 UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENERIA Y TECNOLOGIAS AVANZADAS

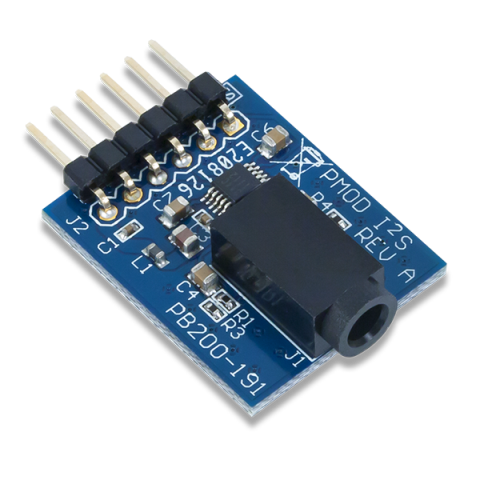
**“Proyecto I2S que genera una melodía de audio”**

Alumno: ZARAZUA AGUILAR LUIS FERNANDO

GRUPO: 2mm9

PROFESOR: Rodríguez fuentes miguel Ángel

MATERIA: DISPOSITIVOS LÓGICOS PROGRAMABLES



**Planteamiento del Problema**

Esta práctica tiene como objetivo poder reproducir una pequeña melodía en base a sus frecuencias, por medio del protocolo I2S, usando un PMOD de Digilent que contiene una DAC de 2 canales y 16 bits con una salida de 3.5mm para jack (conector de audífonos). Este protocolo tiene como conexiones un pin datos, un pin de reloj que detecta cada bit enviado, un pin de entrada de reloj maestro que sirve para operación correcta de la DAC, y sus entradas de voltaje y tierra.

**Código Principal**

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** siren **is**

**Port** **(** clk\_50MHz **:** **in** STD\_LOGIC**;** -- system clock (50 MHz)

dac\_MCLK **:** **out** STD\_LOGIC**;** -- outputs to PMODI2L DAC. Reloj de operación de la DAC.

dac\_LRCK **:** **out** STD\_LOGIC**;**--Selección del canal.

dac\_SCLK **:** **out** STD\_LOGIC**;**--Señal de reloj para sincronizar dato con la DAC.

dac\_SDIN **:** **out** STD\_LOGIC**;**--Salida de dato serial.

Salidas\_7segc **:** **out** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Control\_Disp\_7segc**:** **out** STD\_LOGIC\_VECTOR **(**3 **downto** 0**));**

**end** siren**;**

**architecture** Behavioral **of** siren **is**

**component** Leds\_Display\_7 **is**

**Port** **(** clkin **:** **in** STD\_LOGIC**;**

Entrada\_Disp\_1 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Entrada\_Disp\_2 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Entrada\_Disp\_3 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Entrada\_Disp\_4 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Salidas\_7seg **:** **out** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Control\_Disp\_7seg **:** **out** STD\_LOGIC\_VECTOR **(**3 **downto** 0**));**

**end** **component;**

--constant wail\_speed: UNSIGNED (7 downto 0) := to\_unsigned (8, 8); -- sets wailing speed

**component** dac\_if **is**

**Port** **(** SCLK **:** **in** STD\_LOGIC**;**

L\_start**:** **in** STD\_LOGIC**;**

R\_start**:** **in** STD\_LOGIC**;**

L\_data **:** **in** signed **(**15 **downto** 0**);**

R\_data **:** **in** signed **(**15 **downto** 0**);**

SDATA **:** **out** STD\_LOGIC**);**

**end** **component;**

**component** wail **is**

**Port** **(** wclk **:** **in** STD\_LOGIC**;** -- wailing clock (47.6 Hz), duracion base de una nota.

audio\_clk **:** **in** STD\_LOGIC**;** --Frecuencia de muestreo, Reloj de audio a 48,828.125Hz (t=256 veces MCLK),LRCK,equivalente a 1 muestra analogica.

M**:** **in** STD\_LOGIC\_VECTOR **(**23 **downto** 0**);**--Entrada de la memoria.

rom\_addr **:** **out** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**--Direccion a la que se quiere acceder de la memoria.

audio\_data **:** **out** SIGNED **(**15 **downto** 0**));** -- output audio sequence (wailing tone),Señal de salida.

