UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

I.T. Telecomunicación (Sist. Electrónicos)

"Síntesis de música con dispositivos FPGA"

TRABAJO FINAL DE CARRERA

Autor/es:
Francisco de Borja Pérez Albors

Director/es:
Trinidad M. Sansaloni Balaguer

GANDIA, 2013

A mi familia, en especial a mi madre.

Índice:

Capítulo 1.-Introducción

Este proyecto desarrolla un sintetizador de sonidos musicales con técnicas digitales. En particular, aborda la síntesis del sonido de 3 instrumentos musicales: un oboe, un violín y la tuba. Cada uno de estos instrumentos tiene un timbre y un rango de frecuencias distinto, consideraciones que se han tenido en cuenta en la implementación.

El sistema se ha implementado en la placa DE2-115 de Altera, utilizándose un arpa con 12 láseres y 12 sensores para generar las señales que definen qué notas se están tocando. También es posible generar las notas con los pulsadores y switches disponibles en la misma placa de pruebas.

En el capítulo 2 se hace un breve repaso a los conceptos básicos relacionados con el sonido y las técnicas utilizadas en la síntesis de sonido y se elige la técnica que mejor se adapta a nuestros objetivos.

El capítulo 3 se centra en el modelado en Matlab del sistema. El capítulo incluye la descripción de los parámetros que definen el timbre de los instrumentos elegidos.

El capítulo 4 comienza con una breve descripción de la placa utilizada en la implementación y las características básicas del dispositivo programable que contiene, EP4EC115F29C7 e incluye la descripción de los módulos VHDL que se han utilizado para generar el código completo. El anexo 1 contiene los detalles de dicho código.
En el capítulo 5 se muestra la simulación del sistema y en el capítulo 6 los resultados de su implementación en el dispositivo FPGA.
Esta memoria acaba con la exposición de las principales conclusiones de este trabajo y la presentación de nuevas ideas que pueden mejorar los resultados y prestaciones de este proyecto.

Capítulo 2.-Conceptos básicos.

2.1.- El sonido

Se define como sonido a la vibración periódica del medio elástico material que baña el mecanismo auditivo (generalmente el aire). [1],[6],[9].

Para que se produzca un sonido se requiere la existencia de un cuerpo vibrante, este puede ser debido por ejemplo a una cuerda tensa, una varilla, una lengüeta. En el caso del sonido generado electrónicamente el fenómeno de la agitación molecular del aire se origina en el cono de un altavoz.

Como onda, el sonido responde a las siguientes características:
- Es una onda mecánica: Las ondas mecánicas no pueden desplazarse en el vacío, necesitan hacerlo a través de un medio material (aire, agua, cuerpo sólido), además éste debe ser elástico (ya que un medio rígido no permite la transmisión del sonido, porque no permite las vibraciones). La propagación de la perturbación se produce por la compresión y expansión del medio por el que se propagan. La elasticidad del medio permite que cada partícula transmita la perturbación a la partícula adyacente, dando origen a un movimiento en cadena.
- Es una onda longitudinal: El movimiento de las partículas que transporta la onda se desplaza en la misma dirección de propagación de la onda.
- Es una onda esférica: Las ondas sonoras son ondas tridimensionales, es decir, se desplazan en tres direcciones y sus frentes de ondas son esferas radiales que salen de la fuente de perturbación en todas las direcciones. El principio de Huygens afirma que cada uno de los puntos de un frente de ondas esféricas puede ser considerado como un nuevo foco emisor de ondas secundarias también esféricas, que como la originaria, avanzarán en el sentido de la perturbación con la misma velocidad y frecuencia que la onda primaria.

Cualquier sonido sencillo, como una nota musical, puede describirse en su totalidad especificando tres características de su percepción: el tono, la intensidad y el timbre. Estas características corresponden exactamente a tres características físicas: la frecuencia, la amplitud y la composición armónica o forma de onda.

2.1.1.-Tono y frecuencia

Aunque ambos conceptos están relacionados no son lo mismo. El tono será siempre una magnitud subjetiva de la altura o gravedad del sonido que está formado por la frecuencia fundamental percibida por el oído, acompañada de otras características como el contenido armónico, la intensidad con que se produce y la duración de la nota.
Por su parte, la frecuencia es una definición física cuantitativa, que se puede medir con aparatos sin una referencia auditiva, la percepción puede ser diferente en distintas situaciones, así para una frecuencia específica no siempre será la misma.

A continuación podemos ver las gamas de frecuencia aproximadas de varios instrumentos.

Flauta	DO4 C4	—	DO7 C7	261 Hz a 2093 Hz
Cuerno inglés	SOL3 G3	—	DO6 C6	196 Hz a 1046 Hz
Violín	SOL3 G3	—	DO6 C6	196 Hz a 1046 Hz
Viola	DO3 C3	—	MI4 E4	130 Hz a 329 Hz
Trompeta	MI13 E3	—	LA5 A5	164 Hz a 880 Hz
Piano	LA0 A0	—	DO8 C8	27 Hz a 4186 Hz
Oboe	LA3 A3	—	SOL6 G6	220 Hz a 1568 Hz
Tuba	FA1 F1	—	FA4 F4	43 Hz a 349 Hz
Cuerno francés	SI1 B1	—	FA5 F5	61 Hz a 689 Hz
Violoncelo	DO3 C2	—	MI3 E3	65 Hz a 164 Hz
Contrabajo	MI1 E1	—	MI3 E3	41 Hz a 164 Hz

Figura 1: Frecuencias instrumentos.

2.1.2.-Timbre y estructura armónica

El timbre de un sonido permite la identificación del instrumento o foco emisor del sonido. El sonido emitido por un instrumento musical no es una vibración simple, sino que es una mezcla de señales cuyas frecuencias constituyen valores múltiplos de esta fundamental denominados armónicos. Lo que hace distinguible una misma nota emitida por un instrumento o por otro es la distribución de los porcentajes de cada armónico y su relación de fase con respecto a la fundamental.

Jean Fourier demostró matemáticamente que toda función periódica sinusoidal puede ser descompuesta en una serie de funciones sinusoidales. Por el contrario, una onda sinusoidal no puede ser descompuesta, lo que confirma que estás son las más puras obtenibles. Por tanto cualquier onda compleja la podremos sintetizar con un número definido de ondas sinusoidales.

A continuación podemos ver la descomposición espectral de algunos instrumentos musicales:

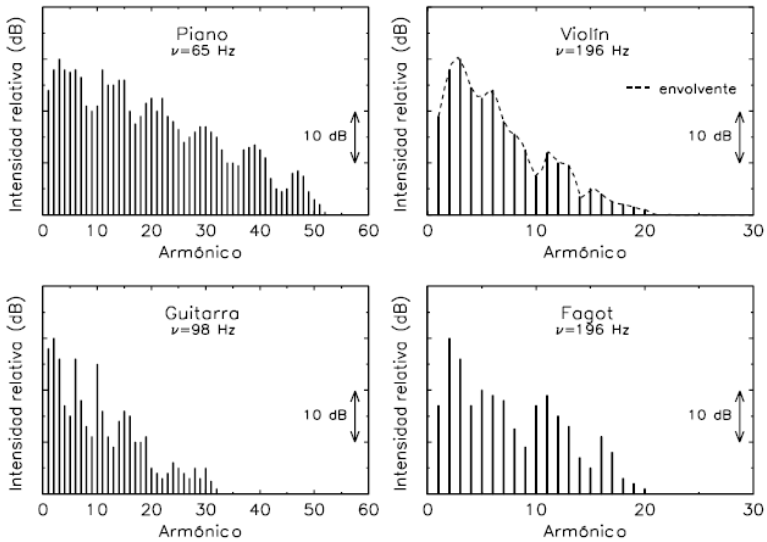


Figura 2: Descomposición espectral

2.1.3.-Intensidad y sonoridad

La intensidad es una magnitud física y medible análoga a la potencia eléctrica. Por definición, es la energía sonora transportada por unidad de tiempo y que atraviesa un área unidad perpendicular a la dirección de propagación.

La intensidad corresponde poco a las sensaciones sonoras que produce, ya que no existe una proporcionalidad entre la intensidad objetiva y la sensación percibida. Por ello a la sensación subjetiva de la intensidad se define como sonoridad, que depende también de otros factores como la frecuencia, el ancho de banda y la duración del sonido percibido.

Para medir la sonoridad se hace uso de la escala logarítmica, en este caso la unidad que se usa es el decibelio. A continuación podemos ver una tabla con niveles sonoros aproximados:

TABLA DE NIVELES SONOROS

	NIVEL SPL APROXIMADO
Estudio de grabación en silencio	+ 0 dB
Murmuración a tres metros	+ 10 dB
Paso de las hojas de un libro	+ 10 dB
Susurro a un metro	+ 20 dB
Calle sin tráfico distrito residencial	+ 30 dB
Dormitorio reposado en pleno día	+ 35 dB
Conversación a tres metros	+ 45 dB
Orquesta de cuerda y de viento	+ 60 dB
Oficina comercial con máquinas escribir	+ 65 dB
Orquesta de metales	+ 75 dB
Despertador a cuarenta centímetros	+ 80 dB
Calle ruidosa con mucho tráfico	+ 90 dB
Fábrica industrial ruidosa	+ 100 dB
Umbral doloroso	+ 120 dB
Avión reactor a doscientos metros	+ 140 dB
Cohete espacial a unos tres mil metros	+ 200 dB

Figura 3: Tabla de niveles sonoros

2.1.4.-Intensidad dinámica

En la variación temporal de la intensidad tenemos tres fases que son el ataque, el sonido sostenido y el decaimiento.

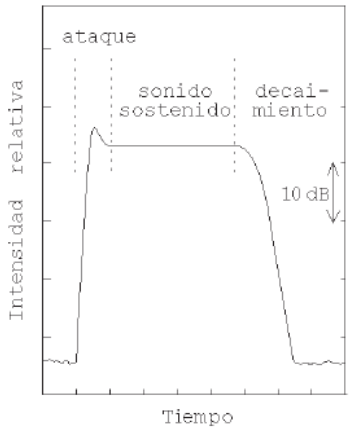


Figura 4: Intensidad dinámica.

El ataque es el intervalo de tiempo requerido para que un sonido procedente de un instrumento musical alcance su valor máximo de intensidad.

El decaimiento esta referido a la rapidez con que un sonido se extingue una vez ha cesado la excitación del elemento vibrante.

El tiempo de sostenimiento describe el intervalo de duración de una nota entre el ataque y el decaimiento. Es un estado estacionario y de emisión continuada de sonido

A continuación podemos ver la intensidad dinámica de varios instrumentos:

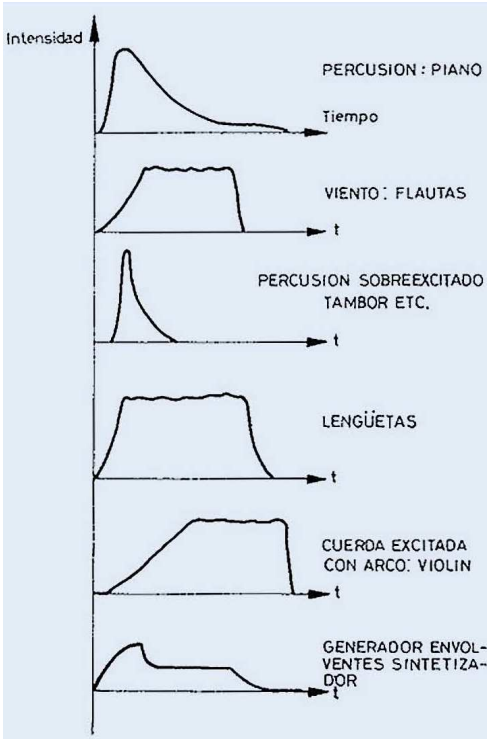


Figura 5: Intensidad dinámica varios instrumentos

2.2.- Métodos de síntesis del sonido

Los métodos de síntesis del sonido pueden ser en función del tiempo o de la frecuencia. Los métodos de síntesis en frecuencia son la síntesis aditiva, la substractiva en FM o la síntesis LPC [1], [5], [7], [8].

2.2.1.-Síntesis en frecuencia: Aditiva

Este método se basa en el teorema de Fourier, que como hemos visto anteriormente, demuestra que podemos obtener cualquier señal compleja mediante la suma de señales sinusoidales con distintas amplitudes y fases. Necesitaremos tres funciones de control para cada oscilador sinusoidal (frecuencia, amplitud y fase).

La expresión que usaremos es:

$$y(n) = \sum_{k=0}^{M-1} A_k(n) \sin\left[2\pi F_k(n)\right]$$

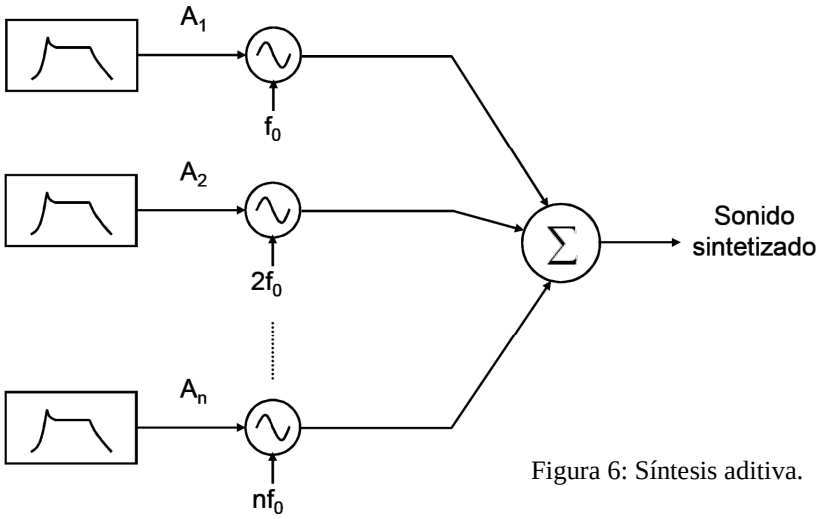


Figura 6: Síntesis aditiva.

2.2.2.-Síntesis en frecuencia: Substractiva

En la síntesis sustractiva el proceso es el opuesto al utilizado en la síntesis aditiva. El sonido se construye, eliminando componentes indeseados a partir de un sonido inicial. En éste intervienen tres elementos: generador, filtro y amplificador. La fuente o generador puede ser cualquier tipo de sonido pero suele usarse un ruido de banda ancha. El filtro seleccionaremos la parte que interesa y el amplificador controla el volumen del sonido. El proceso completo puede emular la característica espectral de un instrumento.

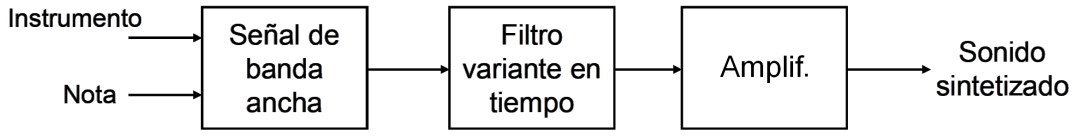


Figura 7: Síntesis substractiva

2.2.3.-Síntesis en frecuencia: FM

La síntesis FM está basada, como su propio nombre indica, en la modulación en frecuencia. La frecuencia de una onda determinada se modula por otra onda de distinta frecuencia. El resultado contiene elementos de ambas frecuencias junto con nuevos armónicos relacionados matemáticamente con las frecuencias originales. Se puede demostrar teóricamente que cualquier sonido, por complejo que sea, puede ser modelado mediante una serie de modulaciones en frecuencia de ondas sinusoidales.

La ecuación general es:

$$y(t) = A_c \cos(2\pi f_c t + 2\pi k_f \int_o^t m(\tau) d\tau)$$

El esquema de este método es:

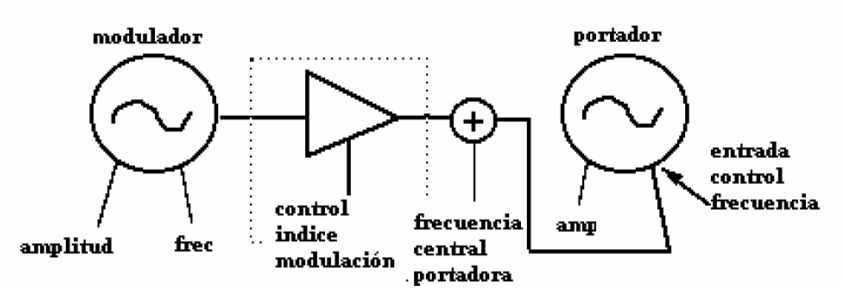


Figura 8: Síntesis FM

2.2.4.-Síntesis en frecuencia LPC

Las siglas LPC se refieren a *Linear Predictive Coding.* Este tipo de síntesis es muy utilizada en síntesis y reconocimiento de voz. En esta síntesis primero tenemos que establecer un modelo que represente los parámetros de la señal que se desea extraer, un algoritmo de análisis basado en el modelo y por ultimo necesitamos un programa de síntesis.

Por ejemplo el modelo para reproducir el habla seria:

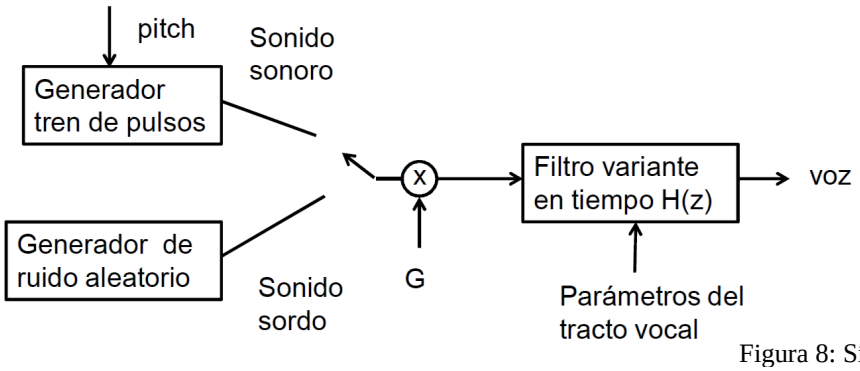


Figura 8: Síntesis LPC

2.2.5.-Síntesis en tiempo: Wavetable

En esta forma de síntesis se digitalizara el sonido original y almacenaremos las muestras en una tabla de onda (Wavetable).

Las ventajas que presenta este método es que es fácil de implementar, obtenemos muy buena calidad, es eficiente en el tratamiento de señales periódicas. Mientras que los inconvenientes son que no es muy útil para crear sonidos nuevos y se necesita una gran cantidad de memoria para almacenar los sonidos muestreados.

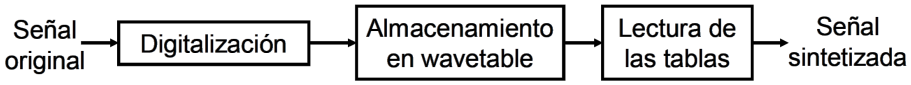


Figura 9: Síntesis Wavetable

2.2.6.-Síntesis en tiempo: Granular

Se basa en sintetizar sonidos a partir de pequeños elementos de señal en el dominio del tiempo llamados sonidos atómicos o granos. Estos granos pueden tener una duración comprendida entre un milisegundo y más de cien. Tenemos diferentes dependiendo de la forma de obtener los granos, estos métodos son:

- Síntesis granular asíncrona (AGS): los granos son dispersados sobre una región en el dominio de la frecuencia denominada nube. Los granos pueden tener formas de onda similares o diferentes. La forma de onda puede ser una sinusoide enventanada, una señal muestreada o bien obtenida mediante un modelo físico.
- Síntesis granular síncrona de Pitch (PSGS), los granos se obtienen de la STFT de la señal original. La longitud de la ventana rectangular usada en la STFT es el periodo del sonido sintetizado, y cada grano corresponde por tanto a un periodo de la señal.

2.2.7.-Síntesis en tiempo: Modelado físico

En este método se usa un modelo matemático para simular las propiedades físicas de la fuente del sonido. Para generar el sonido se usa resonadores (indican cómo vibran los elementos físicos del instrumento y describen los materiales físicos empleados en el instrumento) y generadores (excitación que produce el sonido debida a la interacción del usuario con el instrumento).

2.2.8.- Elección

En nuestro caso se ha decido utilizar la síntesis aditiva ya que la para definir armónico solo necesitaremos tres parámetros (amplitud, frecuencia y fase) que podremos tener almacenados en memoria. Además al cambiar de frecuencia o de instrumento solo deberemos cambiar de memoria y el resto del código será el mismo.

La síntesis substractiva se ha descartado debido a que para lograr generar la misma onda que la síntesis aditiva se necesita una cantidad mayor de ondas. Nosotros solo deberemos sumar veinte ondas para conseguir el instrumento deseado.

El motivo para descartar el modelado físico, la modulación FM y la síntesis en frecuencia LPC ha sido debido a que la función variaría mucho de un instrumento a otro. Por lo que tendríamos mayor cantidad de variables que almacenar y por tanto aumentaría la memoria necesaria.

Por ultimo el motivo para descartar la síntesis Wavetable y la síntesis granular son la gran cantidad de memoria que deberíamos almacenar.

Capítulo 3.-Modelado en Matlab.

En este capítulo vamos a simular y sintetizar el sonido de 3 instrumentos musicales: un oboe, un violín y la tuba utilizando Matlab.

En primer lugar modelaremos varias notas de los distintos instrumentos, para después comparar las simulaciones del programa con los modelos y saber si son correctos.
Los datos de las diferentes amplitudes y fases se han obtenido de "*SHARC TIMBRE DATABASE*"[4].

3.1.- Diagrama de bloques del sistema

En el modelo de matlab partiremos de los datos de la frecuencia fundamental (fundHz), la amplitud de la señal a esta frecuencia (a_freqHz), la amplitud de cada armónico (a_n_a) y su fase (a_n_p).
A continuación podemos ver un esquema del proceso que seguimos (para facilitar la comprensión sólo se han incluido los primeros armónicos).

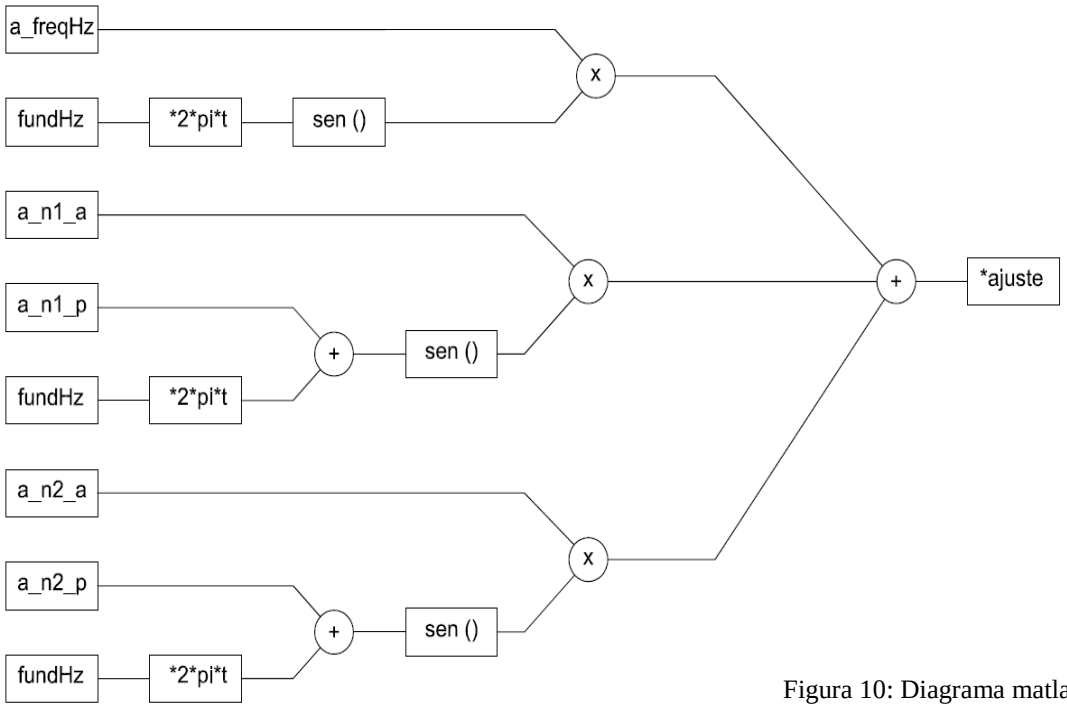


Figura 10: Diagrama matlab

Lo que hacemos es multiplicar la frecuencia fundamental por 2*pi*t de este modo tendremos unos datos que varían en el tiempo a la frecuencia que nosotros queremos (48000Hz). Si esta señal la tenemos desfasada (es uno de los armónicos) debemos sumarle la fase. A continuación utilizamos la función "sen" para obtener una onda sinusoidal. Ahora multiplicaremos esta onda por su amplitud correspondiente. Por ultimo sumaremos todas las señales obtenidas y ajustaremos su amplitud.
Los valores iniciales en VHDL los tendremos almacenados en las diferentes memorias (a_n_p estarán almacenados en fase, a_n_a en amplitud y a_freq los obtendríamos a partir de los valores almacenados en incrementos).

3.2.- Oboe

El primer instrumento que vamos a modelar será el oboe. Para este instrumento se generarán las notas comprendidas entre C4 y C5. El código será el siguiente muestra cómo se genera la primera nota (C4).

```
function c4_oboe
    fs=48000;
    Ts=1/fs;

    fundHz=261.626;
    a_freqHz=9941.78;

    a_n1_p =-1.898;
    a_n1_a =2931.55;
    a_n2_p =-1.427;
    a_n2_a =749.7;
    a_n3_p =-2.018;
    a_n3_a =2679.48;
    a_n4_p = -1.442;
    a_n4_a = 4936.02;
    a_n5_p = -1.666;
    a_n5_a = 5374;
    a_n6_p = -2.94;
    a_n6_a = 2510.78;
    a_n7_p = 2.653;
    a_n7_a = 307.69;
    a_n8_p = -0.539;
    a_n8_a = 442.5;
    a_n9_p = -0.461;
    a_n9_a = 739.29;
    a_n10_p = -0.158;
    a_n10_a = 533.49;
    a_n11_p = 1.196;
    a_n11_a = 461.5;
    a_n12_p = 1.036;
    a_n12_a = 497.54;
    a_n13_p = 0.515;
    a_n13_a = 304.65;
    a_n14_p = 0.372;
    a_n14_a = 24.51;
    a_n15_p = 0.261;
    a_n15_a = 48.97;
    a_n16_p = -0.36;
    a_n16_a = 14.76;
    a_n17_p = -1.328;
    a_n17_a = 81.97;
    a_n18_p = -0.321;
    a_n18_a = 46.85;
    a_n19_p = -1.236;
    a_n19_a = 15.9;
    t_fin=5; %5 segundos
    n_max=t_fin*fs;
    t=[1:n_max]*Ts;
```

```
senyal=((a_freqHz*sin(2*pi*t*fundHz))+(a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))+(a_n2_a*sin(2*
pi*t*3*fundHz+a_n2_p))+(a_n3_a*sin(2*pi*t*4*fundHz+a_n3_p))+(a_n4_a*sin(2*pi*t*5*fundHz+
a_n4_p))+(a_n5_a*sin(2*pi*t*6*fundHz+a_n5_p))+(a_n6_a*sin(2*pi*t*7*fundHz+a_n6_p))+
(a_n7_a*sin(2*pi*t*8*fundHz+a_n7_p))+(a_n8_a*sin(2*pi*t*9*fundHz+a_n8_p))+(a_n9_a*sin(2*
pi*t*10*fundHz+a_n9_p))+(a_n10_a*sin(2*pi*t*11*fundHz+a_n10_p))+(a_n11_a*sin(2*pi*t*12*
fundHz+a_n11_p))+(a_n12_a*sin(2*pi*t*13*fundHz+a_n12_p))+(a_n13_a*sin(2*pi*t*14*fundHz+
a_n13_p))+(a_n14_a*sin(2*pi*t*15*fundHz+a_n14_p))+(a_n15_a*sin(2*pi*t*16*fundHz+
a_n15_p))+(a_n16_a*sin(2*pi*t*17*fundHz+a_n16_p))+(a_n17_a*sin(2*pi*t*18*fundHz+
a_n17_p))+(a_n18_a*sin(2*pi*t*19*fundHz+a_n18_p))+(a_n19_a*sin(2*pi*t*20*fundHz+
a_n19_p)))*10^-4.5;
   %senyal NO puede pasar de +-1
   %senyal = (a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))

   hold on
   plot(t,senyal)
   AXIS([0 0.01 -1 1])
   sound(senyal,fs)
```

Para utilizar esta función debemos introducir en Matlab:

```
c4_oboe
```

Obteniendo:

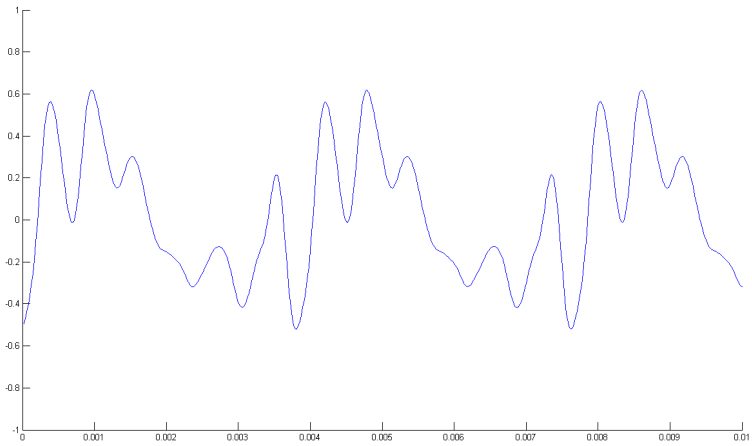


Figura 11: C4 oboe

De forma similar, se modela la nota c5 del oboe. En el anexo2 se muestra el código matlab correspondiente a esta función.

La forma de onda que se genera en este caso se muestra en la siguiente figura.

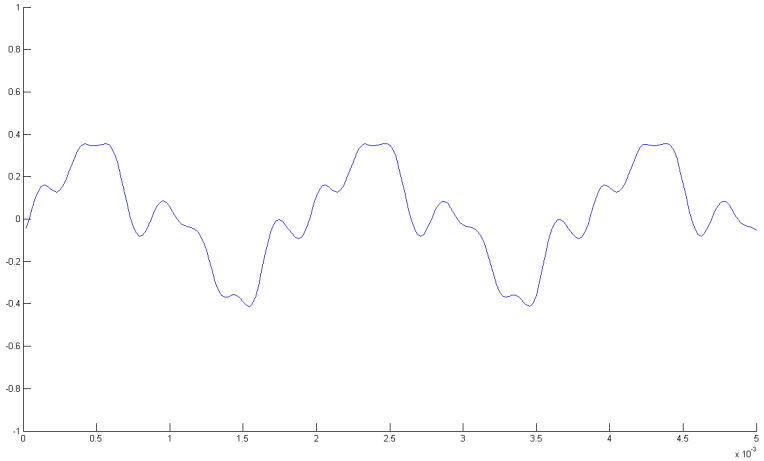


Figura 12: C5 oboe

3.3.-Violín

El segundo instrumento que modelaremos será el violín.
Para este instrumento se generarán las notas comprendidas entre C5 y C6. Los códigos correspondientes a estos ejemplos se muestran en el anexo2.

La forma de onda correspondiente a su primera nota, C5, se muestra en la siguiente figura.

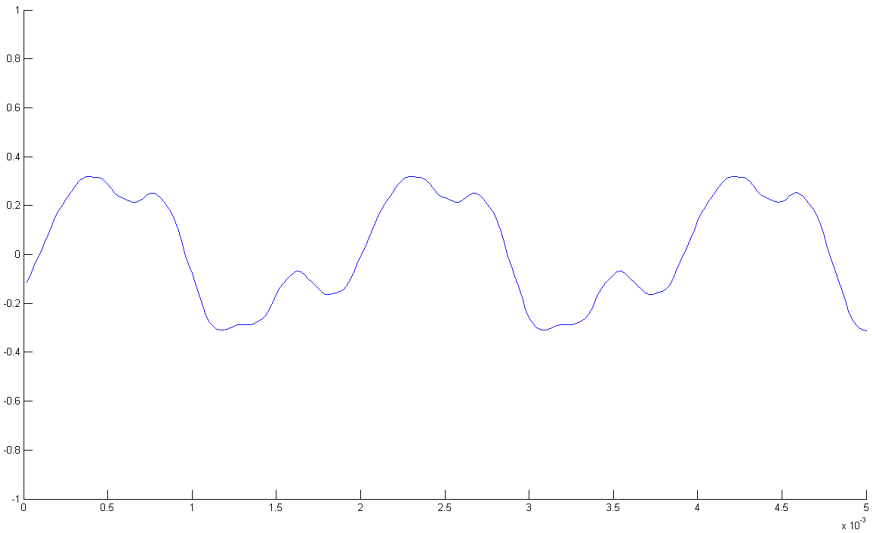


Figura 13: C5 violín

La siguiente figura muestra la forma de onda correspondiente a la nota C6 generada para modelar el sonido de un violín.

