UNIVERSIDAD DE VALLADOLID

ESCUELA TECNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN MENCIÓN EN SISTEMAS ELECTRÓNICOS

DISEÑO DE UNA INTERFAZ ANALÓGICA PARA SOFTWARE-RADIO CON FPGA

Autor: Daniel Santos Sánchez

Tutor: Jesús Arias Álvarez

Índice

1. Idea básica
   - 1.1. Esquema general
   - 1.2. Principales componentes
2. Diseño de la placa prototipo
   - 2.1. Diseño del filtro
   - 2.2. Adición de transformadores
   - 2.3. Puente H de MOSFET
   - 2.4. Alimentación del ADC
   - 2.5. Alimentación del DAC
3. Diseño en Proteus y montaje de la placa base
4. Pruebas del Hardware
   - 4.1. Generación de señal (prueba DAC)
     - 4.1.1. Pines
     - 4.1.2. Módulo principal
     - 4.1.3. PLL
     - 4.1.4. Módulo system
     - 4.1.5. Onda de sierra
     - 4.1.6. Sinusoide
     - 4.1.7. Visualización de la señal
     - 4.1.8. Mejora en el uso de memoria
   - 4.2. Captura de señal y modulación en frecuencia (prueba ADC y ADC + DAC)
   - 4.3. Audio PWM
5. Conclusiones
6. Bibliografía

Agradecimientos

A mi familia y en especial a mi padre, por toda la paciencia que han tenido y por los buenos y los malos momentos que me han llevado hasta aquí.

A Violeta, por todo.

A mi tutor Jesús, por todo lo que me has enseñado, por tu paciencia, por tu sabiduría y por hacer que cosas muy difíciles parezcan tan fáciles.

A mis amigos, por haberme ayudado en tantos momentos difíciles.

A la ETSIT, por haberme permitido formar parte de esta comunidad y enseñarme tanto estos años.

A todos los compañeros que he tenido en la ETSIT, ayudándonos todo esto ha sido más fácil.

Resumen

El mundo radiofónico fue el de los primeros medios de difusión que se transmitió a través de las ondas, base de las telecomunicaciones. Los cambios acaecidos en los últimos años lo han cambiado por completo, haciéndolo accesible, por la parte del emisor, a millones de personas.

Con este trabajo se pretende desarrollar un equipo que ayude en ese desempeño en diversas tareas y abierto a incorporar otras.

Abstract

Radio was one of the first wavelengths transmited and constituted the basis of telecommunications. The changes that have occurred in the recent years have completely changed it, making possible for millions of people to become emitters.

The intention of this work is to develop a device that helps to perform diverse tasks while maintaining the possibility of incorporating additional ones.

1. Idea básica

1.1 Motivación

Con la difusión de contenido por parte de mucha gente anónima cada vez se oye hablar más del software radio. Es un software que permite tener funciones de radio con dispositivos programables. Esto permite una mayor flexibilidad puesto que se puede programar según las necesidades cambiantes.

Puesto que para procesar y manipular señales mediante software se necesitan que estén en formato digital son imprescindibles las características del ADC y del DAC como fronteras entre el mundo analógico y el digital. El ADC transformará una señal analógica de entrada para su manipulación y después el DAC transformará esa señal digital manipulada en una señal analógica de salida.

Como se va a trabajar con frecuencias menores de 15 MHz se necesitará al menos una frecuencia de muestreo de $f = 2B = 2 \cdot 15 = 30\ MHz$, condiciones que un ADC y un DAC cumplen sobradamente.

1.2 Esquema general

Por lo dicho anteriormente la idea básica es diseñar y montar una placa con un ADC y un DAC que, conectada mediante pines a otra placa FPGA (con la que se manipulará la señal digital) pueda dar pie a desarrollar una aplicación de interés. Este sería el esquema general:

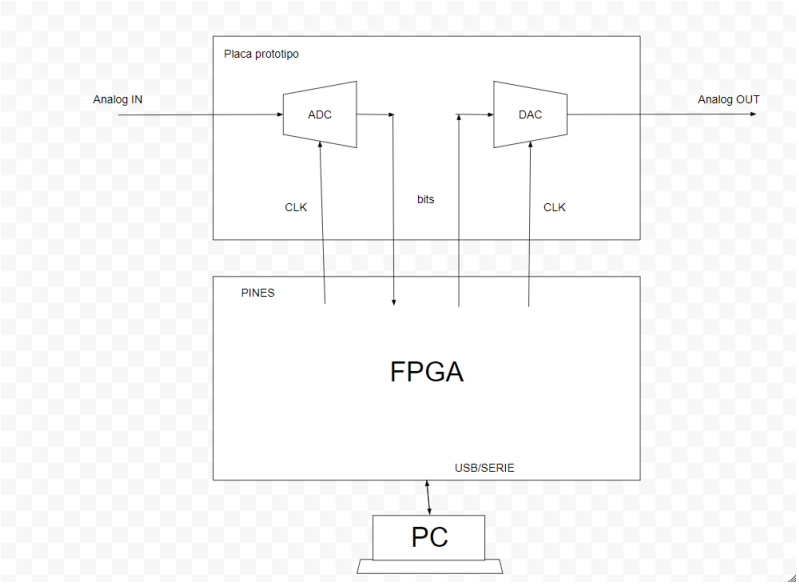


Figura 1.1: Esquema general

1.3 Principales componentes

-FPGA

La FPGA usada es la ICE40HX1K (Lattice). Sus características más reseñables son las siguientes:

- 12 MHz + PLL
- RAM 8192 bytes
- USB serie máximo 3 Mbps

-ADC

El ADC usado es el ADC10065. Sus características principales son las siguientes:

- 10 bits de resolución.
- Tasa de conversión de 65 Msps.
- Entrada analógica diferencial.
- STBY permite ponerlo en stand by ("1").
- DF permite ponerlo sin signo ("0") o en complemento a dos ("1").
- IRS permite elegir el rango de entrada entre 2,0 Vpp ("1"), 1,5 Vpp ("0") y 1,0 Vpp ("Floating").

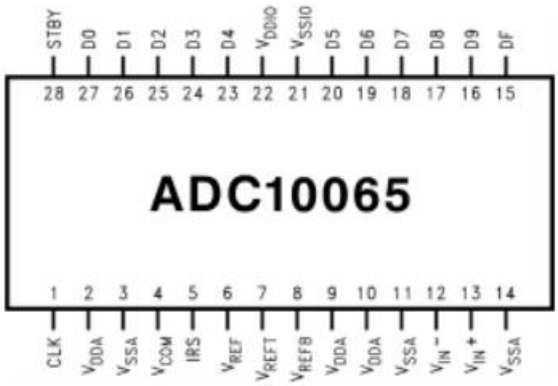


Figure 1. TSSOP Package
See Package Number PW0028A

Figura 1.2: Encapsulado del ADC

-DAC

El DAC usado es el AD9740. Contiene:

- 10 bits de resolución.
- Tasa de conversión de 165 Msps.
- Salida diferencial analógica
- SLEEP permite apagarlo (“0”).
- MODE permite sin signo (“0”) o en complento a dos (“1”).

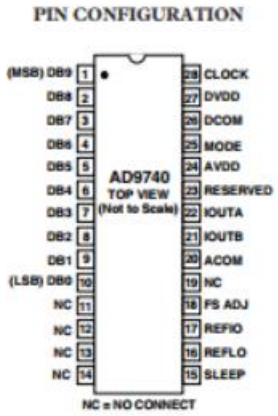


Figura 1.2: Encapsulado del DAC

2. Diseño de la placa prototipo

Aparte de los componentes principales, se añadirán otros secundarios necesarios para el buen funcionamiento del prototipo y para que el diseño esté abierto a más aplicaciones más allá de la que se va a desarrollar en este trabajo.

- Filtros tanto de entrada como de salida.
- Transformadores en la entrada y la salida para la conversión señal modo común a diferencial en la entrada y de diferencial a modo común en la salida.
- Conector BNC en la entrada y en la salida.
- Puente H con transistores MOSFET para amplificar salida de audio.
- Diversos componentes pasivos para la correcta configuración del ADC y del DAC.

2.1 Diseño del filtro

Se realizan primero los cálculos para una frecuencia de corte de 15 MHz y un factor de calidad Butterworth de 0,7. Teniendo en cuenta además que la resistencia elegida debe estar por valores de ~50 Ω por ser los más usados para las configuraciones de los dispositivos y que el filtro va a ser un RLC de segundo orden se tiene que:

$$X_L = X_C = QR$$

$$QR = 0{,}7 \cdot 50 = 35$$

$$QR = 35 = \frac{1}{2\pi f_C C} = X_C$$

$$C = \frac{1}{2\pi f_C 35} = \frac{1}{2\pi \cdot 20 \cdot 10^6 \cdot 35} = 227{,}36\ pF$$

$$QR = 35 = 2\pi f_C L = X_L$$

$$L = \frac{35}{2\pi f_C} = 371{,}36\ nF$$

Se eligen valores comerciales disponibles cercanos y se comprueba si no se aleja mucho del objetivo. Para L = 330 nF y para C = 220 pF y teniendo en cuenta que $X_L = X_C$:

$$f_C = \sqrt{\frac{1}{4\pi^2 LC}} = \sqrt{\frac{1}{4\pi^2 \cdot 330 \cdot 10^{-9} \cdot 220 \cdot 10^{-12}}} = 18{,}68\ MHz$$

Se comprueba también el valor de R:

$$R = \frac{X}{Q} = 55{,}08\ \Omega$$

Puesto que esos valores no se alejan mucho de lo deseado se da por buena la elección de esos componentes.

