



UNIVERSIDAD
POLITÉCNICA
DE MADRID



SISTEMAS ELECTRÓNICOS DIGITALES

TRABAJO DE FPGA

Javier González Corroto 55563

Costin Leonard Matulescu 54080

Javier Tauroni Bernaldo de Quirós 53439

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Tutor: Luis Dávila Gómez

ÍNDICE

1. Introducción
2. Fundamento teórico
3. Elaboración del trabajo
 - 3.1. Fase uno
 - 3.2. Fase dos
 - 3.3. Fase tres
4. Códigos y testbench
5. Conclusiones y observaciones
6. Enlaces de interés
7. Anexo: Idea inicial y desarrollo

8. Introducción

En el trabajo de VHDL se ha optado por la implementación de un transmisor FM, capaz de transmitir señales a través de una antena para que cualquier dispositivo que esté a su alcance sea capaz de detectar dicha información e interpretarla correctamente.

Para llevar a cabo esta implementación se ha dividido el trabajo en varias fases:

- Primero se ha implementado una transmisión FM en la que se transmitía dos notas musicales mediante la pulsación de dos botones.
- En una segunda fase, se ha implementado todas las notas musicales, para que jugando con la posición de los switches que dispone la placa, sea capaz de transmitir las de forma inalámbrica.
- Por último y lo más difícil, se ha optado por transmitir una señal analógica (por ejemplo, un audio) mediante FM. Su desarrollo se verá más adelante.

9. Fundamento teórico

La modulación de frecuencia o frecuencia modulada (FM) se emplea para transmitir señales analógicas de forma inalámbrica empleando ondas electromagnéticas. Dichas ondas viajan por el aire a una determinada frecuencia que acaba llegando a un receptor, que con la electrónica apropiada es capaz de interpretar la señal.

Esta modulación consiste en la unión de dos señales, una de mayor frecuencia que la otra, normalmente mucha mayor frecuencia. La señal de alta frecuencia se denomina señal portadora, pues esta señal sólo se utiliza para transportar la señal de baja frecuencia. La señal de baja frecuencia se denomina señal moduladora.

El porqué de este método es sencillo de entender. Las señales de alta frecuencia viajan mucho más lejos que las de baja frecuencia puesto que tienen más energía. Como el ser humano puede escuchar o “detectar” señales comprendidas entre los 20 Hz y 20 KHz, no tienen suficiente energía para poder transmitirse inalámbricamente a varios kilómetros de distancia. Por tanto, lo que se hace es unir ambas señales (la portadora y la moduladora) para enviarlas mediante una antena a los diferentes receptores a los cuales les interesa dicha señal/información.

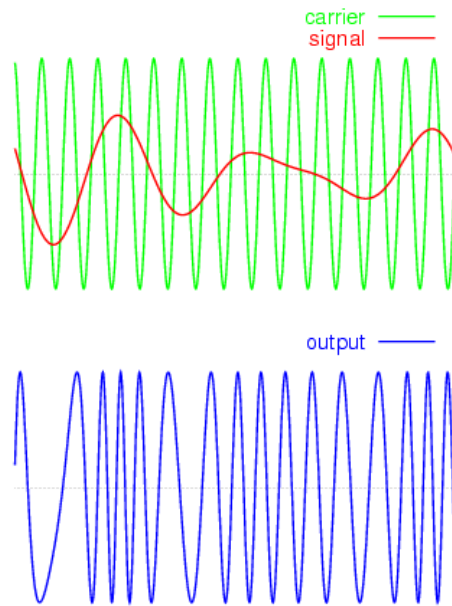


Figura 1: Representación de la modulación en frecuencia.

10. Elaboración del trabajo

Como se ha mencionado en la introducción el trabajo se ha realizado en tres fases, cada una evolución de la anterior.

10.1. Fase uno

Sabiendo cómo funciona la frecuencia modulada se procede a la implementación en VHDL.

Para la generación de señales, tanto la portadora como la moduladora, se ha optado por implementar un acumulador de fase ya que la placa no dispone de convertidor digital-analógico.

El acumulador de fase forma parte de un oscilador y su fundamento teórico es el siguiente:

