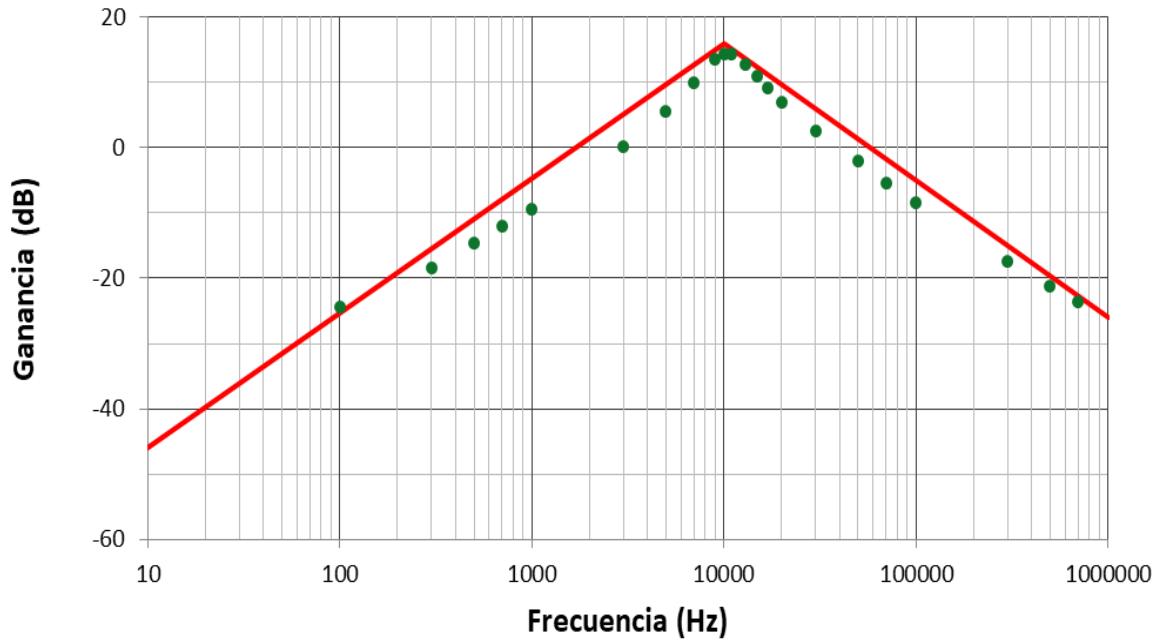
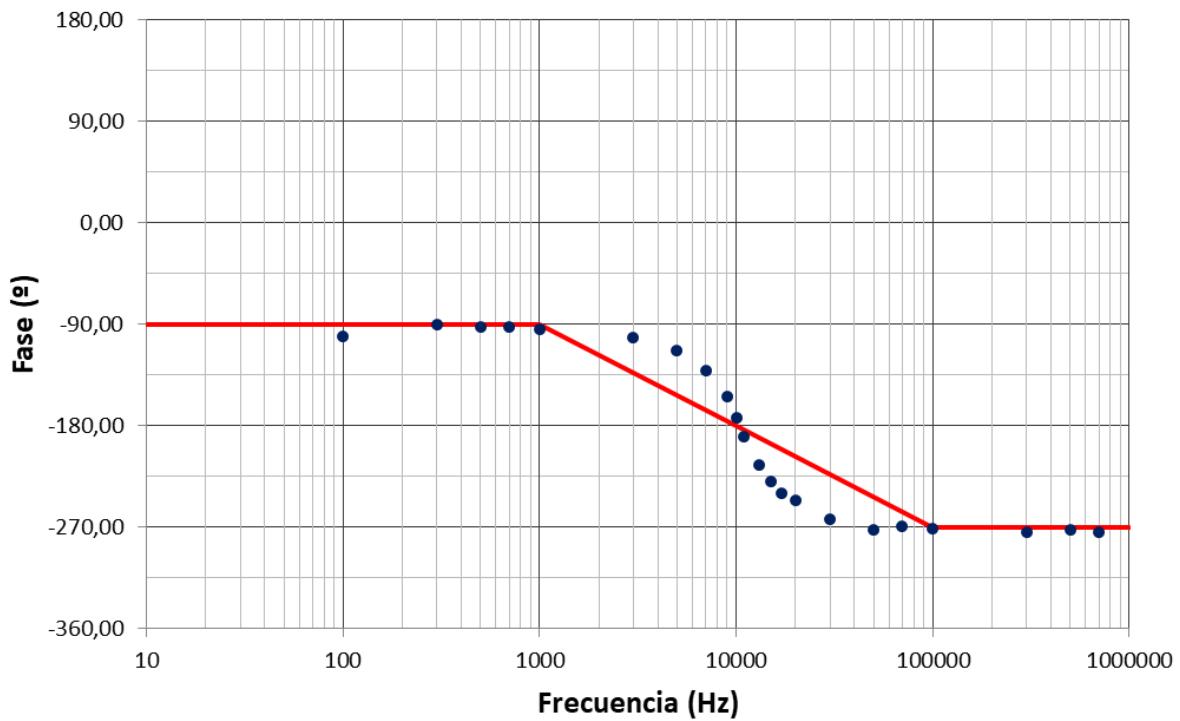
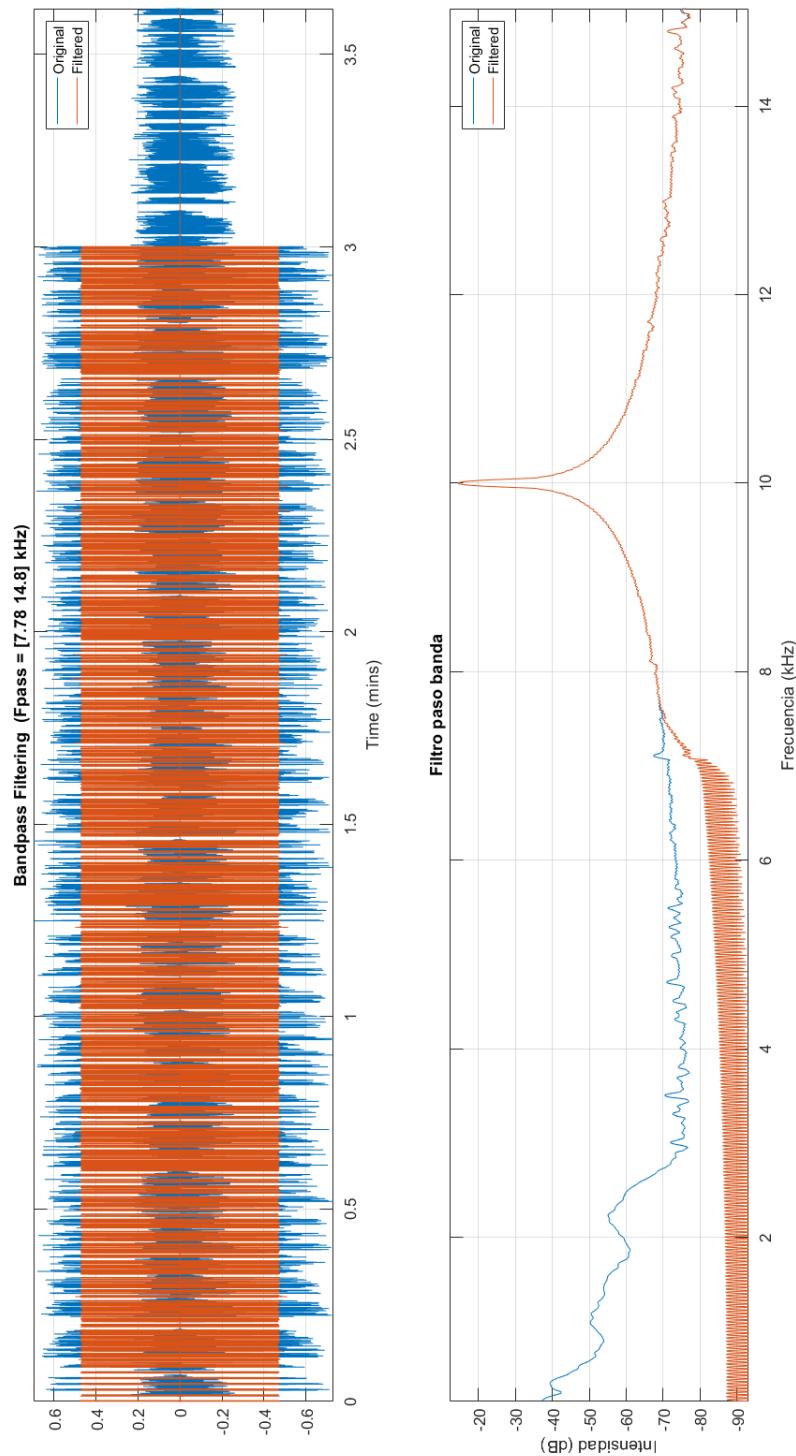


Diagrama de bode de la fase del filtro paso banda



3.4.6 ESPECTRO A LA SALIDA DEL FILTRO

A la salida del filtro paso banda, conseguimos eliminar prácticamente toda la señal en banda base (audio), quedándonos solamente con la señal alrededor de la frecuencia característica, de 10 KHz, que es donde se encuentra la señal de datos.



3.5 DETECTOR DE ENVOLVENTE

El detector de envolvente se encarga de extraer las bandas laterales que contienen la información de baja frecuencia correspondiente a los bits. Está compuesto de un rectificador de onda y un filtro paso bajo.

3.5.1 RECTIFICADOR DE ONDA

Mediante el rectificado de la señal se obtienen las componentes de frecuencia en banda base y en los armónicos de la frecuencia principal. El elemento rectificador es un diodo, 1N4148, trabajando en la zona de conducción directa ($V > 0,6$ [V]), que tiene una buena respuesta a la frecuencia de 10 [kHz]



3.5.2 FILTRO PASO BAJO

Por otro lado, la función del filtro paso bajo es atenuar las componentes no deseadas (portadora) obteniendo una señal con flancos exponenciales y un rizado adicional, permitiendo el paso de los datos presentes en banda base (baja frecuencia) correspondientes a los bits.

Diseño del filtro paso bajo:

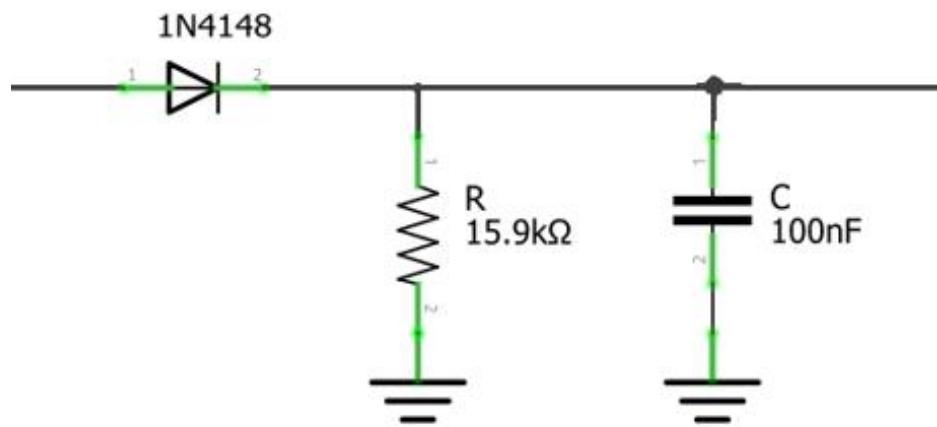
1. Escogemos la frecuencia de corte $f_c = 100$ [Hz], que permite atenuar la portadora de 10 [kHz] dejando la señal de bits (periodo de 100 [ms]) lo suficientemente inalteradas.
2. Elegimos los valores de R y C, para f_c igual a 100 [Hz]

$$f_c = \frac{1}{2 \cdot \pi \cdot R \cdot C}$$

Por ejemplo:

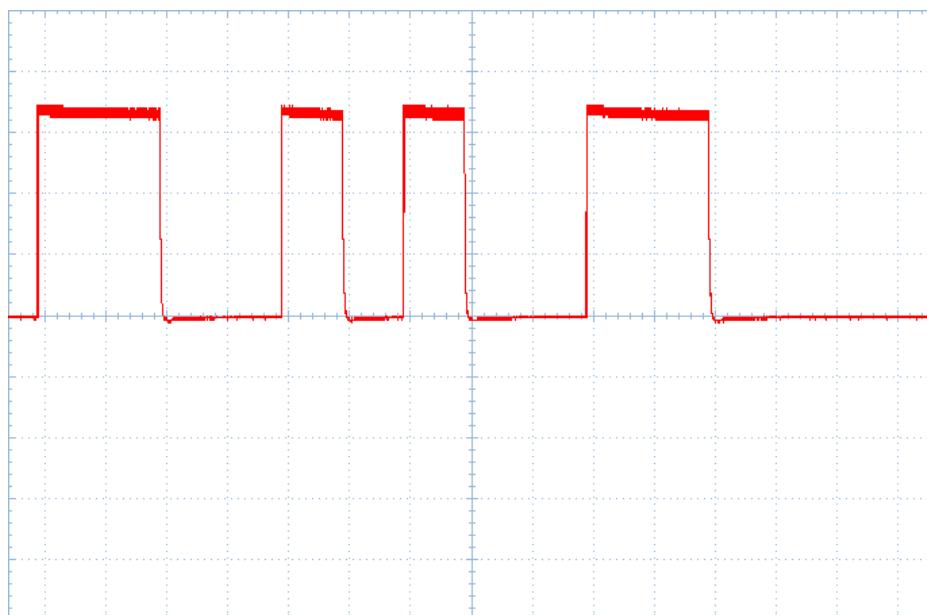
$$C = 100 \text{ [nF]}$$

$$R = 15915,49 \text{ [\Omega]}$$



3.5.3 SEÑAL DE SALIDA EN EL OSCILOSCOPIO

Al introducir la señal de audio en la entrada del circuito, a la salida del detector de envolvente obtenemos la siguiente señal en el osciloscopio (1 [V] y 100 [ms]):



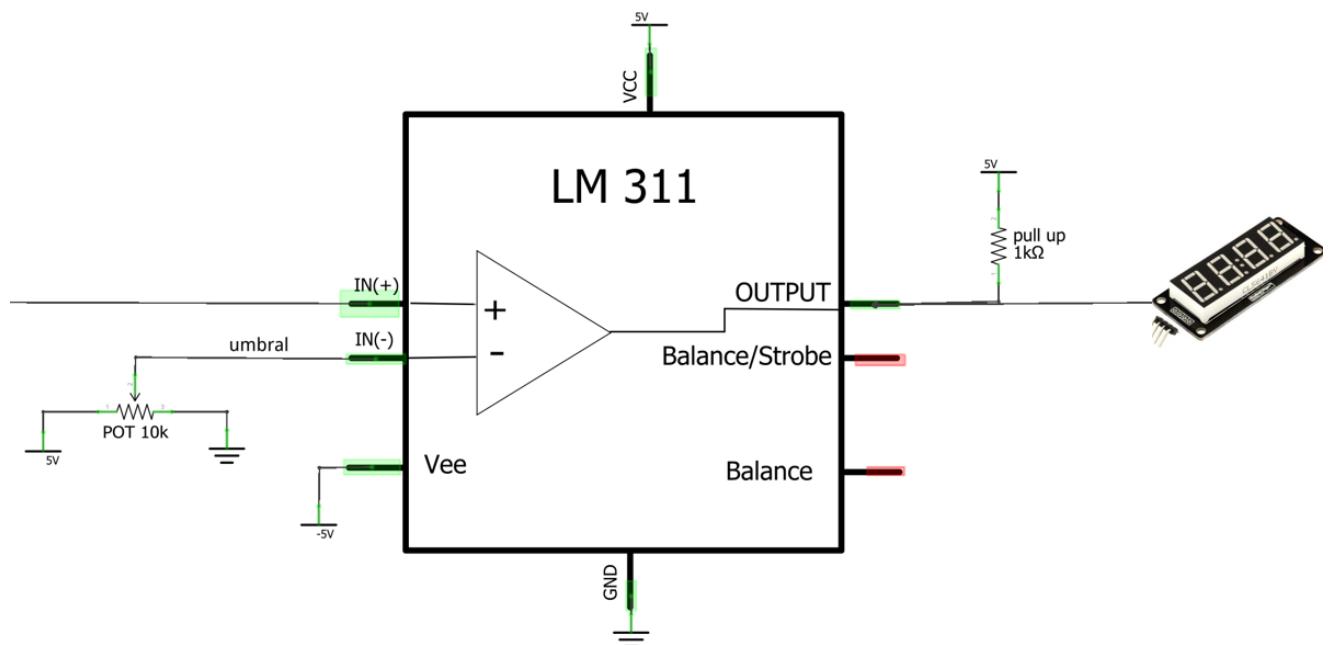
Observando la señal de salida del detector de envolvente podemos observar un pequeño rizado de 10 kHz en los bits a '1'

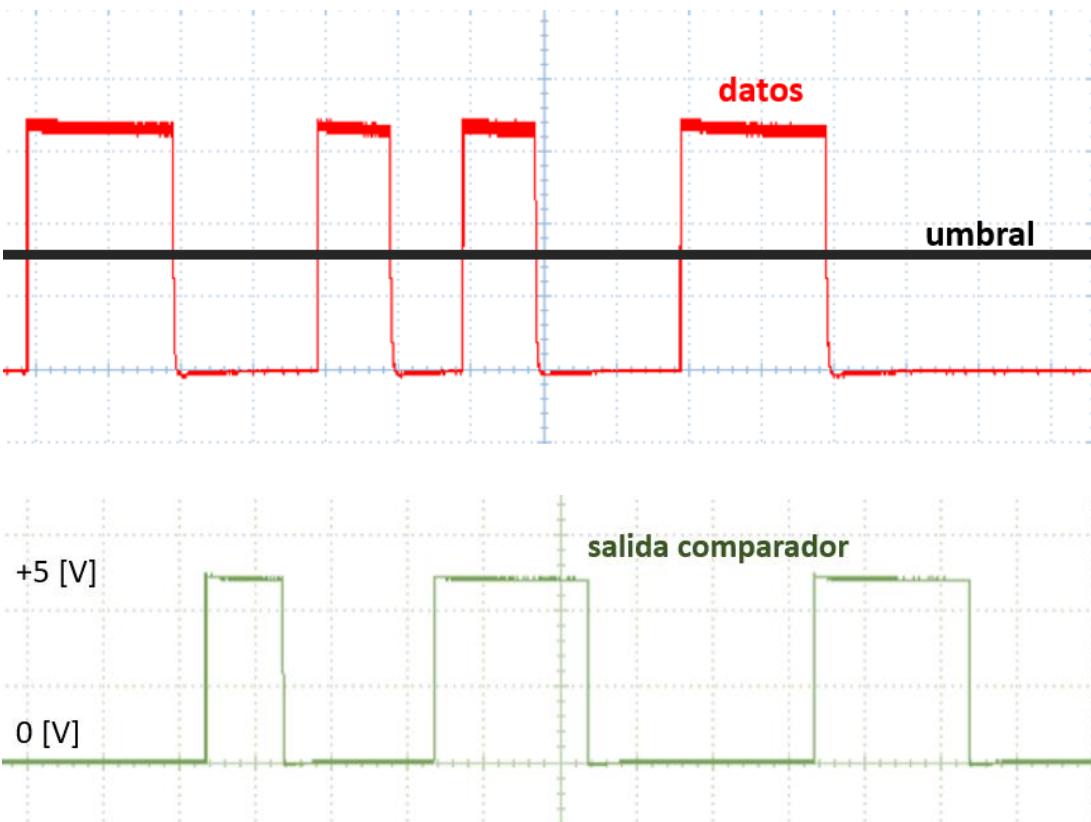
3.6 COMPARADOR

La última etapa del circuito analógico es un comparador, que permite obtener una señal digital. La señal procedente del detector de envolvente tiene flancos exponenciales y rizado. Por lo tanto, la señal no es apropiada para la decodificación de la señal RDS, ya que es necesaria una señal digital con flancos bien definidos entre 0 y 5 [V] para conectarla a una de las entradas de la FPGA.

Por consiguiente, vamos a emplear un módulo del tipo comparador **LM311**, este comparador actúa como un operacional en bucle abierto, donde la tensión que entrega el detector de envolvente se compara con un umbral. Cuando la tensión en V+ supera la tensión en V- entrega +Vcc a la salida, en caso contrario entrega 0 [V]. Finalmente, a la salida, tenemos una señal con flancos definidos y niveles de tensión estables entre 0 y 5 [V].