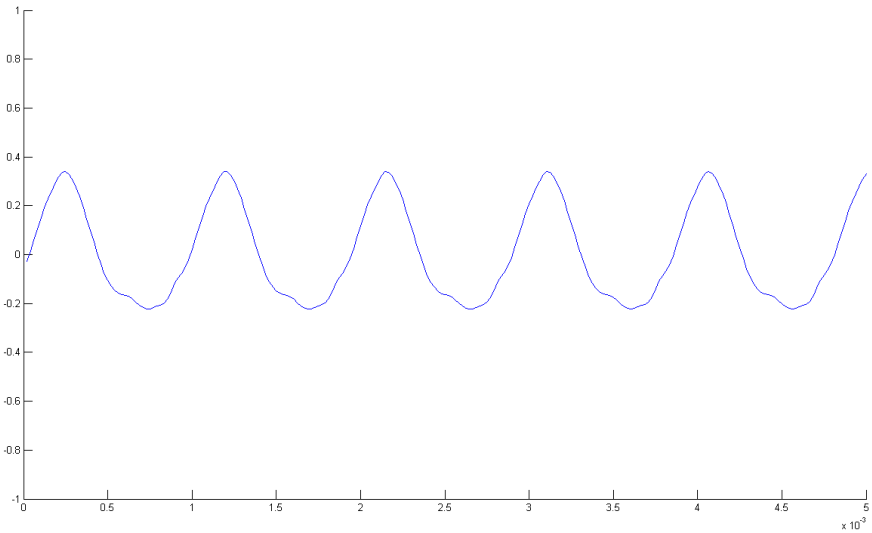


Figura 14: C6 violín

3.4.- Tuba

Por último modelaremos la tuba. Para este instrumento se generarán las notas comprendidas entre C3 y C4. Los códigos correspondientes a estos ejemplos se muestran en el anexo2.

La forma de onda correspondiente a su primera nota, C3, se muestra en la siguiente figura.

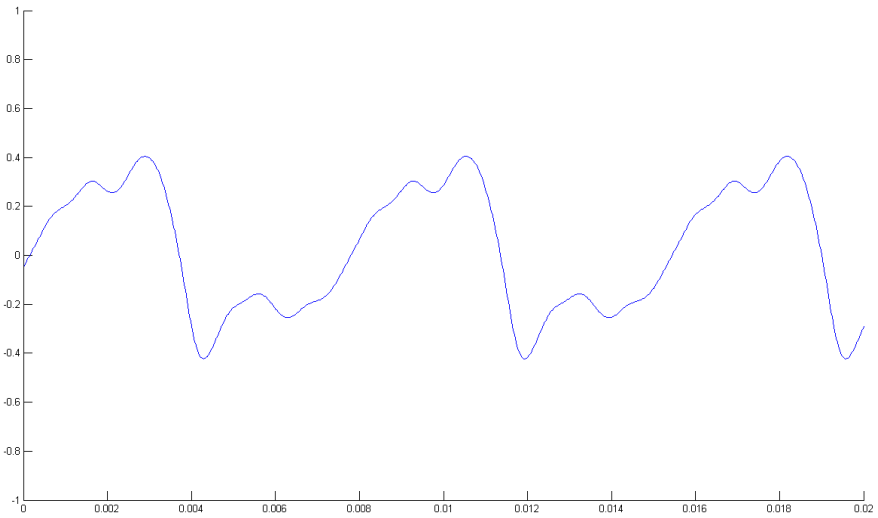


Figura 15: C3 tuba

La siguiente figura muestra la forma de onda correspondiente a la nota C4 generada para modelar el sonido de una tuba.

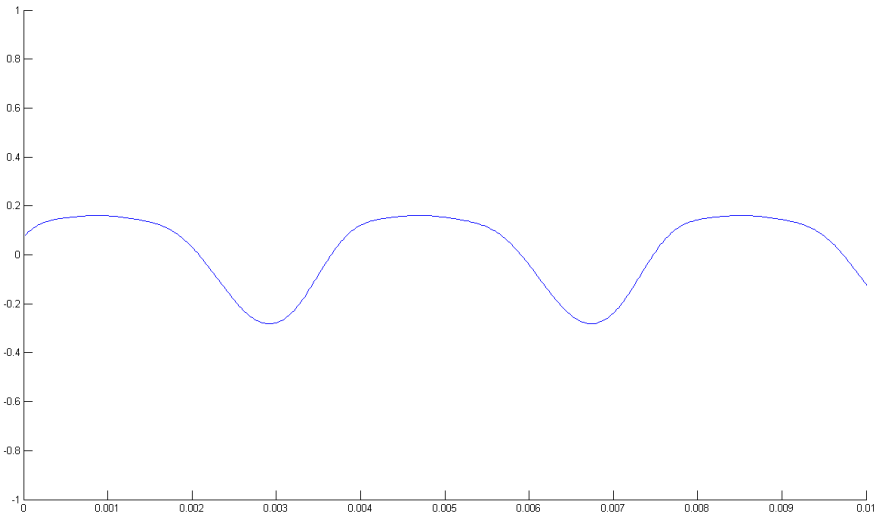


Figura 16: C4 tuba

Capítulo 4.-Implementación del sistema

Para implementar es sistema se ha elegido la placa DE2-115 de Altera que incluye un dispositivo programable, EP4EC115F29C7. En este capítulo se van a resumir brevemente las características de las herramientas HW y SW que han hecho posible esta implementación, así como la descripción de los principales módulos VHDL que configuran el sistema.

4.1. Herramientas utilizadas

4.1.1.- Plataforma Hardware utilizada

En la siguiente figura se muestra el aspecto de la placa escogida para la implementación del sistema, “DE2-115 BOARD” [10], [11].

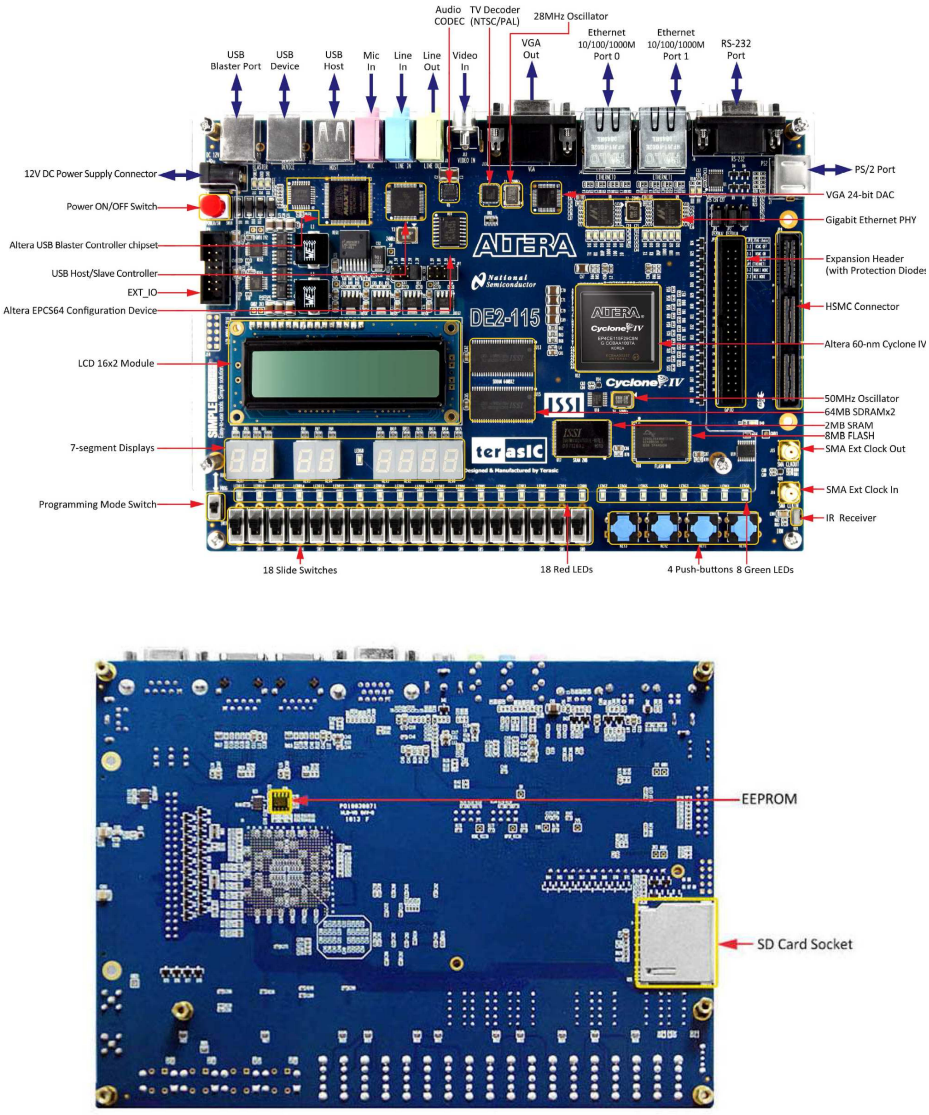


Figura 17: Vista delantera y vista trasera

El diagrama de bloques de la placa lo podemos ver a continuación:

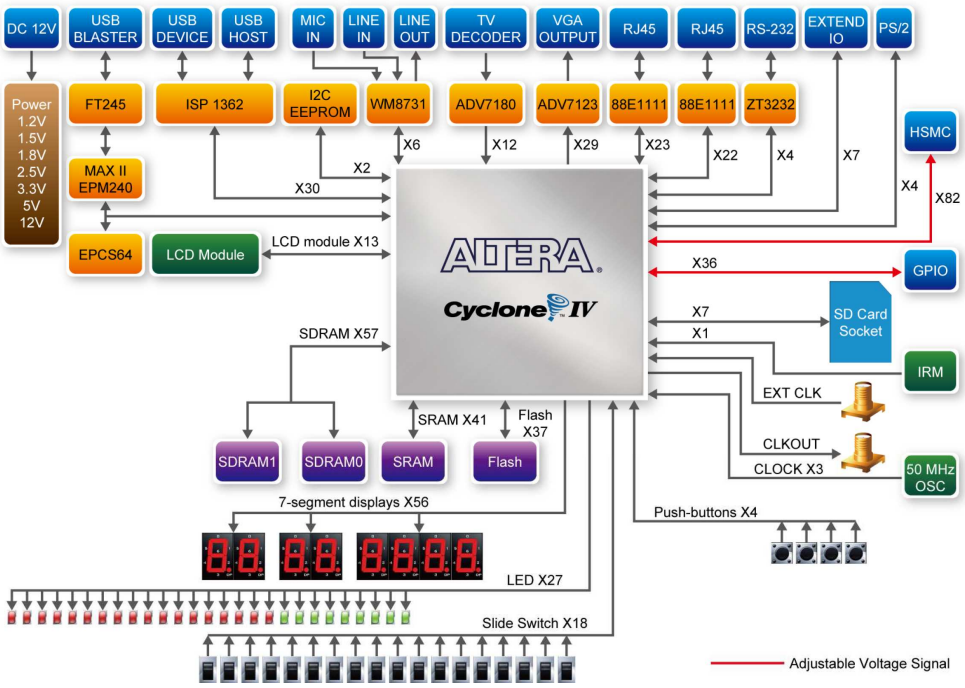


Figura 21: Diagrama.

Características:

- Configuración:
  - Modo de configuración JTAG y AS.
  - Configuración serie EPCS64
  - Configuración de la placa mediante USB.
- Dispositivos de memoria:
  - 128MB (32Mx32bit) SDRAM.
  - 2MB (1Mx16) SRAM.
  - 8 MB (4Mx16) con el modo 8-bit.
  - 32Kb EEPROM.
- Relojes:
  - Tres osciladores a 50MHz.
  - Conexión SMA para reloj externo.
- Audio:
  - Codificador/decodificador de 24 bits
  - Entrada, salida de audio y conector para micrófono.
- Display:
  - LCD de 16x2.
- Entradas e indicadores:
  - 18 interruptores y 4 pulsadores.
  - 18 LEDs rojos y 9 verdes.
  - Display de 7 segmentos.

.- Alimentación:

  - Entrada DC.

- Reguladores LM3150MH.

- Conectores
  - Dos puertos Ethernet 10/100/1000 Mbps.
  - HSMC.
  - I/O configurable a diferentes niveles de tensión (3.3/2.5/1.8/1.5V).
  - USB tipo A y B.
  - Puerto de expansión de 40 pines:

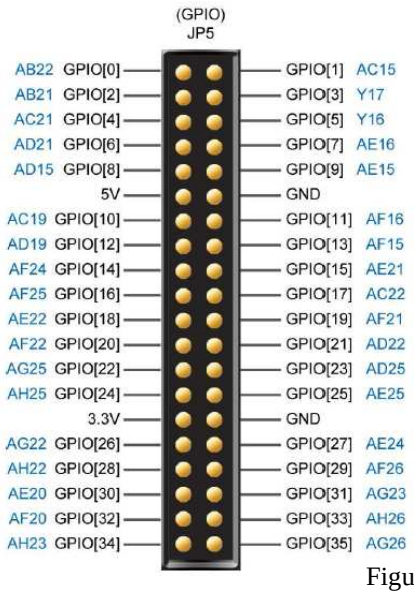


Figura 20: Puerto de expansión.

  - Conector VGA.
  - Conector serie DB9 para RS-232
  - Conector para teclado/ratón PS/2

- Otras características
  - Módulo de recepción de control remoto
  - Conector de entrada de TV y decodificador (NTSC/PAL/SECAM).

De estos recursos se han utilizado los siguientes:

- LABs (*Logic Array Blocks*) y LEs (*Logic Elements*).
- Memoria: Necesitaremos almacenar diversos elementos (fases, incrementos, amplitudes y los valores de una onda sinusoidal).
- Conectores:
  - USB para la programación del dispositivo.
  - Puerto de expansión de 40 pines, para conectar el arpa.
- Relojes: Usaremos uno de los osciladores de 50MHz para obtener los diferentes relojes de nuestro sistema.
- Audio: Nos permitirá escuchar la señal generada (posteriormente veremos mejor este elemento).
- Interruptores: los usaremos para controlar que notas queremos desactivar y para otras señales (RST, WEN y INIT). En ausencia del arpa servirán también para generar notas.
- LEDs rojos: Los usaremos para visualizar cual es la primera nota que estamos tocando.

El dispositivo FPGA incluido en la placa es el *Cyclone IV EP4CE115F29, su* estructura de LABs se muestra a continuación:

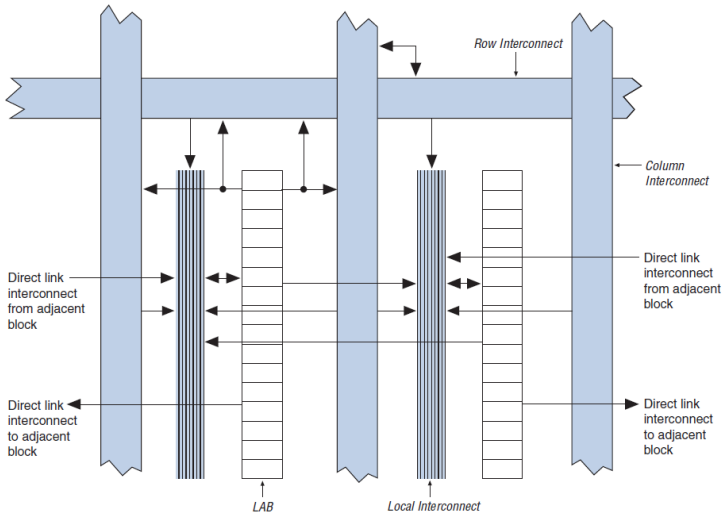


Figura 18: Estructura LAB.

- 114480 LEs cada uno de ellos es como podemos observar en el siguiente esquema:

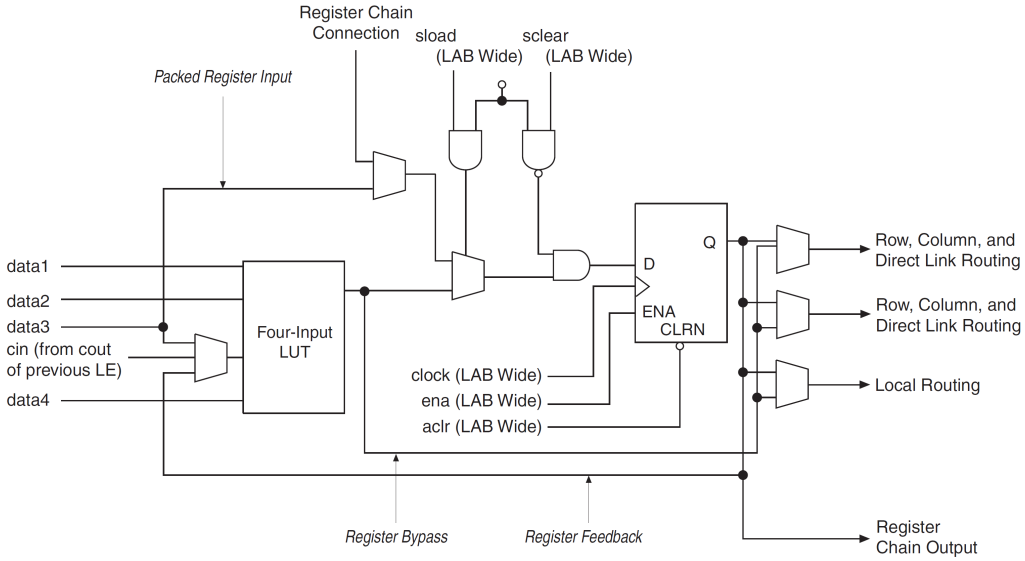


Figura 19: Estructura LE.

- 432 M9K bloques de memoria.
- 3,888 Kbits de memoria embebida.
- 4 PLLs.

4.1.2. Herramientas Software empleadas.

En primer lugar hemos usado el “Xillinx ISE design suite” para crear el proyecto y los diferente módulos. También se ha usado para realizar las simulaciones.
Posteriormente se ha creado otro proyecto usando el “Altera Quartus II”. Se han usado los bloques que teníamos anteriormente. Mediante este software hemos asignado los pines y programado la “DE2-115 BOARD”.

4.2.- Descripción del código VHDL

Tras modelar los diferentes instrumentos vamos a presentar las principales características de los módulos VHDL utilizados para implementar el proyecto.

La estructura del código se muestra en el siguiente diagrama:

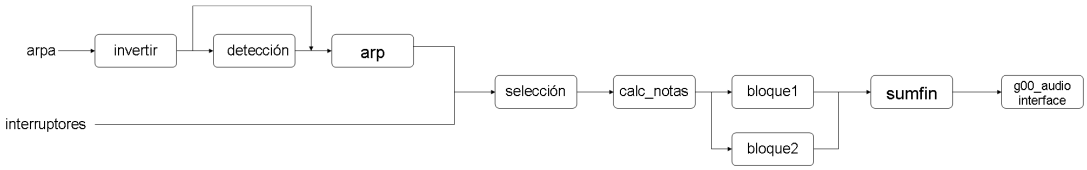


Figura 23: Diagrama.

Invertir: Mediante este bloque invertiremos las señales procedentes del arpa.
Detección: Detectara que LEDs no funcionan
Arp: En este bloque se reciben las señales invertidas y anulara las que no funcionan.
Selección: Seleccionara entre las señales procedentes del arpa y las de los interruptores.
Cal_notas: Es el encargado de convertir las señales procedentes del bloque anterior en notas válidas para nuestro programa.

Los bloques 1 y 2 se han añadido para facilitar la comprensión estos no aparecen como tales en el programa, sino que están los módulos que encontramos dentro de ellos.
A la entrada de cada bloque llegarían los señales de dos relojes, la señal de nota1 (en el caso del bloque 2 nota2) y la señal con el número de instrumento. A la salida de cada uno de los bloques tendríamos dos señales de audio (la del canal derecho y la del canal izquierdo).
A continuación podemos ver cómo son cada uno de estos bloques:

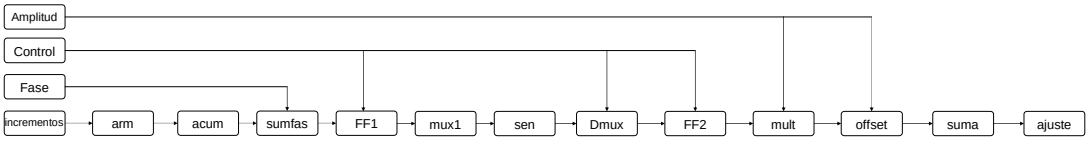


Figura 24: Bloques

Sumfin: Es el encargado de sumar las señales procedentes de los bloques 1 y 2.
G00_audio interface: Es el encargado de transformar las señales que genera nuestro programa en otras válidas para poder ser escuchadas mediante la salida de audio.

Para el correcto funcionamiento del sistema son necesarios 3 relojes.
CLK2: 48000Hz. Es la frecuencia standard para aplicaciones de audio y video profesional y DVD. Este reloj se usará para generar y procesar la señal de audio, en particular, en los bloques fase, amplitud, incrementos, arm, acum, FF1, arp y calc_notas.
CLK3: 25MHz. Este reloj se usará para el bloque g00_audio_interface. Debido a que el interfaz de audio necesita un reloj a 25MHz.
CLK4: 1.008MHz. Este reloj se usara para los bloques control, mux1, dmux y FF2. Este reloj debe de ser 21 veces más rápido que el CLK2, de este modo conseguiremos generar la frecuencia fundamental, los 19 armónicos y un ciclo constante en un ciclo del reloj de 48KHz.

Los tres relojes que tenemos los obtendremos a partir de la señal de reloj de la placa según lo mostrado en la figura 25.

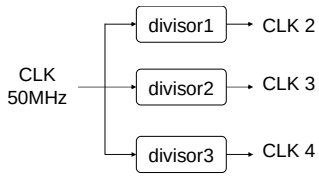


Figura 25: Relojes.

4.2.1.-El DDS (Direct digital synthesizers)

Para generar cada uno de los armónicos representados en la Figura 10 se ha utilizado un DDS (Direct digital synthesizers). Mediante el DDS conseguiremos una onda que puede ser un seno o un coseno.
El sistema se basa en leer valores de una señal senoidal almacenada en una memoria. La distancia entre las muestras leídas determinará la frecuencia de la onda senoidal generada.
El diagrama del DDS es el siguiente:

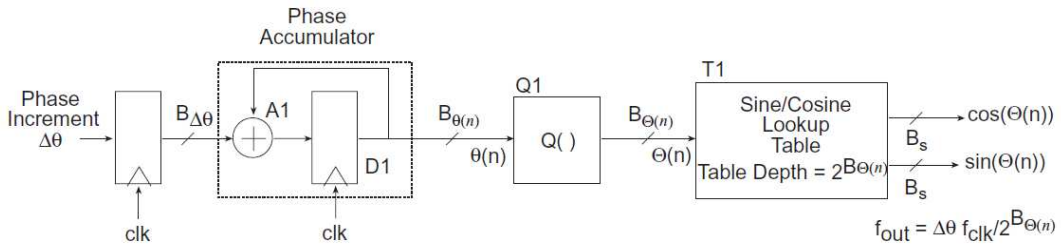


Figura 22: DDS.

Tal y como explicaremos más adelante, el valor del incremento de fase ΔΘ (phase increment) dependerá de la frecuencia a generar. Este incremento de fase servirá de entrada a un circuito acumulador, en el que acumulamos este incremento para determinar qué puntos de la señal serán leídos de la memoria.
La salida del acumulador se trunca en el siguiente bloque, dado que para direccionar la memoria no son necesarios tantos bits como para hacer el cálculo preciso de la suma de los incrementos de fase. Este valor ya truncado indicará qué posición de la onda debe leerse, y, la lectura continua de estas posiciones permitirá generar la señal a la frecuencia deseada.

A continuación se muestra cómo determinar los tamaños (nº de bits) de los diferentes datos. El cálculo de los parámetros correspondientes a los distintos instrumentos se muestran en las siguientes tablas. En ellas se ha representado:

- Nota: nombre de la nota a generar (nótese que cada instrumento opera en una escala distinta)
- F: frecuencia fundamental correspondiente a dicha nota
- P: incremento de fase
- Trunc fase a Mbits: valor del incremento de fase cuando se codifica con M bits
- P: Resolución del parámetro incremento de fase codificado con M bits
- Δf: incremento de frecuencia conseguido cuando se cuantifica el incremento de fase con M bits

- Fgenerada: Valor de la frecuencia generada cuando se utiliza precisión finita
- Error de frecuencia: diferencia entre el valor de la frecuencia deseada y el valor obtenido tras la implementación

nota	f	P=inc fase	trunc fase a M bit	p/2^m	Af (incr. Frec)	FGENERADA	ERROR_FREC	P*4
C_0	16,35	89,2928	89	0,0003	0,183105468750	16,29638672	0,053613281	356
$C^{\#}_0/D^{b}_0$	17,32	94,59029333	94	0,0004	0,183105468750	17,21191406	0,108085938	376
C_4	261,63	1428,84864	1428	0,0054	0,183105468750	261,4746094	0,155390625	5712
$C^{\#}_4/D^{b}_4$	277,18	1513,772373	1513	0,0058	0,183105468750	277,0385742	0,141425781	6052
D_4	293,66	1603,775147	1603	0,0061	0,183105468750	293,5180664	0,141933594	6412
$D^{\#}_4/E^{b}_4$	311,13	1699,18464	1699	0,0065	0,183105468750	311,0961914	0,033808594	6796
E_4	329,63	1800,219307	1800	0,0069	0,183105468750	329,5898438	0,04015625	7200
F_4	349,23	1907,26144	1907	0,0073	0,183105468750	349,1821289	0,047871094	7628
$F^{\#}_4/G^{b}_4$	369,99	2020,63872	2020	0,0077	0,183105468750	369,8730469	0,116953125	8080
G_4	392	2140,842667	2140	0,0082	0,183105468750	391,8457031	0,154296875	8560
$G^{\#}_4/A^{b}_4$	415,3	2268,091733	2268	0,0087	0,183105468750	415,2832031	0,016796875	9072
A_4	440	2402,986667	2402	0,0092	0,183105468750	439,8193359	0,180664063	9608
$A^{\#}_4/B^{b}_4$	466,16	2545,855147	2545	0,0097	0,183105468750	466,003418	0,156582031	10180
B_4	493,88	2697,243307	2697	0,0103	0,183105468750	493,8354492	0,044550781	10788
C_5	523,25	2857,642667	2857	0,0109	0,183105468750	523,1323242	0,117675781	11428
$C^{\#}_5/D^{b}_5$	554,37	3027,59936	3027	0,0115	0,183105468750	554,2602539	0,109746094	12108
D_5	587,33	3207,604907	3207	0,0122	0,183105468750	587,2192383	0,110761719	12828
D_8	4698,64	25660,83925	25660	0,0979	0,183105468750	4698,486328	0,153671875	102640
$D^{\#}_8/E^{b}_8$	4978,03	27186,68117	27186	0,1037	0,183105468750	4977,905273	0,124726562	108744

Figura 26: Tabla oboe

nota	f	P=inc fase	trunc fase a M bit	p/2^m	Af (incr. Frec)	FGENERADA	ERROR_FREC	P*4
C_0	16,35	89,2928	89	0,0003	0,183105468750	16,29638672	0,053613281	356
$C^{\#}_0/D^{b}_0$	17,32	94,59029333	94	0,0004	0,183105468750	17,21191406	0,108085938	376
C_5	523,25	2857,642667	2857	0,0109	0,183105468750	523,1323242	0,117675781	11428
$C^{\#}_5/D^{b}_5$	554,37	3027,59936	3027	0,0115	0,183105468750	554,2602539	0,109746094	12108
D_5	587,33	3207,604907	3207	0,0122	0,183105468750	587,2192383	0,110761719	12828
$D^{\#}_5/E^{b}_5$	622,25	3398,314667	3398	0,0130	0,183105468750	622,1923828	0,057617188	13592
E_5	659,26	3600,438613	3600	0,0137	0,183105468750	659,1796875	0,0803125	14400
F_5	698,46	3814,52288	3814	0,0145	0,183105468750	698,3642578	0,095742188	15256
$F^{\#}_5/G^{b}_5$	739,99	4041,332053	4041	0,0154	0,183105468750	739,9291992	0,060800781	16164
G_5	783,99	4281,63072	4281	0,0163	0,183105468750	783,8745117	0,115488281	17124
$G^{\#}_5/A^{b}_5$	830,61	4536,23808	4536	0,0173	0,183105468750	830,5664063	0,04359375	18144
A_5	880	4805,973333	4805	0,0183	0,183105468750	879,8217773	0,178222656	19220
$A^{\#}_5/B^{b}_5$	932,328	5091,753984	5091	0,0194	0,183105468750	932,1899414	0,138058594	20364
B_5	987,77	5394,541227	5394	0,0206	0,183105468750	987,6708984	0,099101562	21576
C_6	1046,5	5715,285333	5715	0,0218	0,183105468750	1046,447754	0,052246094	22860
$C^{\#}_6/D^{b}_6$	1108,73	6055,144107	6055	0,0231	0,183105468750	1108,703613	0,026386719	24220
D_6	1174,66	6415,209813	6415	0,0245	0,183105468750	1174,621582	0,038417969	25660
D_8	4698,64	25660,83925	25660	0,0979	0,183105468750	4698,486328	0,153671875	102640
$D^{\#}_8/E^{b}_8$	4978,03	27186,68117	27186	0,1037	0,183105468750	4977,905273	0,124726562	108744

Figura 27: Tabla violín

nota	f	P=inc fase	P(trunc fase a M bits)	p/2^m	Af (incr. Frec)	FGENERADA	ERROR_FREC	P*4
C_0	16,35	89,2928	89	0,0003	0,183105468750	16,29638672	0,053613281	356
$C^{\#}_0/D^{b}_0$	17,32	94,59029333	94	0,0004	0,183105468750	17,21191406	0,108085938	376
C_3	130,81	714,3970133	714	0,0027	0,183105468750	130,7373047	0,072695313	2856
$C^{\#}_3/D^{b}_3$	138,59	756,8861867	756	0,0029	0,183105468750	138,4277344	0,162265625	3024
D_3	146,83	801,8875733	801	0,0031	0,183105468750	146,6674805	0,162519531	3204
$D^{\#}_3/E^{b}_3$	155,56	849,5650133	849	0,0032	0,183105468750	155,456543	0,103457031	3396
E_3	164,81	900,0823467	900	0,0034	0,183105468750	164,7949219	0,015078125	3600
F_3	174,61	953,6034133	953	0,0036	0,183105468750	174,4995117	0,110488281	3812
$F^{\#}_3/G^{b}_3$	184,99	1010,292053	1010	0,0039	0,183105468750	184,9365234	0,053476563	4040
G_3	195,99	1070,36672	1070	0,0041	0,183105468750	195,9228516	0,067148438	4280
$G^{\#}_3/A^{b}_3$	207,65	1134,045867	1134	0,0043	0,183105468750	207,6416016	0,008398438	4536
A_3	220	1201,493333	1201	0,0046	0,183105468750	219,909668	0,090332031	4804
$A^{\#}_3/B^{b}_3$	233,08	1272,927573	1272	0,0049	0,183105468750	232,9101563	0,16984375	5088
B_3	246,94	1348,621653	1348	0,0051	0,183105468750	246,8261719	0,113828125	5392
C_4	261,63	1428,84864	1428	0,0054	0,183105468750	261,4746094	0,155390625	5712
$C^{\#}_4/D^{b}_4$	277,18	1513,772373	1513	0,0058	0,183105468750	277,0385742	0,141425781	6052
D_4	293,66	1603,775147	1603	0,0061	0,183105468750	293,5180664	0,141933594	6412
D_8	4698,64	25660,83925	25660	0,0979	0,183105468750	4698,486328	0,153671875	102640
$D^{\#}_8/E^{b}_8$	4978,03	27186,68117	27186	0,1037	0,183105468750	4977,905273	0,124726562	108744

Figura 28: Tabla tuba

La fórmula que se ha usado para calcular el incremento de fase (P=inc fase) es la siguiente:

$P = \frac{f \cdot 2^m}{f_{clk}}$ Donde "m" es el número de bits y " f_{clk} " (frecuencia del reloj) son 48000Hz.