Se añade al filtro de entrada un condensador de 33 nF en serie para filtrar la componente de continua.

Se comprueba ahora su respuesta en frecuencia en una simulación en Proteus:

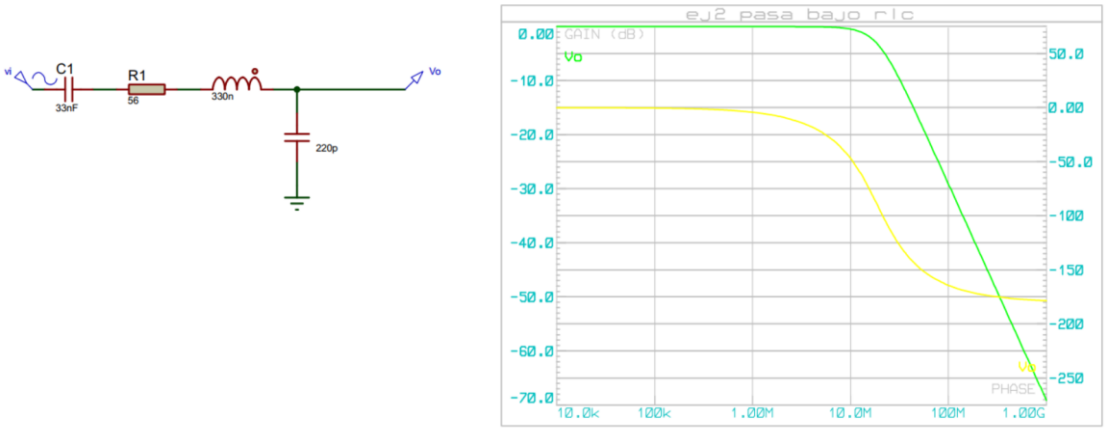


Figura 2.1: Esquema y respuesta en frecuencia del filtro

Se puede observar que acota las frecuencias al régimen de trabajo que se quiere.

2.2 Adición de transformadores

Se añaden transformadores para transformación “single-ended” ↔ diferencial configurándolos según las especificaciones recomendadas. Esto es porque tanto el DAC como el ADC funcionan mejor en modo diferencial. En este modo permite rechazar el ruido común puesto que se elimina al hacer la diferencia, aumenta la precisión ya que una diferencia es menos susceptible a errores en la referencia o en los equipos de medición, se logra mayor rango dinámico y reduce las interferencias.

2.3 Conectores BNC

Se añaden dos conectores BNC por su facilidad para la conexión con los equipos que se usarán tanto para generar señales como para hacer captarlas y poder hacer las comprobaciones pertinentes. Uno en la entrada y otro en la salida.

2.4 Puente H de MOSFET

Para una posible salida de audio se utilizará un puente H dejando una señal como referencia. Se conseguirá así doblar la tensión y por lo tanto cuadriplicar la potencia.

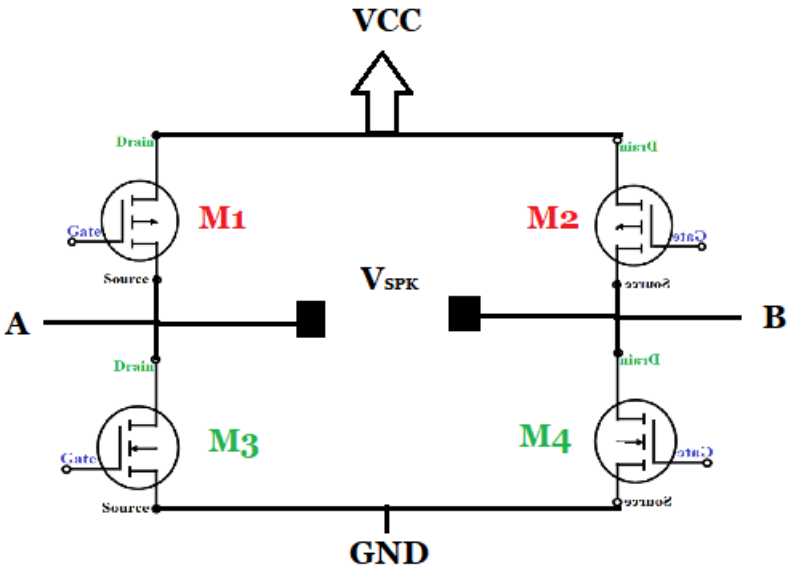


Figura 2.2: Esquema de un puente H

El funcionamiento es el siguiente. Teniendo en cuenta que $V_{SPK} = V_+ - V_-$:

- Si A y B son "0" M3 y M4 conducen y $V_{SPK} = 0 - 0 = 0$
- Si A = "1" y B = "0" M1 y M4 conducen y $V_{SPK} = V_{CC} - 0 = Vcc$
- Si A = "0" y B = "1" M3 y M2 conducen y $V_{SPK} = 0 - Vcc = - V_{CC}$
- Si A y B son "1" M1 y M2 conducen y $V_{SPK} = Vcc - Vcc = 0$

Por tanto es ideal para dos señales PWM en las que una conduzca para las partes positivas y otra para las negativas. Siempre se estaría transmitiendo tensión al altavoz y con la potencia aumentada.

Esta combinación es lo que se conoce como amplificador clase – D, que permite una alta eficiencia, un tamaño compacto y menor distorsión armónica.

2.5 Alimentación del ADC

Se usa las recomendadas por el fabricante:

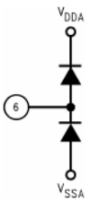
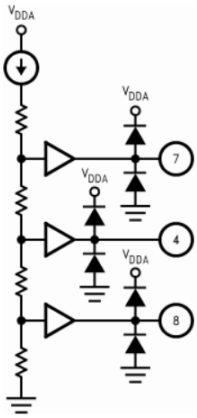
6	V_{REF}		Reference input. This pin should be bypassed to V_{SSA} with a 0.1 µF monolithic capacitor. V_{REF} is 1.20V nominal. This pin may be driven by a 1.20V external reference if desired. Do not load this pin.
7	V_{REFT}		These pins are high impedance reference bypass pins only. Connect a 0.1 µF capacitor from each of these pins to V_{SSA}. These pins should not be loaded. V_{COM} may be used to set the input common mode voltage, V_{CM}.
4	V_{COM}		
8	V_{REFB}		

Figura 2.3: Alimentaciones del ADC

ANALOG POWER			
2, 9, 10	V_{DDA}		Positive analog supply pins. These pins should be connected to a quiet 3.0V source and bypassed to analog ground with a 0.1 µF monolithic capacitor located within 1 cm of these pins. A 4.7 µF capacitor should also be used in parallel.
3, 11, 14	V_{SSA}		Ground return for the analog supply.
DIGITAL POWER			
22	V_{DDIO}		Positive digital supply pins for the ADC10065's output drivers. This pin should be bypassed to digital ground with a 0.1 µF monolithic capacitor located within 1 cm of this pin. A 4.7 µF capacitor should also be used in parallel. The voltage on this pin should never exceed the voltage on V_{DDA} by more than 300 mV.
21	V_{SSIO}		The ground return for the digital supply for the output drivers. This pin should be connected to the ground plane, but not near the analog circuitry.

Figura 2.4: Alimentaciones del ADC

Los pines V_{SS} serán obviamente conectados a tierra y los pines V_{DD} serán conectados a una tensión de 3,3 V mediante condensadores de desacoplo de 4,7 µF. Para evitar interferencias se pondrá un "choque" o alta inductancia para separar la alimentación analógica de la digital.

2.6 Alimentación del DAC

Se usarán también las recomendadas por el fabricante:

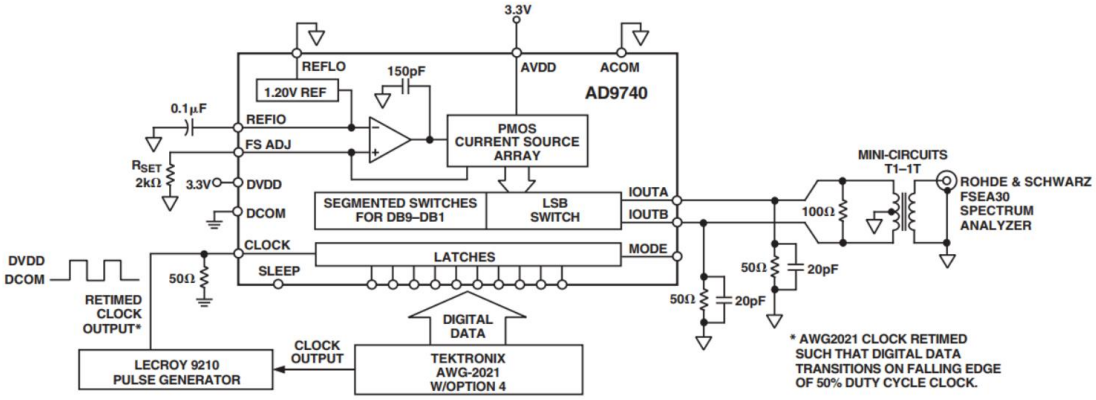


Figura 2.5: Alimentaciones del DAC

Como en el ADC, los pines V_{SS} serán obviamente conectados a tierra y los pines V_{DD} serán conectados a una tensión de 3,3 V mediante condensadores de desacoplo de 4,7 µF. Para evitar interferencias se pondrá un "choque" o alta inductancia para separar la alimentación analógica de la digital.