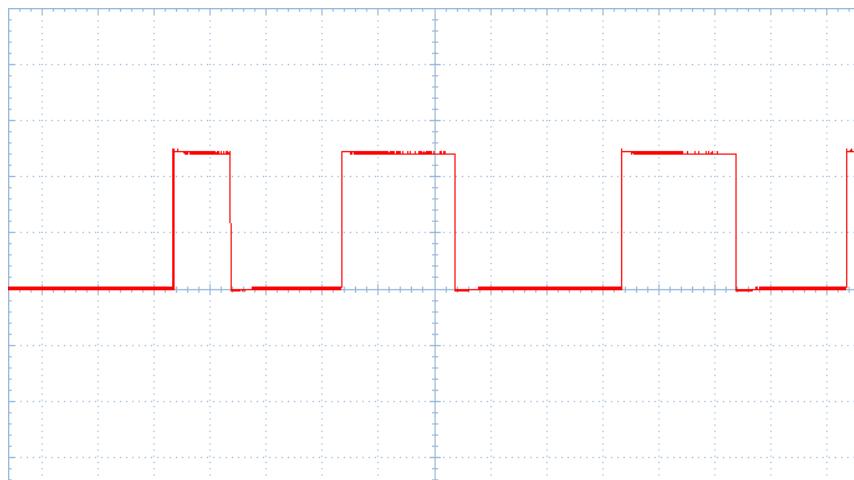
3.6.1 ESQUEMA DEL COMPARADOR





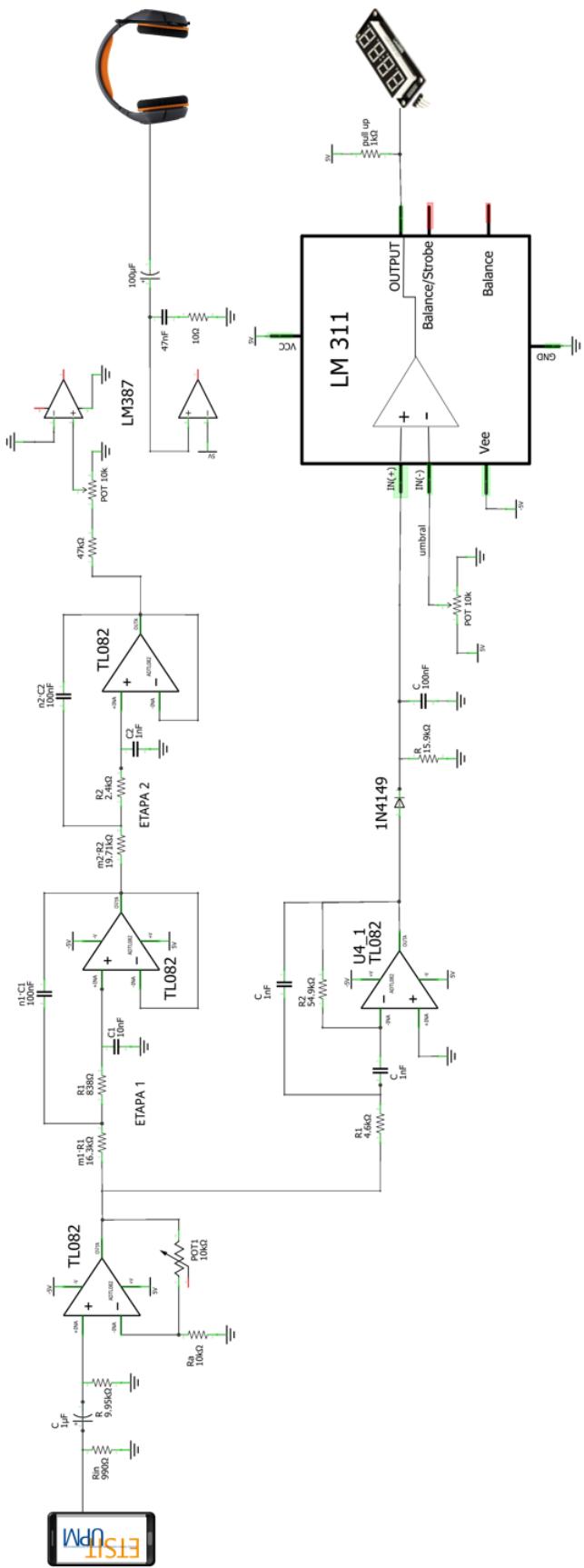
3.6.2 SEÑAL DE SALIDA EN EL OSCILOSCOPIO

Al introducir la señal de audio en la entrada del circuito, a la salida del comparador obtenemos la siguiente señal en el osciloscopio (2 [V] y 100 [ms]):



Finalmente podemos comprobar como la señal final, a la entrada del circuito digital posee flancos bien definidos entre 0 y 5 [V].

3.7 ESQUEMA ELÉCTRICO DEL CIRCUITO ANALÓGICO COMPLETO



3.8 DIFICULTADES PARTE ANALÓGICA

Durante el desarrollo de la parte analógica, hemos tenido que superar diversas dificultades. A continuación, se describen algunas de ellas:

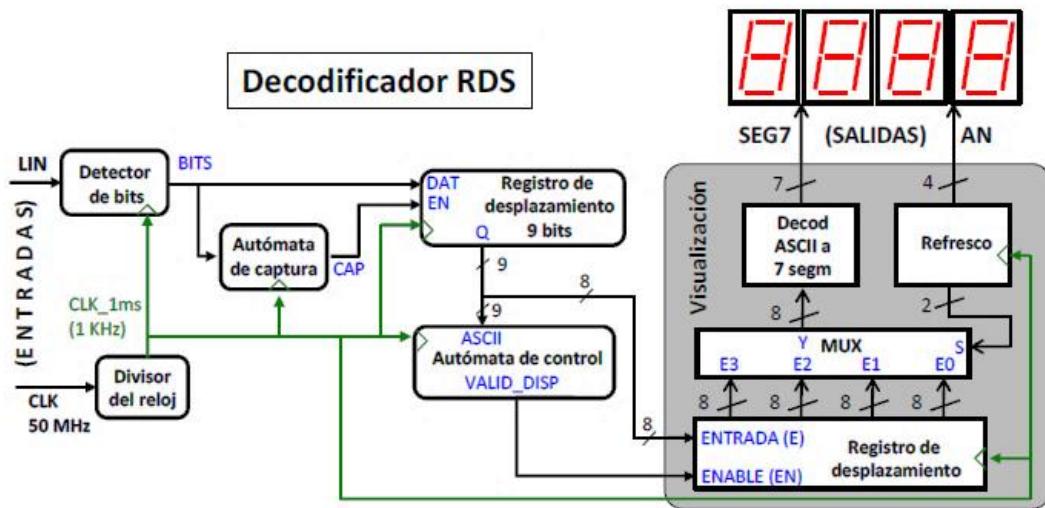
- 1) En la semana 1, tras conectar los condensadores de desacoplo y los leds en la placa, no se encendían los leds. A veces, se encendía solo el led rojo y otras veces solo el verde. Se realizaron las comprobaciones pertinentes con el multímetro. Finalmente, la solución fue acortar la longitud de las patillas de los condensadores de desacoplo, para que los leds funcionarán correctamente.
- 2) En el desarrollo del filtro paso bajo, cuando iniciamos las medidas en el osciloscopio, la frecuencia de corte del filtro no era la correcta (1,5 [kHz]). Nuevamente, comprobamos todos los cálculos, así como los componentes con el multímetro, y observamos que el valor de la resistencia $R_2 \cdot m_2$ era incorrecto.
- 3) En el módulo del amplificador, un condensador nos lo dieron mal en la tienda de “Publicaciones” cuando el valor real era de 1 [nF], cuando teóricamente era de 100 [nF]. Es decir, era incorrecto por dos órdenes de magnitud, y por tanto la ganancia no era la adecuada.
- 4) Al diseñar el filtro paso banda en la placa, cometimos un error, ya que situamos un condensador en cortocircuito. Es decir, el circuito no funcionaba correctamente, ya que teníamos un polo menos. Tras corregir dicho error, no tuvimos más dificultades en el resto de la parte analógica.



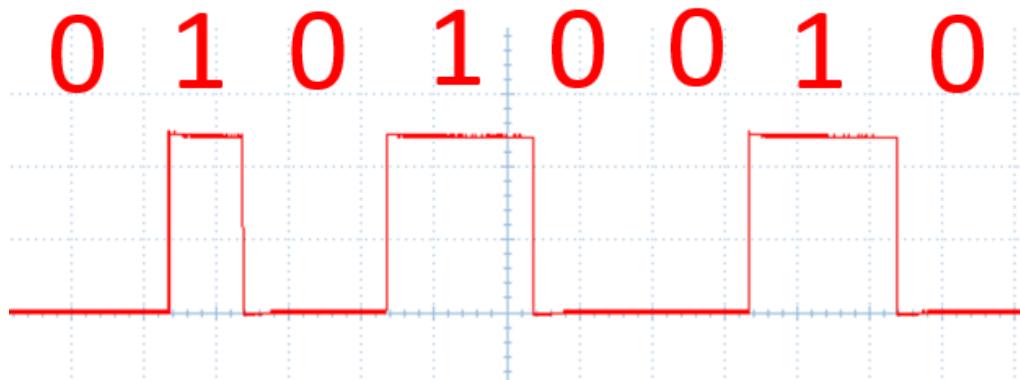
4. PARTE DIGITAL

En esta parte del proyecto, se desarrolla la estructura y el funcionamiento de un decodificador RDS, cuyo objetivo es tomar los bits demodulados de la señal RDS para después interpretar qué símbolos son y mostrar estos en el display de la FPGA.

La estructura de este decodificador es la siguiente:

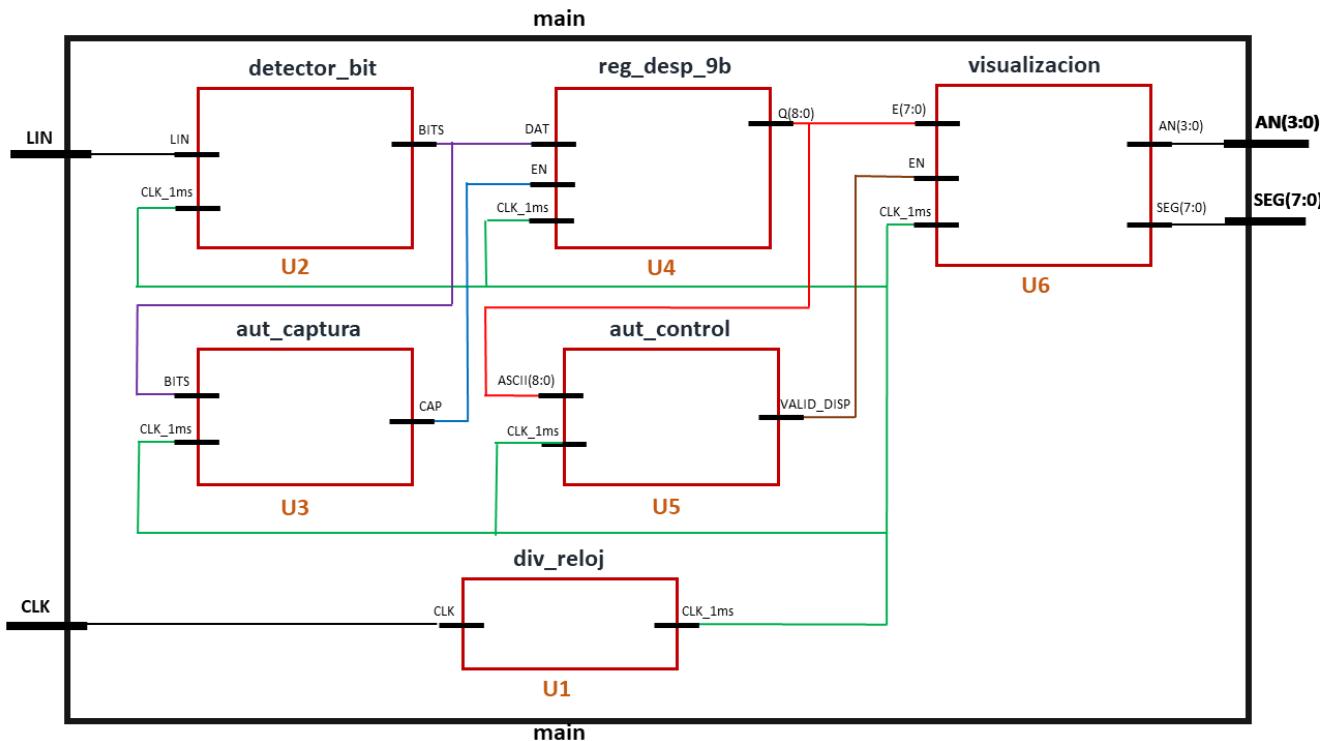


La entrada LIN, es la señal de salida del comparador, que posee valores entre 0 y +5 [V]:



La función de cada componente se describe a continuación:

- **Divisor del reloj:** Dado que la lógica interna del decodificador se sincroniza con un reloj de 1[kHz] de frecuencia (periodo 1 [ms]), y se parte del reloj de la FPGA de frecuencia 50 [MHz] (periodo 20 [ns]), hay que diseñar un divisor que obtenga de este último el primero.
- **Detector de bits:** Este módulo se encarga de detectar con precisión la llegada de un bit, analizando la energía de la señal a la entrada de LIN. Esta se comparará con dos umbrales, el superior determinará, en caso de ser sobrepasado, que ha llegado un “1”, y el inferior, en caso de ser este mayor que la energía, determinará que ha llegado un “0”.
- **Autómata de captura:** Se encarga de introducir el sincronismo de bit, produciendo pulsos de 1 [ms] en la salida CAP sincronizados con la llegada de bits cada 100 [ms].
- **Registro de desplazamiento de 9 bits:** Este registro se encarga de desplazar los bits de entrada (DAT) según los impulsos que recibe mediante CAP, concluyendo en un desplazamiento cada 100 [ms].
- **Autómata de control:** Este autómata se encarga de introducir el sincronismo de carácter. En primera instancia espera a la llegada del carácter SYNC (en binario 10000001) y tras esto activa la señal VALID_DISP en bucle cada 900ms, tiempo que tarda en llegar la cadena de bits que definen el carácter.
- **Visualización:** La función del bloque es tomar el código ASCII del carácter, desplazar los displays una posición hacia la izquierda e introducir el carácter en el display libre de la derecha.



4.1 DIVISOR DE RELOJ

El reloj de la FPGA tiene una frecuencia de 50 [MHz], y por lo tanto cada ciclo de reloj dura 20 [ns]. No obstante, para nuestro sistema digital buscamos un reloj que tenga una frecuencia de 1 [kHz], por ello debemos realizar la división de frecuencia mediante un contador y una señal que cambia de valor ('0' o '1') si alcanza un determinado valor.

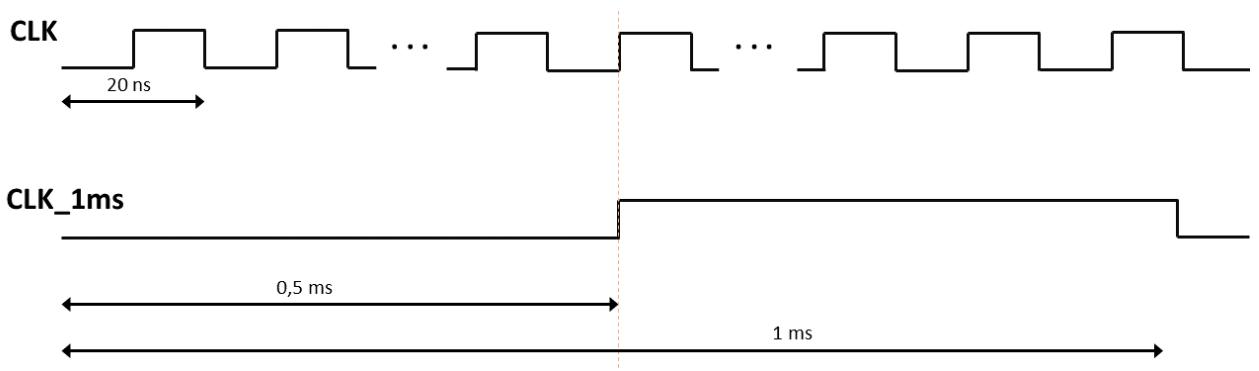


Dicho valor XXXXX se ha calculado con los siguientes razonamientos:

- El periodo de CLK es de 20 [ns], mientras que el periodo deseado de CLK_1ms es de 1 [ms], por tanto, el semiperíodo de CLK_1ms es 0,5 [ms]. Para que cambie de valor debe incrementarse el contador x veces, ya que:

$$XXXXX = \frac{0,5 \cdot 10^{-3} [s]}{20 \cdot 10^{-9} [s]} = 25000 \text{ ciclos de reloj}$$

- Cada 25000 flancos de subida de CLK, se cambia el valor de CLK_1ms, para que su frecuencia sea de 1 [kHz].



El código VHDL del divisor de reloj es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity div_reloj is
    Port ( CLK : in STD_LOGIC;          -- Entrada de reloj de la FPGA de 50 Mhz
            CLK_1ms : out STD_LOGIC);   -- Salida reloj a 1 KHz
end div_reloj;

architecture a_div_reloj of div_reloj is

signal contador : STD_LOGIC_VECTOR (15 downto 0):=(others=>'0'); --contador de 16 bits, inicializado a 0
signal freq_div : STD_LOGIC:='0'; --seal de frecuencia dividida,inicializado a 0

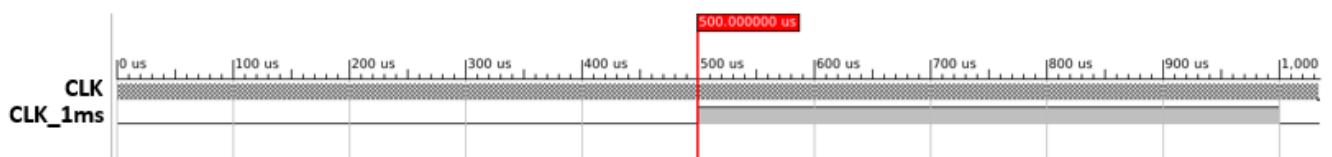
begin

process(CLK)
begin
    if (CLK'event and CLK='1') then --En cada flanco de subida de CLK
        contador<=contador + '1';  --Se incrementa el contador en 1
        if (contador>=25000) then --Si llega al valor de 25000 ciclos de reloj((50MHz/1KHz)/2)
            contador<=(others=>'0'); --poner el contador a 0
            freq_div<=not freq_div; --y conmutar el valor de freq_div
        end if;
    end if;
end process;

CLK_1ms<=freq_div;           -- freq_div esta cableada con la salida

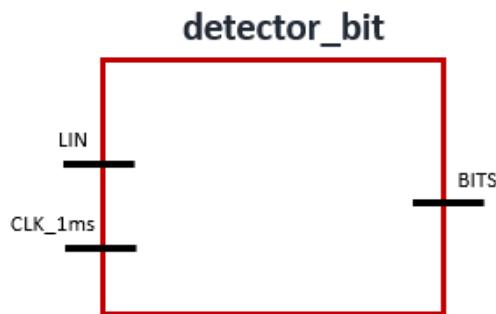
end a_div_reloj;
```

Podemos comprobar el funcionamiento del divisor de reloj mediante un TestBench o asociando la salida CLK_1ms a un terminal de salida de la FPGA y midiendo a través del osciloscopio la nueva frecuencia del decodificador RDS.

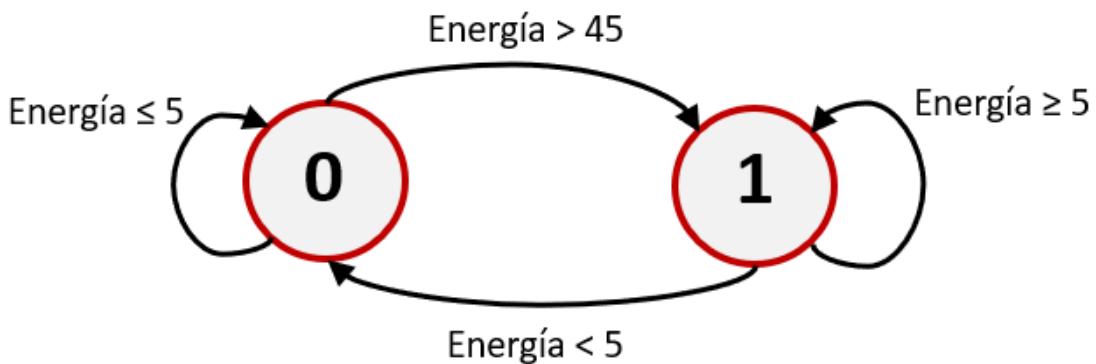


4.2 DETECTOR DE BITS

Este módulo es necesario para detectar correctamente los bits, puesto que la salida del circuito analógico puede contener errores que podrían causar un mal funcionamiento del circuito digital. Para ello, empleamos un registro de desplazamiento de 50 posiciones que muestreará con cada flanco de reloj (periodo 1 [ms]) la entrada *LIN*, añadiendo este bit por la derecha y desplazando los demás hacia la izquierda, en el caso del último bit, este se pierde al ser desplazado. Se define la variable *energía* como la cantidad de '1' que se encuentran en todo momento en el registro de desplazamiento. De igual manera se definen dos umbrales, el superior 45 y el inferior 5, con los cuales se comparará *energía* para determinar si el bit entrante es '1' o '0', evitando así el paso inmediato de posibles glitches. El funcionamiento es el siguiente:



Si el valor de *energía* supera el umbral superior (45), es decir, hay más de 45 '1' en el registro de desplazamiento, determina que la señal entrante es efectivamente un '1', por lo tanto, pone en la salida *BITS* un '1'. Hasta que el valor de *energía* no cae por debajo del umbral inferior (5), es decir, menos de 5 '1' en el registro, no se pone un '0' a la salida *BITS*. Este comportamiento se puede mostrar gráficamente en un diagrama de estados como el siguiente:



Cabe denotar que, por esta forma de comparación, la salida *BITS* está retrasada aproximadamente 45 [ms] respecto a la entrada *LIN*. Esto se puede ver en el caso en el que, si el registro de desplazamiento está lleno de '0', y entra un '1', hasta que la salida *BITS* transmite ese '1', deben entrar al registro de desplazamiento 45 '1', dado que *energía* debe superar el umbral superior. Al muestrearse la señal cada 1 [ms], se concluye que en este caso habría un retraso de 45 [ms].

El código VHDL del detector de bits es el siguiente:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity detector_bit is
    Port ( CLK_lms : in STD_LOGIC;      -- reloj
            LIN    : in STD_LOGIC;      -- Lnea de datos
            BITS  : out STD_LOGIC);   -- Bits detectados
end detector_bit;

architecture a_detector_bit of detector_bit is

constant UMBRAL0 : STD_LOGIC_VECTOR (7 downto 0) := "00000101"; -- 5 umbral para el 0
constant UMBRAL1 : STD_LOGIC_VECTOR (7 downto 0) := "00101101"; -- 45 umbral para el 1

type estados is (E1, E0);
signal estado_actual: estados := E0;
signal estado_siguiente: estados := E0;

signal reg_desp : STD_LOGIC_VECTOR (49 downto 0):=(others=>'0'); --Registro de desplazamiento de 50 bits
signal energia : STD_LOGIC_VECTOR (7 downto 0) := "00000000"; --Energia, 8 bits iniciados en un primer momento a 0
signal s_bits   : STD_LOGIC:='0';

begin

process (CLK_lms)
begin

if (CLK_lms'event and CLK_lms='1') then -- En cada flanco de subida de CLK_lms

    -- Calcular la suma de los elementos del registro (energia)
    if (reg_desp(49)='1' and LIN='1') then      --si sale un 1 y entra un 1, la energia se mantiene igual
        energia<= energia;
    elsif (reg_desp(49)='1' and LIN='0') then    -- si sale un 1 y entra un 0, la energia se decrementa en uno
        energia<=energia-'1';
    elsif (reg_desp(49)='0' and LIN='1') then    -- si sale un 0 y entra un 1, la energia aumenta en uno
        energia<=energia + '1';
    elsif (reg_desp(49)='0' and LIN='0') then    -- si sale un 0 y entra un 0, la energia se mantiene igual
        energia<=energia;
    end if;

    --Tambien se podra hacer:
    -- energia<= energia + LIN - reg_desp(49);

    -- Desplazar los datos del registro capturando la nueva muestra que entra en el registro
    reg_desp(49 downto 1)<=reg_desp(48 downto 0); --Los 49 bits de la derecha, se desplazan 1 hacia la izquierda
    reg_desp(0)<= LIN; --A la entrada del registro de desplazamiento entra la seal LIN

    -- Ver si la energia supera los umbrales y asignar a s_bits el valor adecuado
    if (energia > UMBRAL1) then --Si la energia es mayor que 45, s_bits vale 1
        s_bits <= '1';
    elsif (energia < UMBRAL0) then --Si la energia es menor que 5, s_bits vale 0
        s_bits <= '0';
    end if;
    end if;
end process;

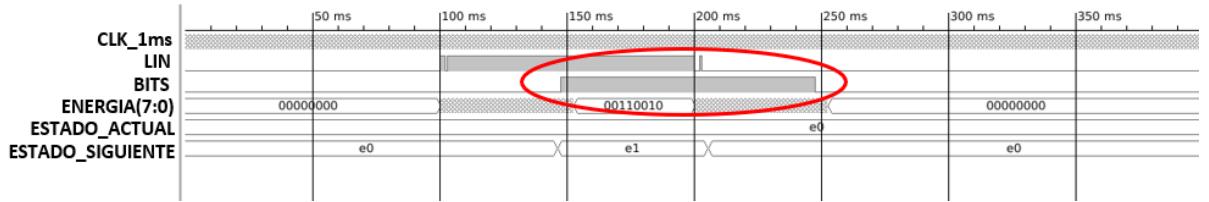
BITS<=s_bits; --Se asigna s_bits a la salida BITS del detector_bit

-- Asignacion de la salida en funcion del estado (Moore)
process (estado_actual, energia)
begin
    case estado_actual is
        when E0 => --Cuando se encuentra en el estado 0
            if energia <= UMBRAL1 then
                estado_siguiente <= E0; --Si la energia es menor o igual que el UMBRAL1 (45) se mantiene en el estado 0
            else
                estado_siguiente <= E1; --Si la energia es mayor que el UMBRAL1 (45) pasa al estado 1
            end if;
        when E1 => -- Cuando se encuentra en el estado 1
            if energia >= UMBRAL0 then
                estado_siguiente <= E1; --Si la energia es menor o igual que el UMBRAL0 (5) se mantiene en el estado 1
            else
                estado_siguiente <= E0; --Si la energia es mayor que el UMBRAL0 (5) pasa al estado 0
            end if;
    end case;
end process;

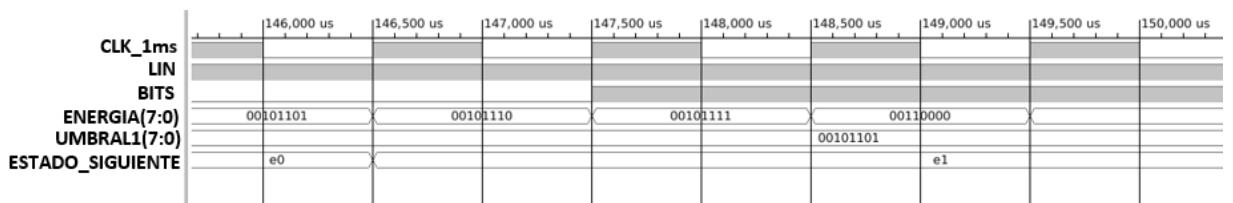
end a_detector_bit;