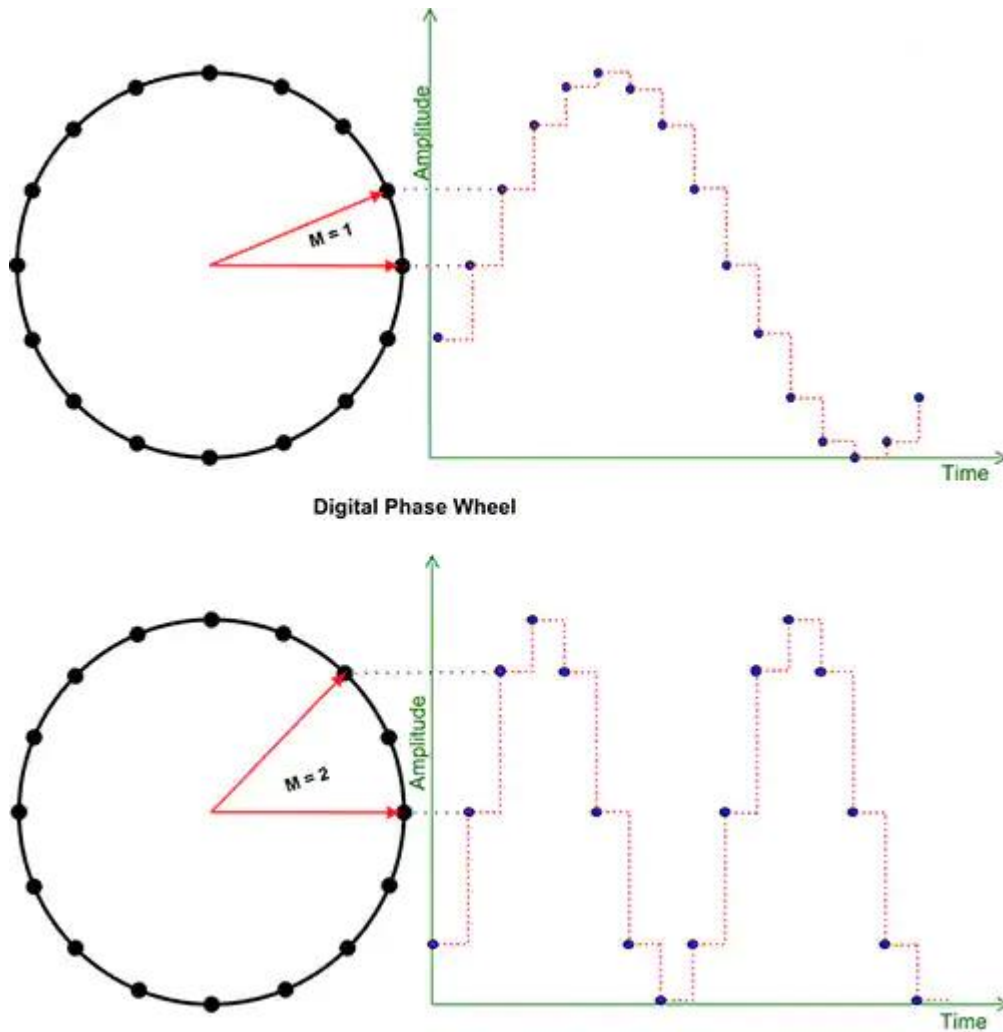


Figura 2: Representación gráfica del acumulador de fase.

Para entender el funcionamiento de un acumulador de fase se puede usar la analogía mostrada en la figura 2 en la que cada división de la circunferencia representa la resolución del acumulador de fase, por ejemplo 16 bits en este caso. Cuando el incremento M aumenta cada 2 bits la frecuencia de la señal resultante también aumenta. Se puede observar que cuando M vale 1 la señal se repite cada 2^N ciclos de reloj, por tanto, para aumentar la frecuencia de señal hay que aumentar el paso M .

La ecuación que sigue este sistema es:

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N$$

Donde M es el incremento, $f_{deseada}$ es la frecuencia que se quiere obtener con el acumulador de fase y $f_{sistema}$ es la frecuencia de reloj.

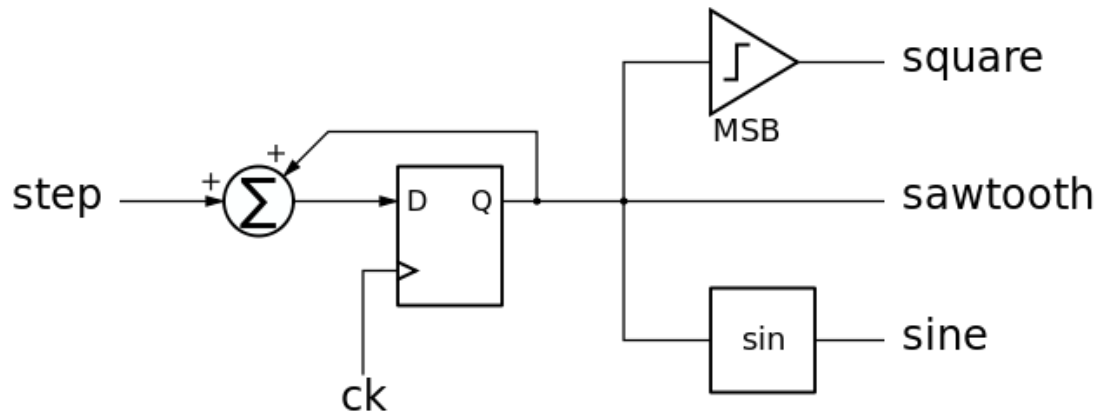


Figura 3: Diagrama de bloques del oscilador.