**end** **component;**

**component** Cancion\_ROM **is**

**Port** **(**clka **:** **IN** STD\_LOGIC**;**

addra **:** **IN** STD\_LOGIC\_VECTOR**(**7 **downto** 0**);**

douta **:** **OUT** STD\_LOGIC\_VECTOR**(**23 **downto** 0**));**

**end** **component;**

**signal** tcount**:** unsigned **(**19 **downto** 0**)** **:=** **(others=>**'0'**);** -- timing counter

**signal** data\_L**,** data\_R**:** SIGNED **(**15 **downto** 0**);** -- 16-bit signed audio data

**signal** dac\_load\_L**,** dac\_load\_R**,**clk50**:** STD\_LOGIC**;** -- timing pulses to load DAC shift reg.

**signal** slo\_clk**,** sclk**,** audio\_CLK**:** STD\_LOGIC**;**

**signal** Direccion\_ROM **:**STD\_LOGIC\_VECTOR**(**7 **downto** 0**);**

**signal** Salida\_ROM**:** STD\_LOGIC\_VECTOR**(**23 **downto** 0**);**

**signal** Display\_1c**,**Display\_2c**,**Display\_3c**,**Display\_4c**:** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

**begin**

-- this process sets up a 20 bit binary counter clocked at 50MHz. This is used

-- to generate all necessary timing signals. dac\_load\_L and dac\_load\_R are pulses

-- sent to dac\_if to load parallel data into shift register for serial clocking

-- out to DAC

Display\_1c**<=**STD\_LOGIC\_VECTOR**(**data\_L**(**7 **downto** 0**));**

Display\_2c**<=**Salida\_ROM**(**7 **downto** 0**);**--rx\_datac;

Display\_3c**<=**Salida\_ROM**(**15 **downto** 8**);**--Data\_2;--Data\_0;

Display\_4c**<=**Salida\_ROM**(**23 **downto** 16**);**--rx\_datac;--Datos\_Rxc;

tim\_pr**:** **process**

**begin**

**wait** **until** **rising\_edge(**clk\_50MHz**);**--Detectar pulso de reloj.

--Pulso para cargar dato al canal Izquierdo.

**if** **(**tcount**(**9 **downto** 0**)>=** X"00F"**)** and **(**tcount**(**9 **downto** 0**)** **<** X"02E"**)** **then**

dac\_load\_L **<=** '1'**;**--"01F"=>31 pulsos de 50MHz=0.620us.

**else**

dac\_load\_L **<=** '0'**;**--"0C3"=>195 pulsos de 50MHz=3.900us.

**end** **if;**

--Pulso para cargar dato al canal Derecho.

**if** **(**tcount**(**9 **downto** 0**)** **>=** X"20F"**)** and **(**tcount**(**9 **downto** 0**)** **<** X"22E"**)** **then**

dac\_load\_R **<=** '1'**;**--"01F"=>31 pulsos de 50MHz=0.620us.

**else**

dac\_load\_R **<=** '0'**;**--"0C3"=>195 pulsos de 50MHz=3.900us.

**end** **if;**

tcount **<=** tcount**+**1**;**

**end** **process;**

dac\_MCLK **<=** not tcount**(**1**);** -- DAC master clock (12.5 MHz), MCLK\_Dac a 12.5MHz.

audio\_CLK **<=** tcount**(**9**);** -- audio sampling rate (48.8 kHz), Reloj de audio a 48,828.125Hz (t=256 veces MCLK).

dac\_LRCK **<=** audio\_CLK**;** -- also sent to DAC as left/right clock, Pulso de canal L/R.

sclk **<=** tcount**(**4**);** -- serial data clock (1.56 MHz),Frecuencia de envio de datos (t=8 veces MCLK) =1562.5KHz.

dac\_SCLK **<=** sclk**;** -- also sent to DAC as SCLK,Reloj de la transmicion de datos.

slo\_clk **<=** tcount**(**19**);** -- clock to control wailing of tone (47.6 Hz), Reloj para controlar que tanto dura una nota.

dac**:** dac\_if **port** **map** **(** SCLK **=>** sclk**,** -- instantiate parallel to serial DAC interface, Frecuencia de envio de datos t=8 veces MCLK) =1562.5KHz.

L\_start **=>** dac\_load\_L**,**--Pulso para cargar dato al canal Izquierdo.

R\_start **=>** dac\_load\_R**,**--Pulso para cargar dato al canal Derecho.

L\_data **=>** data\_L**,**--Datos del canal Izquierdo.

R\_data **=>** data\_R**,**--Datos del canal Derecho.

SDATA **=>** dac\_SDIN **);**--Salida de datos seriales.

w1**:** wail **port** **map(** wclk **=>** slo\_clk**,**--Reloj del tono base.

audio\_clk **=>** audio\_clk**,**-- Reloj de audio a 48,828.125Hz (t=256 veces MCLK),LRCK.

audio\_data **=>** data\_L**,**--Word a enviar a la Dac con el valor que se quiere.