```

Tras analizar el funcionamiento de este módulo mediante su correspondiente TestBench, podemos observar un flanco de subida en la señal de BITS en 147,5 [ms]:



En la siguiente captura podemos ver, como cambia la señal de BITS cuando la energía supera el UMBRAL1, que es igual a 45 (00101101):



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_detector_bit IS
END tb_detector_bit;

ARCHITECTURE behavior OF tb_detector_bit IS

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

COMPONENT detector_bit
PORT (
    CLK_1ms : IN std_logic;
    LIN : IN std_logic;
    BITS : OUT std_logic
);
END COMPONENT;

--Inputs
signal CLK_1ms : std_logic := '0';
signal LIN : std_logic := '0';

--Outputs
signal BITS : std_logic;

-- Clock period definitions
constant CLK_1ms_period : time := 1 ms;

```

```

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: detector_bit PORT MAP (
        CLK_lms => CLK_lms,
        LIN => LIN,
        BITS => BITS
    );

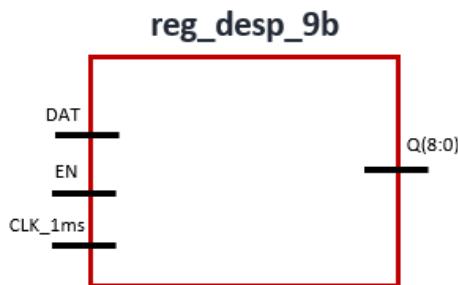
    -- Clock process definitions
    CLK_lms_process :process
    begin
        CLK_lms <= '0';
        wait for CLK_lms_period/2;
        CLK_lms <= '1';
        wait for CLK_lms_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        wait for 100 ms;
        LIN<='1';
        wait for 2 ms;
        LIN<='0';
        wait for 1 ms;
        LIN<='1';
        wait for 97 ms;
        LIN<='0';
        wait for 2 ms;
        LIN<='1';
        wait for 1 ms;
        LIN<='0';
        wait for 97 ms;
    end process;

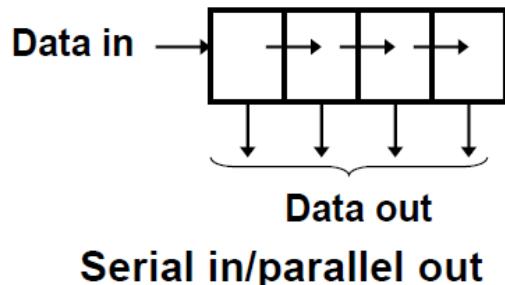
END;

```

4.3 REGISTRO DE DESPLAZAMIENTO DE 9 BITS



La función del registro de desplazamiento de 9 bits consiste en introducir el bit ‘DAT’ en el bit menos significativo (LSB) de Q que es una señal de 9 bits, y desplazar los 8 bits de la derecha un bit hacia la izquierda, cada ciclo de reloj (CLK_1ms), cuando la señal de ‘Enable’ (EN) esté activa.



Este registro de desplazamiento posee una entrada serie, que es la señal DAT, y una salida paralelo, de 9 bits, que es la señal Q.

El código VHDL del registro de desplazamiento de 9 bits, se muestra a continuación:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity reg_desp_9b is
    Port ( CLK_1ms : in STD_LOGIC; -- Reloj del sistema
           DAT : in STD_LOGIC;      -- Entrada de datos
           EN : in STD_LOGIC;       -- Entrada de ENABLE
           Q : out STD_LOGIC_VECTOR (8 downto 0)); --Salida paralelo
end reg_desp_9b;

architecture a_reg_desp_9b of reg_desp_9b is

signal s_Q : STD_LOGIC_VECTOR (8 downto 0):="0000000000";
begin

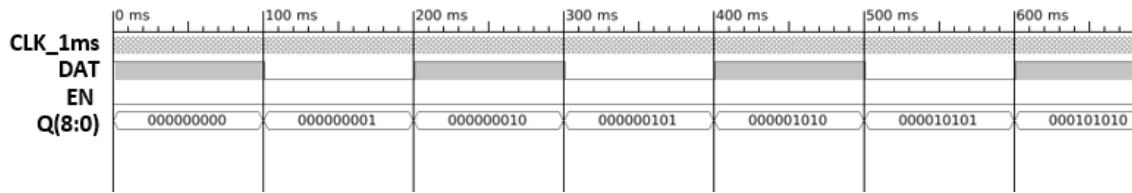
process (CLK_1ms)
begin
    if (EN = '1') then
        if (CLK_1ms'event and CLK_1ms='1') then    --En cada flanco de subida de CLK_1ms
            s_Q(8 downto 1)<=s_Q(7 downto 0); --movemos los 8 bits de la derecha, uno a la izquierda
            s_Q(0)<= DAT; -- se copia la señal de entrada DAT a la derecha de s_Q (bit menos significativo)
        end if;
    end if;
end process;

Q<=s_Q; -- La señal s_Q se cablea con la salida
end a_reg_desp_9b;

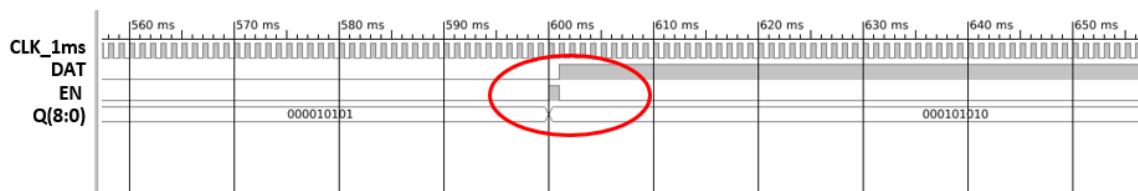
```

El resultado del Test-Bench del registro de desplazamiento de 9 bits se muestra a continuación:

- Se puede observar la entrada de bits al registro de 9 bits Q.



- También podemos ver, un pulso de EN cada 100 [ms] durante 1 periodo de reloj, donde se copia el bit, DAT, en Q



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tb_reg_desp_9b IS
END tb_reg_desp_9b;

ARCHITECTURE behavior OF tb_reg_desp_9b IS

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

COMPONENT reg_desp_9b
PORT (
    CLK_1ms : IN std_logic;
    DAT : IN std_logic;
    EN : IN std_logic;
    Q : OUT std_logic_vector(8 downto 0)
);
END COMPONENT;

--Inputs
signal CLK_1ms : std_logic := '0';
signal DAT : std_logic := '0';
signal EN : std_logic := '0';

--Outputs
signal Q : std_logic_vector(8 downto 0);

-- Clock period definitions
constant CLK_1ms_period : time := 1 ms;

```

```

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: reg_desp_9b PORT MAP (
        CLK_1ms => CLK_1ms,
        DAT => DAT,
        EN => EN,
        Q => Q
    );

    -- Clock process definitions
    CLK_1ms_process :process
    begin
        CLK_1ms <= '1';
        wait for CLK_1ms_period/2;
        CLK_1ms <= '0';
        wait for CLK_1ms_period/2;
    end process;

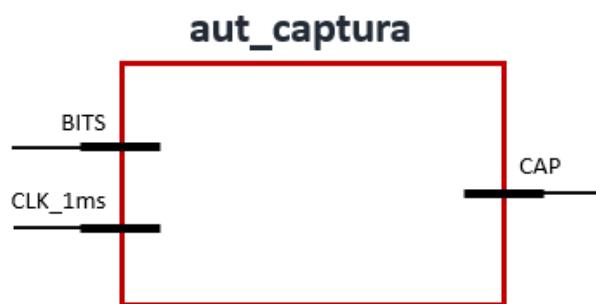
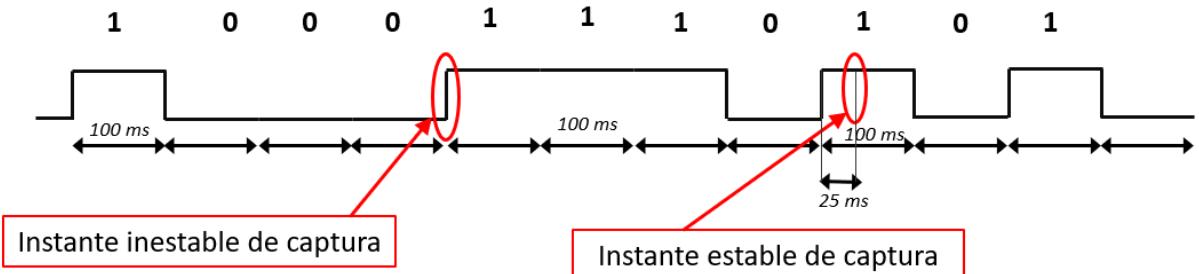
    -- Stimulus process
    stim_proc: process
    begin
        wait for 1 ms;
        DAT<='1';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='0';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='1';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='0';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='1';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='1';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='0';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='1';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='0';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='1';
        wait for 99 ms;
        EN<='1';
        wait for 1 ms;
        EN<='0';
        DAT<='0';
        wait for 99 ms;
    end process;

```

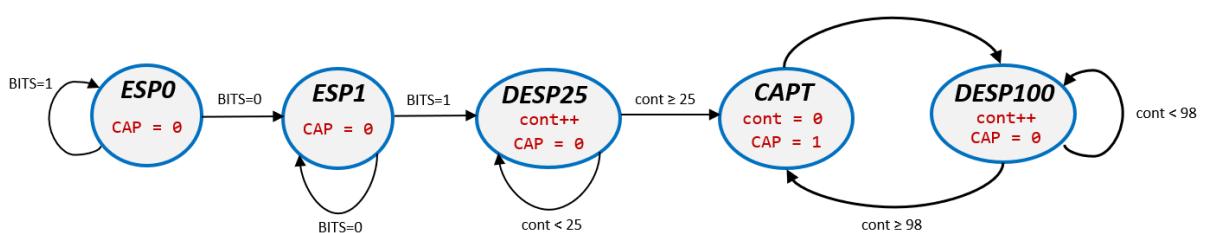
END;

4.4 AUTÓMATA DE CAPTURA

La función del autómata de captura es realizar el sincronismo de bit, capturando los bits en un punto donde son estables, ese punto corresponde con 25 ms después del flanco, es decir, un cuarto de la duración de un bit (100 [ms]). Por tanto, activa una señal de duración un ciclo de reloj (1 [ms]) cada vez que llega un nuevo bit, es decir, cada 100 [ms].



Este sincronismo, se puede realizar mediante un autómata de Moore con 5 diferentes estados: ESP0, ESP1, DESP25, CAPT y DESP100. Inicialmente, el autómata de captura debe esperar a que se reciba un '0' seguido de un '1' (flanco de subida en la señal de entrada, BITS), después de esta operación espera durante 25 ciclos de reloj (25 [ms]) para realizar la captura en un punto estable, y finalmente entra en un bucle donde se activa la señal de salida CAP cada 100 ciclos de reloj (100 [ms]), es decir, la duración de un bit.



El código VHDL del autómata de captura se muestra a continuación:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity aut_captura is
    Port ( CLK_1ms : in STD_LOGIC; -- Reloj del sistema
           BITS : in STD_LOGIC; -- Bits de entrada
           CAP : out STD_LOGIC); -- Señal para el registro de captura
end aut_captura;

architecture a_aut_captura of aut_captura is
type STATE_TYPE is (ESP0,ESP1,DESP25,CAPT,DESP100);

signal ST : STATE_TYPE := ESP0;
signal cont : STD_LOGIC_VECTOR (7 downto 0):="00000000";

begin
begin
--automata

process (CLK_1ms)
begin
    if CLK_1ms'event and CLK_1ms='1' then
        case ST is
            when ESP0 => --Espera por un bit a '0', si lo recibe pasa a ESP1
                if BITS='1' then
                    ST<=ESP0;
                else
                    ST<=ESP1;
                end if;

            when ESP1 => --Espera por un bit a '1', si lo recibe pasa a DESP25
                if BITS='1' then
                    ST<=DESP25;
                else
                    ST<=ESP1;
                end if;

            when DESP25 => --Desplazamiento de 25 ms
                cont<=cont+1; --Incrementa un contador
                if cont>=25 then -- cuando el contador llega a 25 pasa a CAPT
                    ST<=CAPT;
                else
                    ST<=DESP25; --Se repite el estado hasta que el contador llega a 25
                end if;

            when CAPT =>
                cont<="00000000"; --Pone el contador a 0
                ST<=DESP100; -- Pasa siempre a DESP100

            when DESP100 => --Desplazamiento de 100 ms
                cont<=cont+1; --Incrementa un contador
                if cont>=98 then --Espera 98 ms (98 ciclos de reloj, porque los desplazamientos hacia y desde el estado CAPT implican
                    ST<=CAPT; --2 ms, alcanzando los 100 ms que son necesarios) utilizando el contador
                else
                    ST<=DESP100;
                end if;

        end case;
    end if;
end process;

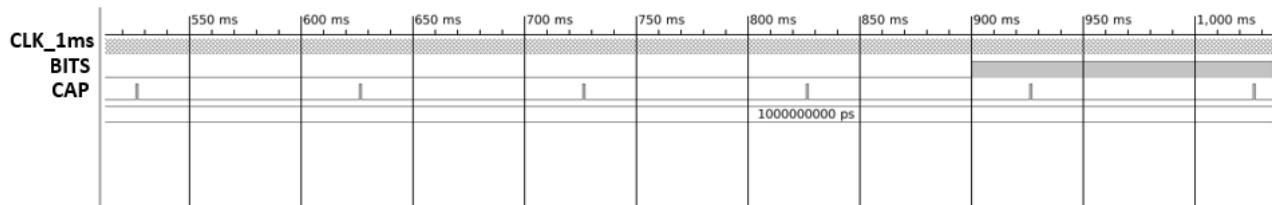
--asignacion de salida
with ST select CAP <=
    '1' when CAPT, --Cuando se encuentra en el estado CAPT, activa la señal CAP
    '0' when others; --En los demás estados CAP=0

end a_aut_captura;

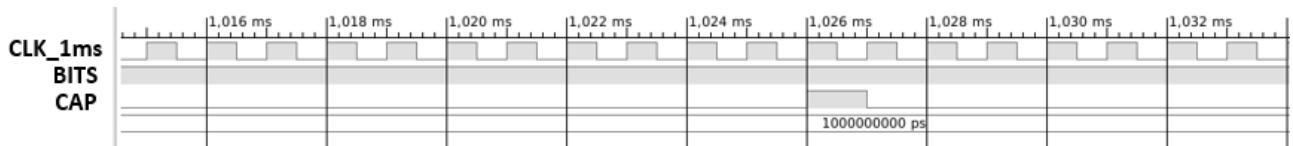
```

En el TestBench del autómata de captura podemos visualizar:

- La activación de la señal CAP, cada 100 ms.



- Cada pulso de la señal CAP, tiene una duración de un ciclo de reloj.



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tb_aut_captura IS
END tb_aut_captura;

ARCHITECTURE behavior OF tb_aut_captura IS

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

COMPONENT aut_captura
PORT (
    CLK_1ms : IN std_logic;
    BITS : IN std_logic;
    CAP : OUT std_logic
);
END COMPONENT;

--Inputs
signal CLK_1ms : std_logic := '0';
signal BITS : std_logic := '0';

--Outputs
signal CAP : std_logic;

-- Clock period definitions
constant CLK_1ms_period : time := 1 ms;

```

```

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: aut_captura PORT MAP (
    CLK_1ms => CLK_1ms,
    BITS => BITS,
    CAP => CAP
);

-- Clock process definitions
CLK_1ms_process :process
begin
    CLK_1ms <= '1';
    wait for CLK_1ms_period/2;
    CLK_1ms <= '0';
    wait for CLK_1ms_period/2;
end process;

-- Stimulus process
stim_proc: process
begin
    BITS<='0';
    wait for 100 ms;
    BITS<='1';          -- comienza SYNC
    wait for 100 ms;
    BITS<='0';
    wait for 100 ms;
    BITS<='1';
    wait for 100 ms;

    BITS<='1';          -- comienza A
    wait for 100 ms;
    BITS<='0';
    wait for 100 ms;
    BITS<='1';
    wait for 100 ms;
    BITS<='0';
    wait for 100 ms;
    BITS<='1';
    wait for 100 ms;