3. Diseño en Proteus y montaje de la placa base

Con todos los supuestos anteriores se procede al diseño en Proteus. Se conecta todo según lo dicho. A esto habría que añadir que cada salida y entrada digital del ADC y del DAC deben estar conectadas a los pines para tener conexión con la FPGA. Los componentes pasivos se hacen con medidas 0508 porque sobra espacio en la placa. Los encapsulados del ADC y del DAC están en el programa y la única huella que hay que hacer manualmente por no estar su encapsulado es la de los transformadores:

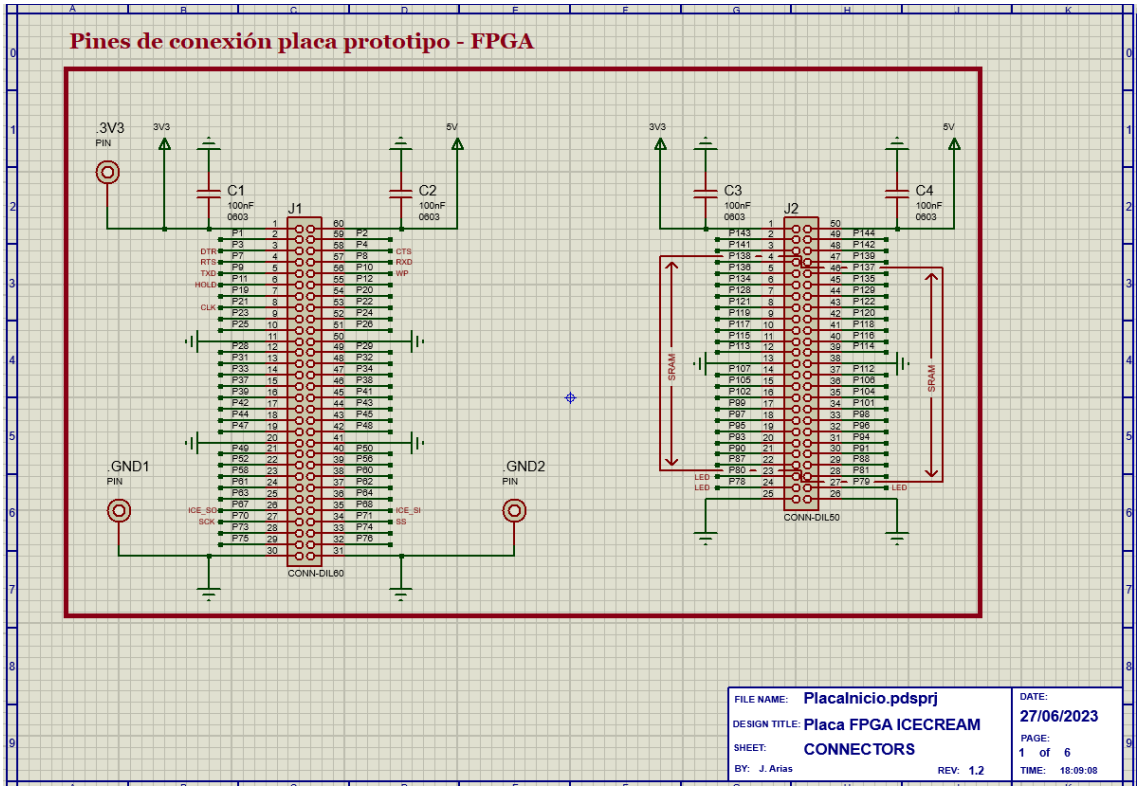


Figura 3.1: Diseño en Proteus de los pines de conexión

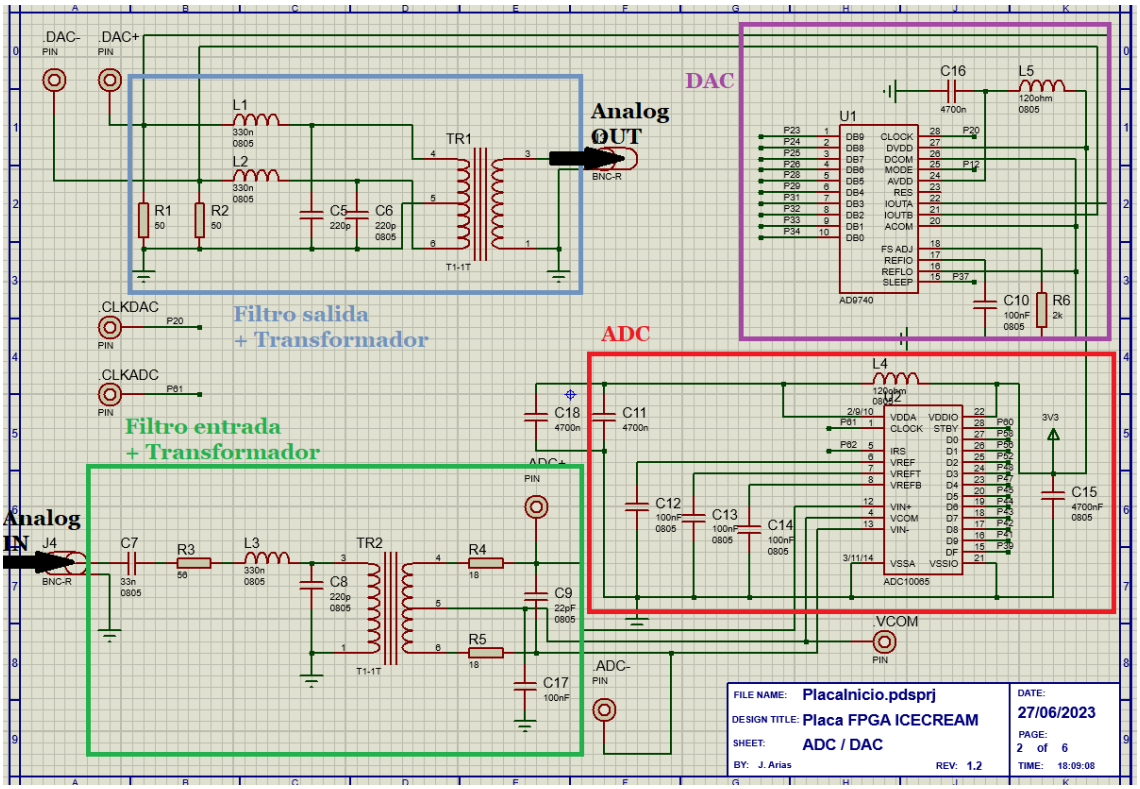


Figura 3.2: Diseño en Proteus de los filtros, el ADC y el DAC

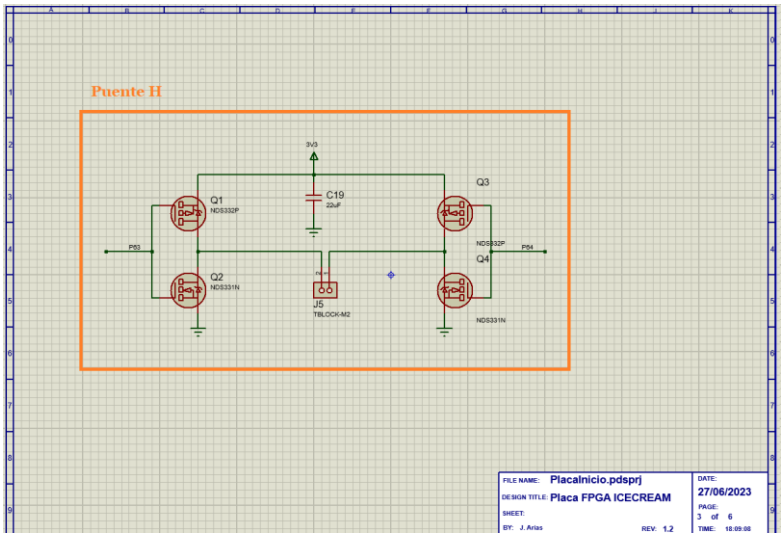


Figura 3.3: Diseño en Proteus del puente H

A la hora de realizar la PCB no hay muchas complicaciones por la sencillez del circuito. Se consigue que casi todas las pistas estén en la capa superior excepto unas pocas de poca longitud que no queda otra opción que pasarlas por debajo, que son las señaladas en amarillo:

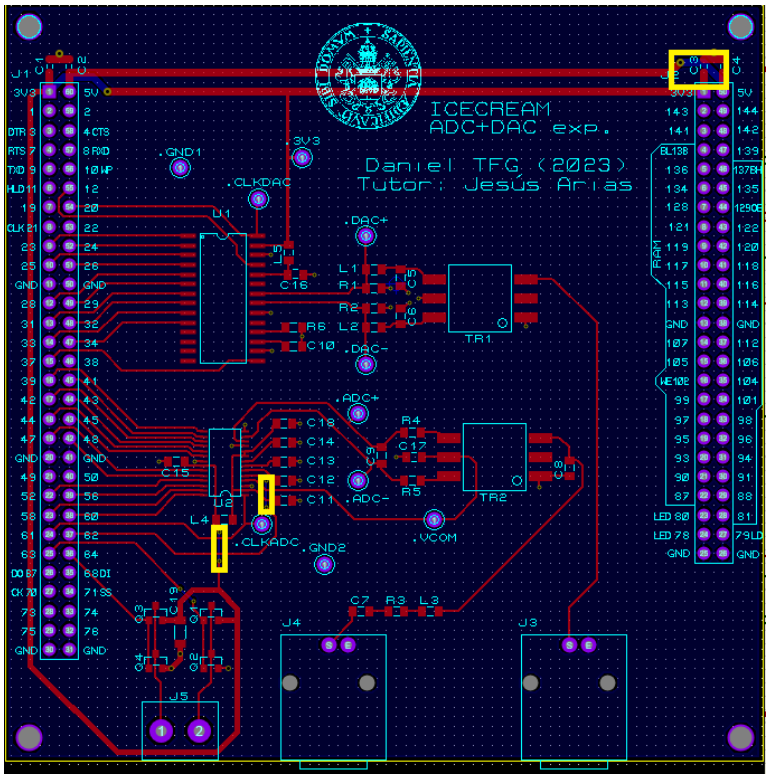


Figura 3.4: Diseño en Proteus de la huella de la PCB

Una vez acabado se encarga el diseño y una vez recibida la placa se procede al soldado de todos sus componentes. Una vez soldados los componentes se procede a introducir los pines en la placa de la FPGA, uniéndolo con la placa prototipo, para que así al soldarlos asegurarse de que encajan perfectamente. El resultado es este:

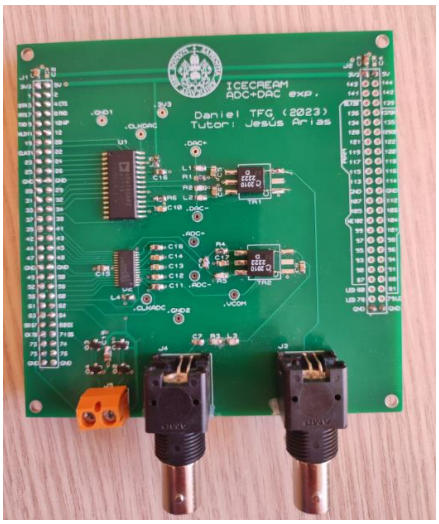


Figura 3.5: Vista de la placa prototipo

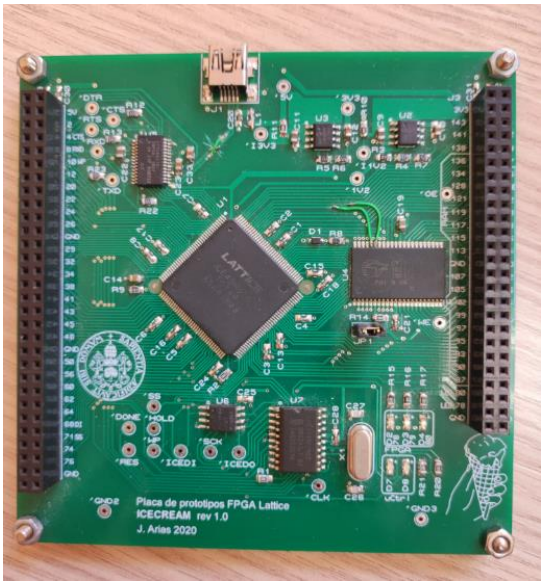


Figura 3.6: Vista de la placa FPGA

Figura 3.7: Vista de la placa prototipo enganchando con la placa FPGA

Figura 3.8: Vista de ambas placas

4. Pruebas del hardware

4.1 Generación de señal (Prueba DAC)

4.1.1 Pines

Sirve para que cada señal digital que pase por un pin se recoja en una variable:

```
set_io CLK 21

set_io DAC[0] 34
set_io DAC[1] 33
set_io DAC[2] 32
set_io DAC[3] 31
set_io DAC[4] 29
set_io DAC[5] 28
set_io DAC[6] 26
set_io DAC[7] 25
set_io DAC[8] 24
set_io DAC[9] 23
set_io DACCLK 20
set_io DACMODE 12
set_io DACSLEEP 37

set_io ADC[0] 58
set_io ADC[1] 56
set_io ADC[2] 52
set_io ADC[3] 48
set_io ADC[4] 47
set_io ADC[5] 45
set_io ADC[6] 44
set_io ADC[7] 43
set_io ADC[8] 42
set_io ADC[9] 41
set_io ADCCLK 61
set_io ADCSTBY 60
set_io ADCIRS 62
set_io ADCDF 39

set_io PWML 63
set_io PWMR 64

set_io LED[0] 78
set_io LED[1] 79
set_io LED[2] 80
```

Figura 4.1: Captura del código

4.1.2 Módulo principal

Se declaran las variables que llegan de los pines:

```
1   `include "pll.v"
2
3   module main(
4       input   CLK,
5       input   [9:0]ADC,
6       output   ADCCLK,
7       output  ADCSTBY,
8       output  ADCIRS,
9       output  ADCDF,
10      output  PWML,
11      output  PWMR,
12      output  [2:0]LED,
13      output [9:0]DAC,
14      output DACCLK,
15      output DACMODE,
16      output DACSLEEP
17
18  );
```

Figura 4.2: Captura del código

4.1.3 PLL

Que la FPGA tenga PLL permite aumentar la señal de reloj. Se usa el comando icepll para generar el módulo que permita aumentar la frecuencia hasta 60 MHz.

```
C:\Users\Daniel\Desktop\TFG\FPGA_nuevo>icepll -i 12 -o 60 -m
/**
 * PLL configuration
 *
 * This Verilog module was generated automatically
 * using the icepll tool from the IceStorm project.
 * Use at your own risk.
 *
 * Given input frequency:        12.000 MHz
 * Requested output frequency:   60.000 MHz
 * Achieved output frequency:    60.000 MHz
 */

module pll(
       input  clock_in,
       output clock_out,
       output locked
       );

SB_PLL40_CORE #(
                .FEEDBACK_PATH("SIMPLE"),
                .DIVR(4'b0000),         // DIVR =  0
                .DIVF(7'b1001111),      // DIVF = 79
                .DIVQ(3'b100),          // DIVQ =  4
                .FILTER_RANGE(3'b001)   // FILTER_RANGE = 1
       ) uut (
                .LOCK(locked),
                .RESETB(1'b1),
                .BYPASS(1'b0),
                .REFERENCECLK(clock_in),
                .PLLOUTCORE(clock_out)
                );

endmodule
```

Figura 4.3: Captura de la salida para la creación del módulo que hace uso del PLL

```
1  /**
2   * PLL configuration
3   *
4   * This Verilog module was generated automatically
5   * using the icepll tool from the IceStorm project.
6   * Use at your own risk.
7   *
8   * Given input frequency:        12.000 MHz
9   * Requested output frequency:   60.000 MHz
10  * Achieved output frequency:    60.000 MHz
11  */
12
13 module pll(
14     input  clock_in,
15     output clock_out,
16     output locked
17     );
18
19 SB_PLL40_CORE #(
20         .FEEDBACK_PATH("SIMPLE"),
21         .DIVR(4'b0000),      // DIVR =  0
22         .DIVF(7'b1001111),   // DIVF = 79
23         .DIVQ(3'b100),       // DIVQ =  4
24         .FILTER_RANGE(3'b001)   // FILTER_RANGE = 1
25     ) uut (
26         .LOCK(locked),
27         .RESETB(1'b1),
28         .BYPASS(1'b0),
29         .REFERENCECLK(clock_in),
30         .PLLOUTCORE(clock_out)
31         );
32
33 endmodule
34
```

Figura 4.4: Captura del código que asocia los pines con variables

Como siempre se va a querer tener esa señal de reloj se llama al módulo desde el main para introducir la señal de reloj de 12 Mhz y que devuelva 60 MHz:

```
19      wire clk;
20      pll pll0 (.clock_in(CLK),.clock_out(clk));
```

Figura 4.5: Captura del código que asocia los pines con variables

4.1.4 Módulo system

Será el módulo que ejecute el trabajo. Se empieza declarando las variables que le tienen que llegar del main:

```
1
2       module system(
3           input   clk,
4           input [9:0]adc,
5           output adcclk,
6           output  pwm1,
7           output  pwm2,
8           output  led1,
9           output  led2,
10          output dacclk,
11          output [9:0]dac
12      );
```

Figura 4.6: Captura del código que asocia los pines con variables

Y desde el main se invoca al módulo para que sus variables recojan el valor de las del main que habían recogido el valor de los pines:

```
26      system system1(.clk(clk),.pwm1(PWML),.pwm2(PWMR),.led1(LED[0]),.led2(LED[1]),
27        .dacclk(DACCLK),.dac(DAC),.adcclk(ADCCLK),.adc(ADC)
```

Figura 4.7: Captura del código que asocia los pines con variables

4.1.5 Onda de sierra

Se crea un registro de 24 bits llamado fase que se inicializará en 0. Ese registro irá aumentando en cada ciclo por el valor de frecuencia, también de 24 bits. Eso hará que el valor de fase aumente de manera lineal hasta llegar al máximo, volviendo a empezar a partir de ahí:

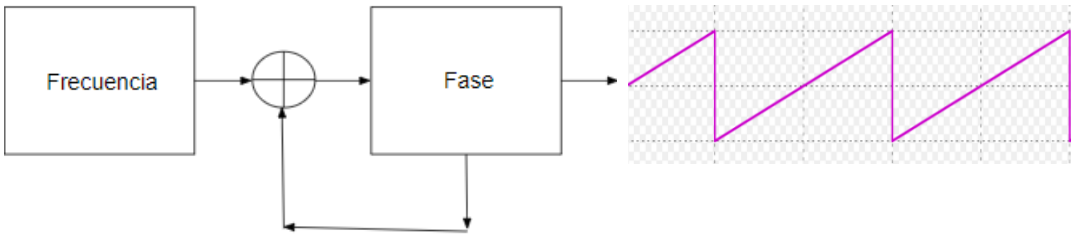


Figura 4.8: Esquema para generar la señal diente de sierra

```
20
21    reg [23:0] fase = 0;
22    wire [23:0] frecuencia = valor;
23
```