Se han probado diferentes valores y se ha decidido escoger una "m" de 18 bits, ya que con este número el error es bastante pequeño (el máximo error es de 0.18 Hz). En nuestro programa los datos que hemos de almacenar son los de la columna "P*4" (esta memoria se encuentra en el bloque incrementos).

Para calcular el desfase inicial usaremos la siguiente fórmula:

$$fasef = \frac{fase \cdot 2^L}{2 \cdot \pi}$$

Donde fase será la fase positiva extraída de la base de datos (en caso de ser negativa le sumaremos $2 \cdot \pi$) y "L" será el número de bits con que se cuantifica dicho valor. El valor entero de fasef es el que guardaremos en memoria, esta memoria se encuentra en el bloque fase.

Para calcular las amplitudes haremos lo siguiente:

$$ampf = \frac{amp}{2^{14-W}}$$

Donde amp será la amplitud extraída de la base de datos y "W" será el número de bits con que se cuantifica.. El valor entero de ampf es el que guardaremos en memoria. Esta memoria se encuentra en el bloque amplitud.

Para calcular los valores de W (número de bits de la amplitud) y L (número de bits del desfase inicial) partimos de las siguientes ecuaciones:

$$SFDR \approx 6.02L - 3.92dB$$
$$SNR_Q \approx 6.02W + 1.76dB$$
$$L = W + 1$$

Interesa que $SFDR$ y SNR_Q sean lo mas similares posible, escogemos W=8 y L=9.

Estos parámetros implican que la señal senoidal se almacenará en una memoria de 512 posiciones (2^9), y que los datos allí almacenados tendrán un tamaño de 8 bits.

Tras el cálculo de todos los parámetros necesarios se va a proceder a explicar los diferentes bloques que constituyen el sistema completo.

Para evitar problemas durante las conversiones de los datos se ha decidido trabajar sin signo.

4.2.2.- Divisores

4.2.2.1.-Divisor1

En este módulo obtendremos un reloj a aproximadamente 48000Hz a partir del reloj del dispositivo, el código para ello es el siguiente:

```
if RST = '1' then Qint <= "00000000000"; sint <= '0';
elsif (CLK' event) and (CLK = '1') then
        if Qint = 1041 then
        Qint <= "00000000000"; sint <= '1';
        else Qint <= Qint+1; sint <= '0';
        end if;
end if;
```

4.2.2.2.-Divisor2

Mediante este divisor obtenemos un reloj a 25MHz.

```
if RST = '1' then Qint <= "00"; sint <= '0';
elsif (CLK' event) and (CLK = '1') then
        if Qint = 1 then
        Qint <= "00"; sint <= '1';
        else Qint <= Qint+1; sint <= '0';
        end if;
end if;
```

A continuación podemos ver como obtenemos un reloj a la mitad de la frecuencia:

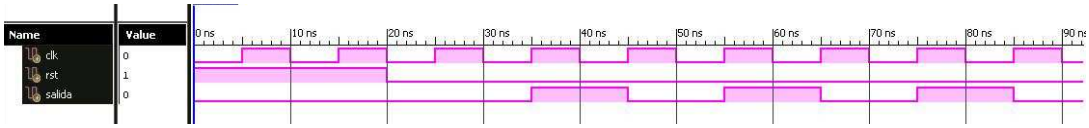


Figura 29: Simulación reloj

4.2.2.3.-Divisor3

En este caso buscamos un reloj a aproximadamente 1.008MHz (21*48kHz) que se utilizará posteriormente para implementar el circuito con un número reducido de memorias. El siguiente código genera uno a 1MHz.

```
if RST = '1' then Qint <= "00000000000"; sint <= '0';
elsif (CLK' event) and (CLK = '1') then
        if Qint = 49 then
        Qint <= "00000000000"; sint <= '1';
        else Qint <= Qint+1; sint <= '0';
        end if;
end if;
```

4.2.3.-G00_audio_interface

Para poder escuchar el sonido creado por nuestro programa debemos utilizar el *Internet Audio CODECWM8731*, el diagrama de este es el siguiente:

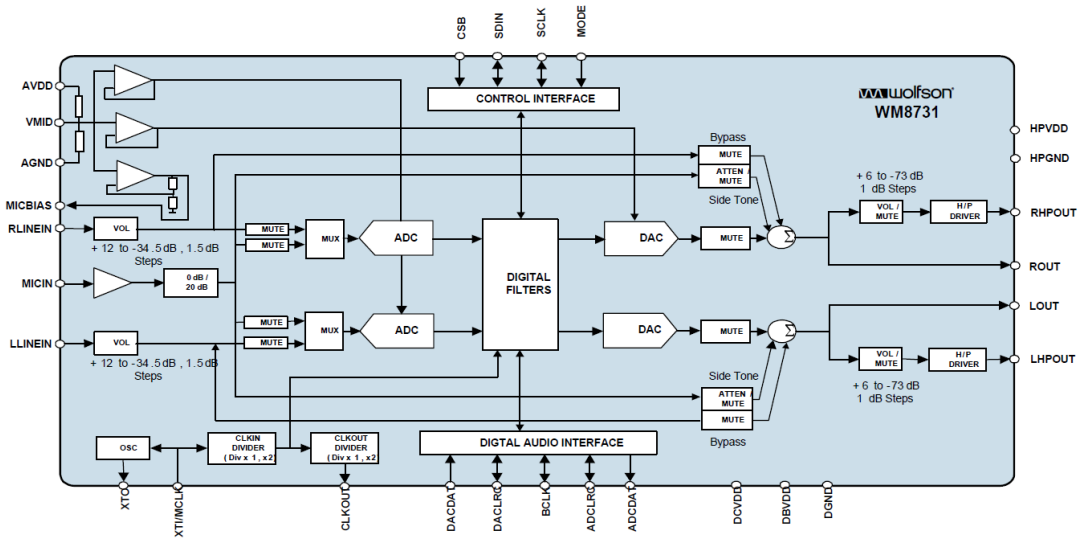


Figura 30: g00_audio_interface

En la parte superior e inferior encontramos las señales de datos que permiten su funcionamiento, en la parte izquierda las entradas provenientes del micrófono y por último a la izquierda las salidas del CODEC.
Las señales que encontramos en nuestro programa son:

- LDATA, RDATA: Son dos entradas de datos en paralelo, en el diagrama se corresponde con DACDAT, que es una línea de datos formada por la estos dos canales multiplexados.
- CLK: Es la entrada de la señal de reloj a 24 MHz (BCLK en el diagrama).
- RST: Es la señal de reset. (El reset depende de la tensión que tenemos en DCVDD).
- INIT: Con esta señal ponemos la condición necesaria para que se inicie.
- W_EN: Dependiendo de si esta activada el CODEC funcionara o no.
- Pulse_48KHz: Salida síncrona con un pulso a 48 KHz. En nuestro caso no usaremos esta salida.
- AUD_MCLK: Salida con la señal de reloj que tenemos en la entrada. (CLKOUT en el esquema).
- AUD_BCLK: Señal de reloj con la que son síncronos los datos (BCLK). En este caso es una señal de salida.
- AUD_DACDAT: Salida de datos analógica (ROUT y LOUT)
- AUD_DACLRCK: Esta señal indica si estamos usando el canal derecho o el izquierdo (DACLRC en el diagrama).
- I2C_SDAT: Salida para el interfaz I2C.
- I2C_SCLK: Salida de reloj para el interfaz I2C.

Dependiendo del modo en que este funcionando el sistema obtendrá los canales de salida derecho e izquierdo de modo diferente.
En nuestro caso trabaja en modo I2S. Los dos canales de audio salen multiplexadados, una vez le llega un flanco de reloj descendente espera un ciclo de reloj y envía la parte de los datos correspondientes al canal de audio derecho. Los datos del otro canal de audio los enviara una vez superada la mitad de la frecuencia de muestreo también esperando un ciclo de reloj. La señal DACLRC es la que controla el canal multiplexado. Este proceso lo podemos ver en la siguiente figura:

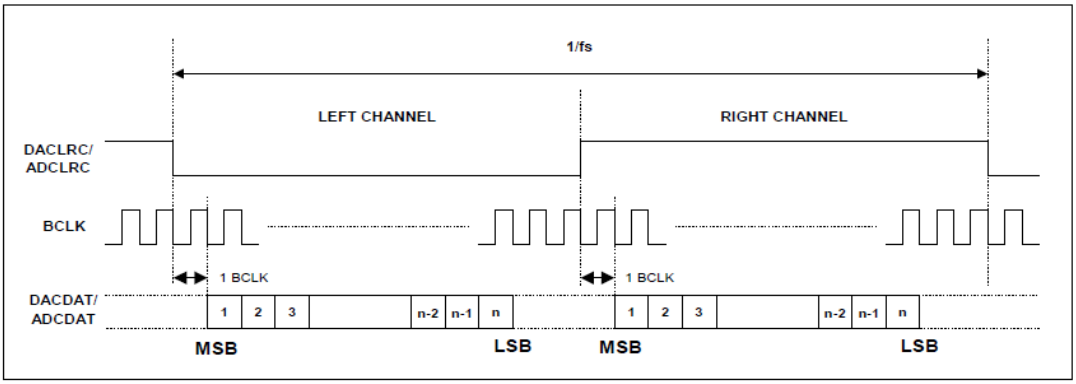


Figura 31: Modo I2S

A continuación podemos ver las entradas y salidas de este bloque

```
ENTITY g00_audio_interface IS
        PORT(  LDATA, RDATA:IN std_logic_vector(23 downto 0);-- parallel external data inputs
                clk, rst, INIT, W_EN : IN std_logic;                    -- clk should be 24MHz
                pulse_48KHz :  OUT std_logic;                           -- sample sync pulse
                AUD_MCLK : OUT std_logic;                               -- codec master clock input
                AUD_BCLK : OUT std_logic;                               -- digital audio bit clock
                AUD_DACDAT : OUT std_logic;                             -- DAC data lines
                AUD_DACLRCK : OUT std_logic;                            -- DAC data left/right select
                I2C_SDAT :  OUT std_logic;                              -- serial interface data line
                I2C_SCLK : OUT std_logic                                -- serial interface clock
        );
END g00_audio_interface;
```

4.2.4.-LED

Este módulo lo que hacía es que al tocar una nota se iluminen una serie de bits, si tocamos la nota “0” no se ilumina ninguna, si tocamos la nota “1” se ilumina uno, si tocamos la “2” se encienden dos y asi sucesivamente. Si tocamos una fuera de rango encontraremos los LEDs encendidos de manera alternada. Este módulo no aparece en el código final, ya que se a sustituido por el de LED2. A continuación podemos ver como conseguiamos esto:

```
with entrada select
        salida <= "0000000000000000" when "00000",  -- 0
                  "0000000000000001" when "00001",  -- 1
                  "0000000000000011" when "00010",  -- 2
                  "0000000000000111" when "00011",  -- 3
                  "0000000000001111" when "00100",  -- 4
                  "0000000000011111" when "00101",  -- 5
                  "0000000000111111" when "00110",  -- 6
                  "0000000001111111" when "00111",  -- 7
                  "0000000011111111" when "01000",  -- 8
                  "0000000111111111" when "01001",  -- 9
                  "0000001111111111" when "01010",  -- 10
                  "0000011111111111" when "01011",  -- 11
                  "0000111111111111" when "01100",  -- 12
                  "0001111111111111" when "01101",  -- 13
                  "0011111111111111" when "01110",  -- 14
                  "0111111111111111" when "01111",  -- 15
                  "1111111111111111" when "10000",  -- 16
                  "0111111111111111" when "10001",  -- 17
                  "1111111111111111" when "10010",  -- 18
                  "1010101010101010" when others;
```

4.2.5.- LED2

Este módulo nos permitirá ver por los LEDs las notas que se tocan. En este bloque se encenderá un LED por cada una de las notas que se toquen. Si se toca la nota 1 se encenderá el primer LED, si se toca la 2 el segundo y así sucesivamente. Si se tocan varias notas se iluminaran varios LEDs. Lo que hacemos es pasar las notas de la entrada a los LEDs:

```
leds <= notas  sel:
```

4.2.6.- LED3

En este bloque podemos ver las señales de control. Si esta activado el INIT se iluminara el LED correspondiente, si esta activado W_EN, se iluminara su LED. Además si no se hace un reset también estará iluminado el LED del RST, mientras que si se hace un RST se apagaran los LEDs de INIT, W_EN y RST. Esto se consigue de la siguiente forma:

```
inter <= INIT & W_EN & RST;
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                if(inter="000") then inter2 <= "101";
                elsif (inter="001") then inter2 <= "000";
                elsif (inter="010") then inter2 <= "111";
                elsif (inter="011") then inter2 <= "000";
                elsif (inter="100") then inter2 <= "001";
                elsif (inter="101") then inter2 <= "000";
                elsif (inter="110") then inter2 <= "011";
                elsif (inter="111") then inter2 <= "000";
                end if;
        end if;
end process;
LED_config <= inter2;
```

También veremos si se esta tocando desde el arpa o los interruptores (LED encendido arpa) y el número de instrumento que tocamos.

```
LED_modo <= sel;
LED_inst <= num_inst;
```

4.2.7.-Memorias

En este proyecto será necesario definir 3 memorias de sólo lectura para almacenar:

- los valores de los incrementos de fase ($\Delta\Theta$) correspondientes a la generación de las frecuencias de cada nota musical (módulo incrementos)
- las amplitudes de los armónicos correspondientes a la nota e instrumento elegido (módulo amplitud)
- los desfases a aplicar a cada armónico según el instrumento deseado y la nota elegida (módulo fase)

4.2.7.1.-Incrementos

Mediante este módulo obtendremos los incrementos que necesitamos para nuestro programa, estos incrementos corresponden con los incrementos de fase necesarios para obtener una determinada frecuencia de salida.
Los incrementos estarán almacenados de la siguiente forma:

```
---------- OBOE
type ROM is array (0 to 18) of integer;                    --ROM P(incremento de fase) * 4
constant MEM1:ROM:=(  356,376,5712,6052,6412,6796,7200,7628,8080,8560,9072,9608,10180,
                      10788,11428,12108,12828,102640,108744);
---------- VIOLIN
type ROM2 is array (0 to 18) of integer;
constant MEM2: ROM2:=(356,376,11428,12108,12828,13592,14400,15256,16164,17124,
                       18144,19220,20364,21576,22860,24220,25660,102640,108744);
```

```
---------- TUBA
type ROM3 is array (0 to 18) of integer;
constant MEM3:ROM3:=(356,376,2856,3024,3204,3396,3600,3812,4040,4280,4536,4804,5088,
                     5392,5712, 6052,6412,102640,108744);
```

Se usaran de la siguiente forma:

```
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                if
                ----------------------------------------------OBOE-----------------------------------------------
                        (num_inst = "01") then
                                INCe <= MEM1(CONV_INTEGER(num_nota));
                                INC <= conv_std_logic_vector(INCe,18);
                ----------------------------------------------VIOLIN----------------------------------------------
                elsif
                        (num_inst = "10") then
                                INCe <= MEM2(CONV_INTEGER(num_nota));
                                INC <= conv_std_logic_vector(INCe,18);
                ----------------------------------------------TUBA------------------------------------------------
                elsif
                        (num_inst = "11") then
                                INCe <= MEM3(CONV_INTEGER(num_nota));
                                INC <= conv_std_logic_vector(INCe,18);
                end if;
        end if;
end Process;
```

Para generar las salidas haremos lo siguiente

```
Sal <= INC;
Sale <= INCe;
```

4.2.7.2-Amplitud

En este módulo lo que hacemos es a partir del instrumento y de la nota que se toca sacamos la amplitud de la frecuencia fundamental y de cada uno de los armónicos.
La amplitud esta almacenada de la siguiente forma:

```
type ROM_a_1 is array (0 to 18) of integer;              -- ROM amplitud frecuencia fundamental inst1
constant MEM_a_1: ROM_a_1:=(0,0,155,138,155,150,154,152,104,116,136,110,152,100,138,
                            129,119,0,0);
```

4.2.7.3.-Fase

En este módulo lo que hacemos es a partir del instrumento y de la nota que se toca sacamos las fases de cada uno de los armónicos (las fases almacenadas son las que hemos calculado anteriormente).
A continuación podemos ver un ejemplo de cómo se almacenan las fases en memoria

```
type ROMf1_1 is array (0 to 18) of integer;
MEMf1_1: ROMf1_1:=(0,0,357,496,13,507,356,44,185,384,268,153,60,191,25,2,492,0,0);
```

A continuación veremos como sacaremos un ejemplo de cómo sacaremos estos valores para un instrumento:

```
Process (CLK)
begin
if (CLK'event and CLK='1') then
        if (num_inst = "01") then
                fase1e <= MEMf1_1(CONV_INTEGER(num_nota));      --guardar en fase1e el
                                                                valor que tenemos en
                                                                la memoria
                fase1 <= conv_std_logic_vector(fase1e,9);       -- convertir el valor
                                                                que tenemos en la
                                                                memoria
                ...
        elsif (num_inst = "10") then
                ...
        ...
        end if;
end if;
end Process;
```

Después estos valores los asignaremos fuera del *process* de la siguiente manera:

```
f1 <= fase1;
```

Con los datos obtenidos de la memoria "amplitud se procede de forma simular.

4.2.8.-Arm

En este módulo a partir de los incrementos obtenidos anteriormente, calcularemos cuanto serán para cada uno de los armónicos. Para ello lo que hacemos es multiplicar el incremento por el número correspondiente dependiendo del armónico que sea (para obtener el primer armónico multiplicamos por 2, para el segundo por 3 y así sucesivamente). Para almacenar estos armónicos necesitaremos 20 bits, ya que si vemos cual es el mayor incremento y lo multiplicamos por 20 obtenemos un valor que necesita 20 bits para almacenarse. Además este bloque también tendrá un *reset.*

Para calcular los armónicos dentro de un *process* haremos lo siguiente:

```
Bf_1 <= "00"& INC;                          -- frecuencia fundamental
B1_1 <= conv_std_logic_vector (INCe*2,20);  -- primer armonico multiplica el incremento*2
B2_1 <= conv_std_logic_vector (INCe*3,20);  -- y lo cuantificamos en 20 bits.
B3_1 <= conv_std_logic_vector (INCe*4,20);
B4_1 <= conv_std_logic_vector (INCe*5,20);
B5_1 <= conv_std_logic_vector (INCe*6,20);
B6_1 <= conv_std_logic_vector (INCe*7,20);
B7_1 <= conv_std_logic_vector (INCe*8,20);
B8_1 <= conv_std_logic_vector (INCe*9,20);
B9_1 <= conv_std_logic_vector (INCe*10,20);
B10_1 <= conv_std_logic_vector (INCe*11,20);
B11_1 <= conv_std_logic_vector (INCe*12,20);
B12_1 <= conv_std_logic_vector (INCe*13,20);
B13_1 <= conv_std_logic_vector (INCe*14,20);
B14_1 <= conv_std_logic_vector (INCe*15,20);
B15_1 <= conv_std_logic_vector (INCe*16,20);
B16_1 <= conv_std_logic_vector (INCe*17,20);
B17_1 <= conv_std_logic_vector (INCe*18,20);
B18_1 <= conv_std_logic_vector (INCe*19,20);
B19_1 <= conv_std_logic_vector (INCe*20,20);
```

4.2.9.-Acum

En este módulo forma parte de la implementación del DDS. Este bloque permite acumular los incrementos de fase (ΔΘ) para determinar la posición de la onda senoidal que debe leerse de la memoria. Lo que hacemos es sumar repetidamente, cada incremento con el valor acumulado (Figura del DDS).
Este bloque está dentro de un proceso regido por una señal de reloj CLK???, lo que permite asegurar que la salida de la suma está registrada.

```
Bf_2 <= Bf_2+Bf_1;
B1_2 <= B1_2+B1_1;
B2_2 <= B2_2+B2_1;
B3_2 <= B3_2+B3_1;
B4_2 <= B4_2+B4_1;
B5_2 <= B5_2+B5_1;
B6_2 <= B6_2+B6_1;
B7_2 <= B7_2+B7_1;
B8_2 <= B8_2+B8_1;
B9_2 <= B9_2+B9_1;
B10_2 <= B10_2+B10_1;
B11_2 <= B11_2+B11_1;
B12_2 <= B12_2+B12_1;
B13_2 <= B13_2+B13_1;
B14_2 <= B14_2+B14_1;
B15_2 <= B15_2+B15_1;
B16_2 <= B16_2+B16_1;
B17_2 <= B17_2+B17_1;
B18_2 <= B18_2+B18_1;
B19_2 <= B19_2+B19_1;
```

4.2.10.-Sumfas

Tal y como se describe en el capítulo 3, para generar la forma de onda correspondiente a cada nota y cada instrumento deben tenerse en cuenta los valores de los desfases iniciales de los mismos (véase capítulo 3). Por ello, este módulo se encarga de sumar al valor que llega del módulo acumulador anterior, el desfase inicial correspondiente a dicho armónico. La salida de este bloque es la que servirá para direccionar la memoria que almacena la onda senoidal.

```
Bf_3 <= Bf_2(19 downto 11);
B1_3 <= B1_2(19 downto 11) +  fase1;
B2_3 <= B2_2(19 downto 11) +  fase2;
B3_3 <= B3_2(19 downto 11) +  fase3;
B4_3 <= B4_2(19 downto 11) +  fase4;
B5_3 <= B5_2(19 downto 11) +  fase5;
B6_3 <= B6_2(19 downto 11) +  fase6;
B7_3 <= B7_2(19 downto 11) +  fase7;
B8_3 <= B8_2(19 downto 11) +  fase8;
B9_3 <= B9_2(19 downto 11) +  fase9;
B10_3 <= B10_2(19 downto 11) +  fase10;
B11_3 <= B11_2(19 downto 11) +  fase11;
B12_3 <= B12_2(19 downto 11) +  fase12;
B13_3 <= B13_2(19 downto 11) +  fase13;
B14_3 <= B14_2(19 downto 11) +  fase14;
B15_3 <= B15_2(19 downto 11) +  fase15;
B16_3 <= B16_2(19 downto 11) +  fase16;
B17_3 <= B17_2(19 downto 11) +  fase17;
B18_3 <= B18_2(19 downto 11) +  fase18;
B19_3 <= B19_2(19 downto 11) +  fase19;
```

4.2.11.-Memoria sen

Hasta ahora se han descrito los módulos que servirán para diseñar un sintetizador con una implementación totalmente paralela. Esto quiere decir, que la implementación requiere tantas memorias como armónicos a generar. En nuestro caso, necesitaríamos 20 memorias con los valores del sen, ya que es necesario sacar los datos de los armónicos y la frecuencia fundamental al mismo tiempo.

Dada la baja frecuencia de reloj a la que opera el circuito, es posible introducir una primera mejora en el diseño original. Esta mejora consiste disminuir los recursos HW necesarios para implementar el circuito, concretamente, en utilizar una única memoria para generar todos los armónicos de una misma nota. Para ello, se genera una señal de reloj más rápida (21 veces más rápida, 1MHz), de modo que en cada periodo de este reloj operamos con un dato diferente. Se ha incluido un ciclo adicional en el que se mantengan constantes para poder registrar todos los valores en ese momento, estos valores los registraremos con el reloj a 48KHz. Con esta modificación, al final en un ciclo de 48KHz tendremos todos calculados todos estos valores con una única memoria.

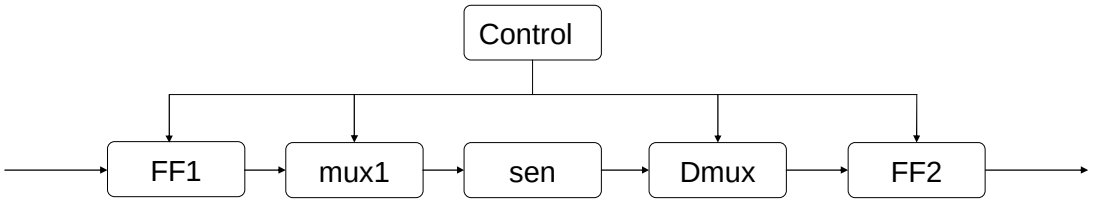


Figura 32: Memoria sen

4.2.11.1.-Control

Mediante este bloque obtendremos unas señales de control que usaremos para controlar los módulos que tendremos posteriormente, básicamente es un contador hasta 21.

```
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                if (RST='1') then
                        Qint <= "00000";
                else
                        if Qint = 20 then Qint <= "00000";
                        else Qint <= Qint+1;                    -- Qint señales control MUX
                        end if;
                end if;
        end if;
end process;
cont <= Qint;
```

4.2.11.2.-FF1

Mediante este módulo registramos la señal a 48000Hz

```
Process (CLK)
        begin
                if (CLK'event and CLK='1') then
                        Ff <= Bf_3;
                        F1 <= B1_3;
                        F2 <= B2_3;
                        F3 <= B3_3;
                        F4 <= B4_3;
                        F5 <= B5_3;
                        F6 <= B6_3;
                        F7 <= B7_3;
                        F8 <= B8  3;
```

```
                        F9 <= B9_3;
                        F10 <= B10_3;
                        F11 <= B11_3;
                        F12 <= B12_3;
                        F13 <= B13_3;
                        F14 <= B14_3;
                        F15 <= B15_3;
                        F16 <= B16_3;
                        F17 <= B17_3;
                        F18 <= B18_3;
                        F19 <= B19_3;
                end if;
end process;
Bf <= Ff;
B1 <= F1;
B2 <= F2;
B3 <= F3;
B4 <= F4;
B5 <= F5;
B6 <= F6;
B7 <= F7;
B8 <= F8;
B9 <= F9;
B10 <= F10;
B11 <= F11;
B12 <= F12;
B13 <= F13;
B14 <= F14;
B15 <= F15;
B16 <= F16;
B17 <= F17;
B18 <= F18;
B19 <= F19;
```

4.2.11.3.-Mux1

Utilizando la señal de control creada anteriormente multiplexaremos las señales del registro anterior. Este módulo funciona 1.008MHz.

```
Process (CLK)
        begin
                if (CLK'event and CLK='1') then
                        if (Qint="00000") then dir_mem <= Ff;
                        elsif (Qint="00001") then dir_mem <= F1;
                        elsif (Qint="00010") then dir_mem <= F2;
                        elsif (Qint="00011") then dir_mem <= F3;
                        elsif (Qint="00100") then dir_mem <= F4;
                        elsif (Qint="00101") then dir_mem <= F5;
                        elsif (Qint="00110") then dir_mem <= F6;
                        elsif (Qint="00111") then dir_mem <= F7;
                        elsif (Qint="01000") then dir_mem <= F8;
                        elsif (Qint="01001") then dir_mem <= F9;
                        elsif (Qint="01010") then dir_mem <= F10;
                        elsif (Qint="01011") then dir_mem <= F11;
                        elsif (Qint="01100") then dir_mem <= F12;
                        elsif (Qint="01101") then dir_mem <= F13;
                        elsif (Qint="01110") then dir_mem <= F14;
                        elsif (Qint="01111") then dir_mem <= F15;
                        elsif (Qint="10000") then dir_mem <= F16;
                        elsif (Qint="10001") then dir_mem <= F17;
```

```
                        elsif (Qint="10010") then dir_mem <= F18;
                        elsif (Qint="10011") then dir_mem <= F19;
                        elsif (Qint="10100") then dir_mem <= F19;
                        end if;
                end if;
end process;
sal <= dir_mem;
```

4.2.11.4.-Sen

Tenemos almacenados 512 valores de una onda sinusoidal positiva en una memoria, nos llegará una dirección de memoria y hemos de sacar el valor que se encuentra en esa dirección (la fase nos indica en qué posición de la onda se encuentra).
La onda esta almacenada de la siguiente forma

```
        type ROM_sen is array (0 to 511) of integer;
        constant MEM_sen:ROM_sen:=(  128,129,131,132,134,135,137,139,140,142,143,145,146,
                                     148,149,151,153,154,156,157,159,160,162,163,165,166,
                                     168,169,171,172,174,175,177,178,179,181,182,184,185,
                                     187,188,189,191,192,193,195,196,197,199,200,201,203,
                                     204,205,206,208,209,210,211,212,214,215,216,217,218,
                                     219,220,221,222,224,225,226,227,228,229,230,230,231,
                                     232,233,234,235,236,237,237,238,239,240,241,241,242,
                                     243,243,244,245,245,246,246,247,248,248,249,249,250,
                                     250,251,251,251,252,252,252,253,253,253,254,254,254,
                                     254,255,255,255,255,255,255,255,255,255,255,255,255,
                                     255,255,255,255,255,255,255,255,254,254,254,254,254,
                                     253,253,253,252,252,252,251,251,250,250,249,249,248,
                                     248, 247, 247,246,246,245,244,244,243,242,242,241,
                                     240,239,239, 238,237,236,235,234,233,232,231,230,229,
                                     228,227,226,225, 224,223,222,221,220,219,218,216,215,
                                     214,213, 212,211,209,208,207,206,205,203,202,201,199,
                                     198,197,195,194,193,191,190,189,187,186,184,183,182,
                                     180,179,177,176,174,173,171,170,168,167,165,164,162,
                                     161,159, 158,156,155, 153,152,150,149,147,146,144,
                                     142,141,139,138,136,135,133,131,130,128,127,125,124,
                                     122,120,119,117,116,114,113,111,109,108,106,105, 103,
                                     102,100,99,97,96,94,93,91,90,88,87,85,84,82,81,79,78,
                                     76, 75,73,72,71,69,68,66,65,64,62,61,60,58,57,56,54,
                                     53,52,50,49,48,47,46,44,43,42,41,40,39, 37,36,35,34,33,
                                     32,31,30,29,28,27,26,25,24,23,22,21,21,20, 19,18,17,16,
                                     16,15,14,13,13,12,11,11,10,9,9,8,8,7,7,6,6,5,5,4,4,3,3,3,
                                     2,2,2,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,
                                     1,1,2,2,2,3,3,3,4,4,4, 5,5,6,6,7,7,8,9, 9,10,10,11,12,12,
                                     13,14,14,15,15,16,17,18,18,19,20,21,22,23,24,25,25,26,
                                     27,28,29,30,31, 33,34,35,36,37,38,39,40,41,43,44,45,46,
                                     47,49,50,51,52,54,55,56,58, 59,60,62,63,64,66,67,68,70,
                                     71,73,74,76,77,78,80,81,83,84,86,87,89,90,92,93,95,
                                     96,98,99,101,102,104,106,107,109, 110,112,113,115,
                                     116,118,120,121, 123,124,126,127);
```

La salida la obtendremos de la siguiente forma:

```
sal <= MEM_sen (CONV_INTEGER(dir_mem));
```

4.2.11.5.-Dmux

Demultiplexaremos la señal anterior en 20 señales (frecuencia fundamental y 19 armónicos). Trabajara a 1.008MHz a partir de la señal de control cada vez que tengamos un flanco de reloj ascendente sacaremos otra salida, menos un ciclo que se mantendrán todas las salidas sin variar.

```
Process (CLK)
        begin
                if (CLK'event and CLK='1') then
                        if (Qint="00001") then Bf_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00010") then B1_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00011") then B2_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00100") then B3_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00101") then B4_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00110") then B5_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00111") then B6_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01000") then B7_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01001") then B8_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01010") then B9_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01011") then B10_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01100") then B11_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01101") then B12_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01110") then B13_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01111") then B14_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10000") then B15_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10001") then B16_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10010") then B17_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10011") then B18_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10100") then B19_5 <= conv_std_logic_vector(sen,8);
                        end if;
                end if;
end process;
```