M **=>** Salida\_ROM**,**--Dato leìdo.

rom\_addr **=>** Direccion\_ROM**);**--Direccion a la que se quiere acceder de la memoria.

data\_R **<=** data\_L**;** -- duplicate data on right channel

ROM1**:** Cancion\_ROM **port** **map(**clka **=>** tcount**(**1**),**

addra **=>** Direccion\_ROM**,**

douta **=>** Salida\_ROM**);**

clk50**<=**clk\_50MHz**;**

U7Seg**:** Leds\_Display\_7 **port** **map** **(** clkin **=>** clk50**,**

Entrada\_Disp\_1 **=>** Display\_1c**,**

Entrada\_Disp\_2 **=>** Display\_2c**,**

Entrada\_Disp\_3 **=>** Display\_3c**,**

Entrada\_Disp\_4 **=>** Display\_4c**,**

Salidas\_7seg **=>** Salidas\_7segc**,**

Control\_Disp\_7seg **=>** Control\_Disp\_7segc**);**

**end** Behavioral**;**

En este código se realizan las conexiones entre solo los bloques principales, ya que el bloque de wail contiene internamente otro paquete con el generador de frecuencias. Además en este bloque se realizan la división del reloj de entrada de 50Mhz a uno de 12.5Mhz que alimenta a la DAC , el pulso con el que se elige el canal derecho o izquierdo y el reloj que se encarga del muestreo de la señal.

**Código Generación de la onda**

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**use** IEEE**.**STD\_LOGIC\_ARITH**.ALL;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**

-- Generates a "wailing siren" sound by instancing a "tone" module and modulating

-- the pitch of the tone. The pitch is increased until it reaches hi\_pitch and then

-- decreased until it reaches lo\_pitch and then increased again...etc.

**entity** wail **is**

**Port** **(** wclk **:** **in** STD\_LOGIC**;** -- wailing clock (47.6 Hz), duracion base de una nota.

audio\_clk **:** **in** STD\_LOGIC**;** --Frecuencia de muestreo, Reloj de audio a 48,828.125Hz (t=256 veces MCLK),LRCK,equivalente a 1 muestra analogica.

M **:** **in** STD\_LOGIC\_VECTOR **(**23 **downto** 0**);**--Entrada de la memoria.

rom\_addr **:** **out** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**--Direccion a la que se quiere acceder de la memoria.

audio\_data **:** **out** SIGNED **(**15 **downto** 0**));** -- Señal de salida con el dato de 16 bits.

**end** wail**;**

**architecture** Behavioral **of** wail **is**

**component** tone **is**

**Port** **(** clk **:** **in** STD\_LOGIC**;**--Reloj.

pitch **:** **in** UNSIGNED **(**13 **downto** 0**);**--Frecuencia.

data **:** **out** SIGNED **(**15 **downto** 0**));**--Dato.

**end** **component;**

**signal** curr\_pitch**:** UNSIGNED **(**13 **downto** 0**);** -- current wailing pitch-Frecuencia que se quiere.

**signal** curr\_pitch1**:** STD\_LOGIC\_VECTOR **(**13 **downto** 0**);**

**signal** count\_direccion**:** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**--Direccion a la que se quiere acceder de la memoria.

**signal** wspeed **:** STD\_LOGIC\_VECTOR **(**7 **downto** 0**):=**0x"00"**;** -- speed of wail in pitch units/wclk.--Numero de repeticiones cada una de duracion (1/47.6).

**begin**

-- this process modulates the current pitch. It keep a variable updn to indicate

-- whether tome is currently rising or falling. Each wclk period it increments

-- (or decrements) the current pitch by wspeed. When it reaches hi\_pitch, it

-- starts counting down. When it reaches lo\_pitch, it starts counting up

wp**:** **process(**wclk**)**

**begin**

**if** **rising\_edge(**wclk**)** **then**

wspeed**<=**M**(**23 **downto** 16**);**

curr\_pitch1**<=**M**(**13 **downto** 0**);**

**if** count\_direccion**<**170 **then**

count\_direccion**<=**count\_direccion**+**1**;**

**else**

count\_direccion**<=(others=>**'0'**);**

**end** **if;**

**end** **if;**

**end** **process;**

rom\_addr**<=**count\_direccion**;**

curr\_pitch**<=**unsigned**(**curr\_pitch1**);**

tgen**:** tone **port** **map(** clk **=>** audio\_clk**,** --Frecuencia de muestreo

pitch **=>** curr\_pitch**,** --Frecuencia que se quiere mandar.

data **=>** audio\_data**);**--Dato de salida para la DAC en un word de 16 bits.