En la figura 3 se puede observar el diagrama de bloques del oscilador el cual está compuesto por un registro y un sumador y dependiendo de lo que se quiera obtener en la salida se aplica un filtro u otro.

Como la placa no tiene convertidor digital analógico se opta por aplicarle el filtro MSB que consiste en tomar únicamente de la salida Q el valor del bit más significativo MSB. Como este bit cambia cada medio ciclo de la frecuencia deseada, al final se obtiene una onda cuadrada.

En cuanto al transmisor FM su función es la de combinar las dos señales, la portadora y la moduladora. La señal portadora tendrá una frecuencia fija mientras que la moduladora tendrá diferente frecuencia dependiendo de la información que contenga. La banda de emisión comercial se encuentra entre los 87 MHz y los 110 MHz, con lo cual se puede cambiar la frecuencia de la portadora en función de esta banda.

Cada canal de emisión tiene un rango de ± 75 KHz para hacer variar su frecuencia lo que significa que la señal moduladora junto con la portadora podrá variar entre la frecuencia central ± 75 KHz para transmitir la información.

El transmisor constará de dos multiplexores. Uno de ellos actuará de tal manera que cuando no hay señal moduladora la antena emita la frecuencia central, es decir la frecuencia de la portadora (elegida entre 87 MHz y 110 MHz). Cuando se pulse un botón la antena emitirá la señal portadora más la moduladora. El segundo multiplexor se encarga de hacer variar la frecuencia entre $f_c + 75$ KHz y $f_c - 75$ KHz.

Para entenderlo mejor véase el siguiente ejemplo:

Suponer que la frecuencia central para emitir es la de 100 MHz. El oscilador deberá tener el siguiente incremento M:

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{100 \text{ MHz}}{320 \text{ MHz}} \cdot 2^{32} = 1342177280$$

Si se supone que la frecuencia del reloj es de 320 MHz y el registro es de 32 bits.

Cuando no hay señal moduladora el oscilador recibe el incremento M y genera una señal cuadrada de frecuencia 100 MHz. Pues el primer multiplexor está actuando y la salida es I_0 .

Ahora bien, si se tiene en cuenta que la frecuencia de emisión puede variar entre la $f_c + 75$ KHz y $f_c - 75$ el incremento M del oscilador es:

$$M_{f_c - 75 \text{ KHz}} = \frac{100 - 0,075 \text{ MHz}}{320 \text{ MHz}} \cdot 2^{32} = 1341170647$$

$$M_{f_c + 75 \text{ KHz}} = \frac{100 + 0,075 \text{ MHz}}{320 \text{ MHz}} \cdot 2^{32} = 1343183913$$

Por tanto, si se quiere emitir por ejemplo la nota musical La cuya frecuencia es de 440 Hz, se debe hacer variar la frecuencia central en $f_c + 75 \text{ KHz}$ y $f_c - 75 \text{ KHz}$ con una frecuencia de 440 Hz por tanto asociando a $M_{f_c - 75 \text{ KHz}}$ un 0 y a $M_{f_c + 75 \text{ KHz}}$ un 1 la salida digital (antena) variará dichos valores con la frecuencia de 440 Hz mencionada anteriormente.

El incremento que se debe introducir al oscilador para obtener los 440 Hz es:

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{440 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 5905,58 = 5906$$

El esquema del transmisor FM es el siguiente:

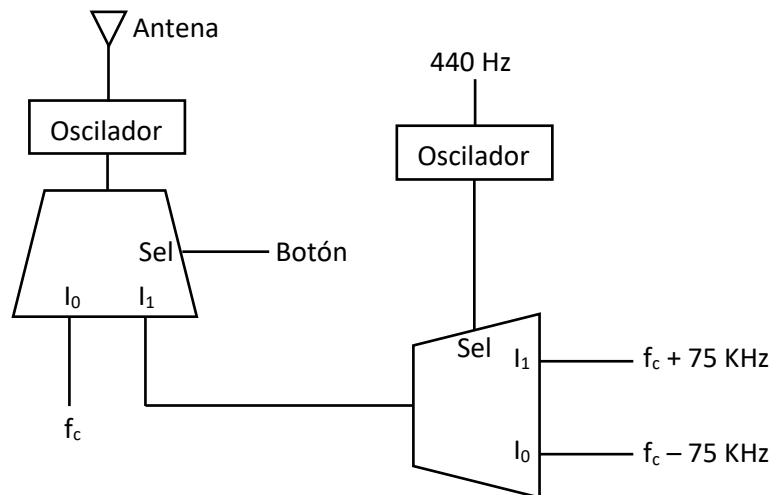


Figura 4: Diagrama de bloques del Transmisor FM.

Para el trabajo se ha optado por transmitir dos tonos diferentes pulsando dos botones.