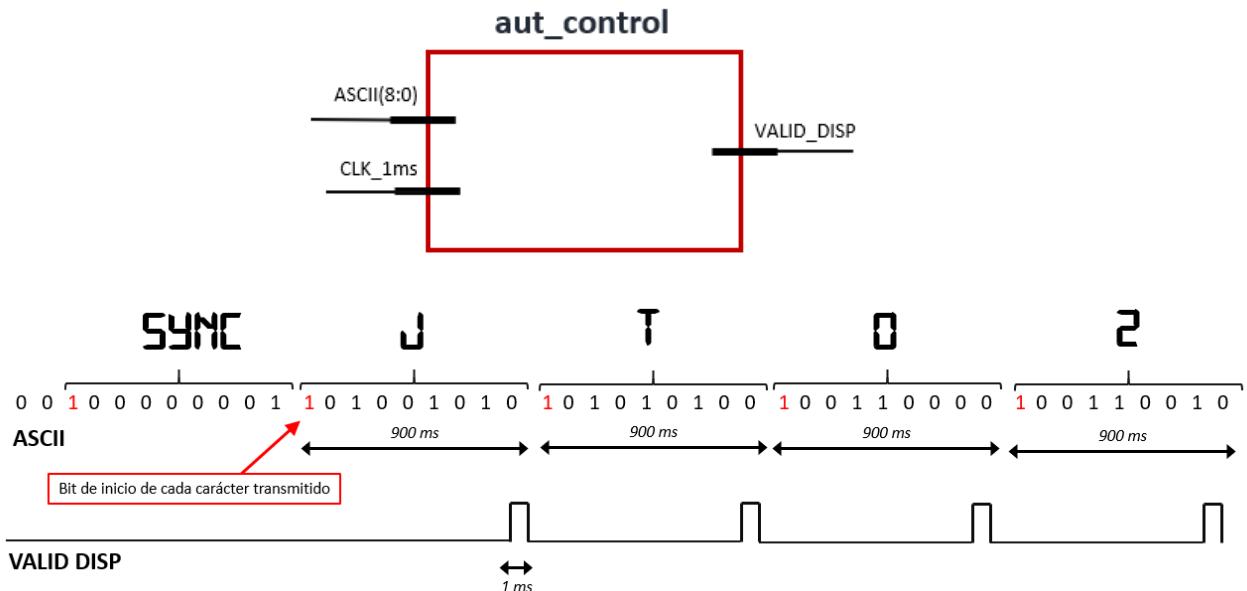
    wait;
end process;

```

END;

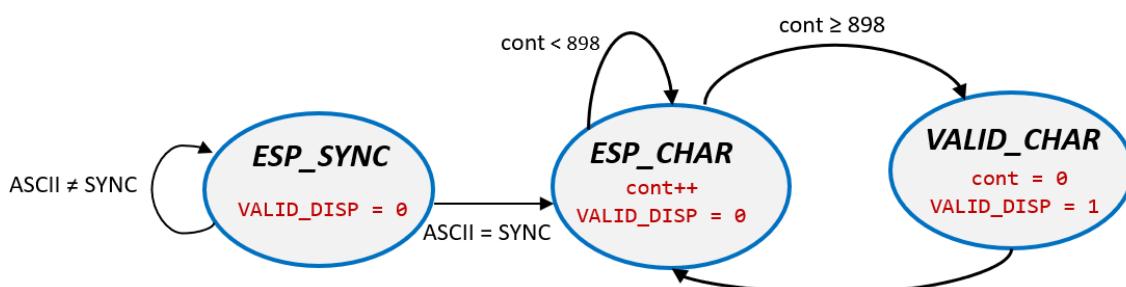
4.5 AUTÓMATA DE CONTROL

La función del autómata de control es proporcionar el sincronismo de carácter. Este sincronismo identifica donde empieza la secuencia de caracteres para capturarlos en el instante adecuado. Para ello, espera por unos bits de sincronismo, "100000001", y a partir de ese momento, activa una señal durante un ciclo de reloj (1 [ms]), VALID_DISP, cada 900 ms (que es la duración de un carácter ASCII de 8 bits, más el bit de inicio, MSB, a 1), lo que permite la visualización del carácter en el display correspondiente.



En la siguiente figura, se puede observar el autómata de Moore, donde la señal solo depende del estado, que se ha utilizado para llevar a cabo este sincronismo de carácter:

- En el estado inicial, **ESP_SYNC**, se espera a que se reciba los bits de sincronización SYNC, en el momento que se reciben pasamos al siguiente estado, llamado **ESP_CHAR**.
- En **ESP_CHAR**, vamos aumentando un contador, mientras que se reciben los 9 bits de cada carácter, es decir, se espera 898 [ms], hasta que el contador alcanza el valor de 898. En ese tiempo, VALID_DISP permanece desactivado.
- Cuando se alcanzan los 898 [ms], se pasa al estado **VALID_CHAR**, donde se reinicia el contador (para reactivar la espera a la llegada de un nuevo carácter) y se activa la señal VALID_DISP durante un ciclo de reloj, finalmente volviendo nuevamente al estado **ESP_CHAR**. Este proceso se repite de forma indefinida.



A continuación, se muestra el código VHDL que se ha diseñado, para realizar el autómata de control:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity aut_control is
    Port ( CLK_1ms : in STD_LOGIC;                      -- Reloj del sistema
           ASCII : in STD_LOGIC_VECTOR (8 downto 0);      -- Datos de entrada del registro
           VALID_DISP : out STD_LOGIC);                  -- Salida para validar el display
end aut_control;

architecture a_aut_control of aut_control is

constant SYNC : STD_LOGIC_VECTOR (8 downto 0) := "100000001"; -- carcter SYNC

type STATE_TYPE is (ESP_SYNC,ESP_CHAR,VALID_CHAR);
signal ST : STATE_TYPE:=ESP_SYNC;
signal cont : STD_LOGIC_VECTOR (15 downto 0):="0000000000000000";

begin
    process (CLK_1ms)
    begin
        if (CLK_1ms'event and CLK_1ms='1') then --En cada flanco de subida de CLK_1ms
            case ST is
                when ESP_SYNC => --Espera que la entrada (de 9 bits) sea igual al valor de SYNC (100000001)
                    if ASCII=SYNC then -- Esperar los bits sincronismo
                        ST<=ESP_CHAR; -- Si se reciben dichos bits, se pasa al estado ESP_CHAR
                    else
                        ST<=ESP_SYNC;
                    end if;

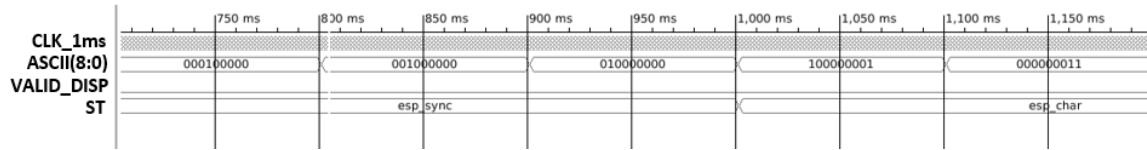
                when ESP_CHAR => --Espera 900 ms
                    cont<=cont+1; -- Incrementa un contador
                    if cont>=898 then --Repite el estado hasta que llega a 898
                        ST<=VALID_CHAR; --Cuando llega a 898 pasa a VALID_CHAR
                    else
                        ST<=ESP_CHAR;
                    end if;

                when VALID_CHAR => --Válida el carácter en el display
                    cont<="0000000000000000"; --Pone a 0 el contador
                    ST<=ESP_CHAR;          --Pasa al estado ESP_CHAR
            end case;
        end if;
    end process;

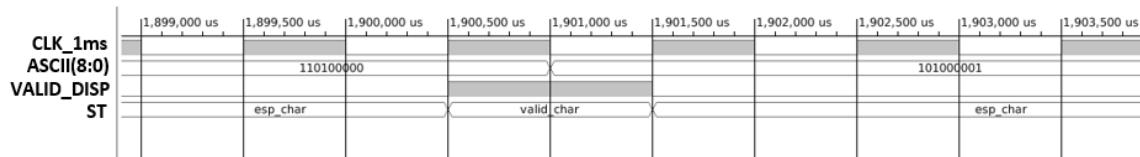
    with ST select VALID_DISP <= -- Asignación de la salida
        '1' when VALID_CHAR, --Activa la señal VALID_DISP en el estado VALID_CHAR
        '0' when others;     --Para el resto de estados VALID_DISP vale 0
end a_aut_control;
```

En el TestBench del autómata de control se puede observar:

- Se espera a la llegada de SYNC, y cuando llega dicha señal, se cambia al estado ESP_CHAR.



- También podemos visualizar el cambio de estado de ESP_CHAR a VALID_CHAR y vuelta, y activación VALID_DISP durante un ciclo de reloj al llegar todos los bits correspondientes a un carácter ASCII completo.



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tb_aut_control IS
END tb_aut_control;

ARCHITECTURE behavior OF tb_aut_control IS

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

COMPONENT aut_control
PORT (
    CLK_1ms : IN std_logic;
    ASCII : IN std_logic_vector(8 downto 0);
    VALID_DISP : OUT std_logic
);
END COMPONENT;

-- Inputs
signal CLK_1ms : std_logic := '0';
signal ASCII : std_logic_vector(8 downto 0) := (others => '0');

-- Outputs
signal VALID_DISP : std_logic;

-- Clock period definitions
constant CLK_1ms_period : time := 1 ms;

```

```

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: aut_control PORT MAP (
    CLK_1ms => CLK_1ms,
    ASCII => ASCII,
    VALID_DISP => VALID_DISP
);

-- Clock process definitions
CLK_1ms_process :process
begin
    CLK_1ms <= '0';
    wait for CLK_1ms_period/2;
    CLK_1ms <= '1';
    wait for CLK_1ms_period/2;
end process;

-- Stimulus process
stim_proc: process
begin
    wait for 201 ms;
    ASCII<="000000001";
    wait for 100 ms;
    ASCII<="000000010";
    wait for 100 ms;
    ASCII<="0000000100";
    wait for 100 ms;
    ASCII<="000000100";
    wait for 100 ms;
    ASCII<="000001000";
    wait for 100 ms;
    ASCII<="0000010000";
    wait for 100 ms;
    ASCII<="000100000";
    wait for 100 ms;
    ASCII<="001000000";
    wait for 100 ms;
    ASCII<="010000000";
    wait for 100 ms;
    ASCII<="100000001";
    wait for 100 ms;      -- SYNC
    ASCII<="000000011";
    wait for 100 ms;
    ASCII<="000000110";
    wait for 100 ms;
    ASCII<="000001101";
    wait for 100 ms;
    ASCII<="000011010";
    wait for 100 ms;
    ASCII<="000110100";
    wait for 100 ms;
    ASCII<="001101000";
    wait for 100 ms;
    ASCII<="011010000";
    wait for 100 ms;
    ASCII<="110100000";
    wait for 100 ms;
    ASCII<="101000001";  -- A
    wait for 100 ms;
    ASCII<="010000011";
    wait for 100 ms;
    ASCII<="100000110";
    wait for 100 ms;
    ASCII<="000001101";
    wait for 100 ms;
    ASCII<="000011010";
    wait for 100 ms;
    ASCII<="000110100";
    wait for 100 ms;
    ASCII<="001101000";
    wait for 100 ms;
    ASCII<="011010000";
    wait for 100 ms;
    ASCII<="110100001";
    wait for 100 ms;
    ASCII<="101000010";  -- B
    wait for 100 ms;

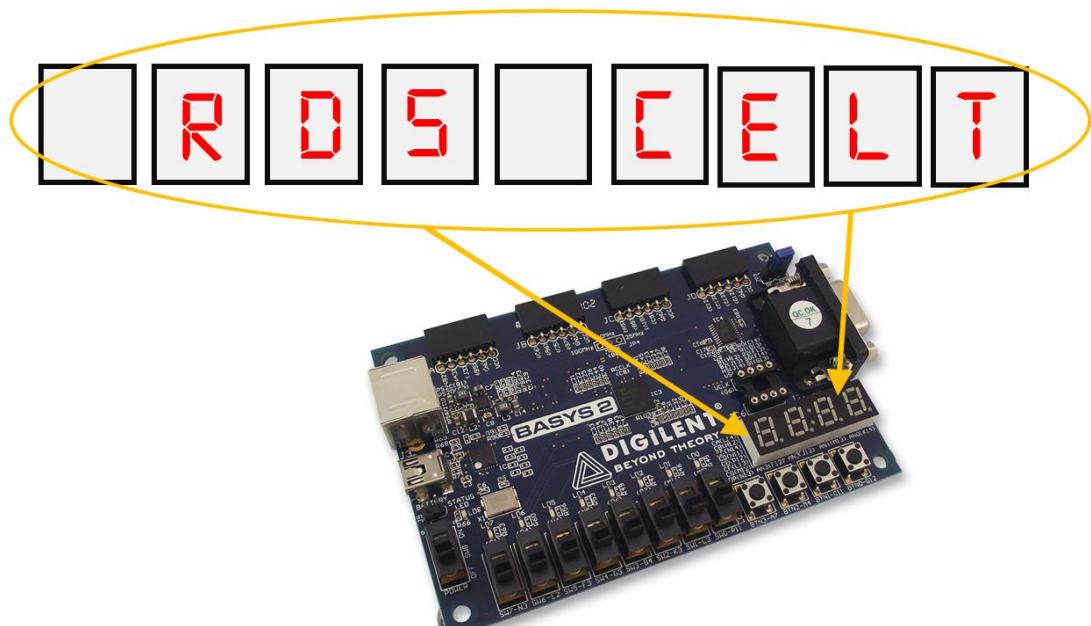
end process;

END;

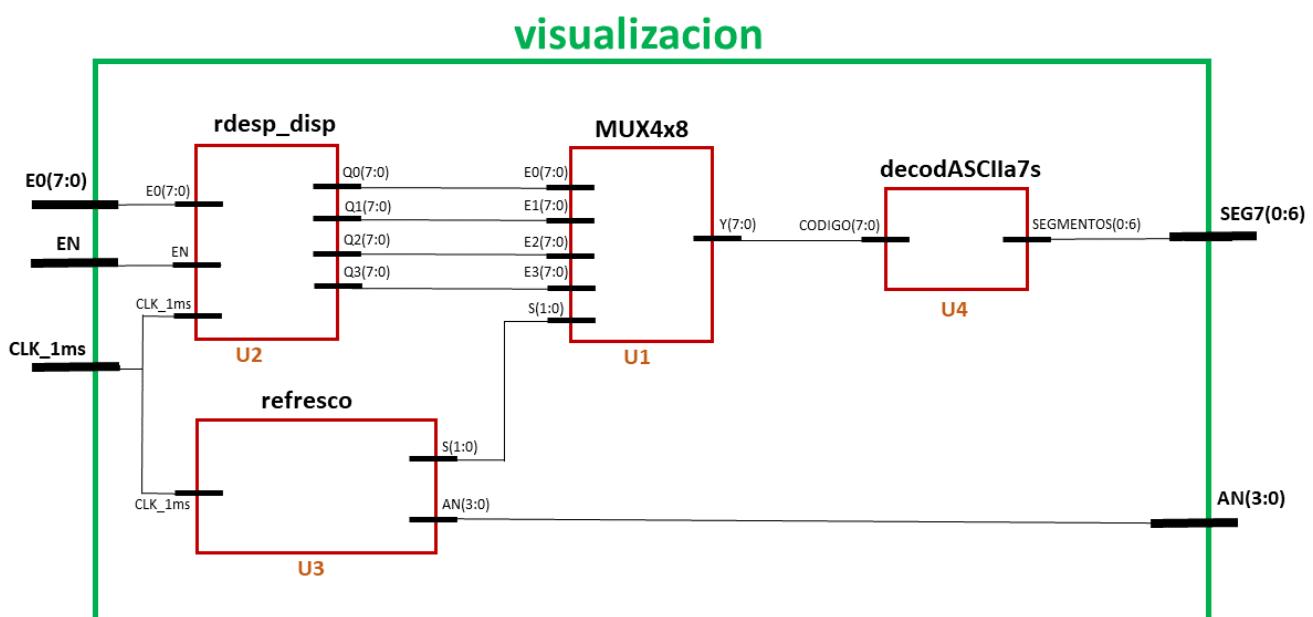
```

4.6 VISUALIZACIÓN

La principal función del bloque de visualización es la representación de una cadena de bits en los displays disponibles. Por tanto, toma el código ASCII del carácter, desplaza los displays hacia la izquierda una posición e introduce el carácter en el display libre de la derecha.

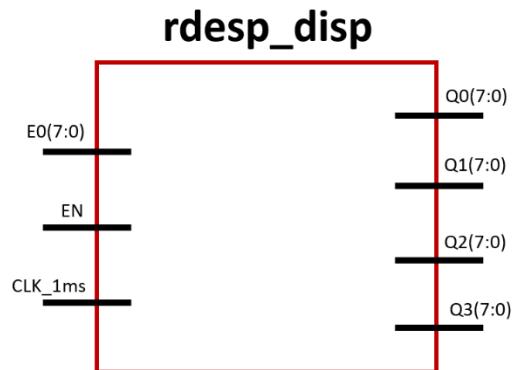


En la siguiente figura, podemos visualizar un diagrama lógico del módulo de visualización:

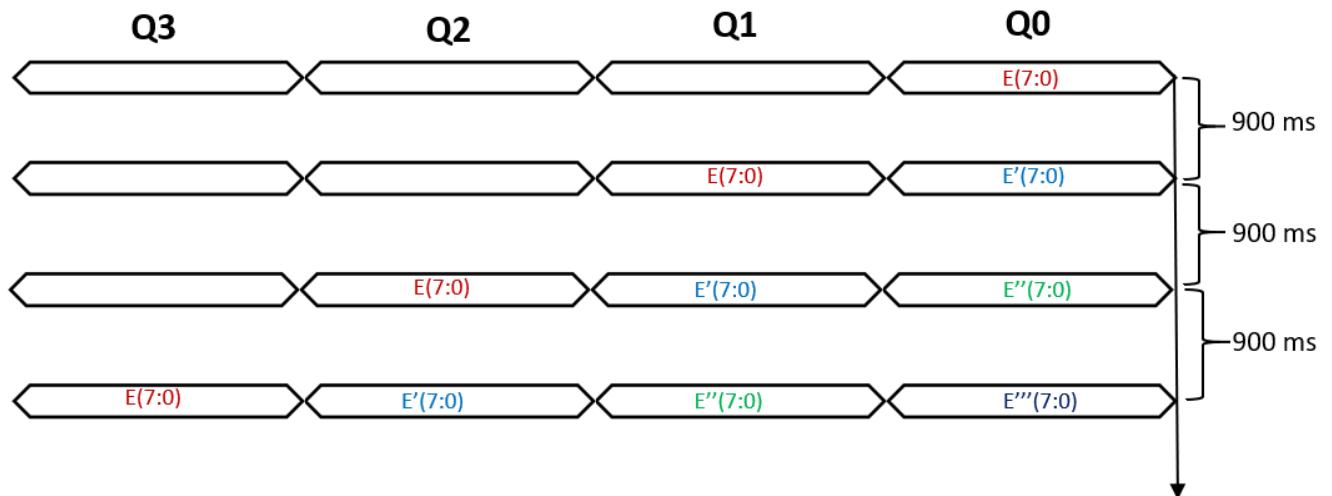


4.6.1 REGISTRO DE DESPLAZAMIENTO

Este módulo permite que los bits se desplacen **hacia la izquierda** cada vez que llega un nuevo carácter. Tiene 3 entradas, una señal de datos E, de 8 bits, una señal de reloj CLK_1ms y una señal de Enable llamada EN. Por otro lado, tiene 4 salidas, cada una de 8 bits, Q0, Q1, Q2 y Q3. Por tanto, la función del registro de desplazamiento es asignar una cadena de caracteres a la salida Q0, cada vez que se activa VALID_DISP en el autómata de control (cada 900 [ms]) y desplazar las demás salidas de 8 bits hacia la izquierda.



En este esquema, podemos observar cómo se introduce los 8 bits de la entrada en Q0 y se desplazan los demás hacia la izquierda, cada 900 [ms].



El código VHDL del módulo de registro de desplazamiento rdesp_disp es el siguiente:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD.TEXT.all;

entity rdesp_disp is
    Port ( CLK_1ms : in STD_LOGIC;           -- entrada de reloj
            EN      : in STD_LOGIC;           -- enable
            E       : in STD_LOGIC_VECTOR(7 downto 0);   -- entrada de datos
            Q0     : out STD_LOGIC_VECTOR(7 downto 0);  -- salida Q0
            Q1     : out STD_LOGIC_VECTOR(7 downto 0);  -- salida Q1
            Q2     : out STD_LOGIC_VECTOR(7 downto 0);  -- salida Q2
            Q3     : out STD_LOGIC_VECTOR(7 downto 0)); -- salida Q3
end rdesp_disp;

architecture a_rdesp_disp of rdesp_disp is

signal QS0 : STD_LOGIC_VECTOR(7 downto 0); -- señal que almacena el valor de Q0
signal QS1 : STD_LOGIC_VECTOR(7 downto 0); -- señal que almacena el valor de Q1
signal QS2 : STD_LOGIC_VECTOR(7 downto 0); -- señal que almacena el valor de Q2
signal QS3 : STD_LOGIC_VECTOR(7 downto 0); -- señal que almacena el valor de Q3

begin

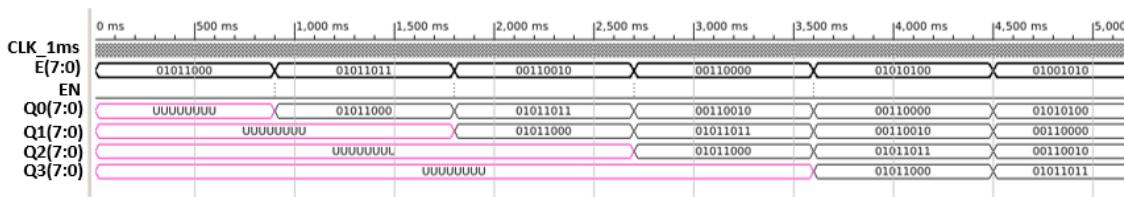
process (CLK_1ms)
begin
    if (EN='1') then -- Enable
        if (CLK_1ms'event and CLK_1ms='1') then -- En cada flanko de subida de CLK_1ms
            QS3<=QS2;
            QS2<=QS1;
            QS1<=QS0;
            QS0<=E;                                -- y se copia el valor de la entrada en Q0
        end if;
    end if;
end process;

QS0<=QS0;                                         -- actualización de las salidas
QS1<=QS1;
QS2<=QS2;
QS3<=QS3;
end a_rdesp_disp;

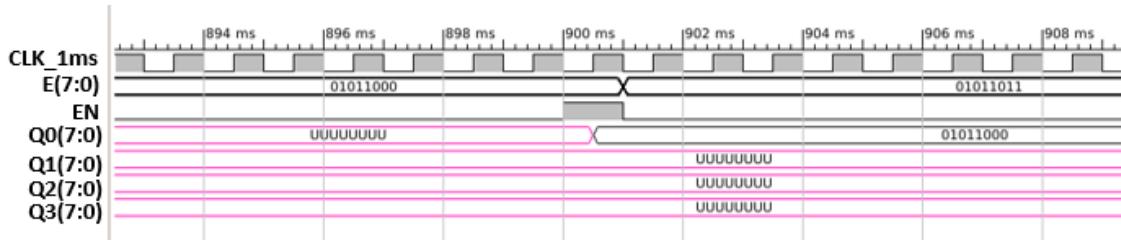
```

En el TestBench del registro de desplazamiento, podemos comprobar varias cosas:

- El desplazamiento de los 8 bits cada 900 ms, desde la entrada, hasta la salida Q3.



- Dicho desplazamiento, se produce únicamente cuando está activa la señal ENABLE (1 ciclo de reloj) que se activa cuando ha pasado el tiempo suficiente para la llegada de un nuevo carácter, por el sincronismo de carácter.