**end** Behavioral**;**

Este código se encarga se generar la onda de salida en un número de 16 bits que contiene un bloque interno que se encarga de convertir de Frecuencia a una salida digital. Además para poder leer la canción se obtienen los datos de una Memoria la cual es leída una vez que se acabó de mandar una nota con la duración indicada. Para esto la memoria de 256 datos y 24 bits se separó en 2 partes una que indica la frecuencia en base de 0.745 Hz, por lo tanto entre más grande sea el número se tendrá un periodo más corto y este dato viene del bit 13 al 0, el otro byte del bit 23 al 16 viene la duración bases de tiempo de 1/47.6 segundos, para el cual se usó un contador que hasta que no igualará lo leído en ese byte no pudiera leer la siguiente dirección todo esto con el fin de tener la duración correcta de la nota. Cabe mencionar que para una mejor operación entre los datos se hicieron conversiones a unsigned.

**Código Convertidor de Frecuencia a Señal Triangular.**

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

-- Generates a 16-bit signed triangle wave sequence at a sampling rate determined

-- by input clk and with a frequency of (clk\*pitch)/65,536

**entity** tone **is**

**Port** **(** clk **:** **in** STD\_LOGIC**;** -- 48.8 kHz audio sampling clock, Frecuencia de muestreo.

pitch **:** **in** UNSIGNED **(**13 **downto** 0**);** -- frequency (in units of 0.745 Hz), Frecuencia que se quiere mandar\*0.745.

data **:** **out** SIGNED **(**15 **downto** 0**));** -- signed triangle wave out.

--pitch=frecuencia\_requerida\*50e6/(pot(2,26)).

**end** tone**;**

**architecture** Behavioral **of** tone **is**

**signal** count**:** unsigned **(**15 **downto** 0**);** -- represents current phase of waveform.

**signal** quad**:** std\_logic\_vector **(**1 **downto** 0**);** --Indica en que región se encuentra.

**signal** index**:** signed **(**15 **downto** 0**);** -- index into current quadrant.

**begin**

-- This process adds "pitch" to the current phase every sampling period. Generates

-- an unsigned 16-bit sawtooth waveform. Frequency is determined by pitch. For

-- example when pitch=1, then frequency will be 0.745 Hz. When pitch=16,384, frequency

-- will be 12.2 kHz.

cnt\_pr**:** **process**

**begin**

**wait** **until** **rising\_edge(**clk**);**

count **<=** count **+** pitch**;**--Contador que aumenta pitch unidades por cada muestreo.

**end** **process;**

quad**<=** std\_logic\_vector**(**count**(**15 **downto** 14**));**--Escoge en que fase de la señal va.

index **<=** signed**(**"00" **&** count**(**13 **downto** 0**));**--Cambia el tipo de dato a signed y asigna el valor de la señal.

**with** quad **select**

data **<=** index **when** "00"**,**--Genera la primera rampa.

16383 **-** index **when** "01"**,**--Genera la primera caida hasta cero.

0 **-**index **when** "10"**,**--Genera la segunda caida hasta -16384.

index **-** 16383 **when** **others;**--Genera la subida hasta cero.

**end** Behavioral**;**

Este código en base a la frecuencia que se le indica va aumentando el contador de 16 bits para poder generar una señal triangular en la DAC considerando que se le manda un valor con signo. Por lo tanto entre mayor sea la frecuencia se tendrá una señal con menor calidad, el reloj que controla este código es el que indica la frecuencia de muestreo con la que trabajará la DAC. Para la resolución del problema se usa un algoritmo de dividir la señal en cuatro partes usando los últimos bits del contador para asignarle una suma o resta y así generar el diente de sierra.

**Código del Transmisor de I2S**

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**entity** dac\_if **is**

**Port** **(** SCLK **:** **in** STD\_LOGIC**;** -- serial clock (1.56 MHz)

L\_start**:** **in** STD\_LOGIC**;** -- strobe to load LEFT data

R\_start**:** **in** STD\_LOGIC**;** -- strobe to load RIGHT data

L\_data **:** **in** SIGNED **(**15 **downto** 0**);** -- LEFT data (15-bit signed)

R\_data **:** **in** SIGNED **(**15 **downto** 0**);** -- RIGHT data (15-bit signed)

SDATA **:** **out** STD\_LOGIC**);** -- serial data stream to DAC, Salida serial de los datos.

**end** dac\_if**;**

**architecture** Behavioral **of** dac\_if **is**

**signal** sreg**:** STD\_LOGIC\_VECTOR **(**15 **downto** 0**);** -- 16-bit shift register to do

-- parallel to serial conversion

**begin**

-- SREG is used to serially shift data out to DAC, MSBit first.

-- Left data is loaded into SREG on falling edge of SCLK when L\_start is active.