10.2. Fase dos

Para aumentar un poco la complejidad se ha decidido transmitir la escala musical por radiofrecuencia, por tanto, con el uso de varios multiplexores conectados a las entradas de los switch se puede implementar. Los incrementos del oscilador para cada switch son:

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{261,63 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 3512$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{277,18 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 3720$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{293,66 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 3941$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{311,13 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 4176$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{329,63 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 4424$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{349,23 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 4687$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{369,99 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 4966$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{392 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 5261$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{415,30 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 5574$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{440 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 5906$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{466 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 6255$$

$$M = \frac{f_{deseada}}{f_{sistema}} \cdot 2^N = \frac{493,88 \text{ Hz}}{320 \text{ MHz}} \cdot 2^{32} = 6629$$

Que corresponden con Do, Do#, Re, Re#, Mi, Fa, Fa#, Sol, Sol#, La, La# y Si respectivamente.

10.3. Fase tres

Por último, para asemejar el trabajo a una radio real se ha optado por enviar una señal de audio. El audio será captado por un pin de la placa, con lo cual, al ser una señal analógica, hará falta un convertidor analógico-digital. Hay muchas formas de implementar un ADC, pero como la placa Nexys 4 DDR cuenta con uno se ha decidido

utilizarlo. Para ello se ha hecho uso de los módulos IP que existen para dicho convertidor.

Los módulos IP son bloques constructivos prediseñados y preverificados con lo cual lo único que hay que hacer es utilizar sus entradas y salidas.

En la ventana de *Flow Navigation* se pulsa sobre *IP Modules* y se busca por el nombre XADC al convertidor. A continuación, aparece una ventana cómo la siguiente:

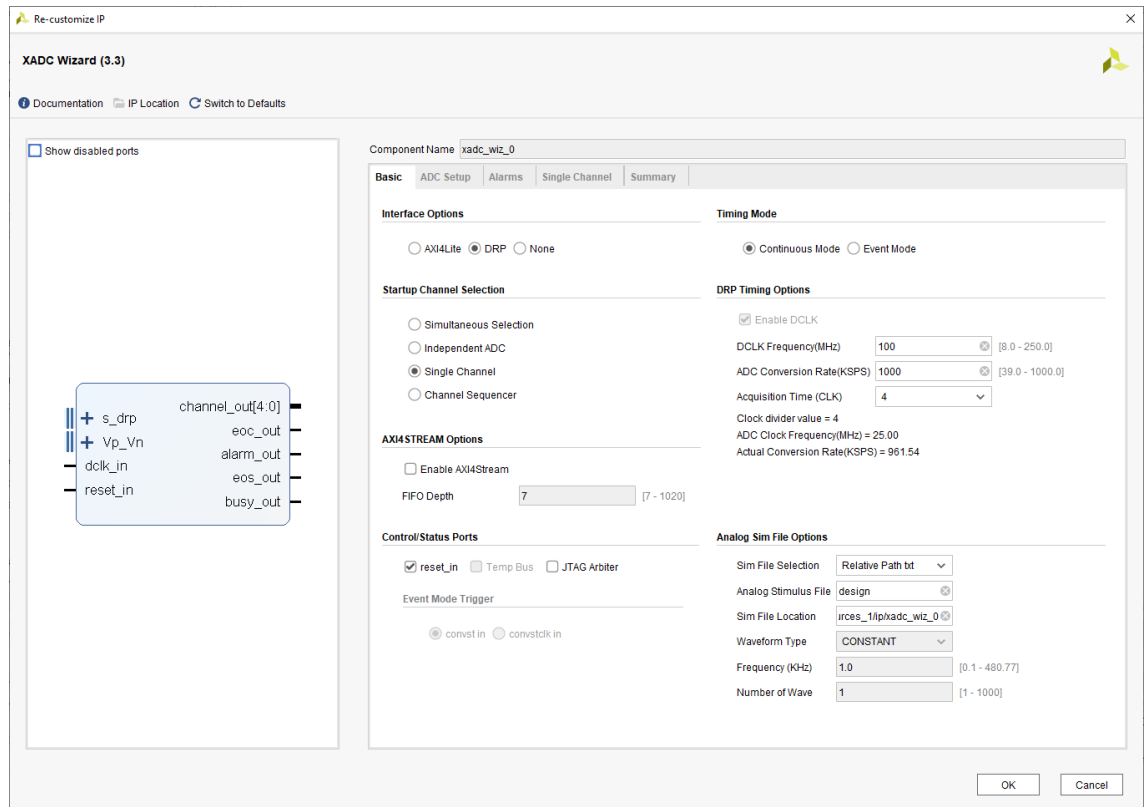


Figura 5: Ventana del asistente Wizard para la configuración del ADC.