Figura 4.9: Captura del código

```
33
34    always @ (posedge clk)
35    begin
36        fase <= fase + frecuencia;
37    end
38
```

Figura 4.10: Captura del código

La frecuencia dependerá del ciclo de reloj, recordemos que de 60 MHz, al haber aprovechado el PLL para multiplicar la frecuencia por 5. Y del cociente de la frecuencia y el número de registros posibles:

$$f_{SW} = f_{CLK} \cdot \frac{f_{REC}}{2^{24}}$$

4.1.6 Sinusoide

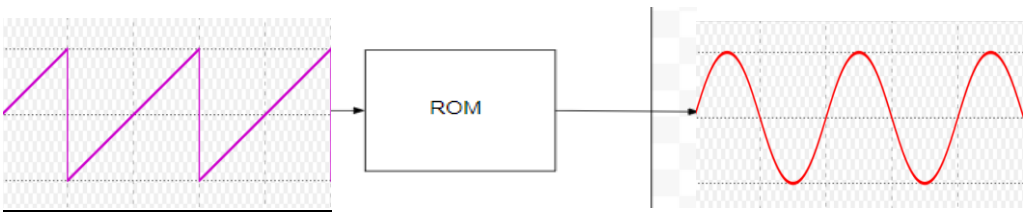


Figura 4.11: Esquema para la generación del seno

Para ello, lo primero que hay que hacer es una función que, introduciendo la dirección de memoria pueda sacar los datos de la sinusoide, que por lo que se crea la función genrom. Esta función crea una matriz de registros en la memoria RAM a partir de un archivo que contiene los datos de la función seno, por lo que en la práctica actúa como una ROM (no se puede escribir, solo leer los datos introducidos en la matriz a partir del archivo, que luego son leídos).

```
50
51  //-----------------------------------------------------------------------
52  //-- Memoria ROM genérica (síncrona)
53  //-----------------------------------------------------------------------
54
55  module genrom (        //-- Puertos
56      input clk,                      //-- Señal de reloj global
57      input wire [AW-1: 0] addr,      //-- Direcciones
58      output reg [DW-1: 0] data       //-- Dato de salida
59  );
60  //-- Parametro: Nombre del fichero con el contenido de la RAM
61  parameter INITFILE = "seno.hex";
62  parameter AW = 10;
63  parameter DW = 10;
64
65    //-- Memoria
66    reg [DW-1: 0] rom [0: (1<<AW)-1];
67
68    //-- Lectura de la memoria
69    always @(posedge clk) begin
70      data <= rom[addr];
71    end
72
73  //-- Cargar en la memoria el fichero ROMFILE
74  //-- Los valores deben estan dados en hexadecimal
75  initial begin
76    $readmemh(INITFILE, rom);
77  end
78
79  endmodule
80
```

Figura 4.12: Captura del código

Como se puede ver en la imagen, la ROM cuenta con diez bits de entrada (dirección) y diez de salida (valor de la sinusoide). Se usan diez bits porque se tienen $2^{10} = 1024$ muestras. Se usan los diez bits más significativos de la fase como dirección de memoria de la ROM:

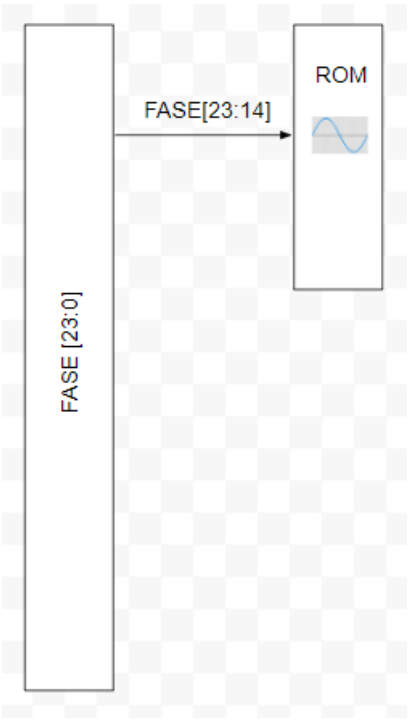


Figura 4.13: Esquema para visualizar los bits usados por la ROM

Esto lo hacemos invocando al módulo genrom desde el módulo system, donde como se ha dicho se introducen los diez bits más significativos de la fase como dirección de memoria y se recogen los valores en la salida que luego va al DAC:

```
36    genrom #(
37        .INITFILE("seno.hex"),
38        .AW(10),
39        .DW(10)
40        ) rom1 (.clk(clk),.addr(fase [23:14]),.data(dac));
```

Figura 4.14: Captura del código

Se le da un valor a la frecuencia:

```
16    wire [23:0] frecuencia = 100000;
```

Figura 4.15: Captura del código

Se calcula teóricamente la frecuencia que debería tener nuestro seno recordando la fórmula citada anteriormente:

$$f_{SW} = f_{CLK} \cdot \frac{f_{REC}}{2^{24}} = 60 \cdot 10^6 \frac{10^5}{2^{24}} = 357{,}63\ KHz$$

Se simula con Simulink para comprobar si se corresponde con los cálculos teóricos.

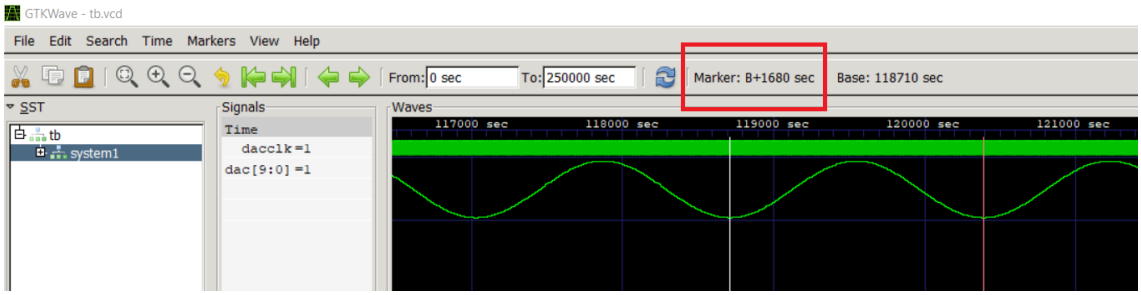


Figura 4.16: Señal simulada en Simulink

Se comprueba que hay 168 ciclos de reloj por cada periodo. Eso significa que la frecuencia es:

$$f_{SW} = \frac{f_{CLK}}{168} = 357{,}14\ KHz$$

Que se corresponde con los cálculos teóricos.

4.1.7 Visualización de la señal

Para visualizar la señal solo hay que recoger la señal de salida del DAC en el conector BNC (después de haber pasado por el filtro y el transformador). Se comprueba en el osciloscopio si tiene la frecuencia esperada:

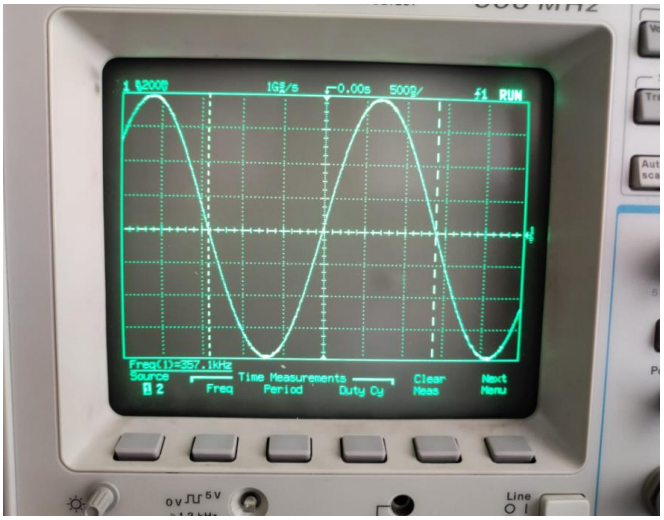


Figura 4.17: Señal visualizada en el osciloscopio

Que como se puede ver, tiene la frecuencia esperada.

4.1.8 Mejora en el uso de memoria

Como bien se sabe los valores de un periodo de una función seno se repiten. Con los valores de un cuarto de periodo se tienen todos los demás. Se va a aprovechar esto para dividir entre cuatro la memoria necesaria al almacenar solamente estos valores.

Lo primero es que como ahora tenemos una cuarta parte de las muestras (256), solo necesitamos ocho bits de dirección. Los otros dos bits (los más significativos) se usarán para saber en qué cuarto de periodo estamos:

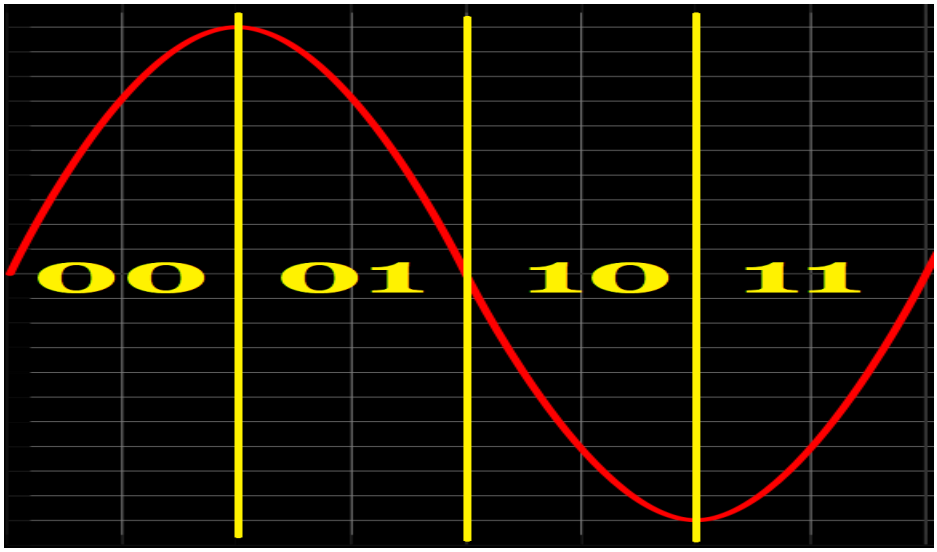


Figura 4.18: División de un periodo de seno en cuatro partes

La utilidad de fase[23], el bit más significativo, está clara: nos indica el signo, por lo que nos serviría de flag para invertir la señal de salida de la ROM.