4.2.11.6.-FF2

Registraremos los datos anteriores a una frecuencia de 48000Hz.

```
Process (CLK)
        begin
                if (CLK'event and CLK='1') then
                        if (Qint="00000") then      -- almacena datos mientras son constantes
                                Bf_FF <= Bf_5;
                                B1_FF <= B1_5;
                                B2_FF <= B2_5;
                                B3_FF <= B3_5;
                                B4_FF <= B4_5;
                                B5_FF <= B5_5;
                                B6_FF <= B6_5;
                                B7_FF <= B7_5;
                                B8_FF <= B8_5;
                                B9_FF <= B9_5;
                                B10_FF <= B10_5;
                                B11_FF <= B11_5;
                                B12_FF <= B12_5;
                                B13_FF <= B13_5;
                                B14_FF <= B14_5;
                                B15_FF <= B15_5;
                                B16_FF <= B16_5;
```

```
                                    B17_FF <= B17_5;
                                    B18_FF <= B18_5;
                                    B19_FF <= B19_5;
                                end if;
                        end if;
end process;
Bf <= Bf_FF;
B1 <= B1_FF;
B2 <= B2_FF;
B3 <= B3_FF;
B4 <= B4_FF;
B5 <= B5_FF;
B6 <= B6_FF;
B7 <= B7_FF;
B8 <= B8_FF;
B9 <= B9_FF;
B10 <= B10_FF;
B11 <= B11_FF;
B12 <= B12_FF;
B13 <= B13_FF;
B14 <= B14_FF;
B15 <= B15_FF;
B16 <= B16_FF;
B17 <= B17_FF;
B18 <= B18_FF;
B19 <= B19_FF;
```

4.2.11.7.-Simulación memoria sen

La simulación de este proceso es la siguiente:

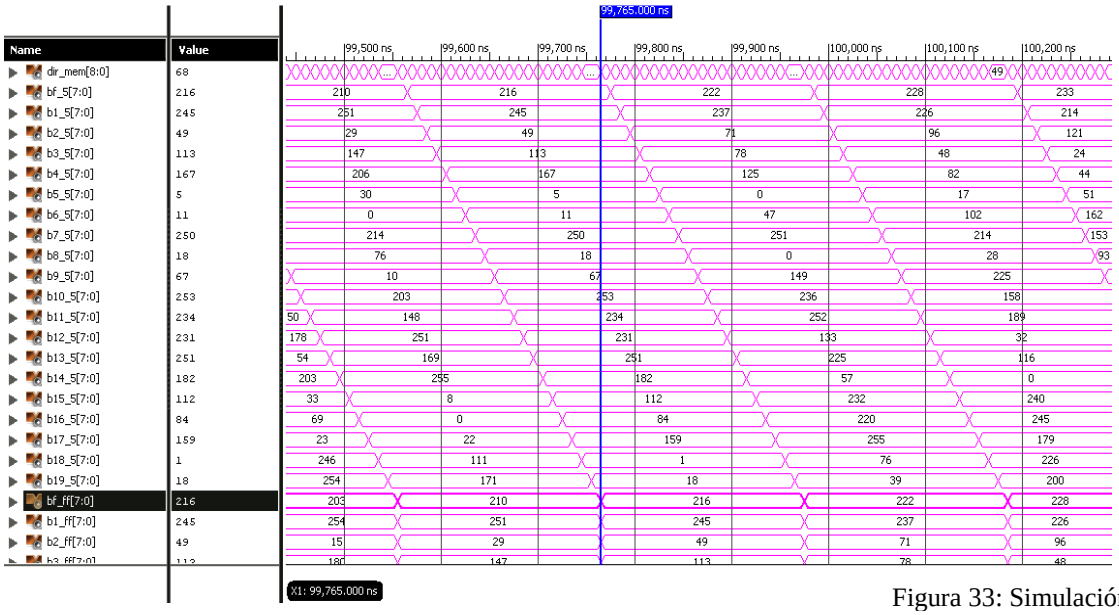


Figura 33: Simulación

4.2.12.-Mult

Llegados a este punto la frecuencia fundamental y los armónicos ya son señales sinusoidales, pero todas ellas tienen la misma amplitud, ahora lo que hacemos es multiplicar cada una por su amplitud correspondiente.

```
Bf_6 <= Bf_FF * ampfund;
B1_6 <= B1_FF * amp1;
B2_6 <= B2_FF * amp2;
B3_6 <= B3_FF * amp3;
B4_6 <= B4_FF * amp4;
B5_6 <= B5_FF * amp5;
B6_6 <= B6_FF * amp6;
B7_6 <= B7_FF * amp7;
B8_6 <= B8_FF * amp8;
B9_6 <= B9_FF * amp9;
B10_6 <= B10_FF * amp10;
B11_6 <= B11_FF * amp11;
B12_6 <= B12_FF * amp12;
B13_6 <= B13_FF * amp13;
B14_6 <= B14_FF * amp14;
B15_6 <= B15_FF * amp15;
B16_6 <= B16_FF * amp16;
B17_6 <= B17_FF * amp17;
B18_6 <= B18_FF * amp18;
B19_6 <= B19_FF * amp19;
```

4.2.13.-Offset

Hacemos que todas las señales tengan el mismo offset para después poder sumarlas correctamente. Miramos cuáles son los máximos valores de la memoria sen y de la amplitud los multiplicamos, este valor será el máximo posible desde el punto más bajo al más alto por lo tanto la mitad de este será el offset que debe tener la señal. Después calcularemos la diferencia entre este punto y el punto medio de cada una de nuestras señales. Por último sumaremos esta diferencia. El código es el siguiente:

```
max1 <= conv_std_logic_vector(255,8);   -- cambiar dependiendo amplitud maxima mem
                                           sen
max2 <= conv_std_logic_vector(78,8);    -- 156/2 cambiar dependiendo de amplitud
                                           maxima entre todos los instrumentos
cont <= (max1 * max2);                  -- valor de offset
med <= conv_std_logic_vector(128,8);    -- 255/2 aprox 128

divf <= (ampfund * med);                -- sacar punto medio señal
div1 <= (amp1 * med);
div2 <= (amp2 * med);
div3 <= (amp3 * med);
div4 <= (amp4 * med);
div5 <= (amp5 * med);
div6 <= (amp6 * med);
div7 <= (amp7 * med);
div8 <= (amp8 * med);
div9 <= (amp9 * med);
div10 <= (amp10 * med);
div11 <= (amp11 * med);
div12 <= (amp12 * med);
div13 <= (amp13 * med);
div14 <= (amp14 * med);
div15 <= (amp15 * med);
div16 <= (amp16 * med);
div17 <= (amp17 * med);
div18 <= (amp18 * med);
div19 <= (amp19 * med);
```

```
Bf_7 <= Bf_6 + cont - divf;                -- sumar diferencia
B1_7 <= B1_6 + cont - div1;
B2_7 <= B2_6 + cont - div2;
B3_7 <= B3_6 + cont - div3;
B4_7 <= B4_6 + cont - div4;
B5_7 <= B5_6 + cont - div5;
B6_7 <= B6_6 + cont - div6;
B7_7 <= B7_6 + cont - div7;
B8_7 <= B8_6 + cont - div8;
B9_7 <= B9_6 + cont - div9;
B10_7 <= B10_6 + cont - div10;
B11_7 <= B11_6 + cont - div11;
B12_7 <= B12_6 + cont - div12;
B13_7 <= B13_6 + cont - div13;
B14_7 <= B14_6 + cont - div14;
B15_7 <= B15_6 + cont - div15;
B16_7 <= B16_6 + cont - div16;
B17_7 <= B17_6 + cont - div17;
B18_7 <= B18_6 + cont - div18;
B19_7 <= B19_6 + cont - div19;
```

A continuación podemos ver una simulación con la frecuencia fundamental y los primeros armónicos de una nota, donde se comprueba su correcto funcionamiento.

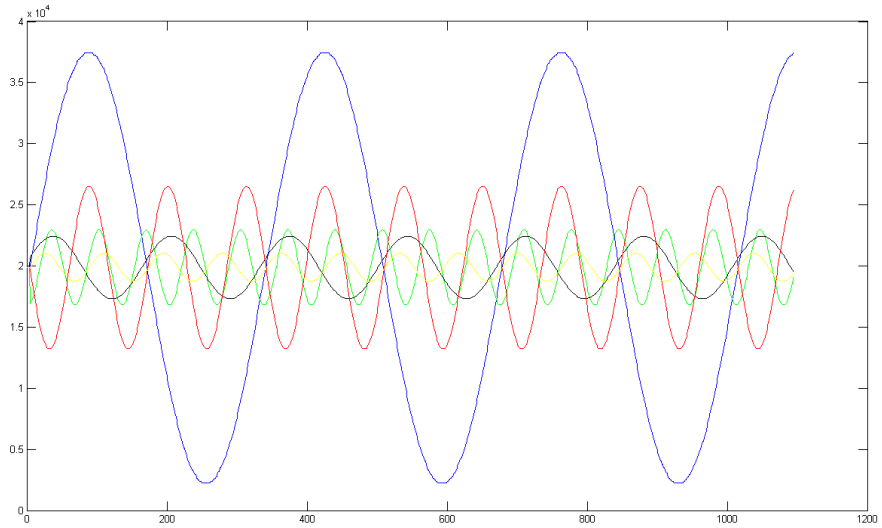


Figura 34: Armónicos sin sumar

4.2.14.-Suma

Sumamos la frecuencia fundamental con los armónicos. De este modo ya tendremos la señal que buscamos.

```
Sal <= "0000" & Bf_7 + B1_7 + B2_7 + B3_7 + B4_7 + B5_7 + B6_7 + B7_7 + B8_7 + B9_7 +
        B10_7 + B11_7 + B12_7 + B13_7 + B14_7 + B15_7 + B16_7 + B17_7 + B18_7 + B19_7;
```

4.2.15.-Ajuste

Una vez hemos obtenido la señal que buscamos cambiamos la amplitud su amplitud (aumentamos o disminuimos el volumen). Tendremos dos salidas la del canal derecho y la del izquierdo.

```
Sal1 <= "0" & Ent & "000";      -- Dependiendo de los 0 que tengamos delante y detras variara el
                                   volumen
Sal2 <= "0" & Ent & "000";      -- otro canal
```

4.2.16.-Entrada

4.2.16.1.-Invertir

Recibimos las señales del arpa y las invertimos, ya que en el arpa si se toca una nota corresponde con un “0” y si no se toca con un “1”. A nosotros nos interesa lo contrario (para que sea igual que las notas procedentes de los interruptores).
Para invertir cada nota se hace de la siguiente forma

```
if (notas(11)='0') then Salint(11) <= '1';
else Salint(11)<='0';
end if;
```

4.2.16.2.-Detección

Permite que al pulsar el pulsador detecte que LEDs del arpa se encuentran fundidos:

```
Process (CLK)
begin
	if (CLK'event and CLK='1') then
		if (en='0')then
			if (notasent(11)='1') then Salint(11) <= '0';
			else Salint(11)<='1';
			end if;
			...
		else Salint <= Salint;
		end if;
	end if;
end process;
des <= Salint;
```

4.2.16.3.-Arp

Recibimos las señales de la entrada (arpa) y se desactivan algunas de las entradas (los LEDs que no funcionen).
A continuación podemos ver un ejemplo de cómo se desactiva una nota (esto se haría dentro de un *process*).

```
If (des(11)='1') then Salint(11) <= notas(11);
else Salint(11)<='0';
end if;
```

4.2.16.4.-Selección

Elegimos si queremos las señales que proceden del arpa o las procedentes de los interruptores

```
Process (CLK)
begin
	if (CLK'event and CLK='1') then
		if (sel='1') then inter <= notas_arp;
		else inter <= notas_int;
		end if;
	end if;
end process;
```

4.2.16.5.-Calc_notas

Convertimos las señales del módulo anterior en otras válidas para nuestro programa (num_nota). Si tocamos una nota, convertiremos ésta. En caso de tocar dos o más convertiremos la nota más aguda y la más grave.
A continuación podemos ver como obtenemos el número de notas que tocamos.

```
Process (CLK)
begin
        if notasent(11)='1' then i11<="00001"; else i11<="00000";end if;
        if notasent(10)='1' then i10<="00001"; else i10<="00000";end if;
        if notasent(9)='1' then i9<="00001"; else i9<="00000";end if;
        if notasent(8)='1' then i8<="00001"; else i8<="00000";end if;
        if notasent(7)='1' then i7<="00001"; else i7<="00000";end if;
        if notasent(6)='1' then i6<="00001"; else i6<="00000";end if;
        if notasent(5)='1' then i5<="00001"; else i5<="00000";end if;
        if notasent(4)='1' then i4<="00001"; else i4<="00000";end if;
        if notasent(3)='1' then i3<="00001"; else i3<="00000";end if;
        if notasent(2)='1' then i2<="00001"; else i2<="00000";end if;
        if notasent(1)='1' then i1<="00001"; else i1<="00000";end if;
        if notasent(0)='1' then i0<="00001"; else i0<="00000";end if;
end process;
cant_notas <= i11+i10+i9+i8+i7+i6+i5+i4+i3+i2+i1+i0;
```

Ahora veremos lo que pasa si tocamos sólo una nota o ninguna

```
        if (CLK'event and CLK='1') then
                if      (cant_notas=0) then
                        nota1int <= "00000";
                        nota2int <= "00000";
                elsif (cant_notas=1) then
                        if (notasent(11)='1') then
                                nota1int <= "01110";
                                nota2int <= "00000";
                        elsif (notasent(10)='1') then
                                nota1int <= "01101";
                                nota2int <= "00000";
                        ...
                        end if;
```

En el siguiente código podemos ver una parte del código por si tocamos dos o más notas:

```
                elsif (cant_notas > 1) then
                        if (notasent(11)='1') then
                                if notasent(0)='1' then
                                        nota1int <= "01110";
                                        nota2int <= "00010";
                                elsif notasent(1)='1' then
                                                nota1int <= "01110";
                                                nota2int <= "00011";
                                elsif notasent(2)='1' then
                                                nota1int <= "01110";
                                                nota2int <= "00100";
                                elsif notasent(3)='1' then
                                                nota1int <= "01110";
                                                nota2int <= "00101";
                                elsif notasent(4)='1' then
                                                nota1int <= "01110":
```

```
                                        nota2int <= "00110";
                                elsif notasent(5)='1' then
                                        nota1int <= "01110";
                                        nota2int <= "00111";
                                elsif notasent(6)='1' then
                                        nota1int <= "01110";
                                        nota2int <= "01001";
                                elsif notasent(7)='1' then
                                        nota1int <= "01110";
                                        nota2int <= "01010";
                                elsif notasent(8)='1' then
                                        nota1int <= "01110";
                                        nota2int <= "01011";
                                elsif notasent(9)='1' then
                                        nota1int <= "01110";
                                        nota2int <= "01100";
                                elsif notasent(10)='1' then
                                        nota1int <= "01110";
                                        nota2int <= "01101";
                                end if;
                        elsif (notasent(10)='1') then
                                ...
```

4.2.16.6.-Sumfin

Sumaremos la nota 1 con la nota 2.

```
Sal1 <= Ent1 + Ent2;
Sal2 <= Ent1_2 + Ent2_2;
```

4.2.17.-Proyecto

Este módulo es el que contendrá todos los módulos anteriores.

```
U1 : divisor1 Port map (CLK, RST, CLK2);                                --divisor aprox
                                                                        48000Hz
U2 : divisor2 Port map (CLK, RST, CLK3);                                --divisor 25MHz
U3 : divisor3 Port map (CLK, RST, CLK4);                                --divisor 1MHz aprox
                                                                        1.008MHz
U4 : g00_audio_interface  PORT map(Sal1 , Sal2, CLK3, RST,INIT, W_EN,   --interfaz de audio
                          pulse_48KHz, AUD_MCLK, AUD_BCLK,
                          AUD_DACDAT, AUD_DACLRCK, I2C_SDAT,
                          I2C_SCLK );
U5 : LED2 Port map ( notas_sel,sal_LED );
U6 : fase Port map (CLK2,nota1,num_inst,f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,
                    f11,f12,f13,f14,f15,f16,f17,f18,f19);
U7 : amplitud Port map (CLK2,nota1,num_inst,af,a1,a2,a3,a4,a5,a6,a7,a8,
                        a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19);
U8 : control Port map (CLK4,RST,cont);                                  -- señales de control

U9 : incrementos Port map (CLK2,nota1,num_inst,ince,inc);
U10 : arm Port map (CLK2,nota1,num_inst,RST,ince,inc,armf,arm1,arm2,    -- Calculo armonicos
                    arm3,arm4,arm5,arm6,arm7,arm8,arm9,arm10,arm11,arm12,
                    arm13,arm14,arm15,arm16,arm17,arm18,arm19);
U11 : acum Port map (CLK2,RST,armf,arm1,arm2,arm3,arm4,arm5,arm6,arm7,-- acumulador
                     arm8,arm9,arm10,arm11,arm12,arm13,arm14,arm15,arm16,
                     arm17,arm18,arm19,Bf,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,
                     B11,B12,B13,B14,B15,B16,B17,B18,B19);
```

```
U12 : sumfas Port map (f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,          -- suma fase
                       f16,f17,f18,f19,Bf,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,
                       B11,B12,B13,B14,B15,B16,B17,B18,B19,Bf_3,B1_3,B2_3,
                       B3_3,B4_3,B5_3,B6_3,B7_3,B8_3,B9_3,B10_3,B11_3,B12_3,
                       B13_3,B14_3,B15_3,B16_3,B17_3,B18_3,B19_3);
U13 : FF1 Port map (CLK2,Bf_3,B1_3,B2_3,B3_3,B4_3,B5_3,B6_3,B7_3,B8_3,B9_3,
                    B10_3,B11_3,B12_3,B13_3,B14_3,B15_3,B16_3,B17_3,B18_3,
                    B19_3,Bf_4,B1_4,B2_4,B3_4,B4_4,B5_4,B6_4,B7_4,B8_4,B9_4,
                    B10_4,B11_4,B12_4,B13_4,B14_4,B15_4,B16_4,B17_4,B18_4,
                    B19_4);
U14 : mux1 Port map (CLK4,cont,Bf_4,B1_4,B2_4,B3_4,B4_4,B5_4,B6_4,B7_4,B8_4,
                     B9_4,B10_4,B11_4,B12_4,B13_4,B14_4,B15_4,B16_4,B17_4,
                     B18_4,B19_4,dir_mem);
U15 : sen Port map (dir_mem,sensal);
U16 : DMUX Port map (CLK4,cont,sensal,Bf_5,B1_5,B2_5,B3_5,B4_5,B5_5,B6_5,
                       B7_5,B8_5,B9_5,B10_5,B11_5,B12_5,B13_5,B14_5,B15_5,
                       B16_5,B17_5,B18_5,B19_5);
U17 : FF2 Port map (CLK4,cont,Bf_5,B1_5,B2_5,B3_5,B4_5,B5_5,B6_5,B7_5,B8_5,
                    B9_5,B10_5,B11_5,B12_5,B13_5,B14_5,B15_5,B16_5,B17_5,
                    B18_5,B19_5,Bf_FF,B1_FF,B2_FF,B3_FF,B4_FF,B5_FF,B6_FF,
                    B7_FF,B8_FF,B9_FF,B10_FF,B11_FF,B12_FF,B13_FF,B14_FF,
                    B15_FF,B16_FF,B17_FF,B18_FF,B19_FF);
U18 : mult Port map (af,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,    --mult por la amplitud
                     a16,a17,a18,a19,Bf_FF,B1_FF,B2_FF,B3_FF,B4_FF,B5_FF,
                     B6_FF,B7_FF,B8_FF,B9_FF,B10_FF,B11_FF,B12_FF,B13_FF,
                     B14_FF,B15_FF,B16_FF,B17_FF,B18_FF,B19_FF,Bf_6,B1_6,
                     B2_6,B3_6,B4_6,B5_6,B6_6,B7_6,B8_6,B9_6,B10_6,B11_6,
                     B12_6,B13_6,B14_6,B15_6,B16_6,B17_6,B18_6,B19_6);
U19 : offset Port map (af,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,
                       a15,a16,a17,a18,a19,Bf_6,B1_6,B2_6,B3_6,B4_6,B5_6,
                       B6_6,B7_6,B8_6,B9_6,B10_6,B11_6,B12_6,B13_6,B14_6,
                       B15_6,B16_6,B17_6,B18_6,B19_6,Bf_7,B1_7,B2_7,B3_7,
                       B4_7,B5_7,B6_7,B7_7,B8_7,B9_7,B10_7,B11_7,B12_7,
                       B13_7,B14_7,B15_7,B16_7,B17_7,B18_7,B19_7);
U20 : suma Port map (Bf_7,B1_7,B2_7,B3_7,B4_7,B5_7,B6_7,B7_7,B8_7,B9_7,B10_7,
                     B11_7,B12_7,B13_7,B14_7,B15_7,B16_7,B17_7,B18_7,B19_7,
                       Salsum);
U21 : ajuste Port map (Salsum,Sintaud,Sintaud2);                    -- ajuste del volumen
U42:  invertir Port map ( CLK2,notas,notas_inv);
U40 : deteccion Port map (CLK2,notas_inv,en,des);                   -- detectar fundidos
U22 : arp Port map (CLK2,notas_inv,des,notasent);                   -- desactivar senyales
U41 : seleccion Port map (sel,clk2,notasent,notas_int,notas_sel);   -- seleccionar notas
                                                                       arp o notas interupt
U23 : calc_notas Port map (CLK2,notas_sel,nota1,nota2);
                          -- Convertir seï¿½ales a notas
U24 : fase Port map (CLK2,nota2,num_inst,f1_2,f2_2,f3_2,f4_2,f5_2,f6_2,f7_2,
                     f8_2,f9_2,f10_2,f11_2,f12_2,f13_2,f14_2,f15_2,f16_2,
                     f17_2,f18_2,f19_2);

U25 : amplitud Port map (CLK2,nota2,num_inst,af_2,a1_2,a2_2,a3_2,a4_2,a5_2,
                         a6_2,a7_2,a8_2,a9_2,a10_2,a11_2,a12_2,a13_2,a14_2,
                         a15_2,a16_2,a17_2,a18_2,a19_2);
U26 : incrementos Port map (CLK2,nota2,num_inst,ince_2,inc_2);
U27 : arm Port map (CLK2,nota2,num_inst,RST,ince_2,inc_2,armf_2,arm1_2,
                    arm2_2,arm3_2,arm4_2,arm5_2,arm6_2,arm7_2,arm8_2,
                    arm9_2,arm10_2,arm11_2,arm12_2,arm13_2,arm14_2,arm15_2,
                    arm16_2,arm17_2,arm18_2,arm19_2);
U28 : acum Port map (CLK2,RST,armf_2,arm1_2,arm2_2,arm3_2,arm4_2,arm5_2,
                     arm6_2,arm7_2,arm8_2,arm9_2,arm10_2,arm11_2,arm12_2,
                     arm13_2,arm14_2,arm15_2,arm16_2,arm17_2,arm18_2,arm19_2,
                     Bf_2,B1_2,B2_2,B3_2,B4_2,B5_2,B6_2,B7_2,B8_2,B9_2,B10_2,
```

```
                        B11_2,B12_2,B13_2,B14_2,B15_2,B16_2,B17_2,B18_2,B19_2);
U29 : sumfas Port map (f1_2,f2_2,f3_2,f4_2,f5_2,f6_2,f7_2,f8_2,f9_2,f10_2,
                       f11_2,f12_2,f13_2,f14_2,f15_2,f16_2,f17_2,f18_2,f19_2,
                       Bf_2,B1_2,B2_2,B3_2,B4_2,B5_2,B6_2,B7_2,B8_2,B9_2,B10_2,
                       B11_2,B12_2,B13_2,B14_2,B15_2,B16_2,B17_2,B18_2,B19_2,
                       Bf_3_2,B1_3_2,B2_3_2,B3_3_2,B4_3_2,B5_3_2,B6_3_2,B7_3_2,
                       B8_3_2,B9_3_2,B10_3_2,B11_3_2,B12_3_2,B13_3_2,B14_3_2,
                       B15_3_2,B16_3_2,B17_3_2,B18_3_2,B19_3_2);
U30 : FF1 Port map (CLK2,Bf_3_2,B1_3_2,B2_3_2,B3_3_2,B4_3_2,B5_3_2,B6_3_2,
                    B7_3_2,B8_3_2,B9_3_2,B10_3_2,B11_3_2,B12_3_2,B13_3_2,
                    B14_3_2,B15_3_2,B16_3_2,B17_3_2,B18_3_2,B19_3_2,Bf_4_2,
                    B1_4_2,B2_4_2,B3_4_2,B4_4_2,B5_4_2,B6_4_2,B7_4_2,B8_4_2,
                    B9_4_2,B10_4_2,B11_4_2,B12_4_2,B13_4_2,B14_4_2,B15_4_2,
                    B16_4_2,B17_4_2,B18_4_2,B19_4_2);
U31 : mux1 Port map (CLK4,cont,Bf_4_2,B1_4_2,B2_4_2,B3_4_2,B4_4_2,B5_4_2,
                     B6_4_2,B7_4_2,B8_4_2,B9_4_2,B10_4_2,B11_4_2,B12_4_2,
                     B13_4_2,B14_4_2,B15_4_2,B16_4_2,B17_4_2,B18_4_2,B19_4_2,
                     dir_mem_2);
U32 : sen Port map (dir_mem_2,sensal_2);
U33 : DMUX Port map (CLK4,cont,sensal_2,Bf_5_2,B1_5_2,B2_5_2,B3_5_2,B4_5_2,
                      B5_5_2,B6_5_2,B7_5_2,B8_5_2,B9_5_2,B10_5_2,B11_5_2,
                      B12_5_2,B13_5_2,B14_5_2,B15_5_2,B16_5_2,B17_5_2,B18_5_2,
                      B19_5_2);
U34 : FF2 Port map (CLK4,cont,Bf_5_2,B1_5_2,B2_5_2,B3_5_2,B4_5_2,B5_5_2,
                    B6_5_2,B7_5_2,B8_5_2,B9_5_2,B10_5_2,B11_5_2,B12_5_2,
                    B13_5_2,B14_5_2,B15_5_2,B16_5_2,B17_5_2,B18_5_2,B19_5_2,
                    Bf_FF_2,B1_FF_2,B2_FF_2,B3_FF_2,B4_FF_2,B5_FF_2,B6_FF_2,
                    B7_FF_2,B8_FF_2,B9_FF_2,B10_FF_2,B11_FF_2,B12_FF_2,
                    B13_FF_2,B14_FF_2,B15_FF_2,B16_FF_2,B17_FF_2,B18_FF_2,
                    B19_FF_2);
U35 : mult Port map (af_2,a1_2,a2_2,a3_2,a4_2,a5_2,a6_2,a7_2,a8_2,a9_2,a10_2,
                     a11_2,a12_2,a13_2,a14_2,a15_2,a16_2,a17_2,a18_2,a19_2,
                     Bf_FF_2,B1_FF_2,B2_FF_2,B3_FF_2,B4_FF_2,B5_FF_2,B6_FF_2,
                     B7_FF_2,B8_FF_2,B9_FF_2,B10_FF_2,B11_FF_2,B12_FF_2,
                     B13_FF_2,B14_FF_2,B15_FF_2,B16_FF_2,B17_FF_2,B18_FF_2,
                     B19_FF_2,Bf_6_2,B1_6_2,B2_6_2,B3_6_2,B4_6_2,B5_6_2,B6_6_2,
                     B7_6_2,B8_6_2,B9_6_2,B10_6_2,B11_6_2,B12_6_2,B13_6_2,
                     B14_6_2,B15_6_2,B16_6_2,B17_6_2,B18_6_2,B19_6_2);
U36 : offset Port map (af_2,a1_2,a2_2,a3_2,a4_2,a5_2,a6_2,a7_2,a8_2,a9_2,a10_2,
                       a11_2,a12_2,a13_2,a14_2,a15_2,a16_2,a17_2,a18_2,a19_2,
                       Bf_6_2,B1_6_2,B2_6_2,B3_6_2,B4_6_2,B5_6_2,B6_6_2,B7_6_2,
                       B8_6_2,B9_6_2,B10_6_2,B11_6_2,B12_6_2,B13_6_2,B14_6_2,
                       B15_6_2,B16_6_2,B17_6_2,B18_6_2,B19_6_2,Bf_7_2,B1_7_2,
                       B2_7_2,B3_7_2,B4_7_2,B5_7_2,B6_7_2,B7_7_2,B8_7_2,B9_7_2,
                       B10_7_2,B11_7_2,B12_7_2,B13_7_2,B14_7_2,B15_7_2,B16_7_2,
                       B17_7_2,B18_7_2,B19_7_2);
U37 : suma Port map (Bf_7_2,B1_7_2,B2_7_2,B3_7_2,B4_7_2,B5_7_2,B6_7_2,B7_7_2,
                     B8_7_2,B9_7_2,B10_7_2,B11_7_2,B12_7_2,B13_7_2,B14_7_2,
                     B15_7_2,B16_7_2,B17_7_2,B18_7_2,B19_7_2,Salsum_2);
U38 : ajuste Port map (Salsum_2,Sintaud_2,Sintaud2_2);
U39 : sumfin Port map (Sintaud,Sintaud2,Sintaud_2,Sintaud2_2,Sal1,Sal2);
U43 : LED3 Port map ( INIT,CLK, W_EN , RST ,sel, num_inst,LED_config,LED_modo,LED_inst);
```

4.2.18.-Test Bench

Mediante el siguiente *Test Bench* obtendremos un fichero con los valores de la onda de uno de los dos canales antes de pasar por el interfaz de audio. Estos valores irán a una frecuencia de 48000Hz.

Para simular los valores que se irán cambiando son:
- notas: es un vector, donde cada bit corresponde con cada una de las notas.
- des: es un vector donde cada bit corresponde con uno de los pulsadores para desactivar.
- num_inst: es el numero del instrumento que queremos probar.

```
signal notas : std_logic_vector(11 downto 0) := "100000000001"; -- introducir notas que toco
signal des : std_logic_vector(11 downto 0) := "000000000000";    --introducir que quiero desactivar
signal num_inst : std_logic_vector(1 downto 0) := "10";
```

Para generar el fichero con los valores en que cambien las salidas debemos definir los siguientes parámetros:

```
file vect_out: text open WRITE_MODE is
"archivo_escritura.txt";
```

```
CLK <= not CLK after 5 ns;
RST <= '0' after 10 ns;

process (CLK)
	begin
		if (CLK'event and CLK='1') then
			if Qint = 1041 then Qint <= "00000000000"; CLK2 <= '1';     -- CLK a
48000
			else Qint <= Qint+1; CLK2 <= '0';             -- Qint contador reloj señales
control MUX
			end if;
		end if;
end process;
process (CLK2)
	variable buf_in,buf_out:line;
	begin
	if (CLK2'event and CLK2='1') then
		Salent <= conv_integer(sim);
		i <= i+1;                -- saber cuantos numeros estoy sacando
		WRITE(buf_out,Salent);
		WRITELINE(vect_out,buf_out);
	end if;
end process;
```

Capítulo 5.- Simulación del funcionamiento del circuito

En este capítulo vamos a describir el proceso de simulación del sistema completo. Para ello se simulará el *Test Bench* descrito en el capítulo anterior con el Simulador de Xilinx. Se definirá un tiempo de simulación largo, ya que el reloj que se simula es el de 50MHz, aunque mediante este *Test Bench* obtendremos una salida a 48KHz. Los resultados de esta simulación se almacenarán en un fichero, que se llamará archivo_escritura.txt.
Los valores de este fichero se llevarán a *Matlab* donde serán asignados a un vector x.

En el entorno de Matlab se definirán también otros parámetros: t, var, t2, que a continuación se definen:

- El vector t es la base de tiempos para la señal x, se define como: forma t=[1:1:n] donde n será el número de valores que tenemos en x.
- La señal var=1/48000, define el periodo a la frecuencia deseada.
- El vector t2 pondera el vector t de acuerdo con el valor de var (t2=[t.*var]) .

De esta forma, la señal x puede verse gráficamente en función de t2 utilizando el comando: plot (t2,x).

A continuación simularemos la nota C6 del violín para ello introducimos en notas "100000000001", en des "000000000001" (de este modo también comprobamos que funciona el desactivar las notas) y en num_inst “10”.

A continuación podemos ver la simulación que obtenemos y comparándola con la del modelo vemos que el periodo y la forma de la onda es el mismo (la amplitud no sera la misma pero sí que será equivalente).