-- Right data is loaded into SREG on falling edge of SCLK when R\_start is active.

-- At other times, falling edge of SCLK causes REG to logically shift one bit left

-- Serial data to DAC is MSBit of SREG

dac\_proc**:** **process**

**begin**

**wait** **until** **falling\_edge(**SCLK**);**

**if** L\_start **=** '1' **then**

sreg **<=** std\_logic\_vector **(**L\_data**);** -- load LEFT data into SREG.

**elsif** R\_start **=** '1' **then**

sreg **<=** std\_logic\_vector **(**R\_data**);** -- load RIGHT data into SREG.

**else**

sreg **<=** sreg**(**14 **downto** 0**)** **&** '0'**;** -- logically shift SREG one bit left.

**end** **if;**

**end** **process;**

SDATA **<=** sreg**(**15**);** -- serial data to DAC is MSBit of SREG.

**end** Behavioral**;**

Este código se encarga de enviar serialmente la información que le llega desde el Convertidor de Frecuencia a Señal Triangular basándose en un registro de corrimiento y un selector que le indica que canal transmitir.

**Código del Display**

**library** IEEE**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**std\_logic\_arith**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

**entity** Leds\_Display\_7 **is**

**Port** **(** clkin **:** **in** STD\_LOGIC**;**

Entrada\_Disp\_1 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Entrada\_Disp\_2 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Entrada\_Disp\_3 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Entrada\_Disp\_4 **:** **in** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Salidas\_7seg **:** **out** STD\_LOGIC\_VECTOR **(**7 **downto** 0**);**

Control\_Disp\_7seg **:** **out** STD\_LOGIC\_VECTOR **(**3 **downto** 0**));**

**end** Leds\_Display\_7**;**

**architecture** Behavioral **of** Leds\_Display\_7 **is**

**signal** clkdiv**:** STD\_LOGIC\_VECTOR **(**16 **downto** 0**);**

**signal** contador\_disp**:** STD\_LOGIC\_VECTOR **(**1 **downto** 0**):=**"00"**;**

**begin**

**process(**clkin**)**

**begin**

**if** **rising\_edge(**clkin**)** **then**

clkdiv **<=** clkdiv **+**1**;**

**end** **if;**

**end** **process;**

**process(**clkdiv**(**16**),**contador\_disp**)**

**begin**

**if** **rising\_edge(**clkdiv**(**16**))** **then**

contador\_disp**<=**contador\_disp**+**1**;**

**if** contador\_disp**=**0 **then**

Control\_Disp\_7seg**<=**"0111"**;**

Salidas\_7seg**<=**not**(**Entrada\_Disp\_1**);**

**elsif** contador\_disp**=**1 **then**

Control\_Disp\_7seg**<=**"1011"**;**

Salidas\_7seg**<=**not**(**Entrada\_Disp\_2**);**

**elsif** contador\_disp**=**2 **then**

Control\_Disp\_7seg**<=**"1101"**;**

Salidas\_7seg**<=**not**(**Entrada\_Disp\_3**);**

**else**

Control\_Disp\_7seg**<=**"1110"**;**

Salidas\_7seg**<=**not**(**Entrada\_Disp\_4**);**

**end** **if;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

Este código se implementa para ver si el dato enviado a los canales está variando, indicándonos así si es que hay alguna falla mostrándonos el resultado en los displays.