La utilidad de fase[22] será hacer que lea los datos de la memoria al revés, por lo que afecta a la dirección que introducimos en la ROM. Esto puede hacerse invirtiendo los bits de dirección cuando fase[22] = 1. Se pone un ejemplo con solo tres bits (más el bit indicador) para ilustrarlo:

Bit indicador	Fase	Dirección
0	00	00
0	01	01
0	10	10
0	11	11
1	00	11
1	01	10
1	10	01
1	11	00

Se ve claramente como fase está haciendo una señal diente de sierra pero como dirección hace una señal triangular, recorriendo el espacio de bits en sentido contrario en su segunda fase:

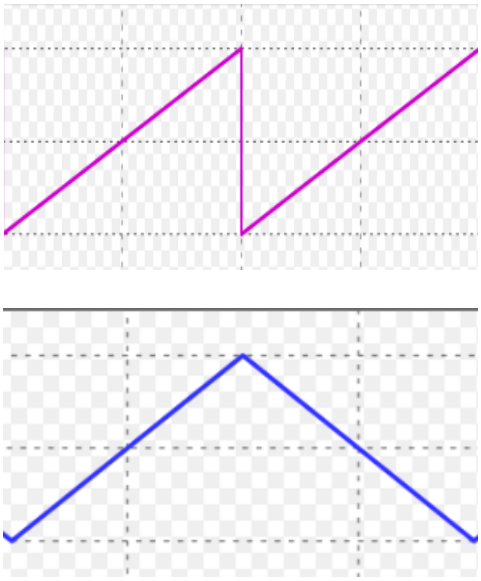


Figura 4.19: Señal diente de sierra y su modificación

Cuando se quiere invertir un dato sin un bit vale 1 se utiliza una puerta XOR, que utilizaremos tanto para cambiar la dirección cuando fase[22] valga 1, como para cambiar la salida cuando fase[23] valga 1:

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Figura 4.20: Esquema de puerta XOR

Se queda, por lo tanto, este esquema:

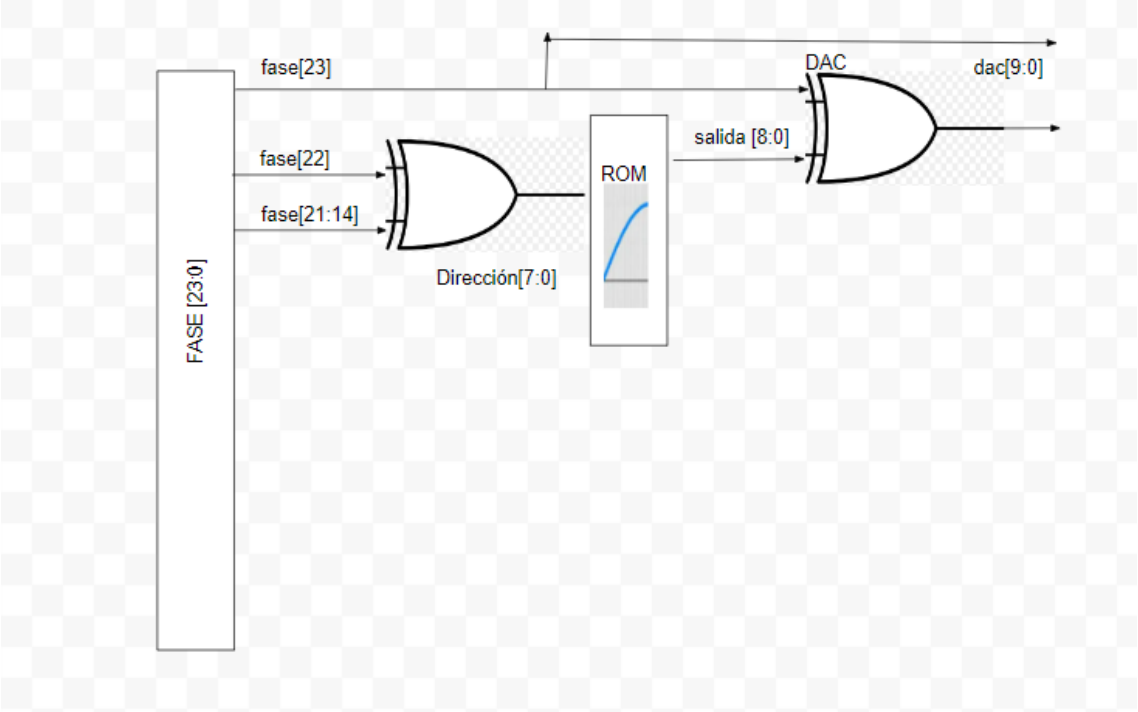


Figura 4.21: Esquema para hacer posible el uso de solo un cuarto de periodo del seno

Esto implica varios cambios en la función. Lo primero la declaración de una dirección y una salida, puesto que ya no son lo mismo que la fase y la entrada al DAC. Serán estas las que interactúen con la función ROM, que ahora tendrá solo 8 bits de entrada y 9 bits de salida. Además de la condición para que se inviertan la entrada (fase[22]=1) y la salida (fase[23]=1):

```
14    reg [23:0] fase = 0;
15    wire [7:0] direccion = fase [22]? ~fase[21:14] : fase[21:14];
16    wire [23:0] frecuencia = 100000;
```

Figura 4.22: Captura del código

```
26    genrom #(
27        .INITFILE("seno.hex"),
28        .AW(8),
29        .DW(9)
30        ) rom1 (.clk(clk),.addr(direccion),.data(salida));
31
32    wire [8:0] salida;
33    assign dac[9] = fase[23];
34    assign dac[8:0] = fase [23]? ~salida : salida;
```

Figura 4.23: Captura del código

Se comprueba en Simulink. Teniendo, evidentemente, la misma frecuencia que antes:

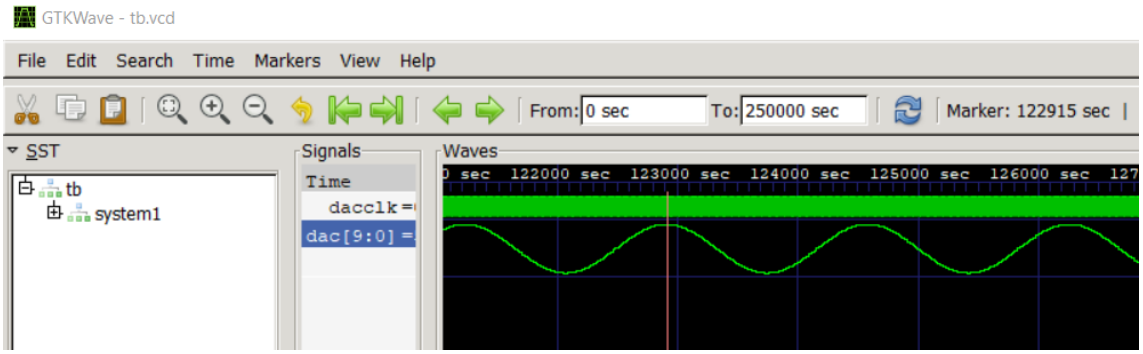


Figura 4.24: Señal de salida en Simulink

Se comprueba también la señal de salida a través del osciloscopio. Cambiando antes el valor MODE del DAC para que tenga en cuenta el signo:

```
25    assign DACMODE = 1;
```