En ella se puede observar las siguientes opciones:

- Basic
 - Interface Options: Puesto que la placa se comunica con el ADC mediante DRP, se dejará esta opción marcada.
 - Startup Channel Selection: Aquí se puede seleccionar el modo de funcionamiento del ADC, puesto que sólo se va a leer un pin, se va a dejar marcada la opción de single channel.
 - AXI4STREAM Options: Esta parte se deja por defecto.
 - Control Status Ports: Se puede seleccionar las señales de control para poner a reset el ADC, pues dejaremos marcada esta opción.
 - Timing Mode: Se puede hacer conversión de forma continua o mediante una señal de activación, en este caso se ha optado por el modo continuo.
 - DRP Timing Options: En esta parte se establece la frecuencia de conversión del ADC.
 - Analog Sim File Options: Puesto que no se pueden simular señales analógicas, se puede crear un archivo en el que vendrán varias muestras en diferentes instantes de tiempo.

- ADC setup

En esta sección no hay que tocar nada, pues se dejarán las opciones marcadas por defecto.

- Alarms

Aquí se desactivarán todas las alarmas ya que no se van a hacer uso de ellas.

- Single channel

En select channel hay que seleccionar VP VN y tener marcada la casilla Channel Enable.

- Summary

En esta pestaña aparece el resumen de la configuración del XADC.

Se pulsa OK y se creará el módulo IP dentro del proyecto. En la ventana *Source* dentro de la pestaña *IP Source* se puede encontrar la carpeta *Instantiation Template* en la cual vienen los archivos que permiten instanciar el ADC.

4. Códigos y testbench

En esta sección se explicarán por encima los puntos más relevantes de la programación VHDL del trabajo, así como la visualización de los testbench.

En primer lugar, los registros no tienen mucha complicación, pues se trata de N biestables que permiten almacenar información.

El oscilador puede ser un poco más complejo, pero no se aleja de la simplicidad del registro:

```
entity Oscilador is
    generic (
        NBits : integer := 4
    );
    port (
        Incremento : in std_logic_vector((NBits-1) downto 0);
        Clk         : in std_logic; -- Reloj
        Reset       : in std_logic; -- Reset
        DataOut     : out std_logic -- Salida del oscilador
    );
end Oscilador;
```

En cuanto al funcionamiento, se ha descrito de la siguiente manera:

```

architecture Behavioral of Oscilador is
component Registro
    generic (
        NBits : integer := 4 );
    port (
        Enable : in std_logic;
        Clk : in std_logic;
        D : in std_logic_vector ((Nbits - 1) downto 0);
        Q : out std_logic_vector ((Nbits - 1) downto 0));
end component;
signal MultiplexorOUT : std_logic_vector ((Nbits - 1) downto 0);
signal RegistroOUT : std_logic_vector ((Nbits - 1) downto 0);
begin
AcumuladorFase: Registro generic map(NBits => NBits)
    port map(Enable => '1', Clk => Clk, D => MultiplexorOut, Q => RegistroOut);

    MultiplexorOut <= (others => '0') when (Reset = '0') else -- Cuando Reset está activado se pone a
        (RegistroOut + Incremento); -- Cuando Reset no está activado el registro se inc

    DataOut <= RegistroOut(NBits - 1); --Tomamos el bit más significativo de la salida
end Behavioral;

```

Se puede ver que el registro forma parte del oscilador por tanto se añade como componente. De acuerdo con la figura 3 harán falta dos señales, una será para un multiplexor cuya función es o poner todo a cero cuando reset es true o ir incrementado la salida del registro cuando reset es false.

Como se ha dicho antes, se tomará el valor del bit más significativo para formar una onda cuadrada.

El transmisor FM es más sencillo de lo que parece pues es conectar los diferentes componentes que aparecen en la siguiente figura:

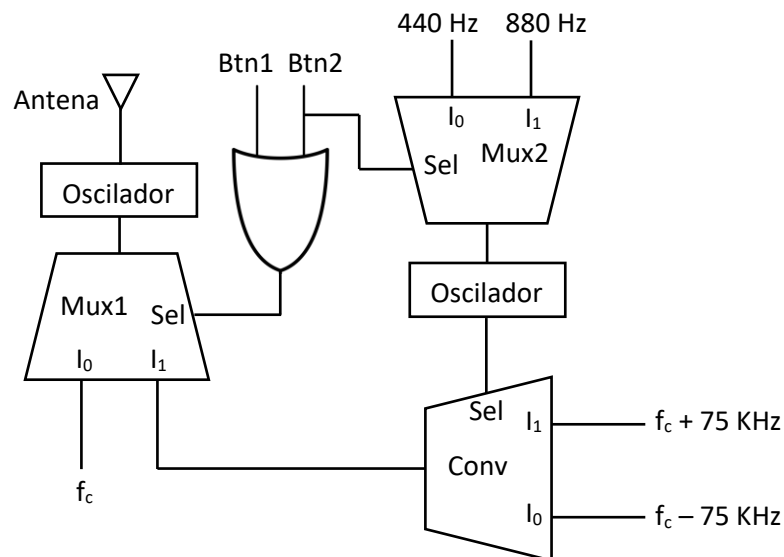


Figura 6: Diagrama de bloques del transmisor FM.