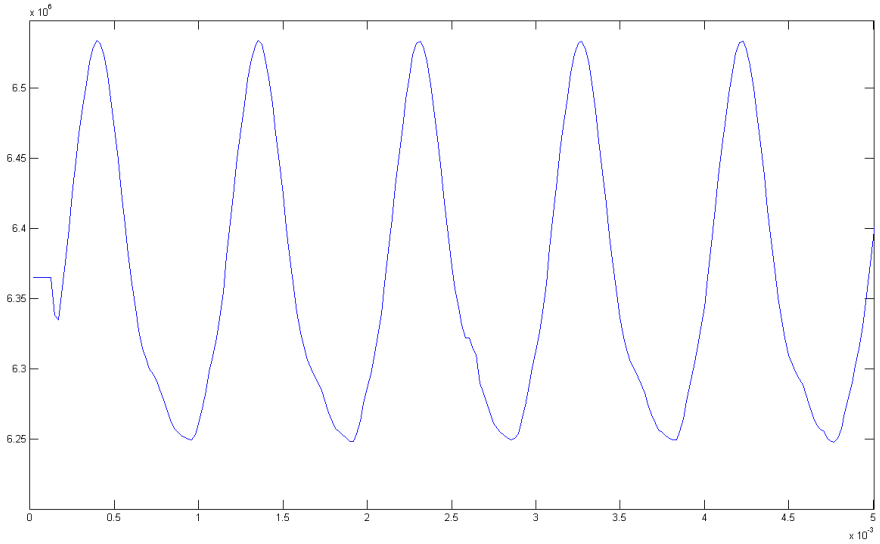


Figura 35: Simulación C6 violín.

A continuación podemos comprobar las otras simulaciones

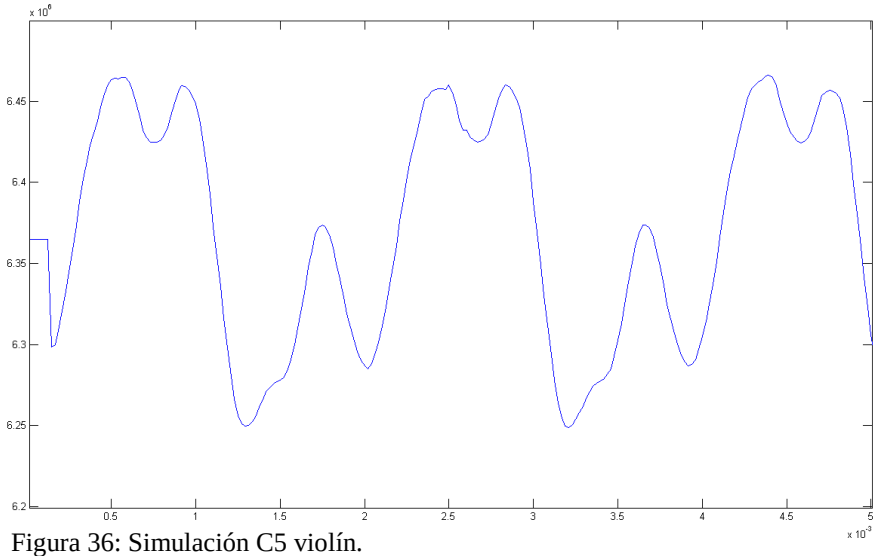


Figura 36: Simulación C5 violín.

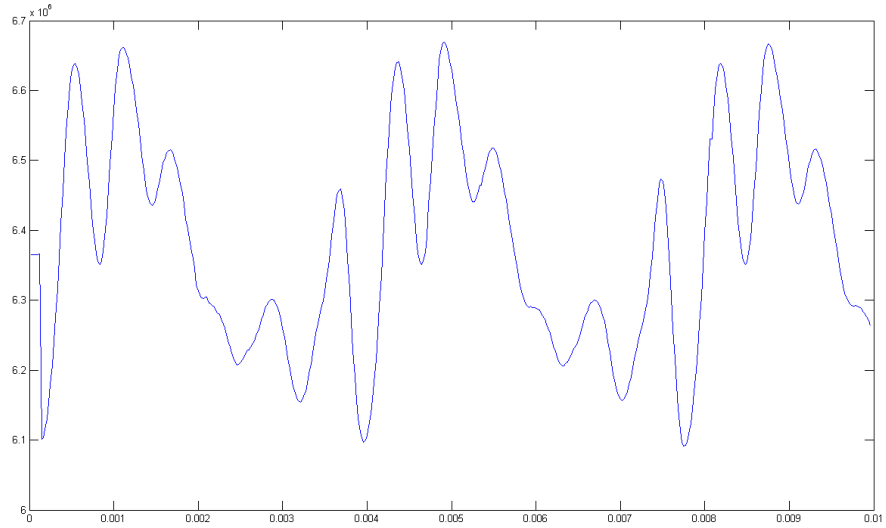


Figura 37: Simulación C4 oboe.

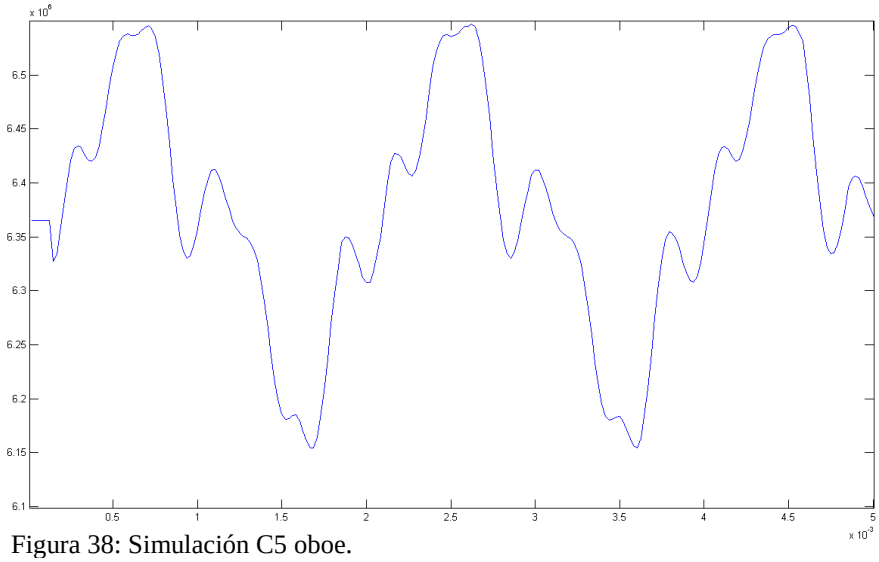


Figura 38: Simulación C5 oboe.

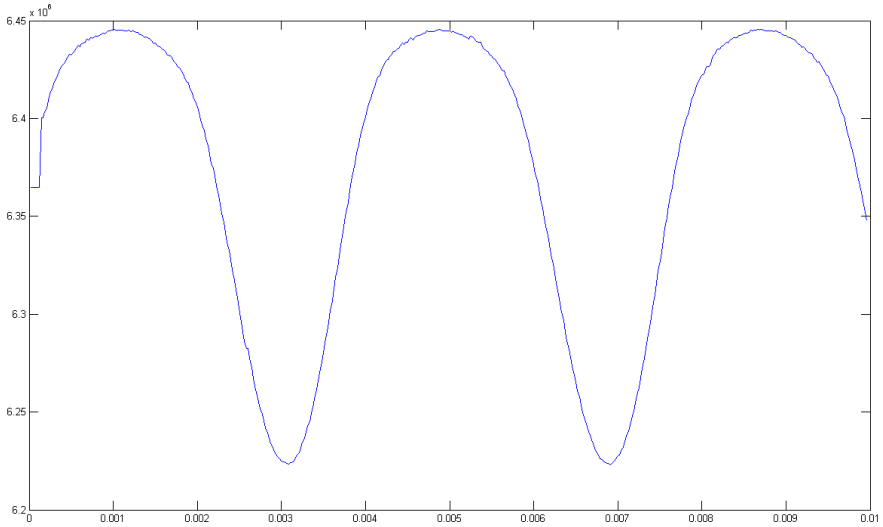


Figura 39: Simulación C4 tuba.

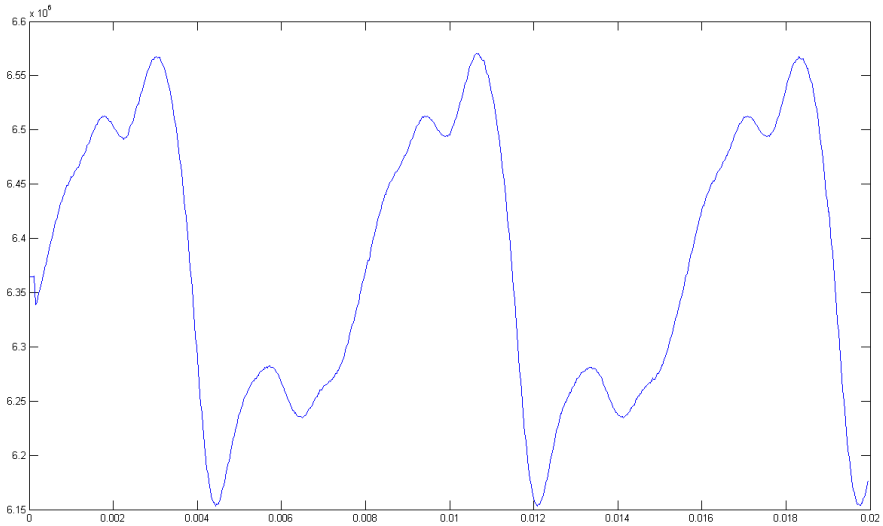


Figura 40: Simulación C3 tuba.

Capítulo 6.-Implementación en la placa

En este capítulo se muestran los detalles de la implementación del sistema en la placa DE2-115.

6.1.- Asignación de pines

En la siguiente tabla podemos ver cómo debemos asignar todos los pines al dispositivo programable FPGA DE2-115. Se han asignado tanto las señales de entrada correspondientes a las notas generadas en el arpa [notas] como las mismas señales si fuesen generadas por los switches de la placa [notas_int].

	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved
1	AUD_BCLK	Output	PIN_F2	1	B1_N1	2.5 V (default)	
2	AUD_DACDAT	Output	PIN_D1	1	B1_N0	2.5 V (default)	
3	AUD_DACLRCK	Output	PIN_E3	1	B1_N0	2.5 V (default)	
4	AUD_MCLK	Output	PIN_E1	1	B1_N0	2.5 V (default)	
5	CLK	Input	PIN_Y2	2	B2_N0	2.5 V (default)	
6	en	Input	PIN_R24	5	B5_N0	2.5 V (default)	
7	I2C_SCLK	Output	PIN_B7	8	B8_N1	2.5 V (default)	
8	I2C_SDAT	Output	PIN_A8	8	B8_N1	2.5 V (default)	
9	INIT	Input	PIN_Y23	5	B5_N2	2.5 V (default)	
10	LED_config[2]	Output	PIN_H15	7	B7_N2	2.5 V (default)	
11	LED_config[1]	Output	PIN_G16	7	B7_N2	2.5 V (default)	
12	LED_config[0]	Output	PIN_G15	7	B7_N2	2.5 V (default)	
13	LED_inst[1]	Output	PIN_H17	7	B7_N2	2.5 V (default)	
14	LED_inst[0]	Output	PIN_J16	7	B7_N2	2.5 V (default)	
15	LED_modo	Output	PIN_F15	7	B7_N2	2.5 V (default)	
16	notas[11]	Input	PIN_AG25	4	B4_N1	2.5 V (default)	
17	notas[10]	Input	PIN_AF22	4	B4_N0	2.5 V (default)	
18	notas[9]	Input	PIN_AE22	4	B4_N0	2.5 V (default)	
19	notas[8]	Input	PIN_AF25	4	B4_N1	2.5 V (default)	
20	notas[7]	Input	PIN_AF24	4	B4_N1	2.5 V (default)	
21	notas[6]	Input	PIN_AD19	4	B4_N0	2.5 V (default)	
22	notas[5]	Input	PIN_AC19	4	B4_N0	2.5 V (default)	
23	notas[4]	Input	PIN_AD15	4	B4_N2	2.5 V (default)	
24	notas[3]	Input	PIN_AD21	4	B4_N0	2.5 V (default)	
25	notas[2]	Input	PIN_AC21	4	B4_N0	2.5 V (default)	
26	notas[1]	Input	PIN_AB21	4	B4_N0	2.5 V (default)	
27	notas[0]	Input	PIN_AB22	4	B4_N0	2.5 V (default)	
28	notas_int[11]	Input	PIN_AB28	5	B5_N1	2.5 V (default)	
29	notas_int[10]	Input	PIN_AC28	5	B5_N2	2.5 V (default)	
30	notas_int[9]	Input	PIN_AC27	5	B5_N2	2.5 V (default)	
31	notas_int[8]	Input	PIN_AD27	5	B5_N2	2.5 V (default)	
32	notas_int[7]	Input	PIN_AB27	5	B5_N1	2.5 V (default)	
33	notas_int[6]	Input	PIN_AC26	5	B5_N2	2.5 V (default)	
34	notas_int[5]	Input	PIN_AD26	5	B5_N2	2.5 V (default)	
35	notas_int[4]	Input	PIN_AB26	5	B5_N1	2.5 V (default)	
36	notas_int[3]	Input	PIN_AC25	5	B5_N2	2.5 V (default)	
37	notas_int[2]	Input	PIN_AB25	5	B5_N1	2.5 V (default)	
38	notas_int[1]	Input	PIN_AC24	5	B5_N2	2.5 V (default)	
39	notas_int[0]	Input	PIN_AB24	5	B5_N2	2.5 V (default)	
40	num_inst[1]	Input	PIN_AA24	5	B5_N2	2.5 V (default)	
41	num_inst[0]	Input	PIN_AB23	5	B5_N2	2.5 V (default)	
42	pulse_48KHz	Output				2.5 V (default)	
43	RST	Input	PIN_AA22	5	B5_N2	2.5 V (default)	
44	sal_LED[11]	Output	PIN_G19	7	B7_N2	2.5 V (default)	
45	sal_LED[10]	Output	PIN_F19	7	B7_N0	2.5 V (default)	
46	sal_LED[9]	Output	PIN_E19	7	B7_N0	2.5 V (default)	
47	sal_LED[8]	Output	PIN_F21	7	B7_N0	2.5 V (default)	
48	sal_LED[7]	Output	PIN_F18	7	B7_N1	2.5 V (default)	
49	sal_LED[6]	Output	PIN_E18	7	B7_N1	2.5 V (default)	
50	sal_LED[5]	Output	PIN_J19	7	B7_N2	2.5 V (default)	
51	sal_LED[4]	Output	PIN_H19	7	B7_N2	2.5 V (default)	
52	sal_LED[3]	Output	PIN_J17	7	B7_N2	2.5 V (default)	
53	sal_LED[2]	Output	PIN_G17	7	B7_N1	2.5 V (default)	
54	sal_LED[1]	Output	PIN_J15	7	B7_N2	2.5 V (default)	
55	sal_LED[0]	Output	PIN_H16	7	B7_N2	2.5 V (default)	
56	sel	Input	PIN_AA23	5	B5_N2	2.5 V (default)	
57	W_EN	Input	PIN_Y24	5	B5_N2	2.5 V (default)	
58	<<new node>>						

Figura 41: Asignación.

6.2.- Recursos utilizados

Tras la implementación del circuito en el dispositivo programable se obtiene las estadísticas de utilización de recursos dentro del dispositivo.
Los recursos para la implementación de nuestro circuito son los siguientes.

Flow Status	Successful - Mon Jun 24 18:55:52 2013
Quartus II 32-bit Version	12.0 Build 263 08/02/2012 SP 2 SJ Web Edition
Revision Name	Proyecto
Top-level Entity Name	proyecto
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	7,551 / 114,480 (7 %)
Total combinational functions	6,856 / 114,480 (6 %)
Dedicated logic registers	3,087 / 114,480 (3 %)
Total registers	3087
Total pins	57 / 529 (11 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	20 / 532 (4 %)
Total PLLs	0 / 4 (0 %)

Figura 42: Recursos.

Podemos ver que para su implementación se han necesitado una cantidad bastante importante de recursos, aunque debido a la gran cantidad de los que dispone nuestro dispositivo esto no es un problema (sólo un 7% del total).

6.3.- Sistema completo

En la siguiente fotografía se muestra el aspecto del sistema completo. Puede verse el arpa y sus conexiones a la placa.

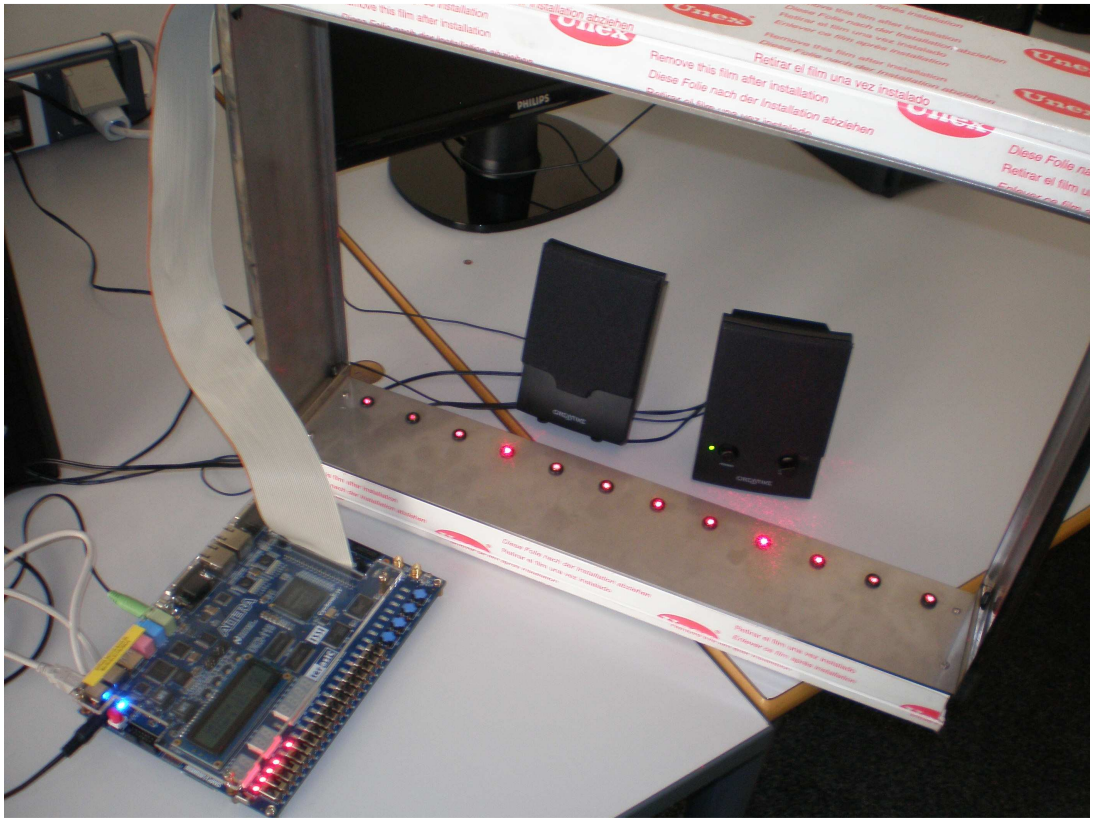


Figura 43: Fotografía.

Capítulo 7.-Conclusiones y futuras mejoras

En este proyecto se ha diseñado un sistema que genera notas musicales correspondientes a tres instrumentos, a partir de las señales procedentes de un arpa láser o a partir de los switches dispuestos en la misma placa.
En apartado 4.2.11., se ha descrito cómo se ha modificado el circuito para conseguir reducir su área, una optimización que permite reducir el número de memorias "sen" de manera significativa, pasando de veinte memorias a una sola.

A continuación se presentan otras mejoras que es posible implementar, para reducir aún más el área del circuito. Concretamente, en los siguientes apartados se muestra cómo puede conseguirse reducir el tamaño de la memoria "sen" utilizando las relaciones trigonométricas.
Para implementar estas mejoras es necesario trabajar con signo, lo que implica la modificación del código descrito anteriormente.

7.1.- Reducción de la memoria del circuito utilizando relaciones aritméticas

7.1.1.- Tono

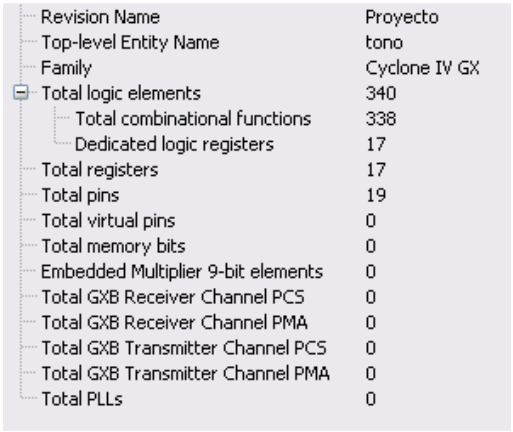
Para describir estas mejoras se partirá de un único tono y se extrapolarán los resultados al caso general.
En el siguiente código se muestra cómo generar un tono utilizando una memoria.

```
type ROM_sen is array (0 to 511) of integer;
        constant
MEM_sen:ROM_sen:=( 0,1,3,4,6,7,9,11,12,14,15,17,18,20,21,23,25,26,28,29,31,32,34,35,37,38,40,
                   41,43,44,46,47,49,50,51,53,54,56,57,59,60,61,63,64,65,67,68,69,71,72,73,
                   75,76,77,78,80,81,82,83,84,86,87,88,89,90,91,92,93,94,96,97,98,99,100,
                   101,102,102,103,104,105,106,107,108,109,109,110,111,112,113,113,114,
                   115,115,116,117,117,118,118,119,120,120,121,121,122,122,123,123,123,
                   124,124,124,125,125,125,126,126,126,126,127,127,127,127,127,127,127,
                   127,127,127,127,127,127,127,127,127,127,127,127,126,126,126,126,
                   126,125,125,125,124,124,124,123,123,122,122,121,121,120,120,119,119,
                   118,118,117,116,116,115,114,114,113,112,111,111,110,109,108,107,106,
                   106,105,104,103,102,101,100,99,98,97,96,95,94,93,92,91,90,88,87,86,85,
                   84,83,81,80,79,78,77,75,74,73,71,70,69,67,66,65,63,62,61,59,58,56,55,54,
                   52,51,49,48,46,45,43,42,40,39,37,36,34,33,31,30,28,27,25,24,22,21,19,18,
                   16,14,13,11,10,8,7,5,3,2,0,-1,-3,-4,-6,-8,-9,-11,-12,-14,-15,-17,-19,-20,-22,
                   -23,-25,-26,-28,-29,-31,-32,-34,-35,-37,-38,-40,-41,-43,-44,-46,-47,-49,-50,
                   -52,-53,-55,-56,-57,-59,-60,-62,-63,-64,-66,-67,-68,-70,-71,-72,-74,-75,-76,
                   -78,-79,-80,-81,-82,-84,-85,-86,-87,-88,-89,-91,-92,-93,-94,-95,-96,-97,-98,
                   -99,-100,-101,-102,-103,-104,-105,-106,-107,-107,-108,-109,-110,-111,
                   -112,-112,-113,-114,-115,-115,-116,-117,-117,-118,-119,-119,-120,-120,
                   -121,-121,-122,-122,-123,-123,-124,-124,-125,-125,-125,-126,-126,-126,
                   -127,-127,-127,-127,-127,-128,-128,-128,-128,-128,-128,-128,-128,-128,
                   -128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-127,-127,-127,
                   -127,-126,-126,-126,-125,-125,-125,-124,-124,-124,-123,-123,-122,-122,
                   -121,-121,-120,-119,-119,-118,-118,-117,-116,-116,-115,-114,-114,-113,
                   -112,-111,-110,-110,-109,-108,-107,-106,-105,-104,-103,-103,-102,-101,
                   -100,-99,-98,-97,-95,-94,-93,-92,-91,-90,-89,-88,-87,-85,-84,-83,-82,-81,
                   -79,-78,-77,-76,-74,-73,-72,-70,-69,-68,-66,-65,-64,-62,-61,-60,-58,-57,-55,
                   -54,-52,-51,-50,-48,-47,-45,-44,-42,-41,-39,-38,-36,-35,-33,-32,-30,-29,-27,
                   -26,-24,-22,-21,-19,-18,-16,-15,-13,-12,-10,-8,-7,-5,-4,-2,-1);
```

El siguiente proceso muestra cómo se genera el tono:

```
begin
        INCe <= 8080;
        INC <= conv_std_logic_vector(INCe,18);
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                ---------- RESET ------------------------
                if (RST='1') then
                                Bf_1 <= (others => '0');
                                Bf_2 <= (others => '0');
                elsif (RST='0') then
                                ----------------------------------------------------amplitudes
                                ampfunde <= 100;
                                ampfund <= conv_std_logic_vector(ampfunde,8);
                                ---------- Primer registro ----------
                                Bf_1 <= INC;        -- frecuencia fundamental
                                ---------- Acumulador ----------
                                Bf_2 <= Bf_2+Bf_1;
                end if;
        end if;
end Process;
                                ---------- suma de la fase ----------
                                Bf_3 <= Bf_2;
                                ---------- truncado para memoria L bits----------
                                Bf_4 <= Bf_3 (17 downto 9);
                                Bf_4mem <= "0" & Bf_4;
                                Bf_i <= MEM_sen (CONV_INTEGER(Bf_4mem));
                                Bf_5 <= conv_std_logic_vector(Bf_i,8);
                                Bf_6 <= Bf_5 * ampfund;
Bf_antsal <= Bf_6;
Bf_pos <= conv_std_logic_vector(13500,16); -- poder convertir a integer para simular
Sal <= (Bf_antsal(15) & Bf_antsal) + (Bf_pos(15) & Bf_pos);
```

La siguiente figura muestra la cantidad de recursos necesarios para generar un tono. Estos valores se tomarán como referencia para la estimación de la bondad de las mejoras propuestas.



Revision Name	Proyecto
Top-level Entity Name	tono
Family	Cyclone IV GX
Total logic elements	340
Total combinational functions	338
Dedicated logic registers	17
Total registers	17
Total pins	19
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0
Total GXB Receiver Channel PMA	0
Total GXB Transmitter Channel PCS	0
Total GXB Transmitter Channel PMA	0
Total PLLs	0

Figura 44: Recursos tono.

La siguiente figura muestra el esquema de la memoria que almacena la onda senoidal correspondiente al tono (memoria de 512x32).

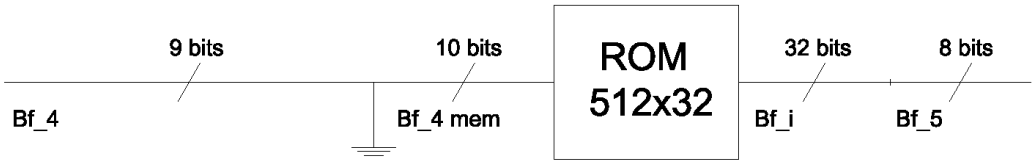


Figura 45: Esquema tono.

7.1.2.- Mejora 1

La primera mejora consigue reducir la memoria descrita antes a la mitad, almacenando únicamente en dicha memoria los valores de la señal senoidal entre 0 y 180º, y obteniéndose el resto de valores a partir de sencillas operaciones aritméticas. El bit de mayor peso es el que indica si la muestra de la señal corresponde al semiciclo positivo o negativo, y por tanto, será la que actúe como señal de control en los multiplexores adicionales que se requieren. Nótese, que se ha incluido un bloque inversor para generar los valores negativos de la onda.

En la siguiente figura se muestra cómo la memoria es de tamaño 256x12 y se han añadido bloques multiplexores 2 a 1. Esto implicará el uso de aproximadamente 32 LEs más que en el caso anterior para la implementación de los muxes.

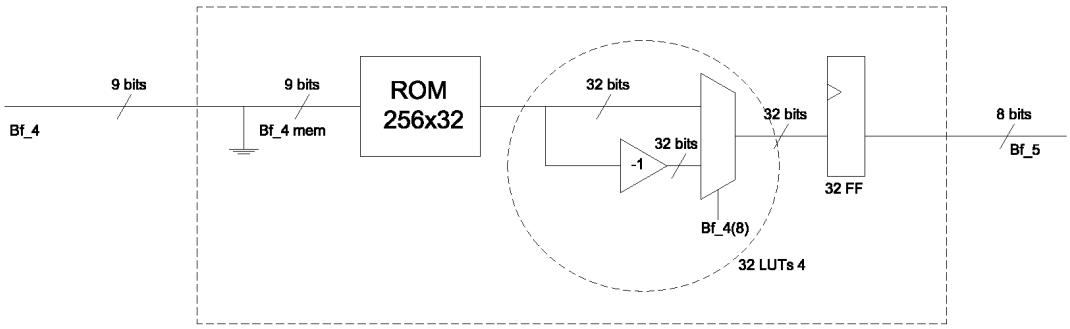


Figura 46: Esquema mejora 1.

A continuación se muestra el código VHDL correspondiente a la descripción de la memoria y a la generación del tono. También se indican los recursos utilizados.

```
type ROM_sen is array (0 to 255) of integer;
constant MEM_sen:ROM_sen:=(  0,1,3,4,6,7,9,11,12,14,15,17,18,20,21,23,25,26,28,29,31,32,34,35,
                             37,38,40,41,43,44,46,47,49,50,51,53,54,56,57,59,60,61,63,64,65,
                             67,68,69,71,72,73,75,76,77,78,80,81,82,83,84,86,87,88,89,90,91,
                             92,93,94,96,97,98,99,100,101,102,102,103,104,105,106,107,108,
                             109,109,110,111,112,113,113,114,115,115,116,117,117,118,118,
                             119,120,120,121,121,122,122,123,123,123,124,124,124,125,125,
                             125,126,126,126,126,127,127,127,127,127,127,127,127,127,127,
                             127,127,127,127,127,127,127,127,127,127,126,126,126,126,126,
                             125,125,125,124,124,124,123,123,122,122,121,121,120,120,119,
                             119,118,118,117,116,116,115,114,114,113,112,111,111,110,109,
                             108,107,106,106,105,104,103,102,101,100,99,98,97,96,95,94,93,
                             92,91,90,88,87,86,85,84,83,81,80,79,78,77,75,74,73,71,70,69,67,
                             66,65,63,62,61,59,58,56,55,54,52,51,49,48,46,45,43,42,40,39,37,
```

```
                                    36,34,33,31,30,28,27,25,24,22,21,19,18,16,14,13,11,10,8,7,5,3,2,
                                    1);
begin
        INCe <= 8080;
        INC <= conv_std_logic_vector(INCe,18);
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                ---------- RESET -------------------------
                if (RST='1') then
                                    Bf_1 <= (others => '0');
                                    Bf_2 <= (others => '0');
                elsif (RST='0') then
                                    ------------------------------------------------------amplitudes
                                    ampfunde <= 100;
                                    ampfund <= conv_std_logic_vector(ampfunde,8);
                                    ---------- Primer registro ----------
                                    Bf_1 <= INC;        -- frecuencia fundamental

                                    ---------- Acumulador ----------
                                    Bf_2 <= Bf_2+Bf_1;
                end if;
        end if;
end Process;
                                    ---------- suma de la fase ----------
                                    Bf_3 <= Bf_2;
                                    ---------- truncado para memoria L bits----------
                                    Bf_4 <= Bf_3 (17 downto 9);
                                    Bf_4mem <= "0" & Bf_4(7 downto 0);
                                    ----------sacar valores mem sen con direcciones bits anteriores---

Process (CLK)
        begin
                if (Bf_4(8)='0') then Bf_i <= MEM_sen (CONV_INTEGER(Bf_4mem));
                else Bf_i <= -MEM_sen (CONV_INTEGER(Bf_4mem));
                end if;
end process;
                Bf_5 <= conv_std_logic_vector(Bf_i,8);
                Bf_6 <= Bf_5 * ampfund;
Bf_antsal <= Bf_6;
Bf_pos <= conv_std_logic_vector(13500,16); -- poder convertir a integer para simular
Sal <= (Bf_antsal(15) & Bf_antsal) + (Bf_pos(15) & Bf_pos);
```

Revision Name	Proyecto
Top-level Entity Name	mejora1
Family	Cyclone IV GX
Total logic elements	243
Total combinational functions	241
Dedicated logic registers	17
Total registers	17
Total pins	19
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0
Total GXB Receiver Channel PMA	0
Total GXB Transmitter Channel PCS	0
Total GXB Transmitter Channel PMA	0
Total PLLs	0

Figura 47: Recursos mejora 1.