Figura 4.25: Captura del código

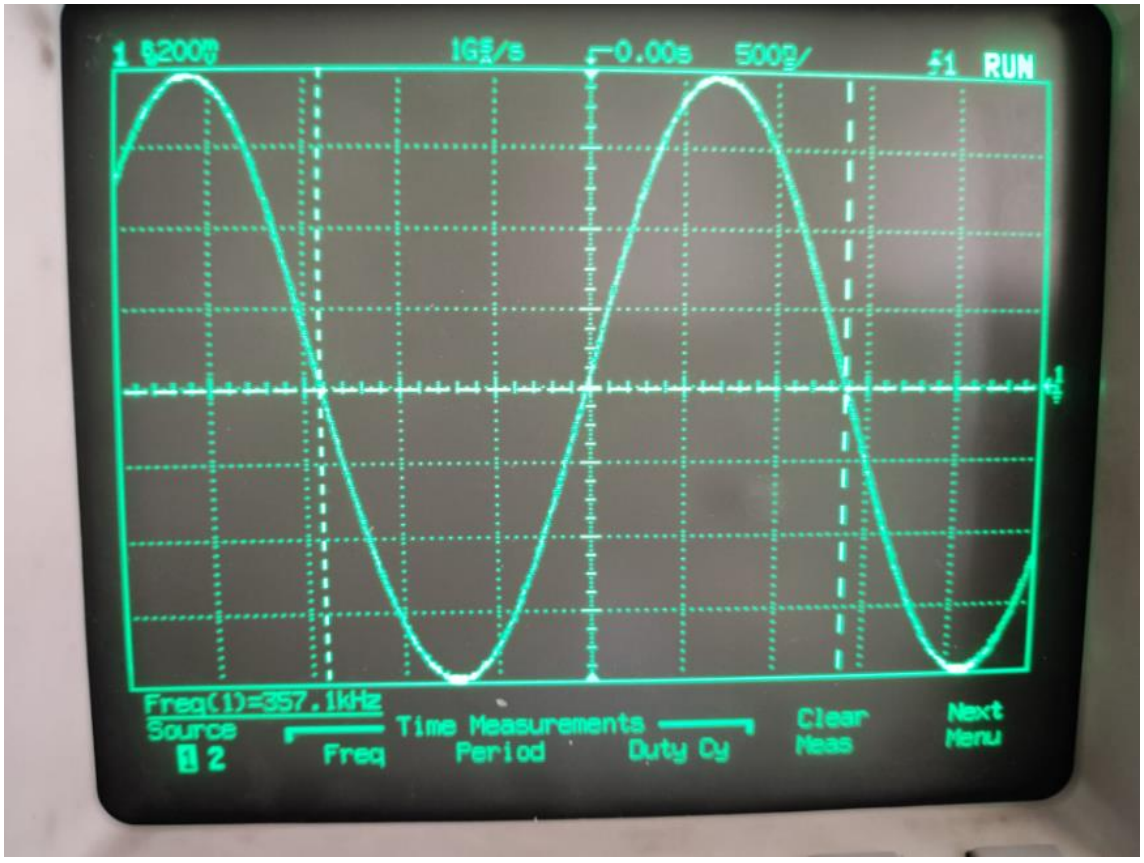


Figura 4.26: Captura de la señal de salida en el osciloscopio

Vuelve a tener la misma frecuencia, la que se calculó teóricamente.

4.2 Captura de señal y modulación en frecuencia (prueba ADC y ADC + DAC)

El último paso sería no poner un valor fijo a la frecuencia sino que esta dependiera de un valor de entrada. El ADC captaría la señal y una vez codificada se introduciría en diez de los veinticuatro bits de la frecuencia, dependiendo entre que valores se quiere que varíe. Se daría por tanto un valor fijo a catorce bits y los otros diez dependerían de la señal de entrada.

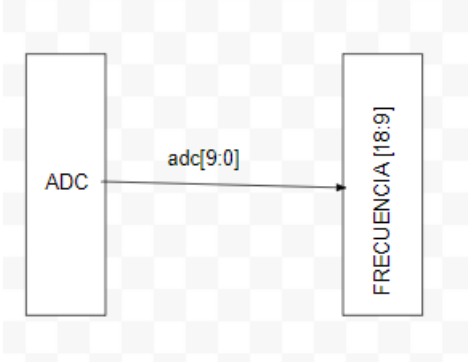


Figura 4.27: Esquema para visualizar la señal de entrada con la frecuencia

Se hace con este esquema. Se piensa por ejemplo en darle a los cinco primeros bits el valor 00001, los diez siguientes dependientes de la frecuencia y los nueve últimos igual a cero.

Con eso variará desde

$$f_1 = 60 \cdot 10^6 \frac{2^{19}}{2^{24}} = 1{,}88\ MHz$$

hasta

$$f_2 = 60 \cdot 10^6 \frac{\sum_{k=9}^{19} 2^k}{2^{24}} = 3{,}75\ MHz$$

Se introducen estos cambios en el módulo system:

```
75    wire [23:0] frecuencia;
76    assign frecuencia [8:0] = 0;
77    assign frecuencia [18:9] = adc;
78    assign frecuencia [23:19] = 5'b00001;
```

Figura 4.28: Captura del código

Se comprueba en el osciloscopio:

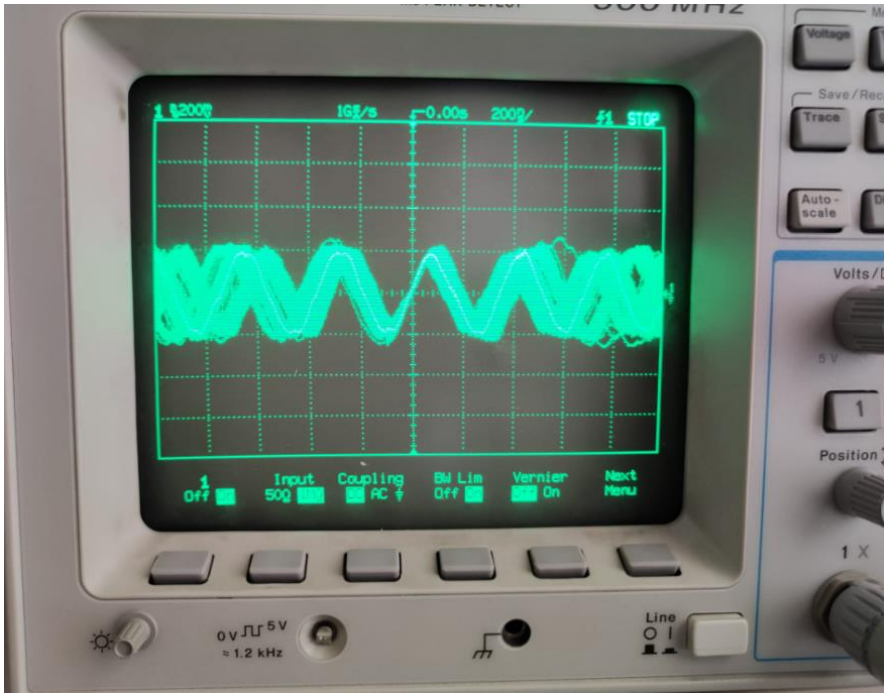


Figura 4.29: Captura en el osciloscopio de la señal modulada

Se tiene dificultad para medirlo pero sí se puede ver que aproximadamente varía entre las frecuencias calculadas.

4.3 Audio PWM

Para poder reproducir sonido a través del altavoz se usará el puente H enviándole una señal PWM teniendo así un amplificador clase – D, como se señaló en el apartado del diseño.

Una señal PWM es una señal que emite pulsos cuya duración depende del valor de la señal. Esto se puede conseguir comparándolo con una señal diente de sierra. Mientras la señal diente de sierra sea menor que la señal introducida se emitirá un pulso que cesará cuando la señal diente de sierra la supere:

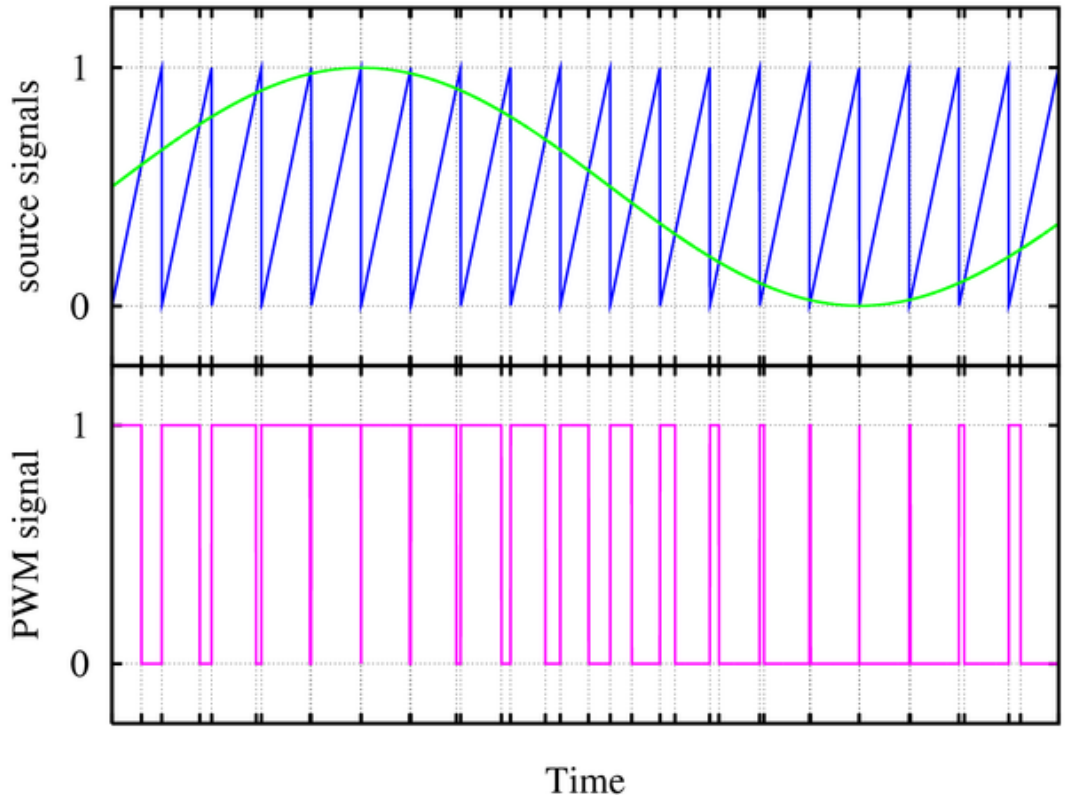


Figura 4.30: Señal de entrada, de sierra y PWM

Esto se conseguirá primero hallando el valor absoluto de la señal. El bit de signo de dicha señal indicará si dejar los bits restantes igual o complementar a 2 en caso de que sea negativo. Un registro introducirá un "1" cada vez que el diente se sierra comience y que introducirá un "0" cuando el diente de sierra y el valor absoluto de la señal se igualen (esto significará que el diente de sierra supera el valor de la señal).