**Código útil de la Memoria.**

**LIBRARY** IEEE**;**

**USE** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**USE** IEEE**.**STD\_LOGIC\_ARITH**.ALL;**

**USE** IEEE**.**STD\_LOGIC\_UNSIGNED**.ALL;**

**LIBRARY** UNISIM**;**

**USE** UNISIM**.**VCOMPONENTS**.ALL;**

--------------------------------------------------------------------------------

-- Entity Declaration

--------------------------------------------------------------------------------

**ENTITY** ROM\_Cancion\_exdes **IS**

**PORT** **(**

--Inputs - Port A

ADDRA **:** **IN** STD\_LOGIC\_VECTOR **(**7 **DOWNTO** 0**);**

DOUTA **:** **OUT** STD\_LOGIC\_VECTOR **(**23 **DOWNTO** 0**);**

CLKA **:** **IN** STD\_LOGIC

**);**

**END** ROM\_Cancion\_exdes**;**

**ARCHITECTURE** xilinx **OF** ROM\_Cancion\_exdes **IS**

**COMPONENT** BUFG **IS**

**PORT** **(**

I **:** **IN** STD\_ULOGIC**;**

O **:** **OUT** STD\_ULOGIC

**);**

**END** **COMPONENT;**

**COMPONENT** ROM\_Cancion **IS**

**PORT** **(**

--Port A

ADDRA **:** **IN** STD\_LOGIC\_VECTOR**(**7 **DOWNTO** 0**);**

DOUTA **:** **OUT** STD\_LOGIC\_VECTOR**(**23 **DOWNTO** 0**);**

CLKA **:** **IN** STD\_LOGIC

**);**

**END** **COMPONENT;**

**SIGNAL** CLKA\_buf **:** STD\_LOGIC**;**

**SIGNAL** CLKB\_buf **:** STD\_LOGIC**;**

**SIGNAL** S\_ACLK\_buf **:** STD\_LOGIC**;**

**BEGIN**

bufg\_A **:** BUFG

**PORT** **MAP** **(**

I **=>** CLKA**,**

O **=>** CLKA\_buf

**);**

bmg0 **:** ROM\_Cancion

**PORT** **MAP** **(**

--Port A

ADDRA **=>** ADDRA**,**

DOUTA **=>** DOUTA**,**

CLKA **=>** CLKA\_buf

**);**

**END** xilinx**;**

Este Código es una representación para poder realizar las operaciones con los registros de la memoria, contiene una entrada de reloj, una entrada de bus de dirección y una salida con el dato. Estos son nombrados con "ADDRA" para la dirección, "DOUTA", para el dato de salida y "CLKA" para la entrada de reloj. Esta memoria se crea a partir de un módulo de implementación y se le carga un archivo .coe con que se le indican los datos a guardar.

**Código para generar el archivo .coe en Matlab**

clc**,** clear all**,** close all

%==Frecuencia de Muestreo==

fm**=**50e6**/**2**^(**10**);**

f**=**8192**;**

**for** octava**=**1**:**8

**for** n**=**1**:**12

freq**(**n**+(**octava**-**1**)\***12**)=**440**\*(**2**^(**1**/**12**))^((**octava**-**4**)\***12**+(**n**-**10**));**

**end**

**end**

**for** octava**=**1**:**8

Do**(**octava**)=**freq**(**1**+(**octava**-**1**)\***12**);**

Dos**(**octava**)=**freq**(**2**+(**octava**-**1**)\***12**);**

Re**(**octava**)=**freq**(**3**+(**octava**-**1**)\***12**);**

Res**(**octava**)=**freq**(**4**+(**octava**-**1**)\***12**);**

Mi**(**octava**)=**freq**(**5**+(**octava**-**1**)\***12**);**

Fa**(**octava**)=**freq**(**6**+(**octava**-**1**)\***12**);**

Fas**(**octava**)=**freq**(**7**+(**octava**-**1**)\***12**);**

Sol**(**octava**)=**freq**(**8**+(**octava**-**1**)\***12**);**

Sols**(**octava**)=**freq**(**9**+(**octava**-**1**)\***12**);**

La**(**octava**)=**freq**(**10**+(**octava**-**1**)\***12**);**

Sib**(**octava**)=**freq**(**11**+(**octava**-**1**)\***12**);**

Si**(**octava**)=**freq**(**12**+(**octava**-**1**)\***12**);**

**end**

do**=**Do**;**

dos**=**Dos**;**

re**=**Re**;**

res**=**Res**;**

mi**=**Mi**;**

fa**=**Fa**;**

fas**=**Fas**;**

sol**=**Sol**;**

sols**=**Sols**;**

la**=**La**;**

sib**=**Sib**;**

si**=**Si**;**

%%Octava base=4

freq**=**freq**';**

%El Silencio, su frecuencia es cero

s**=**0**;**

% ==Parte para Duracion==

% Definimos la duración en segundos de la negra.

% tomada como 1 tiempo.

n**=**0.5**;**

np**=**n**/**8**;**%%pausa entre cada nota.

base**=**4**;**

b0**=**base**;**

b1**=**base**+**1**;**

% Hacemos un vector N = [ nota , duracion ]

NOTAS**=[**mi**(**b0**),**sol**(**b0**),**la**(**b0**),**la**(**b0**),**la**(**b0**),**si**(**b0**),**do**(**b1**),**do**(**b1**),**...

do**(**b1**),**re**(**b1**),**si**(**b0**),**si**(**b0**),**la**(**b0**),**sol**(**b0**),**sol**(**b0**),**la**(**b0**),**s...%%1

mi**(**b0**),**sol**(**b0**),**la**(**b0**),**la**(**b0**),**la**(**b0**),**si**(**b0**),**do**(**b1**),**do**(**b1**),**...

do**(**b1**),**re**(**b1**),**si**(**b0**),**si**(**b0**),**la**(**b0**),**sol**(**b0**),**sol**(**b0**),**la**(**b0**),**s...%%2

mi**(**b0**),**sol**(**b0**),**la**(**b0**),**la**(**b0**),**la**(**b0**),**do**(**b1**),**re**(**b1**),**re**(**b1**),**...