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tb_rdesp_disp IS
END tb_rdesp_disp;

ARCHITECTURE behavior OF tb_rdesp_disp IS

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

COMPONENT rdesp_disp
PORT(
    CLK_lms : IN std_logic;
    EN : IN std_logic;
    E : IN std_logic_vector(7 downto 0);
    Q0 : OUT std_logic_vector(7 downto 0);
    Q1 : OUT std_logic_vector(7 downto 0);
    Q2 : OUT std_logic_vector(7 downto 0);
    Q3 : OUT std_logic_vector(7 downto 0)
);
END COMPONENT;

--Inputs
signal CLK_lms : std_logic := '0';
signal EN : std_logic := '0';
signal E : std_logic_vector(7 downto 0) := (others => '0');

--Outputs
signal Q0 : std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
signal Q2 : std_logic_vector(7 downto 0);
signal Q3 : std_logic_vector(7 downto 0);

-- Clock period definitions
constant CLK_lms_period : time := 1 ms;

BEGIN

-- Instantiate the Unit Under Test (UUT)
ut: rdesp_disp PORT MAP (
    CLK_lms => CLK_lms,
    EN => EN,
    E => E,
    Q0 => Q0,
    Q1 => Q1,
    Q2 => Q2,
    Q3 => Q3);

-- Clock process definitions
CLK_lms_process :process
begin
    CLK_lms <= '0';
    wait for CLK_lms_period/2;
    CLK_lms <= '1';
    wait for CLK_lms_period/2;
end process;

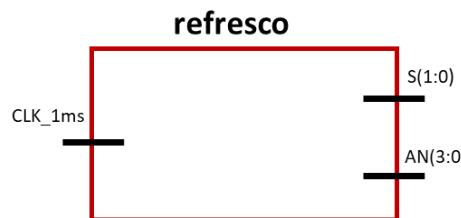
stim_proc: process
begin
    wait for 1 ms;
    E<="01011000";
    wait for 899 ms;
    EN<='1';
    wait for 1 ms;
    EN<='0';
    E<="01011011";
    wait for 899 ms;
    EN<='1';
    wait for 1 ms;
    EN<='0';
    E<="00110010";
    wait for 899 ms;
    EN<='1';
    wait for 899 ms;
    EN<='1';
    wait for 1 ms;
    EN<='0';
    E<="01010100";
    wait for 899 ms;
    EN<='1';
    wait for 1 ms;
    EN<='0';
    E<="01001010";
    wait for 899 ms;
    EN<='1';
    wait for 1 ms;
    EN<='0';
end process;

END;

```

4.6.2 REFRESCO

El bloque de REFRESCO se encarga de controlar la visualización de cada carácter mostrado por el display de forma sucesiva (cada 1 [ms]), para ello se encarga de generar una señal S de 2 bits, que controla las entradas del multiplexor (00, 01, 10 y 11) y otra señal AN, de 4 bits, que permite activar las diferentes entradas de cada display (activas a nivel bajo). De forma que si, por ejemplo, SS vale "00" se activa el primer display, ya que se asigna a AN "0111", y así sucesivamente, y permanecen activas 1 [ms], hasta que se activa otro display.



El código VHDL del módulo de refresco se muestra a continuación:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD.TEXT.all;

entity refresco is
    Port ( CLK_1ms : in STD_LOGIC;           -- reloj de refresco
            S : out STD_LOGIC_VECTOR (1 downto 0);   -- Control para el mux
            AN : out STD_LOGIC_VECTOR (3 downto 0)); -- Control displays individuales
end refresco;

architecture a_refresco of refresco is

signal SS : STD_LOGIC_VECTOR (1 downto 0);

begin

process (CLK_1ms)
begin
    if (CLK_1ms'event and CLK_1ms='1') then      -- En cada flanco de subida de CLK_1ms ( 1 KHz )
        SS<=SS+1;                                -- genera la secuencia 00,01,10 y 11
    end if;
end process;

S<=SS; --Entrada de selección del multiplexor

-- Alterna los datos de entrada del multiplexor con la activación del display correspondiente

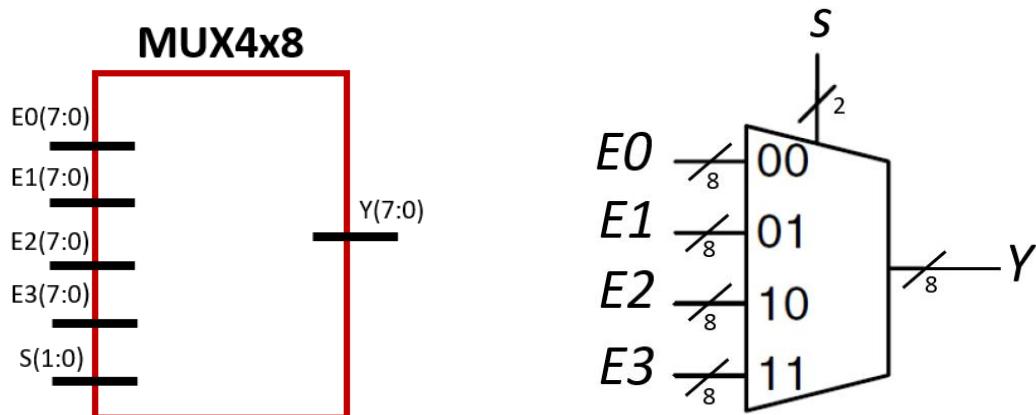
AN<="0111" when SS="00" else    --Entradas de activación de los displays(en función del valor de SS)
"1011" when SS="01" else
"1101" when SS="10" else
"1110" when SS="11";

end a_refresco;

```

4.6.3 MULTIPLEXOR

Mediante el multiplexor, podemos visualizar los caracteres en los diferentes displays. Para llevar a cabo este proceso, y poder visualizar los diferentes caracteres correctamente es necesario presentar cada carácter en cada display muy rápidamente, de manera que el ojo no pueda apreciar el parpadeo. Por lo tanto, necesitamos un multiplexor de 4 entradas de datos de 8 bits (E_0, E_1, E_2 y E_3), controlado por otra entrada de selección de datos S de 2 bits.



El código VHDL del multiplexor MUX4x8 se muestra a continuación:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

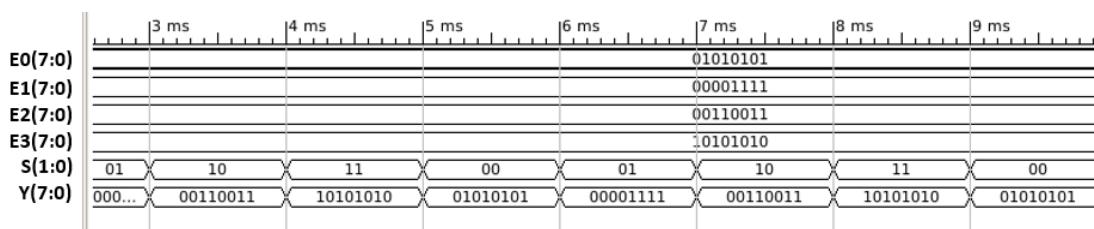
entity MUX4x8 is
    Port ( E0 : in STD_LOGIC_VECTOR (7 downto 0); -- Entrada de datos 0
           E1 : in STD_LOGIC_VECTOR (7 downto 0); -- Entrada de datos 1
           E2 : in STD_LOGIC_VECTOR (7 downto 0); -- Entrada de datos 2
           E3 : in STD_LOGIC_VECTOR (7 downto 0); -- Entrada de datos 3
           S : in STD_LOGIC_VECTOR (1 downto 0); -- Seal de control
           Y : out STD_LOGIC_VECTOR (7 downto 0)); -- Salida
end MUX4x8;

architecture a_MUX4x8 of MUX4x8 is

begin
    -- se selecciona la salida en función de las entradas de control
    Y <= E0 when S="00" else --display 0
        E1 when S="01" else --display 1
        E2 when S="10" else --display 2
        E3 when S="11";      --display 3
end a_MUX4x8;

```

El resultado del TestBench se muestra en la siguiente captura:



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_MUX4x8 IS
END tb_MUX4x8;

ARCHITECTURE behavior OF tb_MUX4x8 IS

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

    COMPONENT MUX4x8
    PORT(
        E0 : IN std_logic_vector(7 downto 0);
        E1 : IN std_logic_vector(7 downto 0);
        E2 : IN std_logic_vector(7 downto 0);
        E3 : IN std_logic_vector(7 downto 0);
        S : IN std_logic_vector(1 downto 0);
        Y : OUT std_logic_vector(7 downto 0)
    );
    END COMPONENT;

    --Inputs
    signal E0 : std_logic_vector(7 downto 0) := (others => '0');
    signal E1 : std_logic_vector(7 downto 0) := (others => '0');
    signal E2 : std_logic_vector(7 downto 0) := (others => '0');
    signal E3 : std_logic_vector(7 downto 0) := (others => '0');
    signal S : std_logic_vector(1 downto 0) := (others => '0');

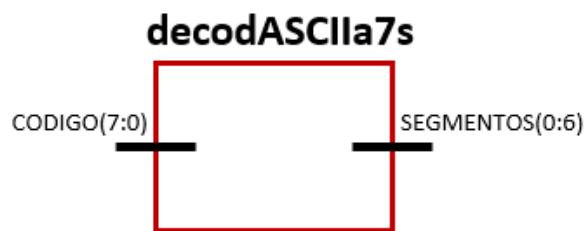
BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: MUX4x8 PORT MAP (
        E0 => E0,
        E1 => E1,
        E2 => E2,
        E3 => E3,
        S => S,
        Y => Y);
    E0<="01010101";
    E1<="00001111";
    E2<="00110011";
    E3<="10101010";

    S<="00" after 1 ms,
    "01" after 2 ms,
    "10" after 3 ms,
    "11" after 4 ms,
    "00" after 5 ms,
    "01" after 6 ms,
    "10" after 7 ms,
    "11" after 8 ms,
    "00" after 9 ms,
    "01" after 10 ms,
    "10" after 11 ms,
    "11" after 12 ms;
END;

```

4.6.4 DESCODIFICADOR ASCII DE 7 SEGMENTOS

Para mostrar el carácter correspondiente en el display, se dispone de un descodificador de ASCII 7 segmentos, que asocia los 8 bits de la señal CODIGO con un vector de 7 bits, SEGMENTOS, que indica qué segmento del display debe encenderse. En este proyecto, únicamente se han utilizado los caracteres correspondientes al espacio, las letras (de la ‘a’ hasta la ‘z’) y las cifras (del 0 al 9).



En la siguiente tabla se muestran los diferentes caracteres, con su código ASCII, así como en hexadecimal y en binario:

Cárcater	ASCII	Binario	Hexadecimal
SYNC	01	00000001	0x01
ESPACIO	32	10000000	0x80
A	65	01000001	0x41
B	66	01000010	0x42
C	67	01000011	0x43
D	68	01000100	0x44
E	69	01000101	0x45
F	70	01000110	0x46
G	71	01000111	0x47
H	72	01001000	0x48
I	73	01001001	0x49
J	74	01001010	0x4A
K	75	01001011	0x4B
L	76	01001100	0x4C
M	77	01001101	0x4D
N	78	01001110	0x4E
O	79	01001111	0x4F
P	80	01010000	0x50
Q	81	01010001	0x51

Cártacter	ASCII	Binario	Hexadecimal
R	82	01010010	0x52
S	83	01010011	0x53
T	84	01010100	0x54
U	85	01010101	0x55
V	86	01010110	0x56
W	87	01010111	0x57
X	88	01011000	0x58
Y	89	01011001	0x59
Z	90	01011010	0x5A
0	48	00110000	0x30
1	49	00110001	0x31
2	50	00110010	0x32
3	51	00110011	0x33
4	52	00110100	0x34
5	53	00110101	0x35
6	54	00110110	0x36
7	55	00110111	0x37
8	56	00111000	0x38
9	57	00111001	0x39

El código VHDL del descodificador ASCII de 7 segmentos, se muestra a continuación:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD.TEXT.all;

entity decodASCIIa7s is
    Port ( CODIGO : in STD_LOGIC_VECTOR (7 downto 0);      -- codigo ASCII
           SEGMENTOS : out STD_LOGIC_VECTOR (0 to 6));        -- Salidas al display
end decodASCIIa7s;

architecture a_decodASCIIa7s of decodASCIIa7s is

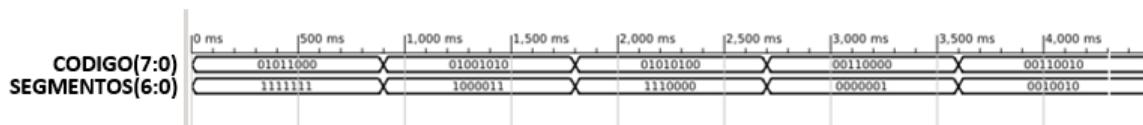
begin
    with CODIGO select SEGMENTOS<=
        -- Señales activas a nivel bajo

        --abcdefg          ASCII
        -----
        "1111111" when "00100000", -- ESPACIO
        "0001000" when "01000001", -- A
        "1100000" when "01000010", -- B
        "0110001" when "01000011", -- C
        "1000010" when "01000100", -- D      ---a---
        "0110000" when "01000101", -- E      |      |
        "0111000" when "01000110", -- F      f      b
        "0000100" when "01000111", -- G      |      |
        "1001000" when "01001000", -- H      ---g---
        "1101111" when "01001001", -- I      |      |
        "1000011" when "01001010", -- J      e      c
        "1111111" when "01001011", -- K      |      |
        "1110001" when "01001100", -- L      ---d---
        "1111111" when "01001101", -- M
        "1101010" when "01001110", -- N
        "1100010" when "01001111", -- O  Segmentos encendidos con '0'
        "0011000" when "01010000", -- P  Segmentos apagados con '1'
        "0001100" when "01010001", -- Q
        "1111010" when "01010010", -- R
        "0100100" when "01010011", -- S
        "1110000" when "01010100", -- T
        "1100011" when "01010101", -- U
        "1111111" when "01010110", -- V
        "1111111" when "01010111", -- W
        "1111111" when "01011000", -- X
        "1000100" when "01011001", -- Y
        "1111111" when "01011010", -- Z
        "1001111" when "00110001", -- 1
        "0010010" when "00110010", -- 2
        "0000110" when "00110011", -- 3
        "1001100" when "00110100", -- 4
        "0100100" when "00110101", -- 5
        "1100000" when "00110110", -- 6
        "0001111" when "00110111", -- 7
        "0000000" when "00111000", -- 8
        "0000010" when "00111001", -- 9
        "0000001" when "00110000", -- 0
        "1111111" when others;   -- apagado

end a_decodASCIIa7s;

```

Y el resultado de su TestBench correspondiente es:



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY tb_decodASCIIa7s IS
END tb_decodASCIIa7s;

ARCHITECTURE behavior OF tb_decodASCIIa7s IS

COMPONENT decodASCIIa7s
PORT(
    CODIGO : IN std_logic_vector(7 downto 0);
    SEGMENTOS : OUT std_logic_vector(0 to 6)
);
END COMPONENT;

--Inputs
signal CODIGO : std_logic_vector(7 downto 0) := (others => '0');

--Outputs
signal SEGMENTOS : std_logic_vector(0 to 6);

BEGIN

uut: decodASCIIa7s PORT MAP (
    CODIGO => CODIGO,
    SEGMENTOS => SEGMENTOS
);

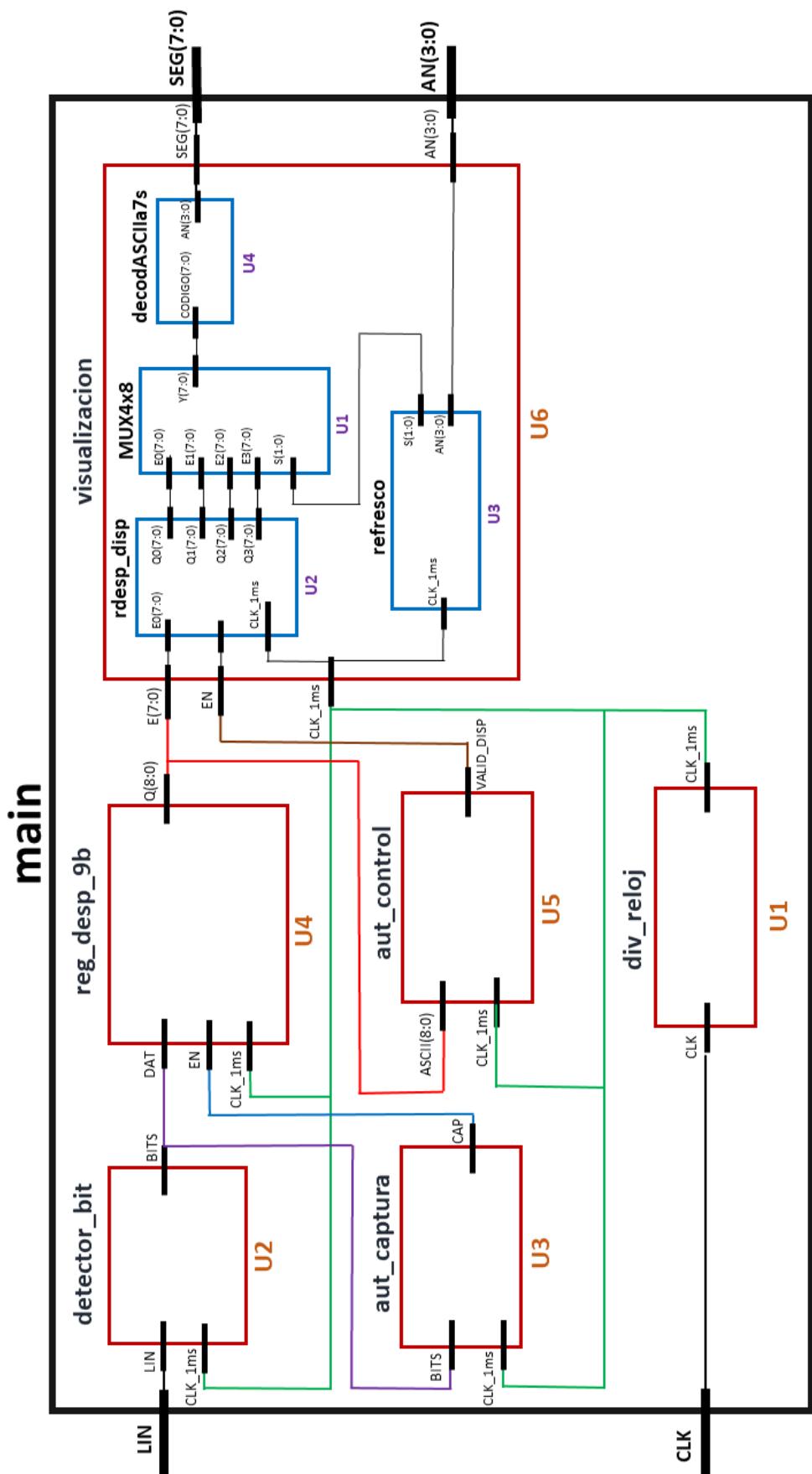
stim_proc: process
begin
    CODIGO<="01011000";
    wait for 900 ms;
    CODIGO<="01001010";
    wait for 900 ms;
    CODIGO<="01010100";
    wait for 900 ms;
    CODIGO<="00110000";
    wait for 900 ms;
    CODIGO<="00110010";
    wait for 900 ms;

end process;

END;

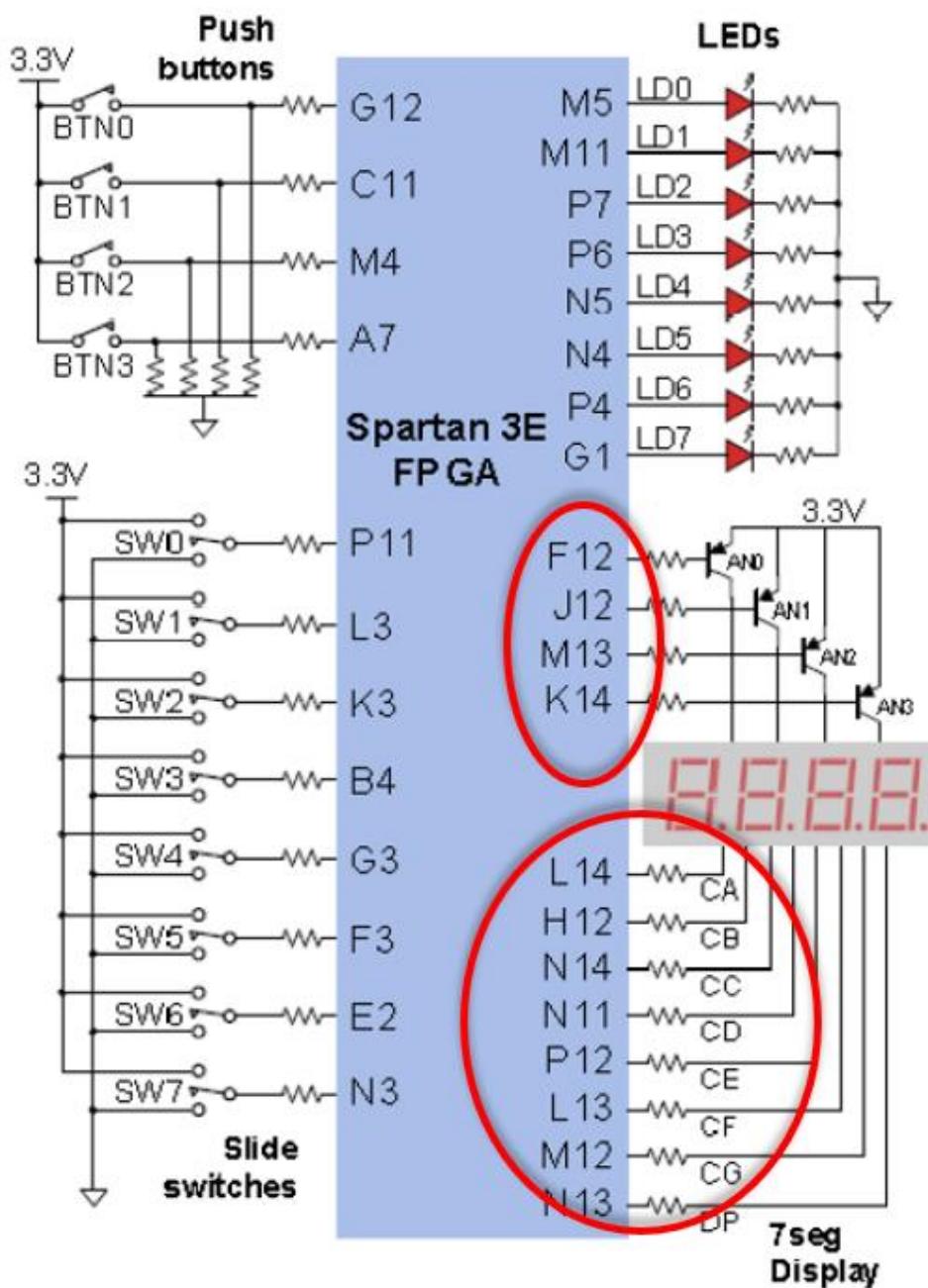
```

4.7 DIAGRAMA LÓGICO DEL CIRCUITO DIGITAL

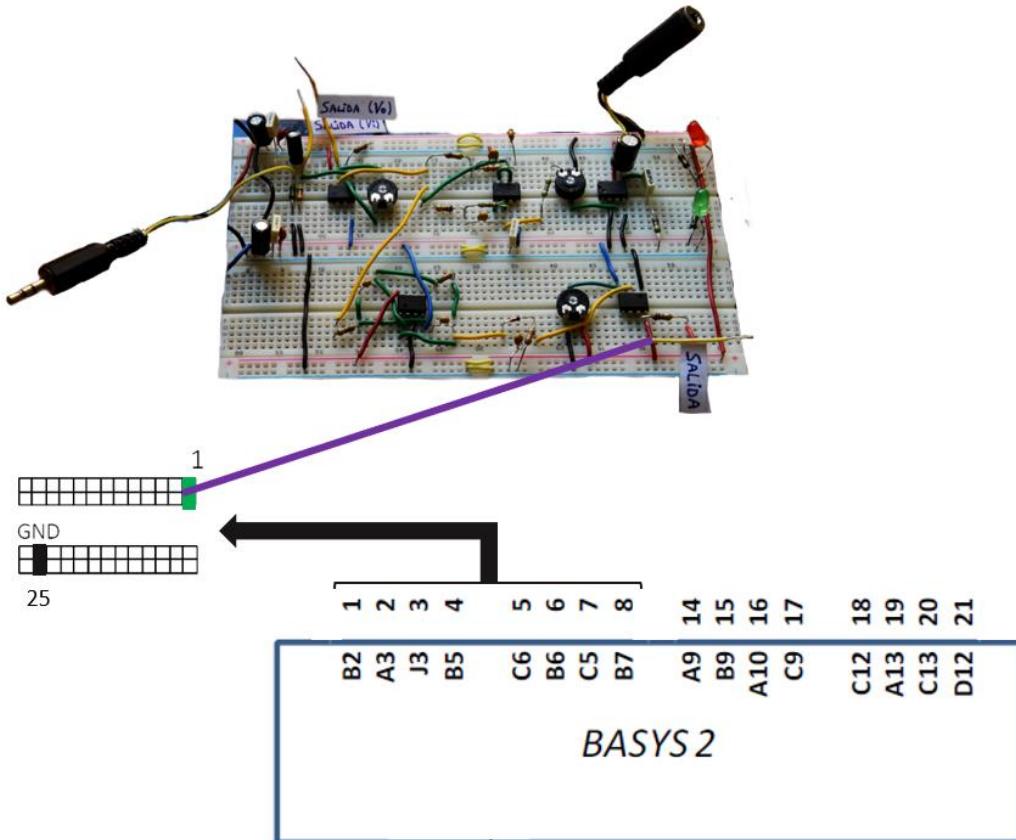


4.8 ASOCIACIONES

El fichero de asociaciones permite indicar las posiciones físicas de las entradas y salidas en las patillas de la FPGA. En la siguiente figura, podemos observar las conexiones para permitir activar los segmentos de cada display, así como la activación de cada uno.