Por tanto, el comportamiento de este sistema puede ser descrito de la siguiente manera:

```

architecture Behavioral of TransmisorFM is
  component Oscilador is
    generic (
      NBits : integer := 32
    );
    port (
      Incremento : in std_logic_vector(31 downto 0);
      Clk         : in std_logic;
      Reset       : in std_logic;
      DataOut     : out std_logic
    );
  end component;
  component Multiplexor is
    generic (
      NBits : integer := 32
    );
    port (
      Sel       : in std_logic;
      In0       : in std_logic_vector((NBits - 1) downto 0);
      In1       : in std_logic_vector((NBits - 1) downto 0);
      DataOut   : out std_logic_vector((NBits - 1) downto 0)
    );
  end component;
  signal Mux1Out : std_logic_vector(31 downto 0);
  signal Mux1Sel : std_logic;
  signal ConvOut : std_logic_vector(31 downto 0);
  signal ConvIn  : std_logic;
  signal Mux2Out : std_logic_vector(31 downto 0);

```

Se declaran los componentes que se van a utilizar, en este caso, el oscilador y un multiplexor.

```

begin
  RadioOsc : Oscilador generic map (
    NBits => 32
  )
  port map (
    Clk => Clk,
    Reset => Reset,
    Incremento => Mux1Out,
    DataOut => AntOut
  );
  Mux1Sel <= Boton440 or Boton880;
  Mux1 : Multiplexor generic map (
    NBits => 32
  )
  port map (
    Sel => Mux1Sel,
    In0 => std_logic_vector(to_unsigned(1342177280, 32)),
    In1 => ConvOut,
    DataOut => Mux1Out
  );

```

El primer oscilador de la radio será el que genera la frecuencia que se emita por la antena, por tanto, el incremento se asocia a la salida del multiplexor 1.

```

Conv : Multiplexor generic map (
    NBits => 32
)
port map (
    Sel => ConvIn,
    In0 => std_logic_vector(to_unsigned(1341170647, 32)),
    In1 => std_logic_vector(to_unsigned(1343183913, 32)),
    DataOut => ConvOut
);

AudioOsc : Oscilador generic map (
    NBits => 32
)
port map (
    Clk => Clk,
    Reset => Reset,
    Incremento => Mux2Out,
    DataOut => ConvIn
);

Mux2 : Multiplexor generic map (
    NBits => 32
)
port map (
    Sel => Boton880,
    In0 => std_logic_vector(to_unsigned(5906, 32)),    -- 440 Hz
    In1 => std_logic_vector(to_unsigned(11812, 32)),   -- 880 Hz
    DataOut => Mux2Out
);

```

Luego el multiplexor 2 será el que dependiendo del botón que se pulse dejará pasar el incremento de los 440 Hz o los 880 Hz al segundo oscilador. El segundo oscilador genera la frecuencia que hará modificar la entrada de selección del multiplexor *conv* de manera que deje pasar el incremento de las frecuencias de ± 75 KHz junto con la frecuencia central al multiplexor 1, y este a su vez, deja pasar dichos incrementos al oscilador 1 que finalmente acaba en la antena.

En cuanto a la fase dos las entradas se multiplican y por tanto se necesita una puerta *or* más grande y más multiplexores, pero el código no se complica demasiado.

Sólo habría que sustituir el código del multiplexor 2 por el siguiente:

```

Mux2Out <= std_logic_vector(to_unsigned(3512, 32)) when sw = "000000000001" else --Do
std_logic_vector(to_unsigned(3720, 32)) when sw = "000000000010" else --Do#
std_logic_vector(to_unsigned(3941, 32)) when sw = "000000000100" else --Re
std_logic_vector(to_unsigned(4176, 32)) when sw = "000000001000" else --Re#
std_logic_vector(to_unsigned(4424, 32)) when sw = "000000010000" else --Mi
std_logic_vector(to_unsigned(4687, 32)) when sw = "000000100000" else --Fa
std_logic_vector(to_unsigned(4966, 32)) when sw = "000001000000" else --Fa#
std_logic_vector(to_unsigned(5261, 32)) when sw = "000010000000" else --Sol
std_logic_vector(to_unsigned(5574, 32)) when sw = "000100000000" else --Sol#
std_logic_vector(to_unsigned(5906, 32)) when sw = "001000000000" else --La
std_logic_vector(to_unsigned(6255, 32)) when sw = "010000000000" else --La#
std_logic_vector(to_unsigned(6629, 32)) when sw = "100000000000" else --Si
std_logic_vector(to_unsigned(0, 32));

```

Por último, en la fase tres se añade el XADC el cual convertirá la señal de audio en valores digitales se implementa mediante el módulo IP. Como la señal de audio es bipolar y el XADC es de 16 bits, para valores negativos corresponde -32768 y para valores positivos corresponde +32768. La señal de audio tiene diferentes amplitudes con lo cual se debe tener en cuenta que la señal de salida puede variar 150 KHz (± 75 KHz). Tomando su incremento:

$$M_{f_c-75\text{ KHz}} = \frac{0,075\text{ MHz}}{320\text{ MHz}} \cdot 2^{32} = 1006633$$