7.1.3.- Mejora 2

Si se siguen utilizando las relaciones aritméticas podemos reducir aún más el tamaño de la memoria. Concretamente, es posible reducir esta memoria a la cuarta parte almacenando únicamente los valores comprendidos entre 0º y 90º y generando el resto a partir de sencillas operaciones matemáticas.

El esquema de este circuito se muestra en la siguiente figura:

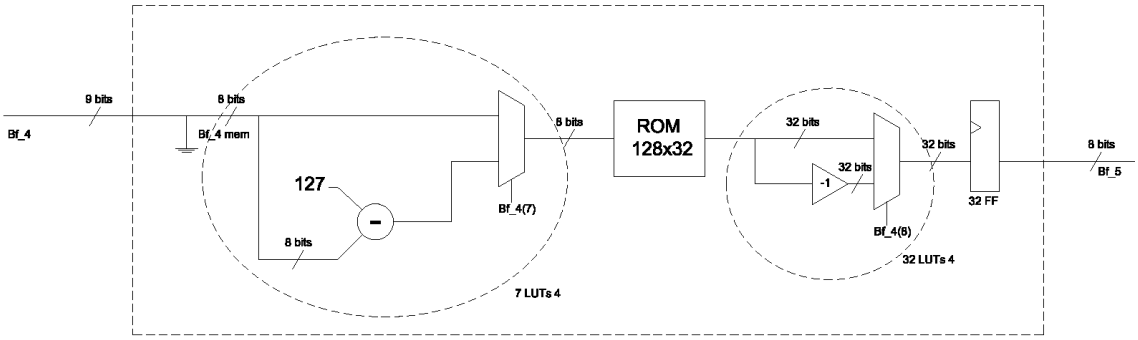


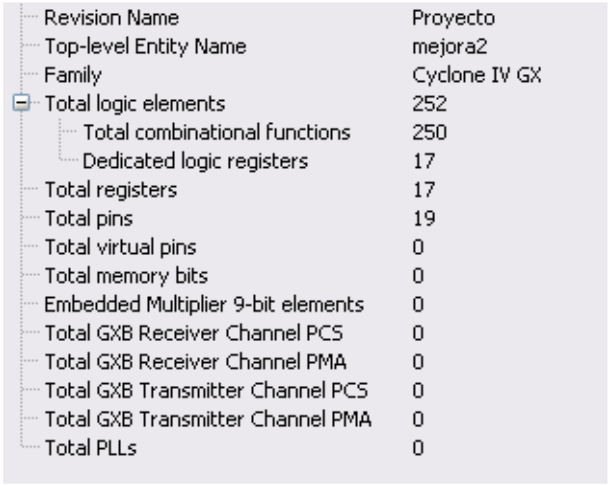
Figura 48: Esquema mejora 1.

Del esquema se deduce que este ahorro de área implica un ligero aumento en el número de LUTs, concretamente son necesarias 39 LUTs más de lo requerido en un principio.
A continuación se muestra el código VHDL correspondiente a la descripción de la memoria y a la generación del tono. También se indican los recursos utilizados.

```
type ROM_sen is array (0 to 127) of integer;
constant MEM_sen:ROM_sen:=(   0,1,3,4,6,7,9,11,12,14,15,17,18,20,21,23,25,26,28,29,31,32,34,
                              35, 37,38,40,41,43,44,46,47,49,50,51,53,54,56,57,59,60,61,63,64,
                              65,67,68,69,71,72,73,75,76,77,78,80,81,82,83,84,86,87,88,89,90,
                              91,92,93,94,96,97,98,99,100,101,102,102,103,104,105,106,107,
                              108,109,109,110,111,112,113,113,114,115,115,116,117,117,118,
                              118,119,120,120,121,121,122,122,123,123,123,124,124,124,125,
                              125,125,126,126,126,126,127,127,127,127,127,127,127,127,127,
                              127);
        INCe <= 8080;
        INC <= conv_std_logic_vector(INCe,18);
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                ----------------- RESET -------------------------
                if (RST='1') then
                                Bf_1 <= (others => '0');
                                Bf_2 <= (others => '0');
                elsif (RST='0')
                                ampfunde <= 100;
                                ampfund <= conv_std_logic_vector(ampfunde,8);
                                ----------------- Primer registro ---------------
                                Bf_1 <= INC;        -- frecuencia fundamental
                                ------------------- Acumulador --------------------
                                Bf_2 <= Bf_2+Bf_1;
                end if;
        end if;
end Process;
                                ---------------- suma de la fase ------------------
                                Bf_3 <= Bf_2;
```

```
                                    ---------- truncado para memoria L bits----------
                                    Bf_4 <= Bf_3 (17 downto 9);
                                    Bf_4mem <= "0" & Bf_4(6 downto 0);

Process (CLK)
        begin
                if (Bf_4(8)='0') then
                        if Bf_4(7)='0' then Bf_i <= MEM_sen (CONV_INTEGER(Bf_4mem));
                        else Bf_i <= MEM_sen (CONV_INTEGER(Bf_res));
                        end if;
                elsif  (Bf_4(8)='1') then
                        if Bf_4(7)='0' then Bf_i <= -MEM_sen (CONV_INTEGER(Bf_4mem));
                        else Bf_i <= -MEM_sen (CONV_INTEGER(Bf_res));
                        end if;
        end if;
end process;
Bf_5 <= conv_std_logic_vector(Bf_i,8);
Bf_6 <= Bf_5 * ampfund;
Bf_antsal <= Bf_6;
Bf_pos <= conv_std_logic_vector(13500,16); -- poder convertir a integer para simular
Sal <= (Bf_antsal(15) & Bf_antsal) + (Bf_pos(15) & Bf_pos);
```



Revision Name	Proyecto
Top-level Entity Name	mejora2
Family	Cyclone IV GX
Total logic elements	252
Total combinational functions	250
Dedicated logic registers	17
Total registers	17
Total pins	19
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total GXB Receiver Channel PCS	0
Total GXB Receiver Channel PMA	0
Total GXB Transmitter Channel PCS	0
Total GXB Transmitter Channel PMA	0
Total PLLs	0

Figura 49: Recursos mejora 2.

7.2.- Otras mejoras

Además de las mejoras descritas y analizadas, es posible implementar otros cambios al sistema.

- Introducir los valores de la memoria en forma de vectores de 8 bits, ya que al trabajar con integers estos ocupan cada uno 32 bits.
- Generar la salida en formato MIDI.
- Utilizar el display LCD de la placa para enviar mensajes al usuario del arpa y facilitar su manejo (mostrar mensajes de fallos, indicar la nota que estamos tocando, etc).

Capítulo 8.-Bibliografía

[1]- BERMUDEZ COSTA, JUAN (1977). Nueva generación de instrumentos musicales electrónicos. Barcelona: Marcombo Boixareu editores.
[2]- FLETCHER, NEVILLE H. y ROSSING THOMAS D.(1998). The Physics of Musical Instruments. Nueva York :Springer.
[3]- JORDÀ PUIG, SERGI (1997). Audio digital y midi. Madrid: Anaya Multimedia.
[4]- SANDELL, GREG. The SHARC Timbre Dataset v. 2.0. (2007) <http://gregsandell.blogspot.com.es/search/label/harmonics> (7 de enero de 2013).
[5]- UNIVERSIDAD DE GRANADA. <http://tstc.ugr.es/pages/actividades_de_extension> (11 de junio de 2013).
[6]- UNIVERSIDAD DE VALLADOLID <http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io2/public_html/introduccion.html> (11 de junio de 2013).
[7] - UNIVERSIDAD DE SEVILLA <http://bibing.us.es/proyectos/abreproy/11055/fichero/SINTESIS+DIGITAL+DE+INSTRUMENTOS+MUSICALES.pdf>(11 de junio de 2013).
[8] - UNIVERSITAT POMPEU FABRA <http://www.tecn.upf.es/~sjorda/ME2003/6-SintesisDigital/ME-6SintesisDigital.pdf> (11 de junio de 2013).
[9]- BUIDE, BENITO. La escala de los armónicos. < http://www.relafare.eu > (11 de junio de 2013).
[10]- TERASIC TECHNOLOGIES (2010). DE2-115 User manual.
[11]- ALTERA CORPORATION (2010). Cyclone IV Device Handbook.
[12]- XILLINX (2005). DDS v5.0
[13]- WOLFSON MICROELECTRONICS (2012). WM8731 / WM8731L datasheet.

Anexo 1.- Manual de usuario del sistema

Este sistema permite sintetizar el sonido de un oboe, un violín y una tuba.
Cada instrumento trabaja en una escala distinta, concretamente:

-Tuba: De C3 a C4
-Oboe: De C4 a C5
-Violín: De C5 a C6.

Dentro de cada escala es posible generar todas las notas salvo F#. Esto es así porque el arpa láser tiene únicamente 12 lásers.

El sistema puede sintetizar sonido actuando sobre los haces de luz del arpa o utilizando los switches de la placa como teclas de un piano.

El sistema es capaz de generar simultáneamente el sonido de dos notas, por lo que si se toca un acorde, sonarán dos notas del mismo. Cuando se pulsan más de dos notas simultáneamente, el sistema detecta la más aguda y la más grave y reproduce únicamente estas dos.

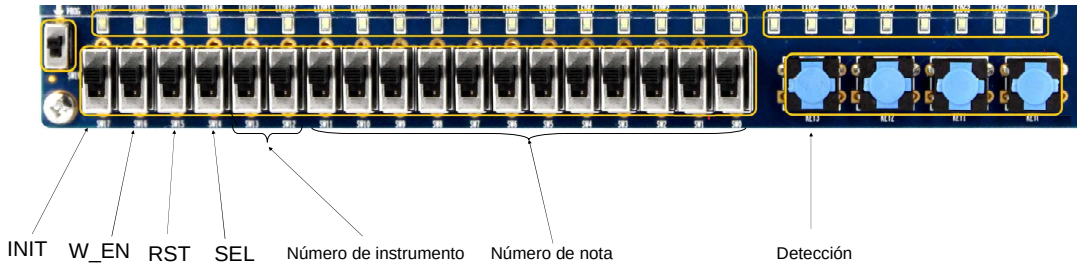


Figura 50: Manual

A continuación vamos a ver las diferentes entradas utilizadas para controlar el circuito:

- INIT: Corresponde con el switch 17. Para que funcione el circuito debe ser “0” (interruptor hacia abajo) y el led 17 debe estar iluminado.
- W_EN: Corresponde con el switch 16. Para que funcione debe ser “1” (interruptor hacia arriba). El led 16 debe estar iluminado.
- RST: Corresponde con el switch 15. Para que funcione debe ser “0”. El led 15 debe estar iluminado, además si hacemos un reset se apagara este led y los que corresponden a INIT y W_EN.
- SEL: Selección entre el arpa o los interruptores para generar las notas: Esta selección se realizara con el switch 14. Si se encuentra encendido el led usaremos el arpa, en caso contrario se estarán usando los interruptores.
- Número de instrumento: El número de instrumento lo controlamos con los switches 13 y 14. El valor se muestra en los leds 13 y 14. Los valores son los siguientes:
  - “00” No se corresponde con ningún instrumento.
  - “01” Oboe.
  - “10” Violin.
  - “11” Tuba.

Interruptores con las notas: Corresponderán con los switches del número 11 (nota más grave) al 0 (nota más aguda). Los leds indican qué notas se están tocando.

- SW 11: C
- SW 10: C#
- SW 9: D

- SW 8: D#
- SW 7: E
- SW 6: F
- SW 5: G
- SW 4: G#
- SW 3: A
- SW 2: A#
- SW 1: B
- SW 0: C

Si al conectar el arpa sin tocar ninguna nota se iluminan los leds de las notas, significa que esas notas del arpa no funciona, por tanto, debemos desactivarlas. Para detectar que leds del arpa se encuentran fundidos y desactivarlos debemos usar el pulsador 3.

Anexo 2: Código VHDL correspondiente a los módulos descritos en el capítulo 4

Divisor1.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity divisor1 is
Port ( CLK : in STD_LOGIC;
       RST : in STD_LOGIC;
       salida : out STD_LOGIC);
end divisor1;
architecture Behavioral of divisor1 is
signal Qint: STD_LOGIC_VECTOR (10 downto 0);
signal sint: STD_LOGIC;
begin
process (CLK, RST)
begin
        if RST = '1' then Qint <= "00000000000"; sint <= '0';
        elsif (CLK' event) and (CLK = '1') then
                if Qint = 1041 then
                Qint <= "00000000000"; sint <= '1';
                else Qint <= Qint+1; sint <= '0';
                end if;
        end if;
end process;
salida <= sint;
end Behavioral;
```

Divisor2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity divisor2 is
Port (  CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        salida : out STD_LOGIC);
end divisor2;
architecture Behavioral of divisor2 is
signal Qint: STD_LOGIC_VECTOR (1 downto 0);
signal sint: STD_LOGIC;
begin
process (CLK, RST)
begin
        if RST = '1' then Qint <= "00"; sint <= '0';
        elsif (CLK' event) and (CLK = '1') then
                if Qint = 1 then
                Qint <= "00"; sint <= '1';
                else Qint <= Qint+1; sint <= '0';
                end if;
        end if;
end process;

salida <= sint;
end Behavioral;
```

Divisor3.

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

entity divisor3 is
Port ( CLK : in STD_LOGIC;
                        RST : in STD_LOGIC;
                        salida : out STD_LOGIC);
end divisor3;

architecture Behavioral of divisor3 is
signal Qint: STD_LOGIC_VECTOR (10 downto 0);
signal sint: STD_LOGIC;
begin

process (CLK, RST)
begin
            if RST = '1' then Qint <= "00000000000"; sint <= '0';
            elsif (CLK' event) and (CLK = '1') then
                        if Qint = 49 then
                        Qint <= "00000000000"; sint <= '1';
                        else Qint <= Qint+1; sint <= '0';
                        end if;
            end if;
end process;

salida <= sint;

end Behavioral;
```

6.4.-g00_audio_interface

```
-- g00_audio_interface
--
-- This module implements the interface to the Wolfson
-- WM8731 Audio Codec chip located on the Altera DE1 board
-- The INIT line when asserted writes configuration data into the registers
-- of the codec chip. These set the sampling rate to 48KHz and selects
-- Slave and USB modes for the interface (implying an external 24MHz input clock).
-- This module only implements the audio output. It does not handle audio
-- input, although it could be easily extended to do so.
--
-- Version 1.0
--
-- Designer: James Clark
-- February 26 2008

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
ENTITY g00_audio_interface IS
	PORT
	(
		LDATA, RDATA		:		IN std_logic_vector(23 downto 0); -- parallel external data inputs
		clk, rst, INIT, W_EN : IN std_logic; -- clk should be 24MHz
		pulse_48KHz :		OUT std_logic; -- sample sync pulse
		AUD_MCLK :		OUT std_logic; -- codec master clock input
		AUD_BCLK :		OUT std_logic; -- digital audio bit clock
		AUD_DACDAT :		OUT std_logic; -- DAC data lines
		AUD_DACLRCK :		OUT std_logic; -- DAC data left/right select
		I2C_SDAT :		OUT std_logic; -- serial interface data line
		I2C_SCLK :		OUT std_logic  -- serial interface clock
	);
END g00_audio_interface;

ARCHITECTURE a OF g00_audio_interface IS

TYPE SCI_state IS (sw_init0,sw_init1,s0,s1,s2,sab1,sab2,sab3,sack11,sack12,sack13,
 sack21,sack22,sack23,sack31,sack32,sack33,sw1b1,sw1b2,sw1b3,sw2b1,sw2b2,sw2b3,send);

TYPE SCI_state2 IS (sw_init0,sw_init1,sw_ready,sw_write);

signal Bcount : unsigned(2 downto 0);
signal BBcount : integer range 0 to 49;
signal Mcount : std_logic;
signal clk_count : integer range 0 to 63;
signal bit_count : integer range 0 to 7;
signal word_count : integer range 0 to 12;
signal LRDATA : std_logic_vector(49 downto 0); -- stores L&R data
signal state : SCI_state;
signal state2 : SCI_state2;
signal SCI_WRITE, SCI_READY : std_logic;
signal SCI_ADDR, SCI_WORD1, SCI_WORD2 : std_logic_vector(7 downto 0);

BEGIN

SCI_ADDR <= "00110100";

-- FSM for controlling audio data transfer to codec
digital_audio_interface: process (clk, rst)

begin

		if rst='1' then
			Mcount <= '1';
			Bcount <= "100";
			BBcount <= 0;
			pulse_48khz <= '0';
			LRDATA <= (others => '0');
			AUD_MCLK <= '0';
			AUD_BCLK <= '0';
			AUD_DACLRCK <= '0';
			AUD_DACDAT <= '0';
		elsif clk='1' and clk'EVENT then
			Mcount <= not Mcount;
```

```
                        if Mcount = '1' then
                                AUD_MCLK <= '1';

                        else -- Mcount = 0
                                AUD_MCLK <= '0';
                                Bcount <= Bcount - "001";
                                if Bcount = "011" then
                                        AUD_BCLK <= '1'; --BCLK is low for 2, high for 3
                                end if; -- Bcount = 3
                                if Bcount = "000" then
                                        AUD_BCLK <= '0';
                                        Bcount <= "100";
                                        BBcount <= BBcount - 1;
                                        if BBcount = 1 then
                                                pulse_48khz <= '1'; -- use for 48Khz sample
                                                                    sync
                                                if W_EN = '1' then
                                                        LRDATA <=
                                                           std_logic_vector(LDATA) &
                                                           std_logic_vector(RDATA) & "00";
                                                end if; -- if W_EN
                                        else
                                                pulse_48khz <= '0';
                                        end if; -- if BBcount
                                        if BBcount = 0 then
                                                BBcount <= 49;
                                        end if; -- if BBcount
                                        if BBcount = 49 then
                                                AUD_DACLRCK <= '1';
                                        else
                                                AUD_DACLRCK <= '0';
                                        end if; -- if BBcount
                                        AUD_DACDAT <= LRDATA(BBcount);
                                end if; -- if Bcount = 0
                        end if; -- if Mcount
                end if; -- if rst
        end process;

-- FSM to control initialization of codec configuration registers
SCI_INIT_FSM: process(clk, rst)
        begin
                if rst='1' then
                        state2 <= sw_init0;
                        word_count <= 0;
                        SCI_WRITE <= '0';
                elsif clk='1' and clk'EVENT then
                        SCI_WRITE <= '0';
                        case state2 is
        when sw_init0 => -- wait for INIT to go low
                                word_count <= 0;
                                if INIT = '0' then
                                        state2 <= sw_init1;
                                end if; -- if INIT='0'
                        when sw_init1 => -- wait for INIT to go high
                                if INIT = '1' then
                                        state2 <= sw_ready;
                                end if; -- if INIT='1'
```

```
                        when sw_ready => -- wait for SCI_READY to go high
                                if SCI_READY = '1' then
                                        state2 <= sw_write;
                                end if; -- if SCI_READY='1'
                        when sw_write => -- write the next word
                                if word_count < 8 then
                                        state2 <= sw_ready; -- wait for next ready cycle
                                        word_count <= word_count + 1;
                                        SCI_WRITE <= '1'; -- begin writing of next word
                                        case word_count is
                                        when 0 =>
                                                SCI_WORD1 <= "00010010"; -- inactivate
                                                                        interface
                                                SCI_WORD2 <= "00000000";
                                        when 1 =>
                                                SCI_WORD1 <= "00000001"; -- set register R0
                                                SCI_WORD2 <= "10010111";
                                        when 2 =>
                                                SCI_WORD1 <= "00001000"; -- set register R4
                                                SCI_WORD2 <= "00010010";
                                        when 3 =>
                                                SCI_WORD1 <= "00001010"; -- set register R5
                                                SCI_WORD2 <= "00010110";
                                        when 4 =>
                                                SCI_WORD1 <= "00001100"; -- set register R6
                                                SCI_WORD2 <= "01100011";
                                        when 5 =>
                                                SCI_WORD1 <= "00001110"; -- set register R7
                                                SCI_WORD2 <= "00001011";
                                        when 6 =>
                                                SCI_WORD1 <= "00010000"; -- set register R8
                                                SCI_WORD2 <= "00000001"; -- sets USB
                                                                        mode
                                        when 7 =>
                                                SCI_WORD1 <= "00010010"; -- reactivate
                                                                        interface
                                                SCI_WORD2 <= "00000001";
                                        when others =>
                                        end case; -- case word_count
                                else
                                                state2 <= sw_init0; -- go back to start
                                end if; -- if word_count
                        end case; -- case state2
                end if; -- if rst
        end process; -- SCI_INIT_FSM

-- FSM controlling the serial transfer of data to the I2C interface
-- to the codec configuration registers.
SCI_FSM: process (clk, rst)
        begin
                if rst='1' then
                        state <= sw_init0;
                        bit_count <= 7;
                        clk_count <= 0;
                        SCI_READY <= '1';
```

```
elsif clk='1' and clk'EVENT then
        I2C_SDAT <= '1'; -- default output values
        I2C_SCLK <= '1';
        case state is
        when sw_init0 => -- wait for SCI_WRITE to go low
                if SCI_WRITE='0' then
                        state <= sw_init1;
                end if; -- if SCI_WRITE='1'
        when sw_init1 => -- wait for SCI_WRITE to go high
                if SCI_WRITE='1' then
                        SCI_READY <= '0';
                        state <= s0;
                end if; -- if SCI_WRITE='0'
        when s0 => -- begin start phase, both I2C_SDAT and I2C_SCLK are high
                clk_count <= clk_count + 1;
                if clk_count = 15 then -- this phase lasts for 16 clocks
                        state <= s1;
                end if; -- if clk_count
        when s1 => -- second part of start phase, I2C_SDAT goes low, while
             I2C_SCLK stays high
                clk_count <= clk_count + 1;
                I2C_SDAT <= '0';
                if clk_count = 47 then -- this phase lasts for 32 clocks
                        clk_count <= 0;
                        state <= s2;
                end if; -- if clk_count
        when s2 => -- end of start phase, both I2C_SDAT and I2C_SCLK are low
                clk_count <= clk_count + 1;
                I2C_SDAT <= '0';
                I2C_SCLK <= '0';
                if clk_count = 15 then -- this phase lasts for 16 clocks
                        bit_count <= 7;
                        state <= sab1;
                        SCI_READY <= '0'; -- indicate we are busy for the next
while
                end if; -- if clk_count
        when sab1 => -- send 8 address bits, MSB first, first phase
                clk_count <= clk_count + 1;
                I2C_SDAT <= SCI_ADDR(bit_count);
                I2C_SCLK <= '0';
                if clk_count = 31 then
                        state <= sab2;
                end if; -- if clk_count

        when sab2 => -- send 8 address bits, MSB first, second phase
                clk_count <= clk_count + 1;
                I2C_SDAT <= SCI_ADDR(bit_count);
                I2C_SCLK <= '1';
                if clk_count = 63 then
                        state <= sab3;
                        clk_count <= 0;
                end if; -- if clk_count

        when sab3 => -- send 8 address bits, MSB first, third phase
                clk_count <= clk_count + 1;
                I2C_SDAT <= SCI_ADDR(bit_count);
```

```
                I2C_SCLK <= '0';
                if clk_count = 15 then
                        if bit_count = 0 then
                                state <= sack11; -- finished all 8 bits, wait for
                                                 ack
                                bit_count <= 7;
                        else
                                state <= sab1; -- write next bit
                                bit_count <= bit_count - 1;
                        end if; -- if bit_count
                end if; -- if clk_count
        when sack11 =>
                clk_count <= clk_count + 1;
                I2C_SDAT <= 'Z'; -- float the tristate data line
                I2C_SCLK <= '0';
                if clk_count = 31 then
                        state <= sack12;
                        clk_count <= 0;
                end if; -- if clk_count
        when sack12 =>
                clk_count <= clk_count + 1;
                I2C_SDAT <= 'Z';
                I2C_SCLK <= '1';
                if clk_count = 31 then
                        state <= sack13;
                        clk_count <= 0;
                end if; -- if clk_count
        when sack13 => -- last phase of acknowledge cycle
                clk_count <= clk_count + 1;
                I2C_SDAT <= 'Z';
                I2C_SCLK <= '0';
                if clk_count = 15 then
                        state <= sw1b1;
                end if; -- if clk_count
        when sw1b1 => -- send 8 bits of first word, MSB first, first phase
                clk_count <= clk_count + 1;
                I2C_SDAT <= SCI_WORD1(bit_count);
                I2C_SCLK <= '0';
                if clk_count = 31 then
                        state <= sw1b2;
                end if; -- if clk_count
        when sw1b2 => -- send 8 bits, MSB first, second phase
                clk_count <= clk_count + 1;
                I2C_SDAT <= SCI_WORD1(bit_count);
                I2C_SCLK <= '1';
                if clk_count = 63 then
                        state <= sw1b3;
                        clk_count <= 0;
                end if; -- if clk_count
        when sw1b3 => -- send 8 address bits, MSB first, third phase
                clk_count <= clk_count + 1;
                I2C_SDAT <= SCI_WORD1(bit_count);
                I2C_SCLK <= '0';
                if clk_count = 15 then
                        if bit_count = 0 then
                                state <= sack21; -- finished all 8 bits, wait for
                                ack
```

```
                                        bit_count <= 7;
                                else
                                        state <= sw1b1; -- write next bit
                                        bit_count <= bit_count - 1;
                                end if; -- if bit_count
                        end if; -- if clk_count
                when sack21 =>
                        clk_count <= clk_count + 1;
                        I2C_SDAT <= 'Z'; -- float the tristate data line
                        I2C_SCLK <= '0';
                        if clk_count = 31 then
                                state <= sack22;
                                clk_count <= 0;
                        end if; -- if clk_count
                when sack22 =>
                        clk_count <= clk_count + 1;
                        I2C_SDAT <= 'Z';
                        I2C_SCLK <= '1';
                        if clk_count = 31 then
                                state <= sack23;
                                clk_count <= 0;
                        end if; -- if clk_count
                when sack23 => -- last phase of acknowledge cycle
                        clk_count <= clk_count + 1;
                        I2C_SDAT <= 'Z';
                        I2C_SCLK <= '0';
                        if clk_count = 15 then
                                state <= sw2b1;
                        end if; -- if clk_count
                when sw2b1 => -- send 8 bits of second word, MSB first, first phase
                        clk_count <= clk_count + 1;
                        I2C_SDAT <= SCI_WORD2(bit_count);
                        I2C_SCLK <= '0';
                        if clk_count = 31 then
                                state <= sw2b2;
                        end if; -- if clk_count
                when sw2b2 => -- send 8 bits, MSB first, second phase
                        clk_count <= clk_count + 1;
                        I2C_SDAT <= SCI_WORD2(bit_count);
                        I2C_SCLK <= '1';
                        if clk_count = 63 then
                                state <= sw2b3;
                                clk_count <= 0;
                        end if; -- if clk_count
                when sw2b3 => -- send 8 address bits, MSB first, third phase
                        clk_count <= clk_count + 1;
                        I2C_SDAT <= SCI_WORD2(bit_count);
                        I2C_SCLK <= '0';
                        if clk_count = 15 then
                                if bit_count = 0 then
                                        state <= sack31; -- finished all 8 bits, wait for
ack
                                        bit_count <= 7;
                                else
                                        state <= sw2b1; -- write next bit
                                        bit_count <= bit_count - 1;
                                end if; -- if bit_count
```

```
        end if; -- if clk_count
                        when sack31 =>
                                clk_count <= clk_count + 1;
                                I2C_SDAT <= 'Z'; -- float the tristate data line
                                I2C_SCLK <= '0';
                                if clk_count = 31 then
                                        state <= sack32;
                                        clk_count <= 0;
                                end if; -- if clk_count
                        when sack32 =>
                                clk_count <= clk_count + 1;
                                I2C_SDAT <= 'Z';
                                I2C_SCLK <= '1';
                                if clk_count = 31 then
                                        state <= sack33;
                                        clk_count <= 0;
                                end if; -- if clk_count
                        when sack33 => -- last phase of acknowledge cycle
                                clk_count <= clk_count + 1;
                                I2C_SDAT <= 'Z';
                                I2C_SCLK <= '0';
                                if clk_count = 15 then
                                        state <= send;
                                end if; -- if clk_count
                        when send => -- last step, raise SCLK, keeping SDAT low for 16 cycles
                                clk_count <= clk_count + 1;
                                I2C_SDAT <= '0';
                                I2C_SCLK <= '1';
                                if clk_count = 31 then
                                        SCI_READY <= '1';
                                        state <= sw_init0; -- go back to start, wait for new write
command
                                end if; -- if clk_count
                        end case;
                end if; -- if rst
        end process;

end a;
```

LED

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

entity LED is
   Port ( entrada : in  STD_LOGIC_VECTOR (4 downto 0);
          salida : out  STD_LOGIC_VECTOR (17 downto 0));
end LED;

architecture Behavioral of LED is

begin
```

```
with entrada select
        salida <= "0000000000000000" when "00000",  -- 0
                  "0000000000000001" when "00001",  -- 1
                  "0000000000000011" when "00010",  -- 2
                  "0000000000000111" when "00011",  -- 3
                  "0000000000001111" when "00100",  -- 4
                  "0000000000011111" when "00101",  -- 5
                  "0000000000111111" when "00110",  -- 6
                  "0000000001111111" when "00111",  -- 7
                  "0000000011111111" when "01000",  -- 8
                  "0000000111111111" when "01001",  -- 9
                  "0000001111111111" when "01010",  -- 10
                  "0000011111111111" when "01011",  -- 11
                  "0000111111111111" when "01100",  -- 12
                  "0001111111111111" when "01101",  -- 13
                  "0011111111111111" when "01110",  -- 14
                  "0111111111111111" when "01111",  -- 15
                  "1111111111111111" when "10000",  -- 16
                  "0111111111111111" when "10001",  -- 17
                  "1111111111111111" when "10010",  -- 18
                  "1010101010101010" when others;

end Behavioral;
```

Fase

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

entity fase is
   Port (                       CLK : in  STD_LOGIC;                    -- 48000Hz
                                num_nota : in STD_LOGIC_VECTOR (4 downto 0);
                                num_inst : in STD_LOGIC_VECTOR (1 downto 0);
                                f1 : out STD_LOGIC_VECTOR (8 downto 0);
                                f2 : out STD_LOGIC_VECTOR (8 downto 0);
                                f3 : out STD_LOGIC_VECTOR (8 downto 0);
                                f4 : out STD_LOGIC_VECTOR (8 downto 0);
                                f5 : out STD_LOGIC_VECTOR (8 downto 0);
                                f6 : out STD_LOGIC_VECTOR (8 downto 0);
                                f7 : out STD_LOGIC_VECTOR (8 downto 0);
                                f8 : out STD_LOGIC_VECTOR (8 downto 0);
                                f9 : out STD_LOGIC_VECTOR (8 downto 0);
                                f10 : out STD_LOGIC_VECTOR (8 downto 0);
                                f11 : out STD_LOGIC_VECTOR (8 downto 0);
                                f12 : out STD_LOGIC_VECTOR (8 downto 0);
                                f13 : out STD_LOGIC_VECTOR (8 downto 0);
                                f14 : out STD_LOGIC_VECTOR (8 downto 0);
                                f15 : out STD_LOGIC_VECTOR (8 downto 0);
                                f16 : out STD_LOGIC_VECTOR (8 downto 0);
                                f17 : out STD_LOGIC_VECTOR (8 downto 0);
                                f18 : out STD_LOGIC_VECTOR (8 downto 0);
                                f19 : out STD_LOGIC_VECTOR (8 downto 0)
                                );
end fase;
```