La interconexión, entre el circuito analógico y el digital, se lleva a cabo a través de la entrada 1 del tablero (patilla B2 de la BASY S2), mientras que la entrada 25 (GND) se conecta a la masa del circuito.



Además, la FPGA tiene una entrada conectada a un reloj, cuya frecuencia es de 50 [MHz] que se corresponde con la patilla M6.

```

# Reloj principal del sistema

NET "CLK" LOC = "M6"; # Señal de reloj del sistema de XTAL

# Conexiones de los DISPLAYS

NET "SEG7<0>" LOC = "L14"; # señal = CA
NET "SEG7<1>" LOC = "H12"; # Señal = CB
NET "SEG7<2>" LOC = "N14"; # Señal = CC
NET "SEG7<3>" LOC = "N11"; # Señal = CD
NET "SEG7<4>" LOC = "P12"; # Señal = CE
NET "SEG7<5>" LOC = "L13"; # Señal = CF
NET "SEG7<6>" LOC = "M12"; # Señal = CG

# Señales de activacion de los displays

NET "AN<0>" LOC = "K14"; # Activacion del display 0 = AN0
NET "AN<1>" LOC = "M13"; # Activacion del display 1 = AN1
NET "AN<2>" LOC = "J12"; # Activacion del display 2 = AN2
NET "AN<3>" LOC = "F12"; # Activacion del display 3 = AN3

# Entrada externa donde se conecta la señal entrante

NET "LIN" LOC = "B2";

```

5. MEJORAS

Partiendo del prototipo básico del decodificador de la señal RDS, se han realizado algunas mejoras, para mejorar su funcionamiento. Las mejoras realizadas han sido: la introducción de un reset manual, así como un reset automático, la detección de caracteres erróneos, un rectificador de precisión y la distinción entre caracteres alfabéticos y numéricos.



5.1 RESETS

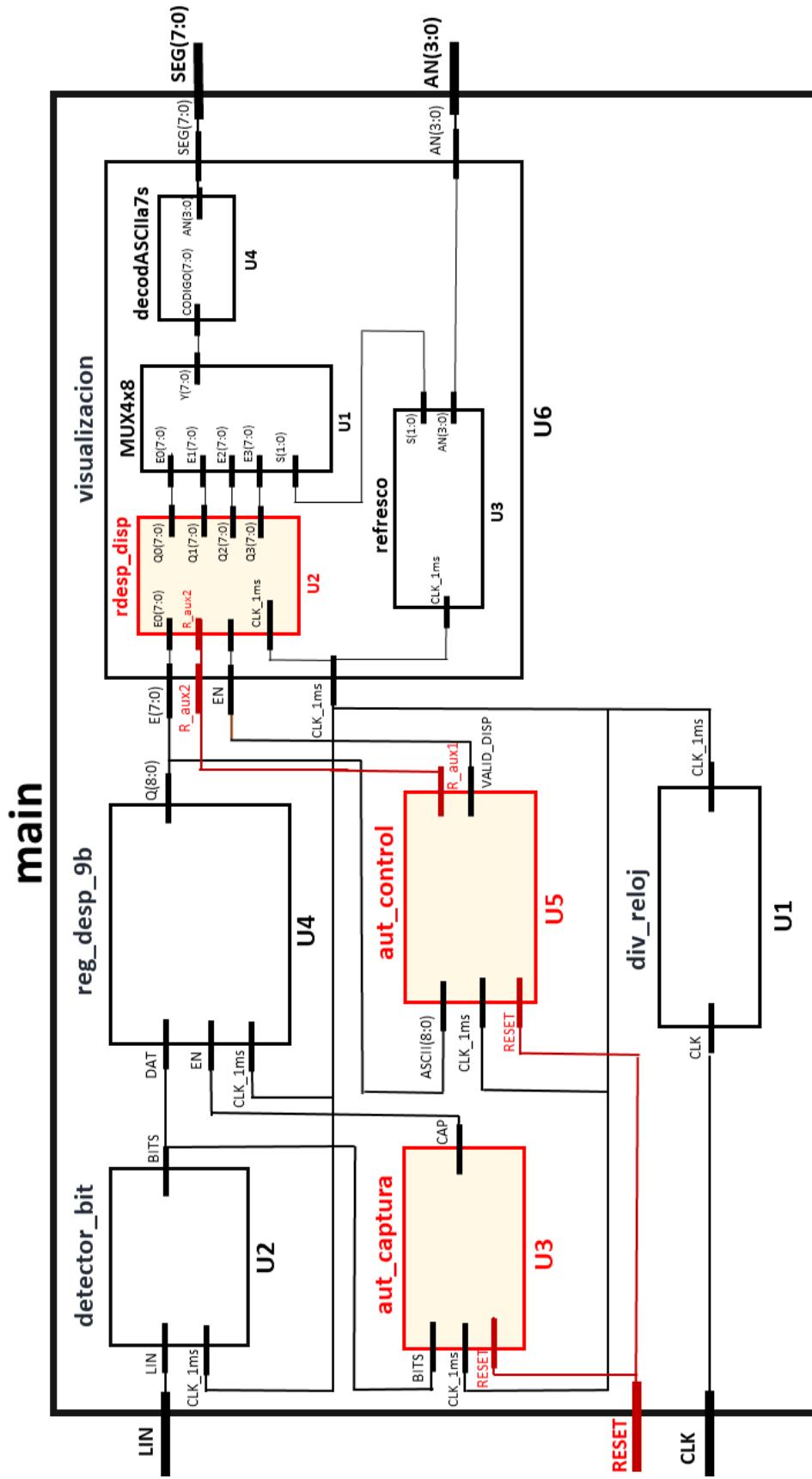
Hasta ahora, una vez el circuito realiza los sincronismos, los autómatas entran en un bucle indefinido de recibir e interpretar bits. En esta situación, en el caso de un corte de la emisión, parando el audio en el móvil, y una vuelta a emitir, el sistema no tenía forma de reiniciarse para poder volver correctamente a hacer su función. Esto por ejemplo llevaba a que los displays se mostraran caracteres que nada tenían que ver con lo esperado, y completamente carentes de sentido alguno.

Al realizar el sincronismo de bit y de carácter en el proyecto básico, se entra en un bucle sin salida, donde se reciben e interpretan los bits de cada carácter. En caso, de una interrupción de la señal de entrada (audio), se pierde el sincronismo, y el sistema no es capaz de volver a interpretar correctamente los bits recibidos.

Por lo tanto, se opta por realizar tanto un reset manual, como un reset automático, para volver a sincronizar los bits de la señal, en caso de una interrupción.

Finalmente, para poder apagar los displays, dado que al cortar la señal de entrada los displays se congelaban con los últimos caracteres cargados, se crea una señal adicional, que se activa, únicamente en caso de que se produzca algún reset, y se interconecta con el registro de desplazamiento (`rdes_disp.vhd`) del módulo de visualización.

A continuación, podemos observar el diagrama lógico completo, después de conectar la nueva entrada del sistema, RESET, y la señal de salida del autómata de control, R_aux2 al módulo de registro de desplazamiento.

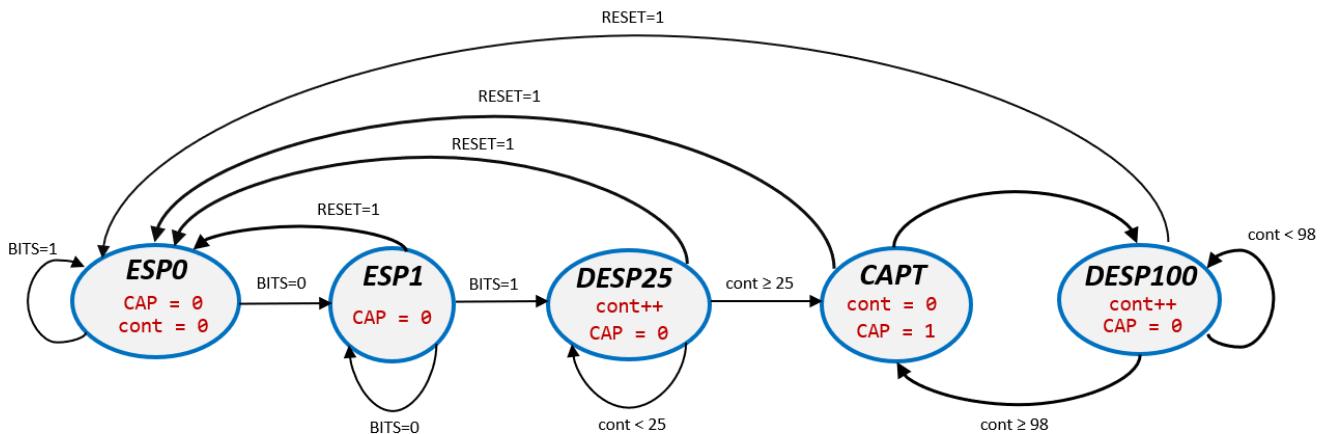


5.1.1 RESET MANUAL

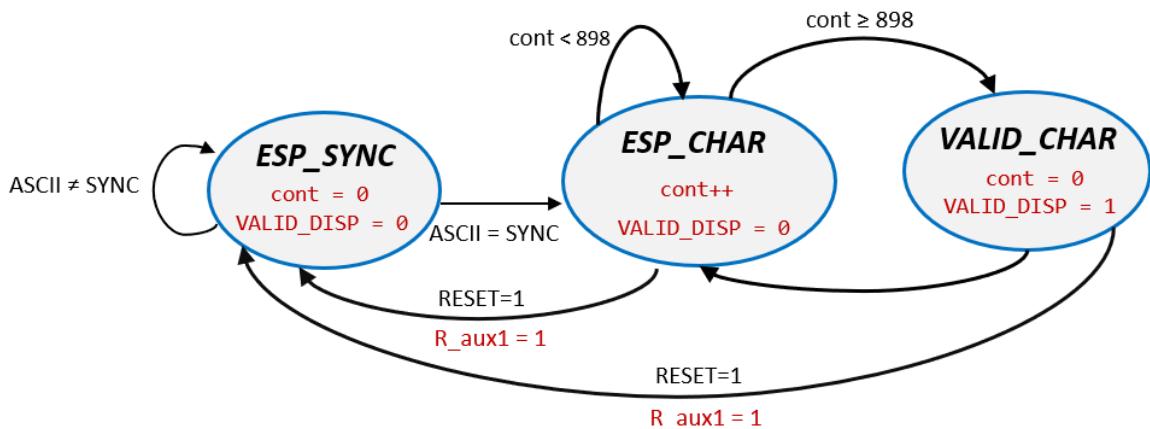
Para arreglar esto, se plantea la introducción de un reset manual, accionado mediante un pulsador de la FPGA, que lleve a los autómatas al estado inicial, de forma que puedan volver a sincronizarse dado que vuelven a estar en el estado correspondiente a la espera de la señal SYNC.

El reset manual, como su propio nombre indica, incluye una entrada adicional de RESET, que se activa, cuando se pulsa un pulsador de la tarjeta BASYS2. Con dicha pulsación, se vuelve a los estados iniciales en los autómatas de control y de captura.

En el autómata de captura, si se pulsa el RESET manual, volvemos al estado inicial, ESP0, donde allí siempre se pone a 0 el contador, para reiniciar la cuenta de los instantes de captura.



Por otro lado, en el autómata de control, de forma análoga, cada vez que se activa la señal de RESET, se vuelve al estado ESP_SYNC, donde se reinicia el contador, y se espera hasta que se recibe nuevamente la señal de sincronismo de carácter SYNC.



Los cambios que se han introducido, en el código VHDL son los siguientes:

```
entity aut_captura is
    Port ( CLK_1ms : in STD_LOGIC; -- Reloj del sistema
           BITS : in STD_LOGIC; -- Bits de entrada
           RESET : in STD_LOGIC; -- Seal reset manual
           CAP : out STD_LOGIC); -- Seal para el registro de captura
end aut_captura;

architecture a_aut_captura of aut_captura is
type STATE_TYPE is (ESP0,ESP1,DESP25,CAPT,DESP100);

signal ST : STATE_TYPE := ESP0;
signal cont : STD_LOGIC_VECTOR (7 downto 0):="00000000";

begin
process (CLK_1ms, RESET)
begin
    if CLK_1ms'event and CLK_1ms='1' then
        case ST is
            when ESP0 =>
                cont<="00000000"; --Se reinicia el contador (en caso de reset)
                if BITS='1' then
                    ST<=ESP0;
                else
                    ST<=ESP1;
                end if;

            when ESP1 =>
                if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
                    ST<=ESP0;
                elsif BITS='1' then
                    ST<=DESP25;
                else
                    ST<=ESP1;
                end if;

            when DESP25 =>
                cont<=cont+1;
                if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
                    ST<=ESP0;
                elsif cont>=25 then
                    ST<=CAPT;
                else
                    ST<=DESP25;
                end if;

            when CAPT =>
                cont<="00000000";
                if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
                    ST<=ESP0;
                end if;
                ST<=DESP100;

            when DESP100 =>
                cont<=cont+1;
                if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
                    ST<=ESP0;
                elsif cont>=98 then
                    ST<=CAPT;
                else
                    ST<=DESP100;
                end if;

        end case;
    end if;
end process;

with ST select CAP <=
    '1' when CAPT,
    '0' when others;

end a_aut_captura;
```

```

entity aut_captura is
  Port ( CLK_1ms : in STD_LOGIC; -- Reloj del sistema
         BITS : in STD_LOGIC; -- Bits de entrada
         RESET : in STD_LOGIC; -- Seal reset manual
         CAP : out STD_LOGIC); -- Seal para el registro de captura
end aut_captura;

architecture a_aut_captura of aut_captura is
type STATE_TYPE is (ESP0,ESP1,DESP25,CAPT,DESP100);

signal ST : STATE_TYPE := ESP0;
signal cont : STD_LOGIC_VECTOR (7 downto 0):="00000000";

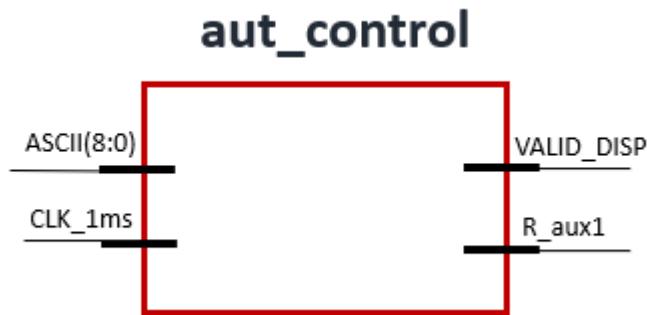
begin
  process (CLK_1ms, RESET)
  begin
    if CLK_1ms'event and CLK_1ms='1' then
      case ST is
        when ESP0 =>
          if cont<="00000000"; --Se reinicia el contador (en caso de reset)
             if BITS='1' then
               ST<=ESP0;
             else
               ST<=ESP1;
             end if;
          end if;
        when ESP1 =>
          if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
            ST<=ESP0;
          elsif BITS='1' then
            ST<=DESP25;
          else
            ST<=ESP1;
          end if;
        when DESP25 =>
          cont<=cont+1;
          if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
            ST<=ESP0;
          elsif cont>=25 then
            ST<=CAPT;
          else
            ST<=DESP25;
          end if;
        when CAPT =>
          cont<="00000000";
          if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
            ST<=ESP0;
          end if;
          ST<=DESP100;
        when DESP100 =>
          cont<=cont+1;
          if RESET='1' then --Si pulsamos el reset manual, volvemos al estado inicial ESP0
            ST<=ESP0;
          elsif cont>=98 then
            ST<=CAPT;
          else
            ST<=DESP100;
          end if;
      end case;
    end if;
  end process;

  with ST select CAP <=
    '1' when CAPT,
    '0' when others;
end a_aut_captura;

```

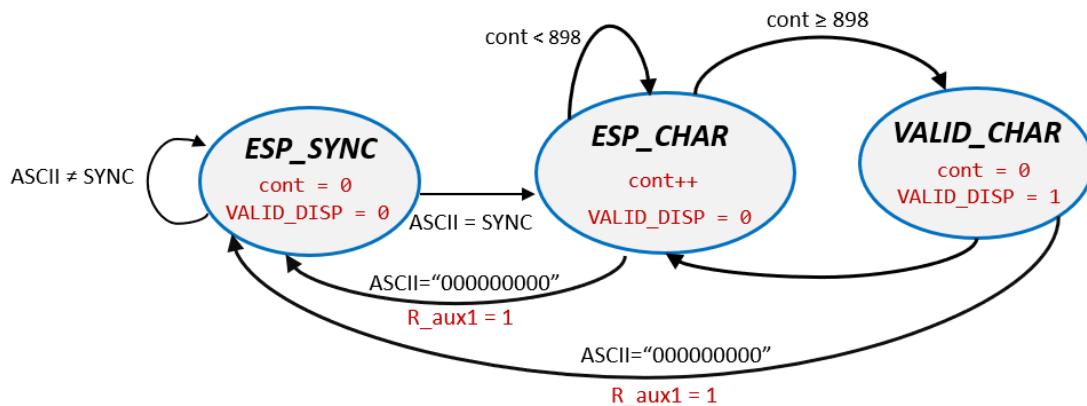
5.1.2 RESET AUTOMÁTICO

En este caso, se plantea el mismo problema que el explicado en el reset manual, con la salvedad de que se introduce un reset automático. Este reset se activa cuando el sistema detecta la llegada de una cadena de bits “000000000”, es decir, cuando no llega señal a la FPGA porque el audio del móvil se ha cortado.



Para esta mejora, solo se ha modificado el autómata de control, así como el registro de desplazamiento para permitir apagar los displays, una vez que se activa algún reset.

En el autómata de control, el diagrama de estados cambia ligeramente, ya que si se detecta una cadena de 9 ceros en ESP_CHAR o en VALID_CHAR, se activa la señal R_aux1 y se vuelve al estado inicial, ESP_SYNC.



El código VHDL del autómata de control, para realizar el reset automático es el siguiente:

```

entity aut_control is
  Port ( CLK_lms : in STD_LOGIC;                                -- Reloj del sistema
         ASCII : in STD_LOGIC_VECTOR (8 downto 0);                -- Datos de entrada del registro
         R_aux1 : out STD_LOGIC;                                     -- Salida reset cableada con rdesp_disp
         VALID_DISP : out STD_LOGIC);                               -- Salida para validar el display

end aut_control;

architecture a_aut_control of aut_control is

constant SYNC : STD_LOGIC_VECTOR (8 downto 0) := "100000001"; -- caracter SYNC

type STATE_TYPE is (ESP_SYNC,ESP_CHAR,VALID_CHAR);
signal ST : STATE_TYPE:=ESP_SYNC;
signal cont : STD_LOGIC_VECTOR (15 downto 0):="0000000000000000";

begin
  process (CLK_lms, RESET)
  begin
    if (CLK_lms'event and CLK_lms='1') then
      case ST is
        when ESP_SYNC =>
          cont<="0000000000000000"; --Se reinicia el contador (en caso de RESET), ya que inicialmente siempre vale 0
          if ASCII=SYNC then    -- Esperar por el SYNC
            ST<=ESP_CHAR;
          else
            ST<=ESP_SYNC;
          end if;
          R_aux1<='0'; --Seal para apagar displays, inicialmente a nivel bajo
        when ESP_CHAR =>
          cont<=cont+1;
          if ASCII="000000000" then --Si se detecta una cadena de 9 ceros
            ST<=ESP_SYNC; --Se vuelve al estado inicial, ESP_SYNC
            R_aux1<='1';-- Activamos la seal R_aux1 para apagar los displays
          elsif cont>=898 then
            ST<=VALID_CHAR;
          else
            ST<=ESP_CHAR;
          end if;
        when VALID_CHAR =>
          cont<="0000000000000000";
          if ASCII="000000000" then --Si se detecta una cadena de 9 ceros
            ST<=ESP_SYNC; --Se vuelve al estado inicial, ESP_SYNC
            R_aux1<='1';-- Activamos la seal R_aux1 para apagar los displays
          end if;
          ST<=ESP_CHAR;
      end case;
    end if;
  end process;

  with ST select VALID_DISP <=
    '1' when VALID_CHAR, -- Asignacion de la salida
    '0' when others;
end a_aut_control;

```

Otra de las funciones de un reset, aparte de realizar el sincronismo de bit y de carácter, es poder apagar los displays, de modo que los caracteres no se queden congelados en los displays, en caso de un reset.

Para poder apagarlos, debemos poner a 0 todas las salidas del registro de desplazamiento, en caso de que se active la señal, del autómata de control, R_aux1 (se pone a 1 cuando está activo el reset manual, o bien el reset automático).