Por tanto, la amplitud máxima del audio que leerá el ADC será 2^{N-1} bits y el ADC al ser de 16 bits, se tiene que la franja de 0.075 MHz queda con una resolución de:

$$\frac{1006633}{2^{15}} = 30,72$$

Ahora bien, pues la señal se moverá entre la frecuencia central y los ± 75 KHz de la siguiente manera:

$$M = M_{f_c} + Salida_{ADC} \cdot 30,72$$

De este modo cuando la señal de audio esté en su punto máximo el incremento M corresponderá con el incremento de $M_{f_c \pm 75\text{ KHz}}$ y cuando la señal de audio esté a cero o apagado corresponderá con el incremento de la frecuencia central.

Para implementar el ADC basta con abrir el archivo **xadc_wiz_0.vho** que se encuentra en *IP Source > IP > xadc_wiz_0 > Instantiation Template* se copia el componente y se pega en el archivo de trabajo.

Por tanto, el transmisor FM esta vez sólo contará con un oscilador y el ADC:

```

RadioOsc : Oscilador generic map (
    NBits => 32
)
port map (
    Clk => Clk,
    Reset => Reset,
    Incremento => IncrementoADC,
    DataOut => AntOut
);

ADC: top_XADC port map(
    Clk => Clk,
    Reset => Reset,
    vp_in => vp_in,
    vn_in => vn_in,
    dout => dout,
    drdy => open,
    channel => open
);

IncrementoADC <= std_logic_vector((signed(dout)*to_signed(ADCgain,16)) + FrecuenciaCentralINC);

```

En el que *IncrementoADC* se encarga de calcular el incremento que debe entrar al oscilador para hacer variar la frecuencia central ± 75 KHz dependiendo de lo que entregue el ADC.

En cuanto al testbench como todos son muy parecidos, sólo se debe observar el comportamiento de la antena, pues al tratarse de frecuencias muy elevadas los cambios casi no se consiguen apreciar. En cambio, el ADC sí que se puede simular:

Como hay que simular con entradas analógicas, se ha creado el siguiente archivo txt:

```

TIME    TEMP    VCCINT    VCCBRAM    VCCAUX    VP    VN    VAUXP[0]    VAUXN[0]    VAUX
0       63.0    1.02    1.02    1.8 0.5  0.0  0.5  0.0  0.5  0.0  0.5  0.0  0.5  0.0  0.5

1250    88.0    1.08    1.08    1.94 0.3  0.0  0.2  0.0  0.2  0.0  0.2  0.0  0.2  0.0  0.
2290    53.0    0.92    0.92    1.7 0.9  0.0  0.9  0.0  0.9  0.0  0.9  0.0  0.9  0.0  0.5

```

En este caso solo es necesario las entradas VP y VN que como se puede observar toman los siguientes valores:

Tiempo	VP	VN
0	0.5	0
1250	0.3	0
2290	0.9	0

Tabla 1: Valores de entrada para la simulación del ADC

La simulación que se obtiene es:

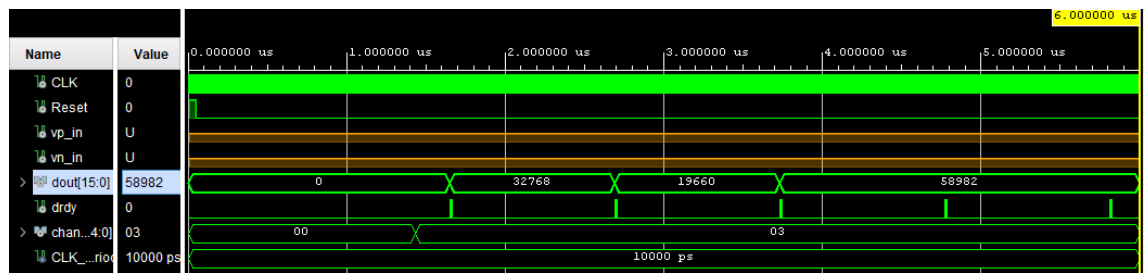


Figura 7: Simulación del ADC.

Teniendo en cuenta que la tensión de referencia del ADC es de 1 V, se comprueban los valores:

$$\frac{32768}{65536} = 0,5 \text{ V}$$

$$\frac{19660}{65536} = 0,3 \text{ V}$$

$$\frac{58982}{65536} = 0,9 \text{ V}$$

Que corresponden con los valores de la *tabla 1*.

5. Conclusiones y observaciones

Una observación interesante es que en la simulación de los transmisores FM la salida de la antena no cambia a pesar de cambiar las entradas, sin embargo, en la simulación con la placa, el transmisor funciona correctamente.

Otra observación es que, a la hora de generar las señales con diferentes frecuencias, se ha optado por un oscilador. Una alternativa que se había pensado era la de utilizar una PWM pero sólo se podría modificar su duty-cycle y además sería más difícil modificar su frecuencia, con lo cual el oscilador es una solución sencilla.