```
architecture Behavioral of fase is

-----------------------------------------ROMs-----------------------------------------------
---------- OBOE

        type ROMf1_1 is array (0 to 18) of integer;          -- ROM fases del primer armonico del
                                                             instrumento 1 (oboe) para cada
                                                             frecuencia
        constant MEMf1_1:
        ROMf1_1:=(0,0,357,496,13,507,356,44,185,384,268,153,60,191,25,2,492,0,0);

        type ROMf2_1 is array (0 to 18) of integer;          -- ROM fases del armonico 2 inst1
        constant MEMf2_1:
        ROMf2_1:=(0,0,395,191,233,237,404,304,142,34,373,24,351,95,253,166,154,0,0);

        type ROMf3_1 is array (0 to 18) of integer;          -- ROM fases del armonico 3 inst1
        constant MEMf3_1:
        ROMf3_1:=(0,0,347,319,402,368,444,67,83,175,351,350,181,248,504,113,162,0,0);

        type ROMf4_1 is array (0 to 18) of integer;          -- ROM fases del armonico 4 inst1
        constant MEMf4_1:
        ROMf4_1:=(0,0,394,61,135,100,345,94,31,439,498,262,420,12,400,29,5,0,0);

        type ROMf5_1 is array (0 to 18) of integer;          -- ROM fases del armonico 5 inst1
        constant MEMf5_1:
        ROMf5_1:=(0,0,376,240,473,118,264,236,438,174,204,330,121,368,462,225,128,0,0);

        type ROMf6_1 is array (0 to 18) of integer;          -- ROM fases del armonico 6 inst1
        constant MEMf6_1:
        ROMf6_1:=(0,0,272,482,149,81,276,275,246,66,286,279,299,493,452,289,158,0,0);

        type ROMf7_1 is array (0 to 18) of integer;          -- ROM fases del armonico 7 inst1
        constant MEMf7_1:
        ROMf7_1:=(0,0,216,248,318,211,264,126,178,126,342,12,415,106,71,476,6,0,0);

        type ROMf8_1 is array (0 to 18) of integer;          -- ROM fases del armonico 8 inst1
        constant MEMf8_1:
        ROMf8_1:=(0,0,468,73,435,51,418,320,484,242,58,197,97,452,237,413,135,0,0);

        type ROMf9_1 is array (0 to 18) of integer;          -- ROM fases del armonico 9 inst1
        constant MEMf9_1:
        ROMf9_1:=(0,0,474,271,260,237,498,100,265,110,40,85,263,239,301,438,277,0,0);

        type ROMf10_1 is array (0 to 18) of integer;         -- ROM fases del armonico 10 inst1
        constant MEMf10_1:
        ROMf10_1:=(0,0,499,478,30,483,352,267,159,298,418,250,391,494,386,108,286,0,0);

        type ROMf11_1 is array (0 to 18) of integer;         -- ROM fases del armonico 11 inst1
        constant MEMf11_1:
        ROMf11_1:=(0,0,97,129,149,125,332,79,477,21,485,6,63,129,286,136,280,0,0);

        type ROMf12_1 is array (0 to 18) of integer;         -- ROM fases del armonico 12 inst1
        constant MEMf12_1:
        ROMf12_1:=(0,0,84,301,340,291,79,326,356,79,7,242,312,414,316,333,167,0,0);

        type ROMf13_1 is array (0 to 18) of integer;         -- ROM fases del armonico 13 inst1
        constant MEMf13_1:
        ROMf13_1:=(0,0,41,472,482,483,98,22,445,30,334,451,287,188,175,509,102,0,0);
```

```
        type ROMf15_1 is array (0 to 18) of integer;          -- ROM fases del armonico 15 inst1
        constant MEMf15_1:
        ROMf15_1:=(0,0,21,310,121,23,33,353,53,391,434,167,130,7,434,420,375,0,0);
        type ROMf16_1 is array (0 to 18) of integer;          -- ROM fases del armonico 16 inst1
        constant MEMf16_1:
        ROMf16_1:=(0,0,482,445,205,371,102,384,237,69,180,157,417,174,348,164,286,0,0);
        type ROMf17_1 is array (0 to 18) of integer;          -- ROM fases del armonico 17 inst1
        constant MEMf17_1:
        ROMf17_1:=(0,0,403,69,414,227,308,424,274,278,433,339,85,116,329,141,0,0,0);
        type ROMf18_1 is array (0 to 18) of integer;          -- ROM fases del armonico 18 inst1
        constant MEMf18_1:
        ROMf18_1:=(0,0,485,277,84,195,175,246,278,205,331,266,488,36,99,0,0,0,0);
        type ROMf19_1 is array (0 to 18) of integer;          -- ROM fases del armonico 19 inst1
        constant MEMf19_1:
        ROMf19_1:=(0,0,411,69,128,24,215,107,31,216,292,92,252,225,510,0,0,0,0);

----------VIOLIN
        type ROMf1_2 is array (0 to 18) of integer;
        constant MEMf1_2:
        ROMf1_2:=(0,0,307,130,195,348,388,404,407,130,401,383,399,137,388,114,120,0,0);
        type ROMf2_2 is array (0 to 18) of integer;
        constant MEMf2_2:
        ROMf2_2:=(0,0,422,498,29,384,342,292,360,509,241,427,208,55,187,227,187,0,0);
        type ROMf3_2 is array (0 to 18) of integer;
        constant MEMf3_2:
        ROMf3_2:=(0,0,306,333,401,412,374,316,263,248,344,216,49,467,292,76,323,0,0);
                type ROMf4_2 is array (0 to 18) of integer;
                constant MEMf4_2:
        ROMf4_2:=(0,0,510,433,231,130,413,405,33,136,147,96,289,44,451,26,40,0,0);

        type ROMf5_2 is array (0 to 18) of integer;
        constant MEMf5_2:
        ROMf5_2:=(0,0,133,126,18,205,217,156,410,11,112,181,335,412,268,103,106,0,0);
        type ROMf6_2 is array (0 to 18) of integer;
        constant MEMf6_2:
        ROMf6_2:=(0,0,279,491,105,246,234,87,99,339,386,132,177,198,284,289,37,0,0);
        type ROMf7_2 is array (0 to 18) of integer;
        constant MEMf7_2:
        ROMf7_2:=(0,0,452,210,194,349,35,235,507,461,84,316,459,142,314,102,89,0,0);
        type ROMf8_2 is array (0 to 18) of integer;
        constant MEMf8_2:
        ROMf8_2:=(0,0,246,214,186,197,430,419,9,35,356,367,467,315,449,5,274,0,0);
        type ROMf9_2 is array (0 to 18) of integer;
        constant MEMf9_2:
        ROMf9_2:=(0,0,287,413,114,405,144,412,408,261,509,32,325,335,399,0,0,0,0);
        type ROMf10_2 is array (0 to 18) of integer;
        constant MEMf10_2:
        ROMf10_2:=(0,0,361,98,43,352,311,306,464,487,171,330,5,489,0,0,0,0,0);
        type ROMf11_2 is array (0 to 18) of integer;
        constant MEMf11_2:
        ROMf11_2:=(0,0,349,462,25,477,268,2,344,242,0,341,0,0,0,0,0,0,0);
        type ROMf12_2 is array (0 to 18) of integer;
        constant MEMf12_2:
        ROMf12_2:=(0,0,24,408,421,288,269,432,27,265,56,0,0,0,0,0,0,0,0);
        type ROMf13_2 is array (0 to 18) of integer;
        constant MEMf13_2:
        ROMf13_2:=(0,0,377,373,221,493,9,340,379,0,0,0,0,0,0,0,0,0,0);
        type ROMf14_2 is array (0 to 18) of integer;
        constant MEMf14  2:
```

```
type ROMf14_2 is array (0 to 18) of integer;
constant MEMf14_2: ROMf14_2:=(0,0,296,36,102,492,510,258,0,0,0,0,0,0,0,0,0,0);
type ROMf15_2 is array (0 to 18) of integer;
constant MEMf15_2: ROMf15_2:=(0,0,346,449,66,273,342,0,0,0,0,0,0,0,0,0,0,0);
type ROMf16_2 is array (0 to 18) of integer;
constant MEMf16_2: ROMf16_2:=(0,0,450,204,52,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROMf17_2 is array (0 to 18) of integer;
constant MEMf17_2: ROMf17_2:=(0,0,390,348,29,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROMf18_2 is array (0 to 18) of integer;
constant MEMf18_2: ROMf18_2:=(0,0,108,236,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROMf19_2 is array (0 to 18) of integer;
constant MEMf19_2: ROMf19_2:=(0,0,292,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
---------- TUBA
type ROMf1_3 is array (0 to 18) of integer;
constant MEMf1_3:
ROMf1_3:=(0,0,319,324,78,442,439,449,443,114,415,413,94,400,114,125,376,0,0);
type ROMf2_3 is array (0 to 18) of integer;
constant MEMf2_3:
ROMf2_3:=(0,0,25,65,114,368,385,383,331,113,302,291,199,277,186,133,52,0,0);
type ROMf3_3 is array (0 to 18) of integer;
constant MEMf3_3:
ROMf3_3:=(0,0,285,412,220,311,325,304,217,136,141,81,135,19,148,36,486,0,0);
type ROMf4_3 is array (0 to 18) of integer;
constant MEMf4_3:
ROMf4_3:=(0,0,26,103,103,428,295,183,22,133,110,358,108,330,129,446,161,0,0);
type ROMf5_3 is array (0 to 18) of integer;
constant MEMf5_3:
ROMf5_3:=(0,0,378,395,117,422,208,181,448,143,131,169,102,94,220,337,355,0,0);
type ROMf6_3 is array (0 to 18) of integer;
constant MEMf6_3:
ROMf6_3:=(0,0,452,270,193,321,431,355,354,283,62,345,53,41,424,511,302,0,0);
type ROMf7_3 is array (0 to 18) of integer;
constant MEMf7_3:
ROMf7_3:=(0,0,184,311,29,411,412,281,192,304,418,169,207,263,80,292,99,0,0);
type ROMf8_3 is array (0 to 18) of integer;
constant MEMf8_3:
ROMf8_3:=(0,0,247,184,202,49,243,203,22,309,310,472,126,243,60,404,320,0,0);
type ROMf9_3 is array (0 to 18) of integer;
constant MEMf9_3:
ROMf9_3:=(0,0,142,340,204,504,153,105,416,352,226,309,272,206,329,286,441,0,0);
type ROMf10_3 is array (0 to 18) of integer;
constant MEMf10_3:
ROMf10_3:=(0,0,442,137,183,381,74,85,344,319,119,171,485,102,495,421,382,0,0);
type ROMf11_3 is array (0 to 18) of integer;
constant MEMf11_3:
ROMf11_3:=(0,0,161,401,282,286,59,376,79,472,76,20,249,46,38,193,51,0,0);
type ROMf12_3 is array (0 to 18) of integer;
constant MEMf12_3:
ROMf12_3:=(0,0,385,184,297,314,87,359,470,50,465,400,310,34,0,429,488,0,0);
type ROMf13_3 is array (0 to 18) of integer;
constant MEMf13_3:
ROMf13_3:=(0,0,102,11,264,223,295,382,278,285,503,19,252,60,184,301,10,0,0);
type ROMf14_3 is array (0 to 18) of integer;
constant MEMf14_3:
ROMf14_3:=(0,0,420,183,375,95,252,236,142,275,164,21,288,98,343,137,495,0,0);
type ROMf15_3 is array (0 to 18) of integer;
constant MEMf15_3:
ROMf15_3:=(0,0,438,125,466,509,221,466,466,288,470,424,291,495,408,338,13,0,0);
```

```
        type ROMf16_3 is array (0 to 18) of integer;
        constant MEMf16_3:
        ROMf16_3:=(0,0,41,415,413,466,280,410,202,335,3,482,264,507,309,340,436,0,0);
        type ROMf17_3 is array (0 to 18) of integer;
        constant MEMf17_3:
        ROMf17_3:=(0,0,348,91,489,417,242,143,79,330,352,152,487,382,95,322,276,0,0);
        type ROMf18_3 is array (0 to 18) of integer;
        constant MEMf18_3:
        ROMf18_3:=(0,0,391,173,133,140,127,132,495,341,214,380,140,507,214,156,188,0,0);
        type ROMf19_3 is array (0 to 18) of integer;
        constant MEMf19_3:
        ROMf19_3:=(0,0,460,271,45,200,11,36,217,406,213,254,305,433,169,319,1,0,0);

-------------------------------------------Signals-----------------------------------------------
signal fasefe : integer:=0;
signal fase1e : integer:=0;
signal fase2e : integer:=0;
signal fase3e : integer:=0;
signal fase4e : integer:=0;
signal fase5e : integer:=0;
signal fase6e : integer:=0;
signal fase7e : integer:=0;
signal fase8e : integer:=0;
signal fase9e : integer:=0;
signal fase10e : integer:=0;
signal fase11e : integer:=0;
signal fase12e : integer:=0;
signal fase13e : integer:=0;
signal fase14e : integer:=0;
signal fase15e : integer:=0;
signal fase16e : integer:=0;
signal fase17e : integer:=0;
signal fase18e : integer:=0;
signal fase19e : integer:=0;
signal fase1 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase2 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase3 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase4 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase5 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase6 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase7 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase8 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase9 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase10 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase11 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase12 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase13 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase14 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase15 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase16 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase17 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase18 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal fase19 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');

begin

Process (CLK)
begin
        if (CLK'event and CLK='1') then
                if
```

```
                    ----------OBOE----------
        (num_inst = "01") then
                    fase1e <= MEMf1_1(CONV_INTEGER(num_nota));
                        --guardar en fase1e el valor que tenemos en la memoria
                    fase1 <= conv_std_logic_vector(fase1e,9);
                        -- convertir el valor que tenemos en la memoria
                    fase2e <= MEMf2_1(CONV_INTEGER(num_nota));
                    fase2 <= conv_std_logic_vector(fase2e,9);
                    fase3e <= MEMf3_1(CONV_INTEGER(num_nota));
                    fase3 <= conv_std_logic_vector(fase3e,9);
                    fase4e <= MEMf4_1(CONV_INTEGER(num_nota));
                    fase4 <= conv_std_logic_vector(fase4e,9);
                    fase5e <= MEMf5_1(CONV_INTEGER(num_nota));
                    fase5 <= conv_std_logic_vector(fase5e,9);
                    fase6e <= MEMf6_1(CONV_INTEGER(num_nota));
                    fase6 <= conv_std_logic_vector(fase6e,9);
                    fase7e <= MEMf7_1(CONV_INTEGER(num_nota));
                    fase7 <= conv_std_logic_vector(fase7e,9);
                    fase8e <= MEMf8_1(CONV_INTEGER(num_nota));
                    fase8 <= conv_std_logic_vector(fase8e,9);
                    fase9e <= MEMf9_1(CONV_INTEGER(num_nota));
                    fase9 <= conv_std_logic_vector(fase9e,9);
                    fase10e <= MEMf10_1(CONV_INTEGER(num_nota));
                    fase10 <= conv_std_logic_vector(fase10e,9);
                    fase11e <= MEMf11_1(CONV_INTEGER(num_nota));
                    fase11 <= conv_std_logic_vector(fase11e,9);
                    fase12e <= MEMf12_1(CONV_INTEGER(num_nota));
                    fase12 <= conv_std_logic_vector(fase12e,9);
                    fase13e <= MEMf13_1(CONV_INTEGER(num_nota));
                    fase13 <= conv_std_logic_vector(fase13e,9);
                    fase14e <= MEMf14_1(CONV_INTEGER(num_nota));
                    fase14 <= conv_std_logic_vector(fase14e,9);
                    fase15e <= MEMf15_1(CONV_INTEGER(num_nota));
                    fase15 <= conv_std_logic_vector(fase15e,9);
                    fase16e <= MEMf16_1(CONV_INTEGER(num_nota));
                    fase16 <= conv_std_logic_vector(fase16e,9);
                    fase17e <= MEMf17_1(CONV_INTEGER(num_nota));
                    fase17 <= conv_std_logic_vector(fase17e,9);
                    fase18e <= MEMf18_1(CONV_INTEGER(num_nota));
                    fase18 <= conv_std_logic_vector(fase18e,9);
                    fase19e <= MEMf19_1(CONV_INTEGER(num_nota));
                    fase19 <= conv_std_logic_vector(fase19e,9);
                    ----------VIOLIN----------
elsif
        (num_inst = "10") then

                    fase1e <= MEMf1_2(CONV_INTEGER(num_nota));
                    --guardar en fase1e el valor que tenemos en la memoria
                    fase1 <= conv_std_logic_vector(fase1e,9);
                    -- convertir el valor que tenemos en la memoria
                    fase2e <= MEMf2_2(CONV_INTEGER(num_nota));
                    fase2 <= conv_std_logic_vector(fase2e,9);
                    fase3e <= MEMf3_2(CONV_INTEGER(num_nota));
                    fase3 <= conv_std_logic_vector(fase3e,9);
                    fase4e <= MEMf4_2(CONV_INTEGER(num_nota));
                    fase4 <= conv_std_logic_vector(fase4e,9);
                    fase5e <= MEMf5_2(CONV_INTEGER(num_nota));
                    fase5 <= conv_std_logic_vector(fase5e,9);
                    fase6e <= MEMf6_2(CONV_INTEGER(num_nota));
                    fase6 <= conv_std_logic_vector(fase6e,9);
```

```
                fase7e <= MEMf7_2(CONV_INTEGER(num_nota));
                fase7 <= conv_std_logic_vector(fase7e,9);
                fase8e <= MEMf8_2(CONV_INTEGER(num_nota));
                fase8 <= conv_std_logic_vector(fase8e,9);
                fase9e <= MEMf9_2(CONV_INTEGER(num_nota));
                fase9 <= conv_std_logic_vector(fase9e,9);
                fase10e <= MEMf10_2(CONV_INTEGER(num_nota));
                fase10 <= conv_std_logic_vector(fase10e,9);
                fase11e <= MEMf11_2(CONV_INTEGER(num_nota));
                fase11 <= conv_std_logic_vector(fase11e,9);
                fase12e <= MEMf12_2(CONV_INTEGER(num_nota));
                fase12 <= conv_std_logic_vector(fase12e,9);
                fase13e <= MEMf13_2(CONV_INTEGER(num_nota));
                fase13 <= conv_std_logic_vector(fase13e,9);
                fase14e <= MEMf14_2(CONV_INTEGER(num_nota));
                fase14 <= conv_std_logic_vector(fase14e,9);
                fase15e <= MEMf15_2(CONV_INTEGER(num_nota));
                fase15 <= conv_std_logic_vector(fase15e,9);
                fase16e <= MEMf16_2(CONV_INTEGER(num_nota));
                fase16 <= conv_std_logic_vector(fase16e,9);
                fase17e <= MEMf17_2(CONV_INTEGER(num_nota));
                fase17 <= conv_std_logic_vector(fase17e,9);
                fase18e <= MEMf18_2(CONV_INTEGER(num_nota));
                fase18 <= conv_std_logic_vector(fase18e,9);
                fase19e <= MEMf19_2(CONV_INTEGER(num_nota));
                fase19 <= conv_std_logic_vector(fase19e,9);
                ---------------------------------TUBA-------------------------------------
elsif
        (num_inst = "11") then
                fase1e <= MEMf1_3(CONV_INTEGER(num_nota));
                --guardar en fase1e el valor que tenemos en la memoria
                fase1 <= conv_std_logic_vector(fase1e,9);
                -- convertir el valor que tenemos en la memoria
                fase2e <= MEMf2_3(CONV_INTEGER(num_nota));
                fase2 <= conv_std_logic_vector(fase2e,9);
                fase3e <= MEMf3_3(CONV_INTEGER(num_nota));
                fase3 <= conv_std_logic_vector(fase3e,9);
                fase4e <= MEMf4_3(CONV_INTEGER(num_nota));
                fase4 <= conv_std_logic_vector(fase4e,9);
                fase5e <= MEMf5_3(CONV_INTEGER(num_nota));
                fase5 <= conv_std_logic_vector(fase5e,9);
                fase6e <= MEMf6_3(CONV_INTEGER(num_nota));
                fase6 <= conv_std_logic_vector(fase6e,9);
                fase7e <= MEMf7_3(CONV_INTEGER(num_nota));
                fase7 <= conv_std_logic_vector(fase7e,9);
                fase8e <= MEMf8_3(CONV_INTEGER(num_nota));
                fase8 <= conv_std_logic_vector(fase8e,9);
                fase9e <= MEMf9_3(CONV_INTEGER(num_nota));
                fase9 <= conv_std_logic_vector(fase9e,9);
                fase10e <= MEMf10_3(CONV_INTEGER(num_nota));
                fase10 <= conv_std_logic_vector(fase10e,9);
                fase11e <= MEMf11_3(CONV_INTEGER(num_nota));
                fase11 <= conv_std_logic_vector(fase11e,9);
                fase12e <= MEMf12_3(CONV_INTEGER(num_nota));
                fase12 <= conv_std_logic_vector(fase12e,9);
                fase13e <= MEMf13_3(CONV_INTEGER(num_nota));
                fase13 <= conv_std_logic_vector(fase13e,9);
                fase14e <= MEMf14_3(CONV_INTEGER(num_nota));
                fase14 <= conv_std_logic_vector(fase14e,9);
                fase15e <= MEMf15_3(CONV_INTEGER(num_nota));
```