re**(**b1**),**mi**(**b1**),**fa**(**b1**),**fa**(**b1**),**mi**(**b1**),**re**(**b1**),**mi**(**b1**),**la**(**b0**),**s**,**...%%3

la**(**b0**),**si**(**b0**),**do**(**b1**),**do**(**b1**),**re**(**b1**),**mi**(**b1**),**la**(**b0**),**s**,**...

la**(**b0**),**do**(**b1**),**si**(**b0**),**si**(**b0**),**do**(**b1**),**la**(**b0**),**si**(**b0**),**s**,**...

mi**(**b0**),**sol**(**b0**),**la**(**b0**),**la**(**b0**),**la**(**b0**),**si**(**b0**),**do**(**b1**),**do**(**b1**),**...

do**(**b1**),**re**(**b1**),**si**(**b0**),**si**(**b0**),**la**(**b0**),**sol**(**b0**),**sol**(**b0**),**la**(**b0**),**s**,**...%6

s**];**

DURACION**=[**n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**,**n**,**...

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**4**,**...%%1

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**,**n**,**...

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**4**,**...%%2

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**,**n**,**...

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**4**,**...%%3

n**/**2**,**n**/**2**,**n**,**n**,**n**,**n**/**2**,**n**/**2**,**n**/**4**,**...%%4

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**,**n**/**4**,**...%%5

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**,**n**,**...

n**/**2**,**n**/**2**,**n**,**n**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**2**,**n**/**4**,**...%%6

n**/**4**];**

%%Agregar silencios

**for** i**=**1**:**length**(**NOTAS**)**

indice1**=**1**+**2**\*(**i**-**1**);**

indice2**=**indice1**+**1**;**

NOTAS2**(**indice1**)=**NOTAS**(**i**);**

NOTAS2**(**indice2**)=**s**;**

DURACION2**(**indice1**)=**DURACION**(**i**);**

DURACION2**(**indice2**)=**np**;**

**end**

N**=[**NOTAS2**',**DURACION2**'];**

y**=[];**

%==Armamos la señal==

**for** i**=**1**:**length**(**N**)**

fr**=**N**(**i**,**1**);**

t**=**N**(**i**,**2**);**

x**=(**0**:(**1**/**fm**):**t**);**

y**=[**y sin**(**fr**\***2**\***pi**.\***x**)];**

**end**

t**=(**0**:**1**:**length**(**y**)-**1**)/**fm**;**

subplot**(**2**,**1**,**1**),**plot**(**t**,**y**)**

t**=(**0**:**1**:**length**(**NOTAS2**)-**1**)\***sum**(**DURACION2**)/(**length**(**NOTAS2**)-**1**);**

subplot**(**2**,**1**,**2**),**stem**(**t**,**NOTAS2**)**

freq\_base**=**50e6**/**2**^(**26**);**

freq\_trabajo**=**50e6**/**2**^(**20**);**

NOTAS\_NEXYS**=**round**(**NOTAS2**/**freq\_base**);**

ParteA**=**dec2hex**(**NOTAS\_NEXYS**,**4**);**

DURACION\_NEXYS**=**round**(**DURACION2**\***freq\_trabajo**);**

ParteB**=**dec2hex**(**DURACION\_NEXYS**,**2**);**

**for** i**=**1**:**length**(**NOTAS\_NEXYS**)**

Palabra**(**i**,:)=**strcat**(**ParteB**(**i**,:),**ParteA**(**i**,:));**

**end**

%%Duracion Guardada de (23-16) y NOTAS (15,0)