Pero para poder aprovechar el puente H se deben generar dos señales, PWM1 que solo se activará si la señal es positiva y PWM2 que solo se activará si la señal es negativa.

La señal de reloj se cambiará a la frecuencia del diente de sierra:

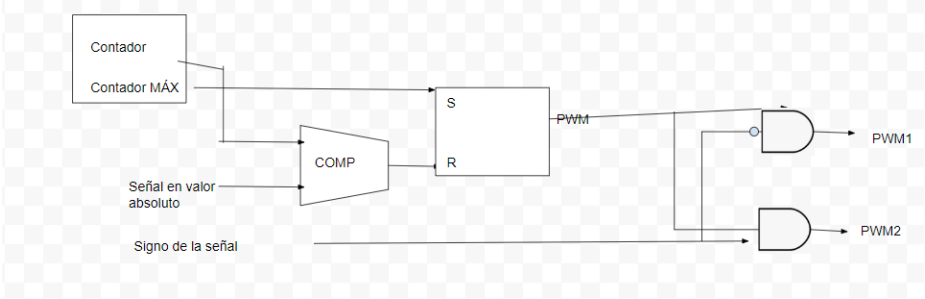


Figura 4.31: Esquema para obtener las dos señales PWM

Se crea el módulo para llevar a cabo estas tareas:

```
82
83  module pwm (
84      input clk,
85      input [NBITS-1:0] d,
86      output pwm1,
87      output pwm2,
88      output clko
89  );
90
91  parameter NBITS = 9; //señal de entrada
92  reg [NBITS-2:0] contador = 0; //contador con los mismos bits que la señal de entrada menos el bit de signo
93
94  always @ (posedge clk)
95      contador <= contador + 1; //señal diente de sierra
96
97  assign clko = ~contador[NBITS-2]; //nueva señal de reloj
98  wire [NBITS-2:0] abs = d[NBITS-1] ? ((~d[NBITS-2:0])+1) : d[NBITS-2:0]; //valor absoluto
99
100 reg pwm;
101
102 always @ (posedge clk)
103     if (contador == abs) pwm <= 0; //reseteo cuando la señal diente de sierra alcanza a la señal de entrada
104     else if (contador == 0) pwm <= 1; //set en el momento en el que comienza cada periodo
105
106 assign pwm1 =~((~d[NBITS-1]) & pwm); //Puerta AND que activa si la señal es positiva
107 assign pwm2 = ~((d[NBITS-1]) & pwm); //Puerta AND que activa si la señal es negativa
108
109 endmodule
110
```

Figura 4.32: Captura del código

Se invoca desde el módulo system, para introducir la señal de reloj y la señal de entrada y recoger la nueva señal de reloj y las señales PWM producidas que permiten escuchar el tono por el altavoz y visualizarlo en el osciloscopio:

```
14      wire clks;
15      pwm #(.NBITS(11)) pwm0( .clk(clk),.clko(clks),.d({dac,1'b0}),.pwm1(pwm1),.pwm2(pwm2));
16
```

Figura 4.33: Captura del código

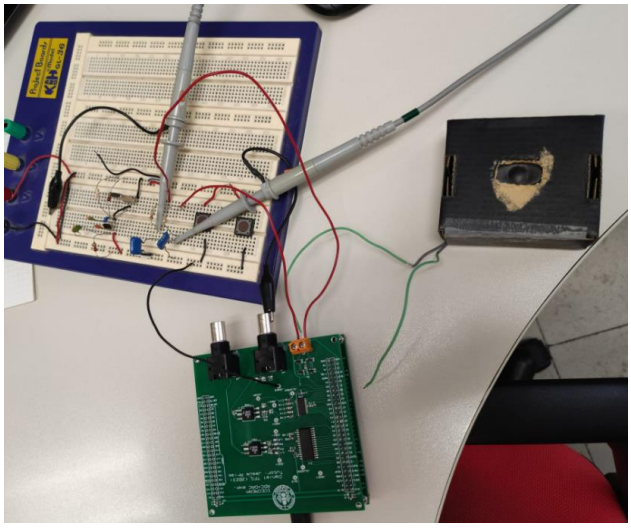


Figura 4.34: Vista del dispositivo conectado al altavoz

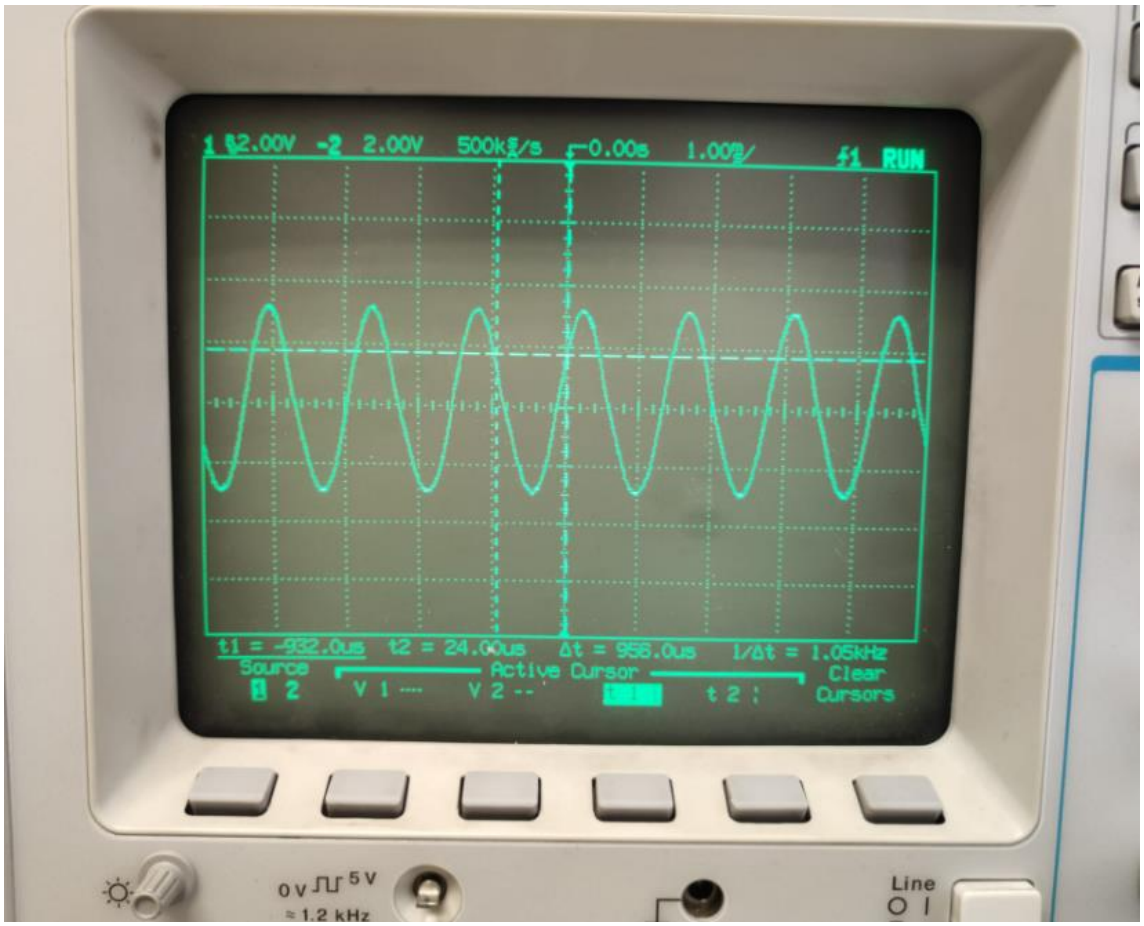


Figura 4.35: Captura del osciloscopio del tono de salida

5. Conclusiones

El mundo de hoy nos ofrece multitud de posibilidades. El mundo de la electrónica muchas. Se ha visto como con una placa sencilla y las ventajas y posibilidades que ofrecen FPGA con ADC y DAC se pueden hacer multitud de aplicaciones. En este caso, en el mundo de la radio. En este trabajo se han mostrado varias de ellas.

6. Bibliografía

Alcalde San Miguel, P. (2008). *Electrónica General.* Paraninfo.

Analog Devices. *10-Bit, 210 MSPS TxDAC D/A Converter.* *https://www.analog.com/media/en/technical-documentation/data-sheets/ad9740.pdf*

Chávez, J. *Manual de Verilog.* *https://marceluda.github.io/rp_dummy/EEOF2018/verilog.pdf*

Mitola, J. (2001). *Software Radio Technologies: Selected Readings.*

Lattice Semiconductor. iCE40 HX-Series Ultra Low-Power FPGA Family. https://pdf1.alldatasheet.es/datasheet-pdf/view/553675/LATTICE/iCE40HX1K-TQ144.html

Los Viveros. *Manual de Proteus.* *http://www.losviveros.es/alumnos/asig8/carpeta714/IntroducciA_n_Proteus_.pdf*

PCBWay. *PCB Instant Qoute. Estándar PCB.* *https://www.pcbway.com/orderonline.aspx*

Texas Instruments. *10-Bit, 65-MSPS Analog-toDigital Converter (ADC).* *https://www.ti.com/product/ADC10065?utm_source=google&utm_medium=cpc&utm_campaign=asc-dc-null-prodfolderdynamic-cpc-pf-google-wwe_int&utm_content=prodfolddynamic&ds_k=DYNAMIC+SEARCH+ADS&DCM=yes&gclid=CjwKCAjw-vmkBhBMEiwAlrMeF4Fv02Fo2rIAN1i37e1SbTMPLLOrqB_vn-sTA2m_k9NasKj5uVzNGBoCm3UQAvD_BwE&gclsrc=aw.ds*