6. Enlaces de interés

https://idus.us.es/bitstream/handle/11441/69386/TFG_Pedro%20Gutierrez%20Lora.pdf?sequence=1&isAllowed=y

<http://avelinoherrera.com/blog/index.php?entry=entry161103-181043>

<http://avelinoherrera.com/blog/index.php?entry=entry201007-010240>

https://reference.digilentinc.com/_media/nexys4-ddr:nexys4ddr_rm.pdf

https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf

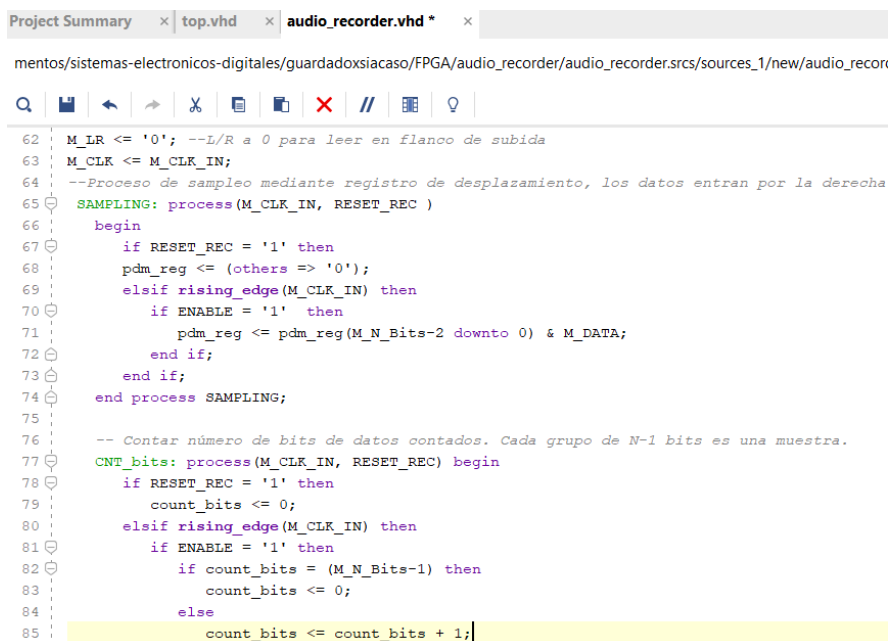
https://www.xilinx.com/support/documentation/ip_documentation/xadc_wiz/v3_0/pg091-xadc-wiz.pdf

Anexo: Idea inicial y desarrollo

En un primer momento, se barajó la idea de combinar el STM32F4 y la Nexys 4DDR de forma que la FPGA recibiese un archivo de audio por Bluetooth, lo procesase y se lo mandase al microcontrolador mediante un emisor FM para reproducirlo por la salida de audio. La idea fue desechada por la dificultad de encontrar tanto componentes como ejemplos en Internet para enviar audio por Bluetooth a la FPGA.

Por tanto, finalmente se decidió separar los trabajos y, en el de FPGA, diseñar un programa capaz de recibir datos de audio desde el micrófono integrado en la Nexys, almacenarlos en la memoria y posteriormente reproducirlos por la salida de audio.

Se consiguió interactuar con el micrófono y recibir los datos de audio correctamente, pero después de incorporar un bloque IP que funcionaba como interfaz de memoria no se pudo progresar sin que surgieran numerosos errores. Por tanto, se desechó la idea y se comenzó a trabajar en el proyecto de transmisión de señales mediante FM.



Project Summary x top.vhd x audio_recorder.vhd * x

mentos/sistemas-electronicos-digitales/guardadoxiacaso/FPGA/audio_recorder/audio_recorder.srcs/sources_1/new/audio_recori

```

62 M_LR <= '0'; --L/R a 0 para leer en flanco de subida
63 M_CLK <= M_CLK_IN;
64 --Proceso de sampleo mediante registro de desplazamiento, los datos entran por la derecha
65 SAMPLING: process(M_CLK_IN, RESET_REC)
66 begin
67     if RESET_REC = '1' then
68         pdm_reg <= (others => '0');
69     elsif rising_edge(M_CLK_IN) then
70         if ENABLE = '1' then
71             pdm_reg <= pdm_reg(M_N_Bits-2 downto 0) & M_DATA;
72         end if;
73     end if;
74 end process SAMPLING;
75
76 -- Contar número de bits de datos contados. Cada grupo de N-1 bits es una muestra.
77 CNT_bits: process(M_CLK_IN, RESET_REC) begin
78     if RESET_REC = '1' then
79         count_bits <= 0;
80     elsif rising_edge(M_CLK_IN) then
81         if ENABLE = '1' then
82             if count_bits = (M_N_Bits-1) then
83                 count_bits <= 0;
84             else
85                 count_bits <= count_bits + 1;

```

Fragmento de código del proyecto anterior