% Hacemos el sonido

% en este comando

% esta toda la magia

sound**(**y**,**fm**,**16**)**

outfile**=**'Frecuencias.coe'**;**

s **=** fopen**(**outfile**,**'w+'**);** %opens the output file

fprintf**(**s**,**'%s\n'**,**'; VGA Memory Map '**);**

fprintf**(**s**,**'%s\n'**,**'; .COE file with hex coefficients '**);**

fprintf**(**s**,**'%s\n'**,**'memory\_initialization\_radix=16;'**);**

fprintf**(**s**,**'%s\n'**,**'memory\_initialization\_vector='**);**

**for** i**=**1**:**length**(**NOTAS\_NEXYS**)**

fprintf**(**s**,**'%c'**,**Palabra**(**i**,:));**

**if** i**~=**length**(**NOTAS\_NEXYS**)**

fprintf**(**s**,**'%c'**,**','**);**

**else**

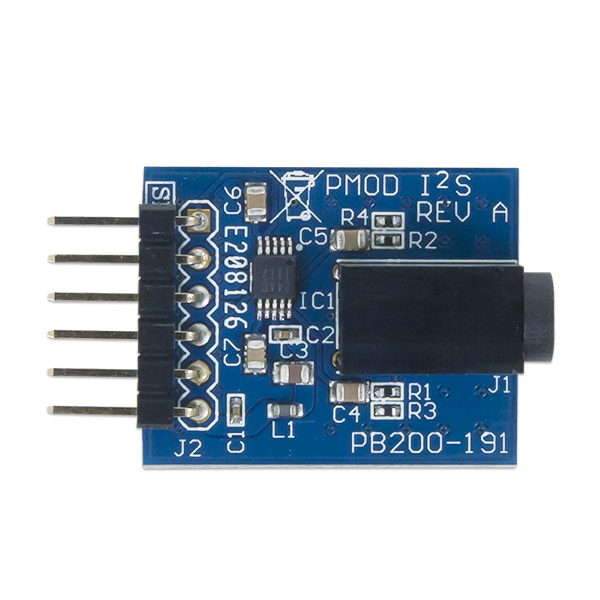
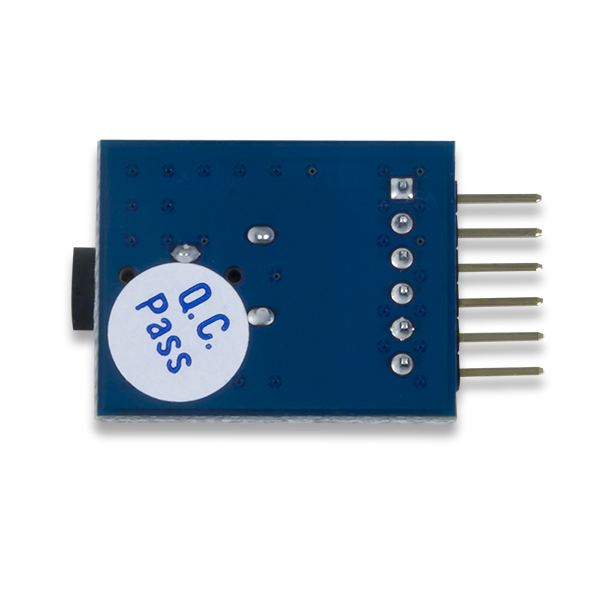
fprintf**(**s**,**'%c'**,**';'**);**

**end**

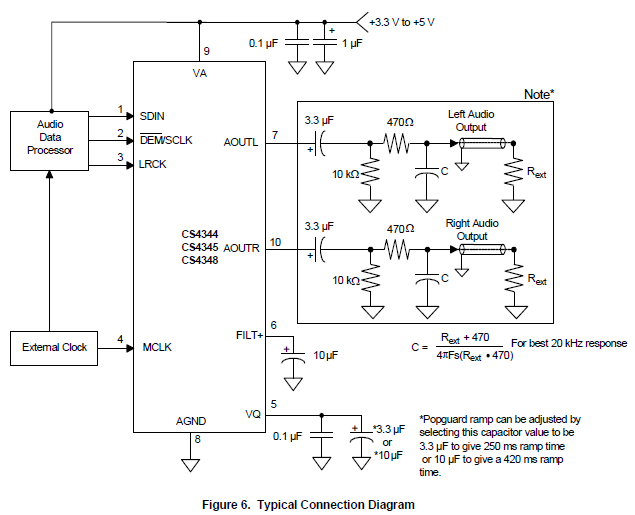
**end**

Genera el archivo Frecuencias.coe que contiene la frecuencia y la duración de cada nota incluyendo los silencios, para la realización de este programa se encontró la fórmula para calcular la frecuencia que tiene cada nota en base a las octavas y luego se guardó en un arreglo con el nombre correspondiente a cada nota, a la par se tiene otro arreglo en el que se específica la duración en segmentos de la nota negra por ejemplo la corchea = n/2, posteriormente se hacen 2 nuevos arreglos que contienen las notas de la canción y sus duraciones, a estos arreglos se le agrega un silencio entre cada nota para respetar sus tiempos y que no se junten por ejemplo 2 notas negras y que parezcan una blanca, una vez teniendo las notas en hertz y las duraciones en segundos se escalan según los tiempos con los que trabaja la Nexys para la frecuencia y cada cuando se cambia de nota (1/47.6). Luego se reproduce el sonido en Matlab para saber si es correcto y se genera el archivo .coe con la sintaxis adecuada y los números en hexadecimal.

**Placa Utilizada**

**Circuito Utilizado**



**Protocolo**