Por lo tanto, en el código VHDL, se han añadido los siguientes cambios:

```

entity rdesp_disp is
  Port ( CLK_lms : in STD_LOGIC;           -- entrada de reloj
         EN      : in STD_LOGIC;           -- enable
         E       : in STD_LOGIC_VECTOR (7 downto 0); -- entrada de datos
         R_aux2 : in STD_LOGIC;           -- Seal reset aut_control
         Q0     : out STD_LOGIC_VECTOR (7 downto 0); -- salida Q0
         Q1     : out STD_LOGIC_VECTOR (7 downto 0); -- salida Q1
         Q2     : out STD_LOGIC_VECTOR (7 downto 0); -- salida Q2
         Q3     : out STD_LOGIC_VECTOR (7 downto 0)); -- salida Q3
end rdesp_disp;

architecture a_rdesp_disp of rdesp_disp is

  signal QS0 : STD_LOGIC_VECTOR (7 downto 0); -- seal que almacena el valor de Q0
  signal QS1 : STD_LOGIC_VECTOR (7 downto 0); -- seal que almacena el valor de Q1
  signal QS2 : STD_LOGIC_VECTOR (7 downto 0); -- seal que almacena el valor de Q2
  signal QS3 : STD_LOGIC_VECTOR (7 downto 0); -- seal que almacena el valor de Q3

begin

  process (CLK_lms, R_aux2)
  begin
    if R_aux2='1' then --Si se ha detectado un reset (Manual o automatico)
      QS0<="00000000"; --Asignamos ceros a todas las salidas del regitstro de desplazamiento, para apagar los displays
      QS1<="00000000";
      QS2<="00000000";
      QS3<="00000000";
    end if;
    elsif (EN ='1') then
      if (CLK_lms'event and CLK_lms='1') then -- con cada flanco activo
        QS3<=QS2;                           -- se desplazan todas las salidas
        QS2<=QS1;
        QS1<=QS0;
        QS0<=E;                           -- y se copia el valor de la entrada en Q0
      end if;
    end if;
  end process;

  Q0<=QS0;                                     -- actualizacion de las salidas
  Q1<=QS1;
  Q2<=QS2;
  Q3<=QS3;
end a_rdesp_disp;

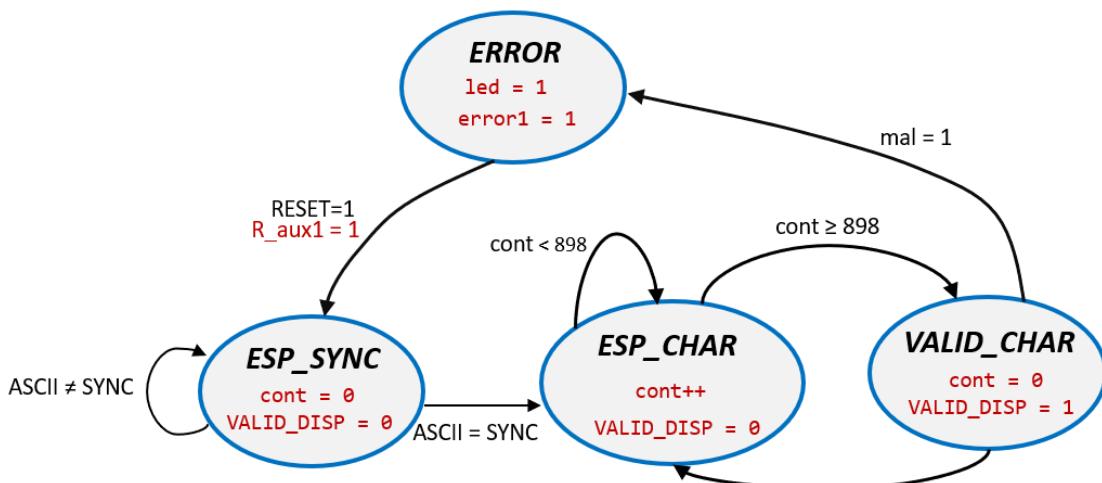
```

5.3 DETECCIÓN DE LA LLEGADA DE CARACTERES ERRÓNEOS

En esta mejora, se ha pretendido detectar la llegada de un carácter erróneo. Todos los caracteres de la señal RDS.mp3 empiezan por un 1, y los caracteres representados son el espacio, todas las letras, y las cifras. En caso de que no lleguen uno de estos caracteres se entenderá que se ha recibido un error. Por tanto, se producen errores en los siguientes casos:

- Primer bit es un 0, pero los ocho bits restantes son distintos de 0.
- Si su código ASCII no es uno de los siguientes:
 - ✚ 0x20 -> Espacio
 - ✚ 0x41 al 0x5A -> Letras
 - ✚ 0x30 al 0x39 -> Números

El nuevo diagrama de estados, diseñado para el autómata de control es el siguiente:



- En caso de que se produzca un error, se activa la señal “mal”, si dicha señal está activa en **VALID_CHAR** (ya que debemos comprobar los bits del carácter, una vez que ha llegado el carácter completo, 900ms), si dicha señal esta activa, nos lleva a un estado llamado **ERROR**.
- En **ERROR**, se activa el LED, y la señal `error1`. En caso de producirse un reset manual, nos lleva al estado **ESP_SYNC**, donde se vuelve esperar la señal de sincronismo.

El código VHDL, con los cambios implementados para detectar caracteres erróneos es el siguiente:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity aut_control is
  Port ( CLK_lms : in STD_LOGIC;          -- Reloj del sistema
         ASCII : in STD_LOGIC_VECTOR (8 downto 0); -- Datos de entrada del registro
         RESET : in STD_LOGIC;                      -- Señal de reset
         R_aux1 : out STD_LOGIC;                    -- Salida reset rdsp_dsp
         error1 : out STD_LOGIC;                   -- Señal de error
         led : out STD_LOGIC;                     -- Led de salida, en caso de error
         VALID_DISP : out STD_LOGIC);            -- Salida para validar el display

end aut_control;

architecture a_aut_control of aut_control is

constant SYNC : STD_LOGIC_VECTOR (8 downto 0) := "100000001"; -- carácter SYNC
constant ESPACIO : STD_LOGIC_VECTOR (8 downto 0) := "100100000"; -- carácter espacio

type STATE_TYPE is (ESP_SYNC,ESP_CHAR,VALID_CHAR,ERROR);
signal ST : STATE_TYPE:=ESP_SYNC;
signal cont : STD_LOGIC_VECTOR (15 downto 0):="0000000000000000";
signal mal : STD_LOGIC:='0';

begin

process (CLK_lms)
begin

  if (CLK_lms'event and CLK_lms='1') then
    case ST is
      when ESP_SYNC =>
        cont<="0000000000000000";
        led<='0'; --Iniciamos a 0 las nuevas señales
        error1<='0';
        mal<='0';
        if ASCII=SYNC then -- Esperar por el SYNC
          ST<=ESP_CHAR;
        else
          ST<=ESP_SYNC;
        end if;

        R_aux1<='0';
        when ESP_CHAR =>
          cont<=cont+1;
          if cont>=898 then
            ST<=VALID_CHAR;
          else
            ST<=ESP_CHAR;
          end if;

      when VALID_CHAR =>
        cont<="0000000000000000";
        if ((ASCII(8)='0') and (ASCII(7 downto 0)!="00000000")) then --Si el bit 9 es un 0 y los 8 restantes son distintos de 0
          mal<='1';
        elsif (ASCII=ESPACIO) then --Si es un espacio, no hay error
          mal<='0';
        elsif ("00110000"≤ASCII) and (ASCII≤"00111001") then --Si es un número, no hay error
          mal<='0';
        elsif ("01000001"≤ASCII) and (ASCII≤"01011010") then --Si es una letra, no hay error
          mal<='0';
        else
          mal<='1'; --Si no es ni espacio, número ni letra, entonces hay un error
        end if;
        if mal='1' then --Si hay un error pasamos, a ERROR
          ST<=ERROR;
        end if;
        ST<=ESP_CHAR;
    end case;
  end if;
end process;

```

```

when ERROR =>
    error1<='1';
    led<='1';
    if RESET='1' then --En caso de reset manual, volvemos a ESP_SYNC
        ST<=ESP_SYNC;
        R_aux1<='1';

    end if;

    end case;
end if;
end process;

with ST select VALID_DISP <=
    '1' when VALID_CHAR, -- Asignacion de la salida
    '0' when others;

```

Por otra parte, en el fichero asociaciones.ucf se han añadido las siguientes conexiones de la entrada (reset) y de la salida (led, en caso de error)

```

# RESET

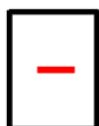
NET "RESET" LOC = "G12"; # Reset manual

# LED en caso de error

NET "led" LOC="M11";

```

En caso de producirse un error, se debe encender un guión, es decir todos los segmentos del display deben permanecer apagados excepto el “CG”. Además, hemos asignado en el decodificador ASCII de 7 segmentos un valor de CODIGO, que no estuviera asignado antes al valor del guion, por ejemplo, “10000000”, al valor de SEGMENTOS igual a “11111101”.



```

"00000000" when "00111000", -- 8
"00001000" when "00111001", -- 9
"00000010" when "00110000", -- 0
"11111101" when "10000000", -- -
"11111111" when others;      -- apagado

```

Además, se ha modificado el fichero rdesp_disp.vhd, del módulo de visualización, para que en caso de que se produzca un error o se pulse el reset manual, se apliquen los cambios pertinentes al display de la BASYS2. Por tanto, se han realizado los siguientes cambios:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity rdesp_disp is
  Port ( CLK_lms : in STD_LOGIC;                      -- entrada de reloj
         EN      : in STD_LOGIC;                      -- enable
         E       : in STD_LOGIC_VECTOR (7 downto 0);    -- entrada de datos
         R_aux2  : in STD_LOGIC;                      -- Señal reset aut_control
         error2  : in STD_LOGIC;
         Q0      : out STD_LOGIC_VECTOR (7 downto 0);   -- salida Q0
         Q1      : out STD_LOGIC_VECTOR (7 downto 0);   -- salida Q1
         Q2      : out STD_LOGIC_VECTOR (7 downto 0);   -- salida Q2
         Q3      : out STD_LOGIC_VECTOR (7 downto 0));  -- salida Q3
end rdesp_disp;

architecture a_rdesp_disp of rdesp_disp is

signal Q0 : STD_LOGIC_VECTOR (7 downto 0); -- señal que almacena el valor de Q0
signal Q1 : STD_LOGIC_VECTOR (7 downto 0); -- señal que almacena el valor de Q1
signal Q2 : STD_LOGIC_VECTOR (7 downto 0); -- señal que almacena el valor de Q2
signal Q3 : STD_LOGIC_VECTOR (7 downto 0); -- señal que almacena el valor de Q3

begin

process (CLK_lms, R_aux2)
begin
  if R_aux2='1' then --Si se pulsa el reset manual, apagamos los displays
    Q0<="00000000";
    Q1<="00000000";
    Q2<="00000000";
    Q3<="00000000";

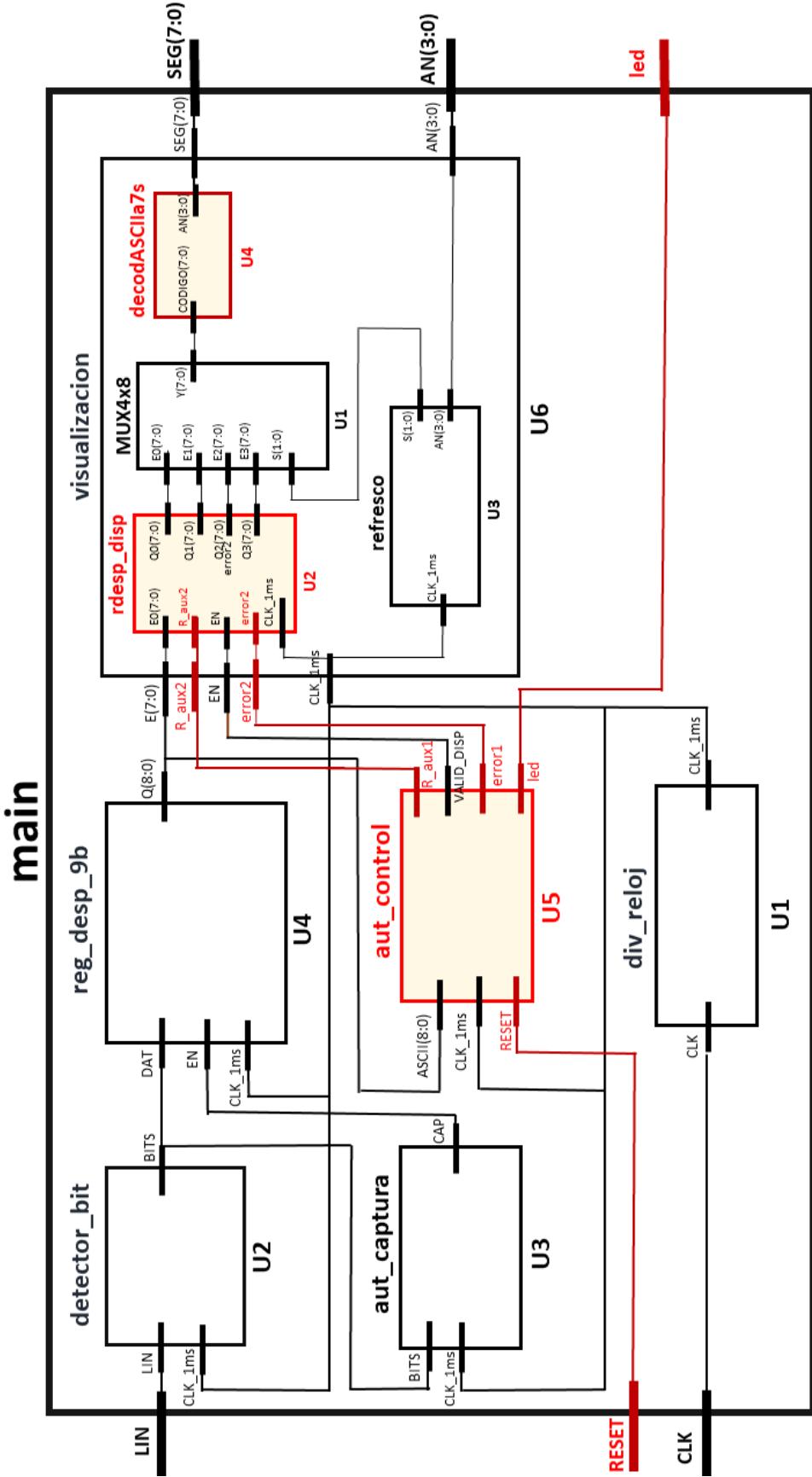
  elsif (EN ='1' and error2='1') then --Si se produce un error, se introduce el código ASCII de un guion
    Q0<="10000000";
    Q1<="10000000";
    Q2<="10000000";
    Q3<="10000000";

  elsif (CLK_lms'event and CLK_lms='1') then -- con cada flanco activo
    Q3<=Q2;                                -- se desplazan todas las salidas
    Q2<=Q1;
    Q1<=Q0;
    Q0<=E;                                -- y se copia el valor de la entrada en Q0
  end if;
end process;

Q0<=Q0;                                     -- actualizacin de las salidas
Q1<=Q1;
Q2<=Q2;
Q3<=Q3;
end a_rdesp_disp;

```

Finalmente, el diagrama lógico completo tras realizar la mejora de detección de errores sería el siguiente:

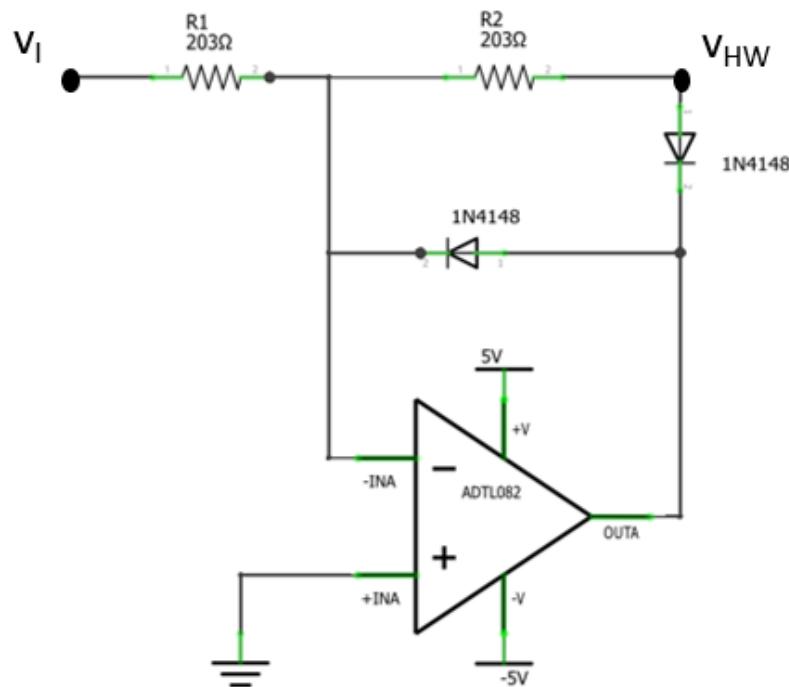


5.4 RECTIFICADOR DE PRECISIÓN

Mediante el rectificado de la señal se obtienen las componentes de frecuencia en banda base y en los armónicos de la frecuencia principal. Pero para operar con señales de entrada débiles en el rectificador, es necesario sustituir el diodo rectificador (1N4147) por un rectificador de precisión de onda completa con ganancia 5, ya que el diodo, necesita una tensión mínima (0,6 [V]) para entrar en la zona de conducción directa. Con el rectificador de precisión se puede detectar señales débiles, ya que permite la conducción desde la señal de entrada de 0 [V]

Un rectificador de onda completa, en la combinación de una señal rectificada en media onda inversora, en una relación de 1 a 2, sumada a la señal de entrada. De esta forma, se consigue una onda rectificada de forma completa, con una ganancia en valor absoluto igual a 5.

El rectificador está compuesto por dos amplificadores operacionales, el primero de ellos, se encarga de rectificar la señal en media onda inversora:

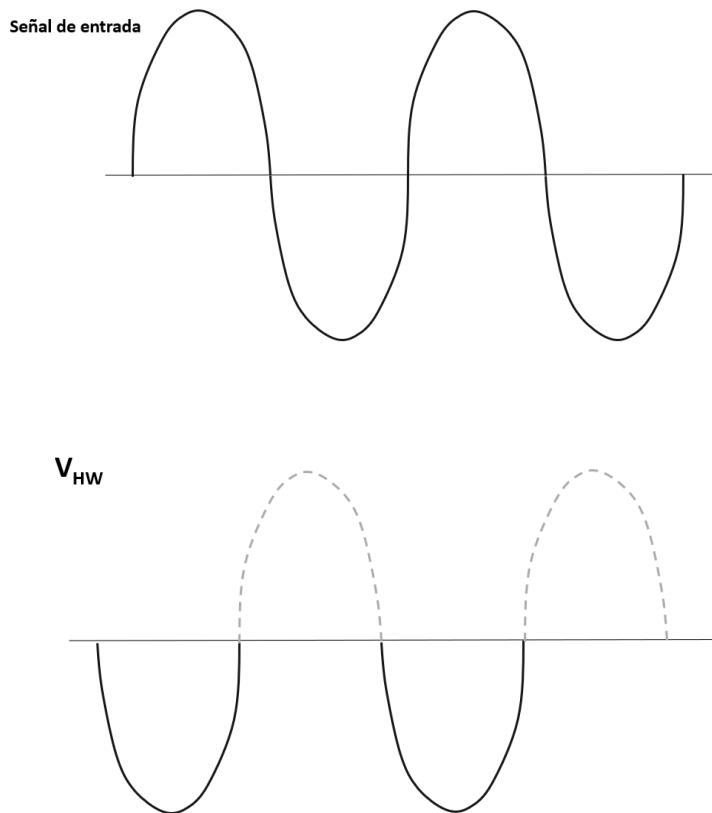


La señal de salida v_{HW} es igual a:

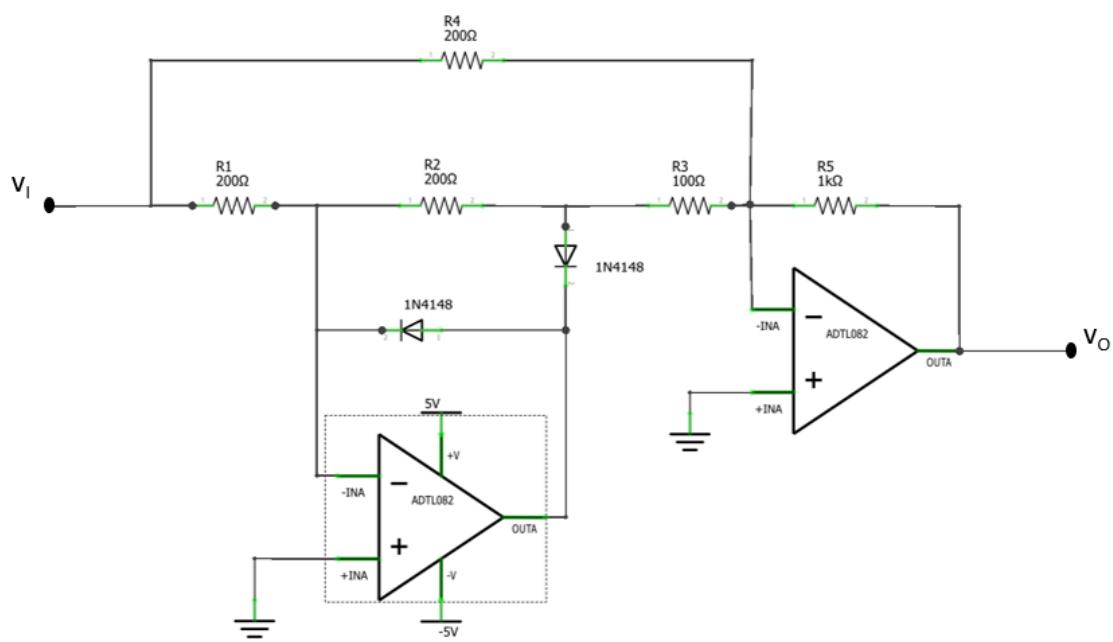
$$v_{HW} = - \frac{R_2}{R_1} \cdot v_I \quad \text{para} \quad v_I > 0$$

$$v_{HW} = 0 \quad \text{para} \quad v_I < 0$$

La señal de salida teórica de este primer amplificador, que rectifica en media onda inversora es:

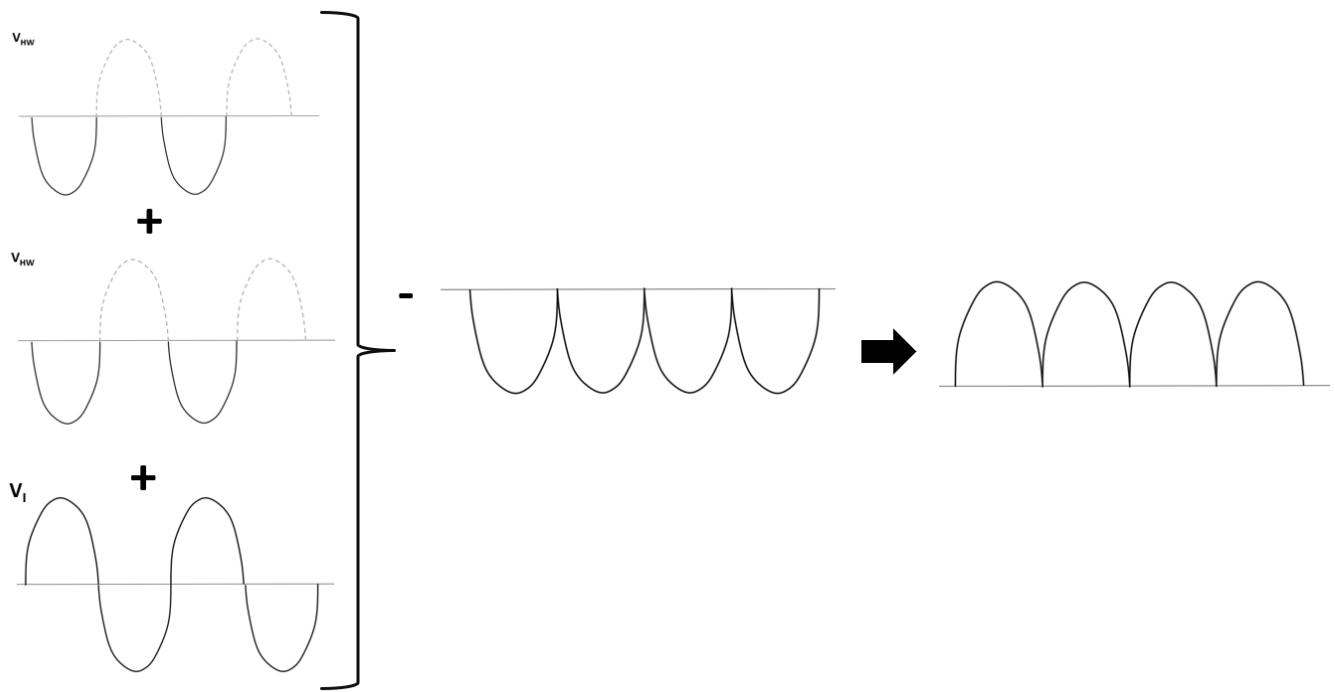


El segundo amplificador es un **sumador inversor**, con realimentación negativa, que se encarga de sumar por una parte la señal de entrada, con la señal de salida del primer amplificador en una relación de 1 a 2:

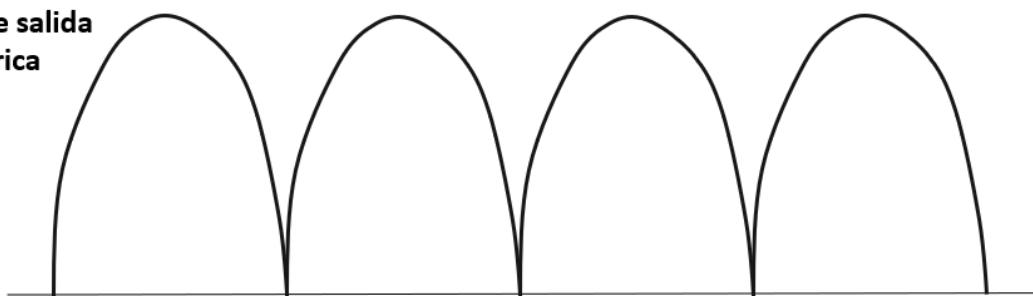


$$v_0 = \frac{-R_5}{R_4} \cdot v_I + \frac{-R_5}{R_3} \cdot v_{HW}$$

$$v_0 = \begin{cases} \frac{-R_5}{R_4} \cdot v_I + \frac{R_5}{R_3} \cdot \frac{R_2}{R_1} v_I & \text{para } v_I > 0 \\ \frac{-R_5}{R_4} \cdot v_I & \text{para } v_I < 0 \end{cases}$$



Señal de salida teórica



Mediante el multímetro, se ha comprobado que, aunque se han diseñado los filtros con resistencias que poseen muy poca tolerancia (1%), no son componentes perfectos, y los valores obtenidos de las resistencias son:

$$R_1 = 203 \Omega$$

$$R_2 = 203 \Omega$$

$$R_3 = 98 \Omega$$

$$R_4 = 203 \Omega$$

$$R_5 = 997 \Omega$$

Sustituyendo el valor de los componentes reales, en la ecuación de la señal de salida, la ganancia será:

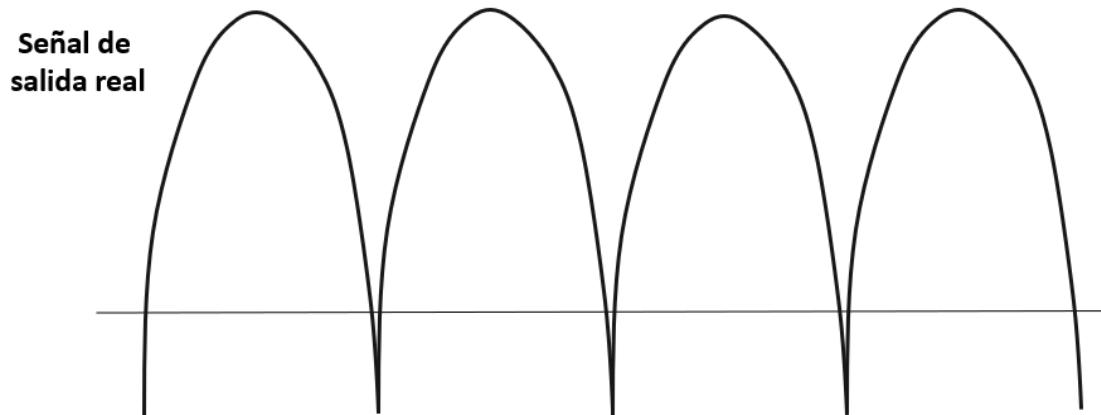
$$v_0 \approx 5 \cdot v_I \quad \text{para } v_I > 0 [V]$$

$$v_0 \approx -5 \cdot v_I \quad \text{para } v_I < 0 [V]$$

O lo que es lo mismo:

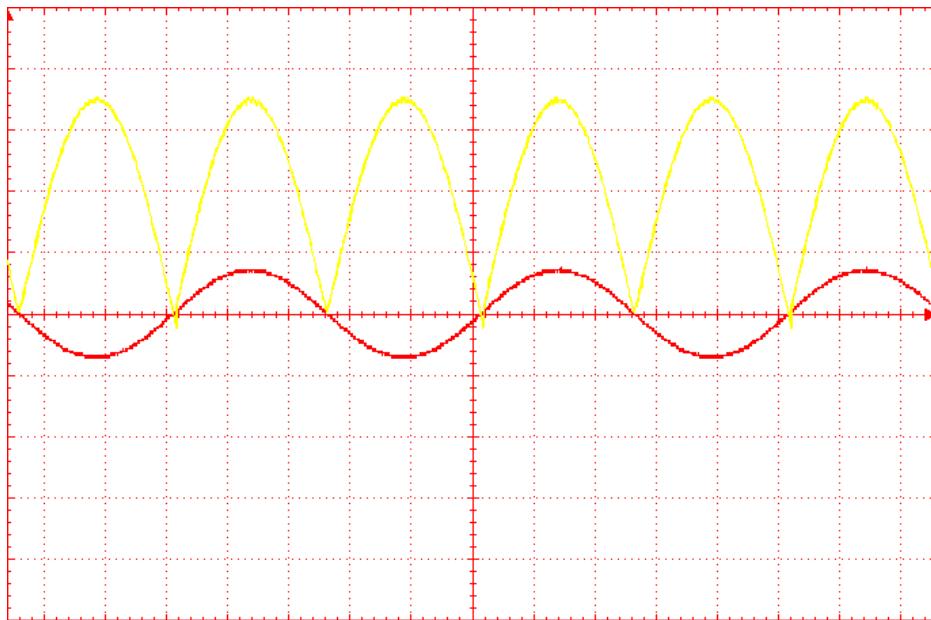
$$v_0 \approx 5 \cdot |v_I|$$

Aun teniendo valores muy próximos a los deseados, el funcionamiento del rectificador de precisión no es el deseado, debido a los valores reales de los componentes, y el resultado obtenido, al introducir una señal sinusoidal generada por el generador de funciones es:

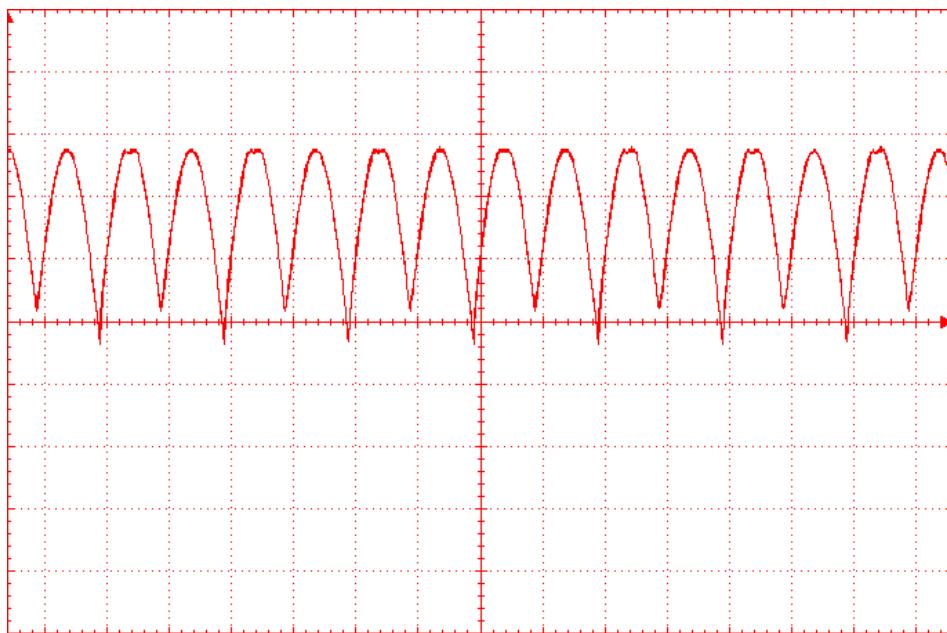


A continuación, se muestran capturas del osciloscopio, realizadas al detector de precisión:

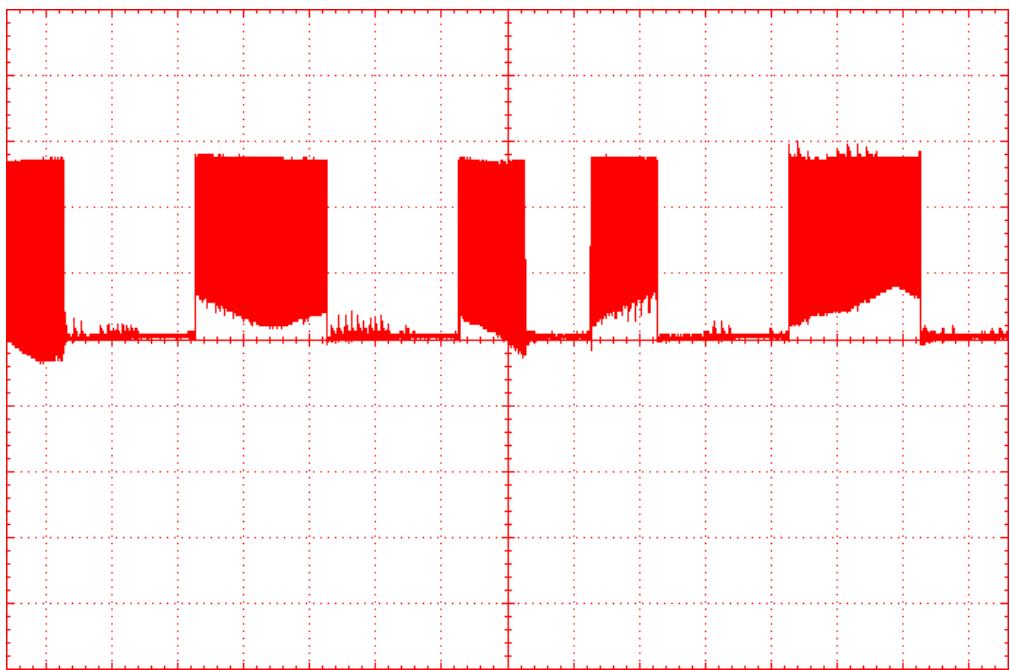
CAPTURA 1: Señal de entrada sinusoidal (rojo) y señal a la salida del rectificador de precisión (amarilla), con 5 [mV] y 200 [μ s], donde se aprecia una ganancia igual a 5.



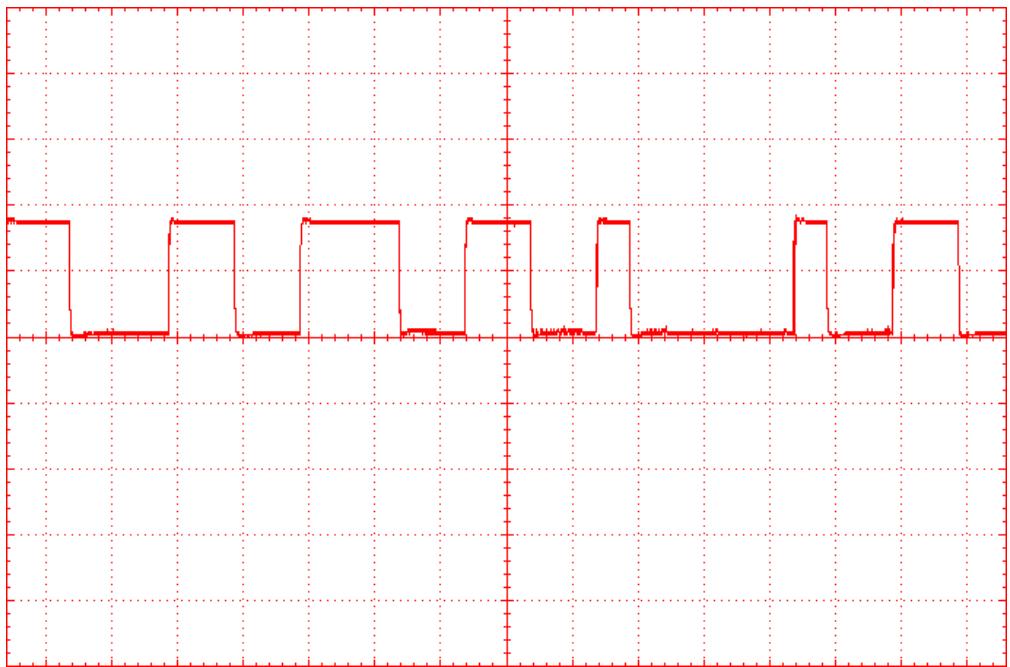
CAPTURA 2: Señal del audio, tras pasar el rectificador de precisión, justo antes de la entrada del filtro paso bajo, con 1 [V] y 50 [μ s].



CAPTURA 3: Señal del audio, tras pasar el rectificador de precisión, justo antes de la entrada del filtro paso bajo, con 1 [V] y 100 [ms].



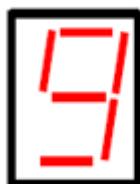
CAPTURA 4: Señal del audio, tras pasar filtro paso banda, justo antes de la entrada del comparador, con 1 [V] y 100 [ms].



5.5. DISTINCIÓN ENTRE CARACTERES ALFABÉTICOS Y NUMÉRICOS

El decodificador ASCII a 7 segmentos asigna la misma representación en el display a los caracteres "S" y "5", así como los caracteres "G" y "9". Por lo tanto, existe la necesidad de diferenciar dichos caracteres en el display. Para ello cuando se muestre un carácter numérico (del 0 al 9), se encenderá un punto decimal en el display, mientras que, si se representa un carácter alfabético, dicho punto permanecerá apagado.

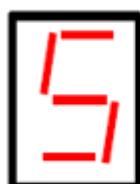
Carácter "g"



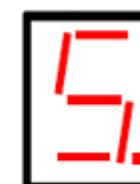
Carácter "9"



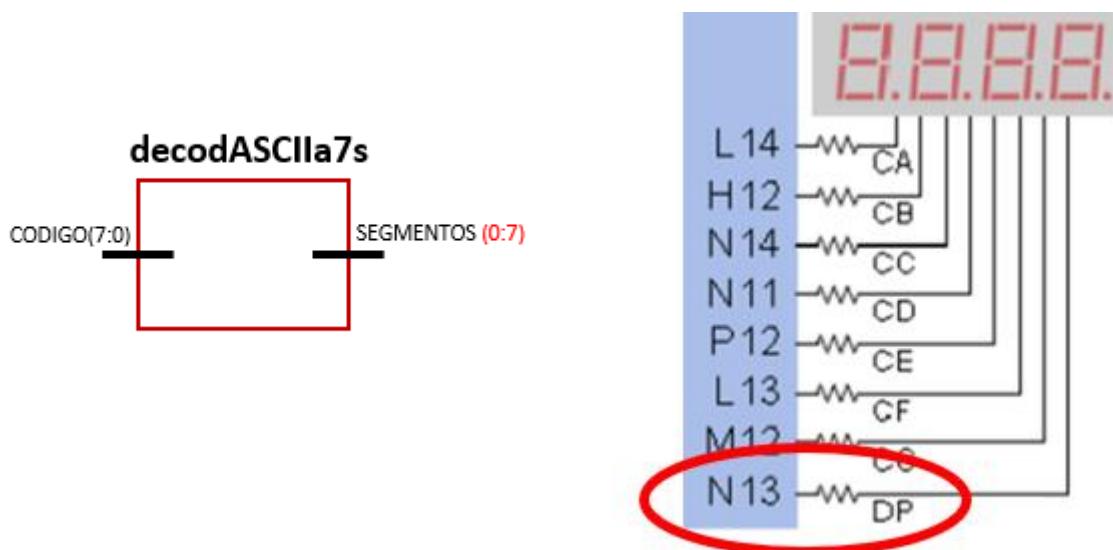
Carácter "s"



Carácter "5"



Para ello, debemos añadir un bit adicional a la señal de salida SEGMENTOS, de modo que tenga 8 bits, siete de ellos para los segmentos del display (CA, CB, ..., CG) y otro adicional, para añadir el punto decimal, DP, que se conecta a la BASYS 2, por medio de la patilla N13.



En el fichero decodASCIIa7s.vhd se ha añadido un bit adicional a la salida de SEGMENTOS, así dicha salida será del tipo “x x x x x x x 1” si es un carácter alfabético (punto decimal apagado, ya que está activo a nivel bajo) y “x x x x x x x 0” si es un carácter numérico (punto decimal encendido), como se muestra a continuación:

```

entity decodASCIIa7s is
    Port ( CODIGO : in STD_LOGIC_VECTOR (7 downto 0);      -- codigo ASCII
           SEGMENTOS : out STD_LOGIC_VECTOR (0 to 7));      -- Salidas al display
end decodASCIIa7s;

architecture a_decodASCIIa7s of decodASCIIa7s is

begin
    with CODIGO select SEGMENTOS<=
    --abcdefg      ASCII
    -----
    "11111111" when "00100000", -- ESPACIO
    "00010001" when "01000001", -- A
    "11000001" when "01000010", -- B
    "01100001" when "01000011", -- C
    "10000001" when "01000100", -- D      --a--
    "01100001" when "01000101", -- E      |   |
    "01110001" when "01000110", -- F      f   b
    "00001001" when "01000111", -- G      |   |
    "10010001" when "01001000", -- H      --g--
    "11011101" when "01001001", -- I      |   |
    "10000101" when "01001010", -- J      e   c
    "11111101" when "01001011", -- K      |   |
    "11100001" when "01001100", -- L      --d--
    "11111111" when "01001101", -- M
    "11010101" when "01001110", -- N
    "11000101" when "01001111", -- O Segundos encendidos con '0'
    "00110001" when "01010000", -- P Segundos apagados con '1'
    "00011001" when "01010001", -- Q
    "11110101" when "01010010", -- R
    "01001001" when "01010011", -- S
    "11100001" when "01010100", -- T
    "11000101" when "01010101", -- U
    "11111101" when "01010110", -- V
    "11111111" when "01010111", -- W
    "11111111" when "01011000", -- X
    "10001001" when "01011001", -- Y
    "11111111" when "01011010", -- Z

    "10011100" when "00110001", -- 1
    "00100100" when "00110010", -- 2
    "00001100" when "00110011", -- 3
    "10011000" when "00110100", -- 4
    "01001000" when "00110101", -- 5
    "11000000" when "00110110", -- 6
    "00011100" when "00110111", -- 7
    "00000000" when "00111000", -- 8
    "00001000" when "00111001", -- 9
    "00000000" when "00111000", -- 0
    "11111111" when others;     -- apagado

end a_decodASCIIa7s;

```

Además, se ha añadido la conexión del segmento del punto decimal, a la patilla “N13” de la tarjeta BASYS 2, en el fichero de asociaciones.ucf.

```

# Conexiones de los DISPLAYS

NET "SEG7<0>" LOC = "L14"; # señal = CA
NET "SEG7<1>" LOC = "H12"; # Señal = CB
NET "SEG7<2>" LOC = "N14"; # Señal = CC
NET "SEG7<3>" LOC = "N11"; # Señal = CD
NET "SEG7<4>" LOC = "P12"; # Señal = CE
NET "SEG7<5>" LOC = "L13"; # Señal = CF
NET "SEG7<6>" LOC = "M12"; # Señal = CG
NET "SEG7<7>" LOC = "N13"; # Señal = CP

```

6. CONCLUSIÓN

Durante este proyecto, basado en un decodificador de la señal RDS, hemos aprendido bastante sobre circuitos electrónicos, ya que, durante varias semanas, nos centramos en la parte analógica, montando circuitos, calculando los valores de los componentes que más tarde íbamos a probar, realizando medias, y llevando a la práctica todos los conocimientos previos sobre circuitos. Pero sin lugar a duda, con lo que más hemos aprendido, es cuando nos ha sucedido algún error, el hecho de no saber que ocurre, e intentar solucionarlo.

Por otra parte, las últimas semanas del curso, el trabajo se centró en la parte digital, programando los diferentes módulos con los que se compone el circuito. Esta parte se ha desarrollado en un tiempo menor que el asignado, ya que teníamos un buen nivel en el lenguaje de VHDL. No obstante, ha sido una asignatura muy útil para el Grado de Tecnologías y Servicios de Telecomunicación porque te permite poner por primera vez aplicar los conocimientos teóricos sobre electrónica a la práctica.

MEMORIA PROYECTO: DECODIFICADOR DE LA SEÑAL RDS



JAVIER LÓPEZ INIESTA DÍAZ DEL CAMPO

ANDRES CASTILLEJO AMESCUA

CIRCUITOS ELECTRÓNICOS – CELT
GITST