```
                                    fase15 <= conv_std_logic_vector(fase15e,9);
                                    fase16e <= MEMf16_3(CONV_INTEGER(num_nota));
                                    fase16 <= conv_std_logic_vector(fase16e,9);
                                    fase17e <= MEMf17_3(CONV_INTEGER(num_nota));
                                    fase17 <= conv_std_logic_vector(fase17e,9);
                                    fase18e <= MEMf18_3(CONV_INTEGER(num_nota));
                                    fase18 <= conv_std_logic_vector(fase18e,9);
                                    fase19e <= MEMf19_3(CONV_INTEGER(num_nota));
                                    fase19 <= conv_std_logic_vector(fase19e,9);
                        end if;
                end if;
end Process;
f1 <= fase1;
f2 <= fase2;
f3 <= fase3;
f4 <= fase4;
f5 <= fase5;
f6 <= fase6;
f7 <= fase7;
f8 <= fase8;
f9 <= fase9;
f10 <= fase10;
f11 <= fase11;
f12 <= fase12;
f13 <= fase13;
f14 <= fase14;
f15 <= fase15;
f16 <= fase16;
f17 <= fase17;
f18 <= fase18;
f19 <= fase19;
end Behavioral;
```

Amplitud

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

entity amplitud is
    Port (          CLK : in  STD_LOGIC;                            -- 48000Hz
                    num_nota : in STD_LOGIC_VECTOR (4 downto 0);
                    num_inst : in STD_LOGIC_VECTOR (1 downto 0);
                    af : out STD_LOGIC_VECTOR (7 downto 0);
                    a1 : out STD_LOGIC_VECTOR (7 downto 0);
                    a2 : out STD_LOGIC_VECTOR (7 downto 0);
                    a3 : out STD_LOGIC_VECTOR (7 downto 0);
                    a4 : out STD_LOGIC_VECTOR (7 downto 0);
                    a5 : out STD_LOGIC_VECTOR (7 downto 0);
                    a6 : out STD_LOGIC_VECTOR (7 downto 0);
                    a7 : out STD_LOGIC_VECTOR (7 downto 0);
                    a8 : out STD_LOGIC_VECTOR (7 downto 0);
                    a9 : out STD_LOGIC_VECTOR (7 downto 0);
                    a10 : out STD_LOGIC_VECTOR (7 downto 0);
                    a11 : out STD_LOGIC_VECTOR (7 downto 0);
                    a12 : out STD_LOGIC_VECTOR (7 downto 0);
```

```
                    a13 : out STD_LOGIC_VECTOR (7 downto 0);
                    a14 : out STD_LOGIC_VECTOR (7 downto 0);
                    a15 : out STD_LOGIC_VECTOR (7 downto 0);
                    a16 : out STD_LOGIC_VECTOR (7 downto 0);
                    a17 : out STD_LOGIC_VECTOR (7 downto 0);
                    a18 : out STD_LOGIC_VECTOR (7 downto 0);
                    a19 : out STD_LOGIC_VECTOR (7 downto 0)
                    );
end amplitud;

architecture Behavioral of amplitud is
-----------------------------------------------ROMs-----------------------------------------------
------------ OBOE
type ROM_a_1 is array (0 to 18) of integer;             -- ROM amplitud frecuencia fundamental inst1
constant MEM_a_1:
ROM_a_1:=(0,0,155,138,155,150,154,152,104,116,136,110,152,100,138,129,119,0,0);
type ROM_a1_1 is array (0 to 18) of integer;            -- ROM amplitud arm 1 inst1
constant MEM_a1_1: ROM_a1_1:=(0,0,45,62,52,61,35,55,61,34,36,26,40,34,20,111,76,0,0);
type ROM_a2_1 is array (0 to 18) of integer;            -- ROM amplitud arm 2 inst1
constant MEM_a2_1: ROM_a2_1:=(0,0,11,68,67,74,60,57,82,55,64,96,78,76,52,137,111,0,0);
type ROM_a3_1 is array (0 to 18) of integer;            -- ROM amplitud arm 3 inst1
constant MEM_a3_1: ROM_a3_1:=(0,0,41,46,46,67,42,61,85,84,115,48,21,6,9,23,8,0,0);
type ROM_a4_1 is array (0 to 18) of integer;            -- ROM amplitud arm 4 inst1
constant MEM_a4_1: ROM_a4_1:=(0,0,77,48,90,97,79,34,71,39,97,48,14,6,24,12,5,0,0);
type ROM_a5_1 is array (0 to 18) of integer;            -- ROM amplitud arm 5 inst1
constant MEM_a5_1: ROM_a5_1:=(0,0,83,36,25,13,21,30,18,14,15,8,6,4,4,31,24,0,0);
type ROM_a6_1 is array (0 to 18) of integer;            -- ROM amplitud arm 6 inst1
constant MEM_a6_1: ROM_a6_1:=(0,0,39,61,11,2,1,4,9,4,24,18,4,18,24,16,9,0,0);
type ROM_a7_1 is array (0 to 18) of integer;            -- ROM amplitud arm 7 inst1
constant MEM_a7_1: ROM_a7_1:=(0,0,4,20,1,3,1,7,8,19,5,6,18,6,1,4,1,0,0);
type ROM_a8_1 is array (0 to 18) of integer;            -- ROM amplitud arm 8 inst1
constant MEM_a8_1: ROM_a8_1:=(0,0,6,17,2,12,4,5,1,7,9,13,7,4,4,1,2,0,0);
type ROM_a9_1 is array (0 to 18) of integer;            -- ROM amplitud arm 9 inst1
constant MEM_a9_1: ROM_a9_1:=(0,0,11,21,9,4,6,3,6,2,3,2,2,5,2,2,1,0,0);
type ROM_a10_1 is array (0 to 18) of integer;           -- ROM amplitud arm 10 inst1
constant MEM_a10_1: ROM_a10_1:=(0,0,8,8,13,11,1,3,1,1,0,2,1,0,1,1,2,0,0);
type ROM_a11_1 is array (0 to 18) of integer;           -- ROM amplitud arm 11 inst1
constant MEM_a11_1: ROM_a11_1:=(0,0,2,11,8,4,0,0,0,1,1,5,0,1,0,1,0,0,0);
type ROM_a12_1 is array (0 to 18) of integer;           -- ROM amplitud arm 12 inst1
constant MEM_a12_1: ROM_a12_1:=(0,0,7,13,3,1,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a13_1 is array (0 to 18) of integer;           -- ROM amplitud arm 13 inst1
constant MEM_a13_1: ROM_a13_1:=(0,0,4,8,2,0,0,0,0,0,1,0,0,0,0,0,0,0,0);
type ROM_a14_1 is array (0 to 18) of integer;           -- ROM amplitud arm 14 inst1
constant MEM_a14_1: ROM_a14_1:=(0,0,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a15_1 is array (0 to 18) of integer;           -- ROM amplitud arm 15 inst1
constant MEM_a15_1: ROM_a15_1:=(0,0,0,5,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0);
type ROM_a16_1 is array (0 to 18) of integer;           -- ROM amplitud arm 16 inst1
constant MEM_a16_1: ROM_a16_1:=(0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a17_1 is array (0 to 18) of integer;           -- ROM amplitud arm 17 inst1
constant MEM_a17_1: ROM_a17_1:=(0,0,1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a18_1 is array (0 to 18) of integer;           -- ROM amplitud arm 18 inst1
constant MEM_a18_1: ROM_a18_1:=(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a19_1 is array (0 to 18) of integer;           -- ROM amplitud arm 19 inst1
constant MEM_a19_1: ROM_a19_1:=(0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
------------VIOLIN
type ROM_a_2 is array (0 to 18) of integer;
constant MEM_a_2: ROM_a_2:=(0,0,81,155,100,136,92,141,80,97,90,151,145,138,130,86,128,0,0);
type ROM_a1_2 is array (0 to 18) of integer;
constant MEM_a1_2: ROM_a1_2:=(0,0,37,55,24,31,17,57,43,32,31,29,45,30,30,25,70,0,0);
```

```
type ROM_a2_2 is array (0 to 18) of integer;
constant MEM_a2_2: ROM_a2_2:=(0,0,40,8,28,61,7,19,40,11,8,9,16,13,7,3,0,0,0);
type ROM_a3_2 is array (0 to 18) of integer;
constant MEM_a3_2: ROM_a3_2:=(0,0,12,1,6,5,3,9,9,3,4,5,1,3,1,7,3,0,0);
type ROM_a4_2 is array (0 to 18) of integer;
constant MEM_a4_2: ROM_a4_2:=(0,0,12,2,5,2,0,6,3,4,2,1,1,0,0,7,2,0,0);
type ROM_a5_2 is array (0 to 18) of integer;
constant MEM_a5_2: ROM_a5_2:=(0,0,2,2,1,1,0,1,5,3,0,2,3,1,2,0,0,0,0);
type ROM_a6_2 is array (0 to 18) of integer;
constant MEM_a6_2: ROM_a6_2:=(0,0,3,1,3,4,1,1,2,1,1,0,0,0,1,2,2,0,0);
type ROM_a7_2 is array (0 to 18) of integer;
constant MEM_a7_2: ROM_a7_2:=(0,0,0,3,1,1,1,1,0,1,0,1,0,2,0,1,0,0,0);
type ROM_a8_2 is array (0 to 18) of integer;
constant MEM_a8_2: ROM_a8_2:=(0,0,0,1,1,0,1,1,0,0,0,1,1,1,0,2,0,0,0);
type ROM_a9_2 is array (0 to 18) of integer;
constant MEM_a9_2: ROM_a9_2:=(0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,0,0,0,0);
type ROM_a10_2 is array (0 to 18) of integer;
constant MEM_a10_2: ROM_a10_2:=(0,0,0,0,0,0,1,0,1,1,0,2,0,0,0,0,0,0,0);
type ROM_a11_2 is array (0 to 18) of integer;
constant MEM_a11_2: ROM_a11_2:=(0,0,0,0,0,0,0,0,3,1,0,0,0,0,0,0,0,0,0);
type ROM_a12_2 is array (0 to 18) of integer;
constant MEM_a12_2: ROM_a12_2:=(0,0,1,0,0,0,1,1,0,1,0,0,0,0,0,0,0,0,0);
type ROM_a13_2 is array (0 to 18) of integer;
constant MEM_a13_2: ROM_a13_2:=(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a14_2 is array (0 to 18) of integer;
constant MEM_a14_2: ROM_a14_2:=(0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a15_2 is array (0 to 18) of integer;
constant MEM_a15_2: ROM_a15_2:=(0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a16_2 is array (0 to 18) of integer;
constant MEM_a16_2: ROM_a16_2:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a17_2 is array (0 to 18) of integer;
constant MEM_a17_2: ROM_a17_2:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a18_2 is array (0 to 18) of integer;
constant MEM_a18_2: ROM_a18_2:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a19_2 is array (0 to 18) of integer;
constant MEM_a19_2: ROM_a19_2:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
---------- TUBA
type ROM_a_3 is array (0 to 18) of integer;
constant MEM_a_3:
ROM_a_3:=(0,0,155,134,112,63,72,120,86,107,146,96,138,131,106,108,105,0,0);
type ROM_a1_3 is array (0 to 18) of integer;
constant MEM_a1_3: ROM_a1_3:=(0,0,45,36,41,41,27,31,55,23,51,93,82,33,31,26,35,0,0);
type ROM_a2_3 is array (0 to 18) of integer;
constant MEM_a2_3: ROM_a2_3:=(0,0,54,23,37,33,53,40,51,42,24,40,33,9,3,10,4,0,0);
type ROM_a3_3 is array (0 to 18) of integer;
constant MEM_a3_3: ROM_a3_3:=(0,0,31,32,18,30,22,14,21,7,1,9,7,2,2,4,1,0,0);
type ROM_a4_3 is array (0 to 18) of integer;
constant MEM_a4_3: ROM_a4_3:=(0,0,12,13,5,1,5,7,10,8,1,8,3,1,0,0,1,0,0);
type ROM_a5_3 is array (0 to 18) of integer;
constant MEM_a5_3: ROM_a5_3:=(0,0,2,11,7,6,0,2,5,3,0,1,1,0,0,0,0,0,0);
type ROM_a6_3 is array (0 to 18) of integer;
constant MEM_a6_3: ROM_a6_3:=(0,0,7,2,3,4,2,1,3,1,0,0,0,0,0,0,0,0,0);
type ROM_a7_3 is array (0 to 18) of integer;
constant MEM_a7_3: ROM_a7_3:=(0,0,5,4,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a8_3 is array (0 to 18) of integer;
constant MEM_a8_3: ROM_a8_3:=(0,0,0,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a9_3 is array (0 to 18) of integer;
constant MEM_a9_3: ROM_a9_3:=(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a10_3 is array (0 to 18) of integer;
```

```
type ROM_a10_3 is array (0 to 18) of integer;
constant MEM_a10_3: ROM_a10_3:=(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a11_3 is array (0 to 18) of integer;
constant MEM_a11_3: ROM_a11_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a12_3 is array (0 to 18) of integer;
constant MEM_a12_3: ROM_a12_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a13_3 is array (0 to 18) of integer;
constant MEM_a13_3: ROM_a13_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a14_3 is array (0 to 18) of integer;
constant MEM_a14_3: ROM_a14_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a15_3 is array (0 to 18) of integer;
constant MEM_a15_3: ROM_a15_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a16_3 is array (0 to 18) of integer;
constant MEM_a16_3: ROM_a16_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a17_3 is array (0 to 18) of integer;
constant MEM_a17_3: ROM_a17_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a18_3 is array (0 to 18) of integer;
constant MEM_a18_3: ROM_a18_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
type ROM_a19_3 is array (0 to 18) of integer;
constant MEM_a19_3: ROM_a19_3:=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
-------------------------------------------Signals-----------------------------------------------
signal ampfunde : integer:=0;
signal amp1e : integer:=0;
signal amp2e : integer:=0;
signal amp3e : integer:=0;
signal amp4e : integer:=0;
signal amp5e : integer:=0;
signal amp6e : integer:=0;
signal amp7e : integer:=0;
signal amp8e : integer:=0;
signal amp9e : integer:=0;
signal amp10e : integer:=0;
signal amp11e : integer:=0;
signal amp12e : integer:=0;
signal amp13e : integer:=0;
signal amp14e : integer:=0;
signal amp15e : integer:=0;
signal amp16e : integer:=0;
signal amp17e : integer:=0;
signal amp18e : integer:=0;
signal amp19e : integer:=0;

signal ampfund : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp1 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp2 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp3 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp4 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp6 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp7 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp8 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp9 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp10 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp11 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp12 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp13 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp14 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp15 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp16 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp17 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
```

```
signal amp18 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal amp19 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
begin
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                if
                        ----------------------------------OBOE----------------------------------------
                        (num_inst = "01") then
                                ampfunde <= MEM_a_1(CONV_INTEGER(num_nota));
                                ampfund <= conv_std_logic_vector(ampfunde,8);
                                amp1e <= MEM_a1_1(CONV_INTEGER(num_nota));
                                --guardar en amp1e el valor que tenemos en la memoria
                                amp1 <= conv_std_logic_vector(amp1e,8);
                                -- convertir el valor que tenemos en la memoria
                                amp2e <= MEM_a2_1(CONV_INTEGER(num_nota));
                                amp2 <= conv_std_logic_vector(amp2e,8);
                                amp3e <= MEM_a3_1(CONV_INTEGER(num_nota));
                                amp3 <= conv_std_logic_vector(amp3e,8);
                                amp4e <= MEM_a4_1(CONV_INTEGER(num_nota));
                                amp4 <= conv_std_logic_vector(amp4e,8);
                                amp5e <= MEM_a5_1(CONV_INTEGER(num_nota));
                                amp5 <= conv_std_logic_vector(amp5e,8);
                                amp6e <= MEM_a6_1(CONV_INTEGER(num_nota));
                                amp6 <= conv_std_logic_vector(amp6e,8);
                                amp7e <= MEM_a7_1(CONV_INTEGER(num_nota));
                                amp7 <= conv_std_logic_vector(amp7e,8);
                                amp8e <= MEM_a8_1(CONV_INTEGER(num_nota));
                                amp8 <= conv_std_logic_vector(amp8e,8);
                                amp9e <= MEM_a9_1(CONV_INTEGER(num_nota));
                                amp9 <= conv_std_logic_vector(amp9e,8);
                                amp10e <= MEM_a10_1(CONV_INTEGER(num_nota));
                                amp10 <= conv_std_logic_vector(amp10e,8);
                                amp11e <= MEM_a11_1(CONV_INTEGER(num_nota));
                                amp11 <= conv_std_logic_vector(amp11e,8);
                                amp12e <= MEM_a12_1(CONV_INTEGER(num_nota));
                                amp12 <= conv_std_logic_vector(amp12e,8);
                                amp13e <= MEM_a13_1(CONV_INTEGER(num_nota));
                                amp13 <= conv_std_logic_vector(amp13e,8);
                                amp14e <= MEM_a14_1(CONV_INTEGER(num_nota));
                                amp14 <= conv_std_logic_vector(amp14e,8);
                                amp15e <= MEM_a15_1(CONV_INTEGER(num_nota));
                                amp15 <= conv_std_logic_vector(amp15e,8);
                                amp16e <= MEM_a16_1(CONV_INTEGER(num_nota));
                                amp16 <= conv_std_logic_vector(amp16e,8);
                                amp17e <= MEM_a17_1(CONV_INTEGER(num_nota));
                                amp17 <= conv_std_logic_vector(amp17e,8);
                                amp18e <= MEM_a18_1(CONV_INTEGER(num_nota));
                                amp18 <= conv_std_logic_vector(amp18e,8);
                                amp19e <= MEM_a19_1(CONV_INTEGER(num_nota));
                                amp19 <= conv_std_logic_vector(amp19e,8);
                        ---------------------------------------VIOLIN---------------------------------------
                elsif
                        (num_inst = "10") then
                                ampfunde <= MEM_a_2(CONV_INTEGER(num_nota));
                                ampfund <= conv_std_logic_vector(ampfunde,8);
                                amp1e <= MEM_a1_2(CONV_INTEGER(num_nota));
                                --guardar en amp1e el valor que tenemos en la memoria
                                amp1 <= conv_std_logic_vector(amp1e,8);
                                -- convertir el valor que tenemos en la memoria
```

```
                amp2e <= MEM_a2_2(CONV_INTEGER(num_nota));
                amp2 <= conv_std_logic_vector(amp2e,8);
                amp3e <= MEM_a3_2(CONV_INTEGER(num_nota));
                amp3 <= conv_std_logic_vector(amp3e,8);
                amp4e <= MEM_a4_2(CONV_INTEGER(num_nota));
                amp4 <= conv_std_logic_vector(amp4e,8);
                amp5e <= MEM_a5_2(CONV_INTEGER(num_nota));
                amp5 <= conv_std_logic_vector(amp5e,8);
                amp6e <= MEM_a6_2(CONV_INTEGER(num_nota));
                amp6 <= conv_std_logic_vector(amp6e,8);
                amp7e <= MEM_a7_2(CONV_INTEGER(num_nota));
                amp7 <= conv_std_logic_vector(amp7e,8);
                amp8e <= MEM_a8_2(CONV_INTEGER(num_nota));
                amp8 <= conv_std_logic_vector(amp8e,8);
                amp9e <= MEM_a9_2(CONV_INTEGER(num_nota));
                amp9 <= conv_std_logic_vector(amp9e,8);
                amp10e <= MEM_a10_2(CONV_INTEGER(num_nota));
                amp10 <= conv_std_logic_vector(amp10e,8);
                amp11e <= MEM_a11_2(CONV_INTEGER(num_nota));
                amp11 <= conv_std_logic_vector(amp11e,8);
                amp12e <= MEM_a12_2(CONV_INTEGER(num_nota));
                amp12 <= conv_std_logic_vector(amp12e,8);
                amp13e <= MEM_a13_2(CONV_INTEGER(num_nota));
                amp13 <= conv_std_logic_vector(amp13e,8);
                amp14e <= MEM_a14_2(CONV_INTEGER(num_nota));
                amp14 <= conv_std_logic_vector(amp14e,8);
                amp15e <= MEM_a15_2(CONV_INTEGER(num_nota));
                amp15 <= conv_std_logic_vector(amp15e,8);
                amp16e <= MEM_a16_2(CONV_INTEGER(num_nota));
                amp16 <= conv_std_logic_vector(amp16e,8);
                amp17e <= MEM_a17_2(CONV_INTEGER(num_nota));
                amp17 <= conv_std_logic_vector(amp17e,8);
                amp18e <= MEM_a18_2(CONV_INTEGER(num_nota));
                amp18 <= conv_std_logic_vector(amp18e,8);
                amp19e <= MEM_a19_2(CONV_INTEGER(num_nota));
                amp19 <= conv_std_logic_vector(amp19e,8);
-----------------------------------------------TUBA-----------------------------------------------
elsif
        (num_inst = "11") then
                ampfunde <= MEM_a_3(CONV_INTEGER(num_nota));
                ampfund <= conv_std_logic_vector(ampfunde,8);
                amp1e <= MEM_a1_3(CONV_INTEGER(num_nota));
                amp1 <= conv_std_logic_vector(amp1e,8);
                amp2e <= MEM_a2_3(CONV_INTEGER(num_nota));
                amp2 <= conv_std_logic_vector(amp2e,8);
                amp3e <= MEM_a3_3(CONV_INTEGER(num_nota));
                amp3 <= conv_std_logic_vector(amp3e,8);
                amp4e <= MEM_a4_3(CONV_INTEGER(num_nota));
                amp4 <= conv_std_logic_vector(amp4e,8);
                amp5e <= MEM_a5_3(CONV_INTEGER(num_nota));
                amp5 <= conv_std_logic_vector(amp5e,8);
                amp6e <= MEM_a6_3(CONV_INTEGER(num_nota));
                amp6 <= conv_std_logic_vector(amp6e,8);
                amp7e <= MEM_a7_3(CONV_INTEGER(num_nota));
                amp7 <= conv_std_logic_vector(amp7e,8);
                amp8e <= MEM_a8_3(CONV_INTEGER(num_nota));
                amp8 <= conv_std_logic_vector(amp8e,8);
                amp9e <= MEM_a9_3(CONV_INTEGER(num_nota));
                amp9 <= conv_std_logic_vector(amp9e,8);
```

```
                                amp10e <= MEM_a10_3(CONV_INTEGER(num_nota));
                                amp10 <= conv_std_logic_vector(amp10e,8);
                                amp11e <= MEM_a11_3(CONV_INTEGER(num_nota));
                                amp11 <= conv_std_logic_vector(amp11e,8);
                                amp12e <= MEM_a12_3(CONV_INTEGER(num_nota));
                                amp12 <= conv_std_logic_vector(amp12e,8);
                                amp13e <= MEM_a13_3(CONV_INTEGER(num_nota));
                                amp13 <= conv_std_logic_vector(amp13e,8);
                                amp14e <= MEM_a14_3(CONV_INTEGER(num_nota));
                                amp14 <= conv_std_logic_vector(amp14e,8);
                                amp15e <= MEM_a15_3(CONV_INTEGER(num_nota));
                                amp15 <= conv_std_logic_vector(amp15e,8);
                                amp16e <= MEM_a16_3(CONV_INTEGER(num_nota));
                                amp16 <= conv_std_logic_vector(amp16e,8);
                                amp17e <= MEM_a17_3(CONV_INTEGER(num_nota));
                                amp17 <= conv_std_logic_vector(amp17e,8);
                                amp18e <= MEM_a18_3(CONV_INTEGER(num_nota));
                                amp18 <= conv_std_logic_vector(amp18e,8);
                                amp19e <= MEM_a19_3(CONV_INTEGER(num_nota));
                                amp19 <= conv_std_logic_vector(amp19e,8);
                        end if;
                end if;
        end Process;
        af <= ampfund;
        a1 <= amp1;
        a2 <= amp2;
        a3 <= amp3;
        a4 <= amp4;
        a5 <= amp5;
        a6 <= amp6;
        a7 <= amp7;
        a8 <= amp8;
        a9 <= amp9;
        a10 <= amp10;
        a11 <= amp11;
        a12 <= amp12;
        a13 <= amp13;
        a14 <= amp14;
        a15 <= amp15;
        a16 <= amp16;
        a17 <= amp17;
        a18 <= amp18;
        a19 <= amp19;
        end Behavioral;
```

Control

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

```
entity control is
   Port ( CLK : in STD_LOGIC;              -- Tiene que funcionar a 48000*21= 1008000
          RST : in STD_LOGIC;
          cont : out STD_LOGIC_VECTOR (4 downto 0)
          );
end control;

architecture Behavioral of control is
signal Qint: STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
begin

Process (CLK)
begin
          if (CLK'event and CLK='1') then
                    if (RST='1') then
                              Qint <= "00000";
                    else
                              if Qint = 20 then Qint <= "00000";
                              else Qint <= Qint+1;                    -- Qint señales control MUX
                              end if;
                    end if;
          end if;
end process;
cont <= Qint;
end Behavioral;
```

6.9.- Incrementos

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity incrementos is
   Port ( CLK : in  STD_LOGIC;                    -- Tiene que funcionar a 48000
          num_nota : in STD_LOGIC_VECTOR (4 downto 0);
          num_inst : in STD_LOGIC_VECTOR (1 downto 0);
          Sale : out integer;
          Sal : out STD_LOGIC_VECTOR (17 downto 0)
          );
end incrementos;

architecture Behavioral of incrementos is
------------- OBOE
          type ROM is array (0 to 18) of integer;          --ROM P(incremento de fase) * 4
          constant MEM1: ROM:=(356,376,5712,6052,6412,6796,7200,7628,8080,8560,9072,
                              9608,10180,10788,11428, 12108,12828,102640,108744);
------------- VIOLIN
          type ROM2 is array (0 to 18) of integer;
          constant MEM2: ROM2:=(356,376,11428,12108,12828,13592,14400,15256,16164,17124,
                              18144,19220,20364,21576,22860,24220,25660,102640,108744);
```

```
------------ TUBA
        type ROM3 is array (0 to 18) of integer;
        constant MEM3: ROM3:=(356,376,2856,3024,3204,3396,3600,3812,4040,4280,4536,
                              4804,5088,5392,5712, 6052,6412,102640,108744);
-----------------------------------------Signals-----------------------------------------
signal INCe : integer:=0;
signal INC : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
begin
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                if
                -----------------------------------------OBOE-----------------------------------------
                        (num_inst = "01") then
                        INCe <= MEM1(CONV_INTEGER(num_nota));
                        INC <= conv_std_logic_vector(INCe,18);
                -----------------------------------------VIOLIN---------------------------------------
                elsif
                        (num_inst = "10") then
                        INCe <= MEM2(CONV_INTEGER(num_nota));
                        INC <= conv_std_logic_vector(INCe,18);
                -----------------------------------------TUBA-----------------------------------------
                elsif
                        (num_inst = "11") then
                        INCe <= MEM3(CONV_INTEGER(num_nota));
                        INC <= conv_std_logic_vector(INCe,18);
                end if;
        end if;
end Process;
Sal <= INC;
Sale <= INCe;
end Behavioral;
```

Arm

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

entity arm is
   Port ( CLK : in  STD_LOGIC;                          -- Tiene que funcionar a 48000
        num_nota : in STD_LOGIC_VECTOR (4 downto 0);
        num_inst : in STD_LOGIC_VECTOR (1 downto 0);
        RST : in STD_LOGIC;
        INCe : in integer;
        INC : in STD_LOGIC_VECTOR (17 downto 0);
        Bf : out STD_LOGIC_VECTOR (19 downto 0);
        B1 : out STD_LOGIC_VECTOR (19 downto 0);
        B2 : out STD_LOGIC_VECTOR (19 downto 0);
        B3 : out STD_LOGIC_VECTOR (19 downto 0);
        B4 : out STD_LOGIC_VECTOR (19 downto 0);
        B5 : out STD_LOGIC_VECTOR (19 downto 0);
        B6 : out STD_LOGIC_VECTOR (19 downto 0);
        B7 : out STD_LOGIC_VECTOR (19 downto 0);
        B8 : out STD_LOGIC_VECTOR (19 downto 0);
        B9 : out STD_LOGIC_VECTOR (19 downto 0);
        B10 : out STD_LOGIC_VECTOR (19 downto 0);
```

```
        B11 : out STD_LOGIC_VECTOR (19 downto 0);
        B12 : out STD_LOGIC_VECTOR (19 downto 0);
        B13 : out STD_LOGIC_VECTOR (19 downto 0);
        B14 : out STD_LOGIC_VECTOR (19 downto 0);
        B15 : out STD_LOGIC_VECTOR (19 downto 0);
        B16 : out STD_LOGIC_VECTOR (19 downto 0);
        B17 : out STD_LOGIC_VECTOR (19 downto 0);
        B18 : out STD_LOGIC_VECTOR (19 downto 0);
        B19 : out STD_LOGIC_VECTOR (19 downto 0)
        );
end arm;

architecture Behavioral of arm is
signal Bf_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B1_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B2_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B3_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B4_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B5_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B6_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B7_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B8_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B9_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B10_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B11_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B12_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B13_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B14_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B15_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B16_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B17_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B18_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B19_1 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal Bf_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B1_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B2_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B3_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B4_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B5_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B6_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B7_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B8_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B9_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B10_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B11_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B12_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B13_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B14_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B15_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B16_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B17_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B18_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B19_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');

begin
```

```
Process (CLK)
begin
	if (CLK'event and CLK='1') then
	---------- RESET ------------------------
		if (RST='1') then
				Bf_1 <= (others => '0');
				B1_1 <= (others => '0');
				B2_1 <= (others => '0');
				B3_1 <= (others => '0');
				B4_1 <= (others => '0');
				B5_1 <= (others => '0');
				B6_1 <= (others => '0');
				B7_1 <= (others => '0');
				B8_1 <= (others => '0');
				B9_1 <= (others => '0');
				B10_1 <= (others => '0');
				B11_1 <= (others => '0');
				B12_1 <= (others => '0');
				B13_1 <= (others => '0');
				B14_1 <= (others => '0');
				B15_1 <= (others => '0');
				B16_1 <= (others => '0');
				B17_1 <= (others => '0');
				B18_1 <= (others => '0');
				B19_1 <= (others => '0');
		else
				Bf_1 <= "00"& INC;			-- frecuencia fundamental
				B1_1 <= conv_std_logic_vector (INCe*2,20);-- primer armonico
				B2_1 <= conv_std_logic_vector (INCe*3,20);
				B3_1 <= conv_std_logic_vector (INCe*4,20);
				B4_1 <= conv_std_logic_vector (INCe*5,20);
				B5_1 <= conv_std_logic_vector (INCe*6,20);
				B6_1 <= conv_std_logic_vector (INCe*7,20);
				B7_1 <= conv_std_logic_vector (INCe*8,20);
				B8_1 <= conv_std_logic_vector (INCe*9,20);
				B9_1 <= conv_std_logic_vector (INCe*10,20);
				B10_1 <= conv_std_logic_vector (INCe*11,20);
				B11_1 <= conv_std_logic_vector (INCe*12,20);
				B12_1 <= conv_std_logic_vector (INCe*13,20);
				B13_1 <= conv_std_logic_vector (INCe*14,20);
				B14_1 <= conv_std_logic_vector (INCe*15,20);
				B15_1 <= conv_std_logic_vector (INCe*16,20);
				B16_1 <= conv_std_logic_vector (INCe*17,20);
				B17_1 <= conv_std_logic_vector (INCe*18,20);
				B18_1 <= conv_std_logic_vector (INCe*19,20);
				B19_1 <= conv_std_logic_vector (INCe*20,20);
		end if;
	end if;
end Process;
Bf <= Bf_1;
B1 <= B1_1;
B2 <= B2_1;
B3 <= B3_1;
B4 <= B4_1;
B5 <= B5_1;
B6 <= B6_1;
B7 <= B7_1;
B8 <= B8_1;
B9 <= B9_1;
B10 <= B10_1;
```

```
B11 <= B11_1;
B12 <= B12_1;
B13 <= B13_1;
B14 <= B14_1;
B15 <= B15_1;
B16 <= B16_1;
B17 <= B17_1;
B18 <= B18_1;
B19 <= B19_1;
end Behavioral;
```

Acum

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity acum is
   Port ( CLK : in  STD_LOGIC;                          -- Tiene que funcionar a 48000
          RST : in STD_LOGIC;
          Bf_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B1_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B2_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B3_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B4_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B5_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B6_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B7_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B8_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B9_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B10_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B11_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B12_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B13_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B14_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B15_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B16_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B17_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B18_1 : in STD_LOGIC_VECTOR (19 downto 0);
          B19_1 : in STD_LOGIC_VECTOR (19 downto 0);
          Bf : out STD_LOGIC_VECTOR (19 downto 0);
          B1 : out STD_LOGIC_VECTOR (19 downto 0);
          B2 : out STD_LOGIC_VECTOR (19 downto 0);
          B3 : out STD_LOGIC_VECTOR (19 downto 0);
          B4 : out STD_LOGIC_VECTOR (19 downto 0);
          B5 : out STD_LOGIC_VECTOR (19 downto 0);
          B6 : out STD_LOGIC_VECTOR (19 downto 0);
          B7 : out STD_LOGIC_VECTOR (19 downto 0);
          B8 : out STD_LOGIC_VECTOR (19 downto 0);
          B9 : out STD_LOGIC_VECTOR (19 downto 0);
          B10 : out STD_LOGIC_VECTOR (19 downto 0);
          B11 : out STD_LOGIC_VECTOR (19 downto 0);
          B12 : out STD_LOGIC_VECTOR (19 downto 0);
          B13 : out STD_LOGIC_VECTOR (19 downto 0);
          B14 : out STD_LOGIC_VECTOR (19 downto 0);
          B15 : out STD_LOGIC_VECTOR (19 downto 0);
          B16 : out STD_LOGIC_VECTOR (19 downto 0);
```

```
        B17 : out STD_LOGIC_VECTOR (19 downto 0);
        B18 : out STD_LOGIC_VECTOR (19 downto 0);
        B19 : out STD_LOGIC_VECTOR (19 downto 0)
        );
end acum;

architecture Behavioral of acum is
signal Bf_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B1_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B2_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B3_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B4_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B5_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B6_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B7_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B8_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B9_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B10_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B11_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B12_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B13_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B14_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B15_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B16_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B17_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B18_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
signal B19_2 : STD_LOGIC_VECTOR (19 downto 0):= (others=>'0');
begin
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                ---------- RESET -------------------------
                if (RST='1') then
                                Bf_2 <= (others => '0');
                                B1_2 <= (others => '0');
                                B2_2 <= (others => '0');
                                B3_2 <= (others => '0');
                                B4_2 <= (others => '0');
                                B5_2 <= (others => '0');
                                B6_2 <= (others => '0');
                                B7_2 <= (others => '0');
                                B8_2 <= (others => '0');
                                B9_2 <= (others => '0');
                                B10_2 <= (others => '0');
                                B11_2 <= (others => '0');
                                B12_2 <= (others => '0');
                                B13_2 <= (others => '0');
                                B14_2 <= (others => '0');
                                B15_2 <= (others => '0');
                                B16_2 <= (others => '0');
                                B17_2 <= (others => '0');
                                B18_2 <= (others => '0');
                                B19_2 <= (others => '0');
                else
                                Bf_2 <= Bf_2+Bf_1;
                                B1_2 <= B1_2+B1_1;
                                B2_2 <= B2_2+B2_1;
                                B3_2 <= B3_2+B3_1;
                                B4_2 <= B4_2+B4_1;
                                B5_2 <= B5_2+B5_1;
```

```
                                        B6_2 <= B6_2+B6_1;
                                        B7_2 <= B7_2+B7_1;
                                        B8_2 <= B8_2+B8_1;
                                        B9_2 <= B9_2+B9_1;
                                        B10_2 <= B10_2+B10_1;
                                        B11_2 <= B11_2+B11_1;
                                        B12_2 <= B12_2+B12_1;
                                        B13_2 <= B13_2+B13_1;
                                        B14_2 <= B14_2+B14_1;
                                        B15_2 <= B15_2+B15_1;
                                        B16_2 <= B16_2+B16_1;
                                        B17_2 <= B17_2+B17_1;
                                        B18_2 <= B18_2+B18_1;
                                        B19_2 <= B19_2+B19_1;
                        end if;
                end if;
end Process;

Bf <= Bf_2;
B1 <= B1_2;
B2 <= B2_2;
B3 <= B3_2;
B4 <= B4_2;
B5 <= B5_2;
B6 <= B6_2;
B7 <= B7_2;
B8 <= B8_2;
B9 <= B9_2;
B10 <= B10_2;
B11 <= B11_2;
B12 <= B12_2;
B13 <= B13_2;
B14 <= B14_2;
B15 <= B15_2;
B16 <= B16_2;
B17 <= B17_2;
B18 <= B18_2;
B19 <= B19_2;

end Behavioral;
```

Sumfas

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

entity sumfas is
    Port ( fase1 : in STD_LOGIC_VECTOR (8 downto 0);
           fase2 : in STD_LOGIC_VECTOR (8 downto 0);
           fase3 : in STD_LOGIC_VECTOR (8 downto 0);
           fase4 : in STD_LOGIC_VECTOR (8 downto 0);
           fase5 : in STD_LOGIC_VECTOR (8 downto 0);
           fase6 : in STD_LOGIC_VECTOR (8 downto 0);
```

```
        fase7 : in STD_LOGIC_VECTOR (8 downto 0);
        fase8 : in STD_LOGIC_VECTOR (8 downto 0);
        fase9 : in STD_LOGIC_VECTOR (8 downto 0);
        fase10 : in STD_LOGIC_VECTOR (8 downto 0);
        fase11 : in STD_LOGIC_VECTOR (8 downto 0);
        fase12 : in STD_LOGIC_VECTOR (8 downto 0);
        fase13 : in STD_LOGIC_VECTOR (8 downto 0);
        fase14 : in STD_LOGIC_VECTOR (8 downto 0);
        fase15 : in STD_LOGIC_VECTOR (8 downto 0);
        fase16 : in STD_LOGIC_VECTOR (8 downto 0);
        fase17 : in STD_LOGIC_VECTOR (8 downto 0);
        fase18 : in STD_LOGIC_VECTOR (8 downto 0);
        fase19 : in STD_LOGIC_VECTOR (8 downto 0);
        Bf_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B1_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B2_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B3_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B4_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B5_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B6_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B7_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B8_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B9_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B10_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B11_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B12_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B13_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B14_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B15_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B16_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B17_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B18_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B19_2 : in STD_LOGIC_VECTOR (19 downto 0);
        Bf_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B1_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B2_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B3_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B4_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B5_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B6_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B7_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B8_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B9_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B10_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B11_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B12_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B13_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B14_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B15_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B16_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B17_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B18_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B19_3 : out STD_LOGIC_VECTOR (8 downto 0)
        );
end sumfas;

architecture Behavioral of sumfas is

begin
```

```
                                    Bf_3 <= Bf_2(19 downto 11);
                                    B1_3 <= B1_2(19 downto 11) +  fase1;
                                    B2_3 <= B2_2(19 downto 11) +  fase2;
                                    B3_3 <= B3_2(19 downto 11) +  fase3;
                                    B4_3 <= B4_2(19 downto 11) +  fase4;
                                    B5_3 <= B5_2(19 downto 11) +  fase5;
                                    B6_3 <= B6_2(19 downto 11) +  fase6;
                                    B7_3 <= B7_2(19 downto 11) +  fase7;
                                    B8_3 <= B8_2(19 downto 11) +  fase8;
                                    B9_3 <= B9_2(19 downto 11) +  fase9;
                                    B10_3 <= B10_2(19 downto 11) +  fase10;
                                    B11_3 <= B11_2(19 downto 11) +  fase11;
                                    B12_3 <= B12_2(19 downto 11) +  fase12;
                                    B13_3 <= B13_2(19 downto 11) +  fase13;
                                    B14_3 <= B14_2(19 downto 11) +  fase14;
                                    B15_3 <= B15_2(19 downto 11) +  fase15;
                                    B16_3 <= B16_2(19 downto 11) +  fase16;
                                    B17_3 <= B17_2(19 downto 11) +  fase17;
                                    B18_3 <= B18_2(19 downto 11) +  fase18;
                                    B19_3 <= B19_2(19 downto 11) +  fase19;

end Behavioral;
```

6.13.-FF1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FF1 is
   Port ( CLK : in  STD_LOGIC;                          -- Tiene que funcionar a 48000
          Bf_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B1_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B2_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B3_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B4_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B5_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B6_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B7_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B8_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B9_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B10_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B11_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B12_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B13_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B14_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B15_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B16_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B17_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B18_3 : in STD_LOGIC_VECTOR (8 downto 0);
          B19_3 : in STD_LOGIC_VECTOR (8 downto 0);
          Bf : out STD_LOGIC_VECTOR (8 downto 0);
          B1 : out STD_LOGIC_VECTOR (8 downto 0);
          B2 : out STD_LOGIC_VECTOR (8 downto 0);
          B3 : out STD_LOGIC_VECTOR (8 downto 0);
          B4 : out STD_LOGIC_VECTOR (8 downto 0);
          B5 : out STD_LOGIC_VECTOR (8 downto 0);
          B6 : out STD_LOGIC_VECTOR (8 downto 0);
          B7 : out STD_LOGIC_VECTOR (8 downto 0);
```

```
        B8 : out STD_LOGIC_VECTOR (8 downto 0);
        B9 : out STD_LOGIC_VECTOR (8 downto 0);
        B10 : out STD_LOGIC_VECTOR (8 downto 0);
        B11 : out STD_LOGIC_VECTOR (8 downto 0);
        B12 : out STD_LOGIC_VECTOR (8 downto 0);
        B13 : out STD_LOGIC_VECTOR (8 downto 0);
        B14 : out STD_LOGIC_VECTOR (8 downto 0);
        B15 : out STD_LOGIC_VECTOR (8 downto 0);
        B16 : out STD_LOGIC_VECTOR (8 downto 0);
        B17 : out STD_LOGIC_VECTOR (8 downto 0);
        B18 : out STD_LOGIC_VECTOR (8 downto 0);
        B19 : out STD_LOGIC_VECTOR (8 downto 0)
        );
end FF1;
architecture Behavioral of FF1 is
signal Ff : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F1 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F2 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F3 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F4 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F5 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F6 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F7 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F8 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F9 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F10 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F11 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F12 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F13 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F14 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F15 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F16 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F17 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F18 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal F19 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
begin
Process (CLK)
        begin
                if (CLK'event and CLK='1') then
                        Ff <= Bf_3;
                        F1 <= B1_3;
                        F2 <= B2_3;
                        F3 <= B3_3;
                        F4 <= B4_3;
                        F5 <= B5_3;
                        F6 <= B6_3;
                        F7 <= B7_3;
                        F8 <= B8_3;
                        F9 <= B9_3;
                        F10 <= B10_3;
                        F11 <= B11_3;
                        F12 <= B12_3;
                        F13 <= B13_3;
                        F14 <= B14_3;
                        F15 <= B15_3;
                        F16 <= B16_3;
                        F17 <= B17_3;
                        F18 <= B18_3;
                        F19 <= B19_3;
                end if;
```

```
end process;
Bf <= Ff;
B1 <= F1;
B2 <= F2;
B3 <= F3;
B4 <= F4;
B5 <= F5;
B6 <= F6;
B7 <= F7;
B8 <= F8;
B9 <= F9;
B10 <= F10;
B11 <= F11;
B12 <= F12;
B13 <= F13;
B14 <= F14;
B15 <= F15;
B16 <= F16;
B17 <= F17;
B18 <= F18;
B19 <= F19;
end Behavioral;
```

6.14.-Mux1

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

entity mux1 is
   Port ( CLK : in  STD_LOGIC;                          -- Tiene que funcionar a 48000 * 21
          Qint : in STD_LOGIC_VECTOR (4 downto 0);
          Ff : in STD_LOGIC_VECTOR (8 downto 0);
          F1 : in STD_LOGIC_VECTOR (8 downto 0);
          F2 : in STD_LOGIC_VECTOR (8 downto 0);
          F3 : in STD_LOGIC_VECTOR (8 downto 0);
          F4 : in STD_LOGIC_VECTOR (8 downto 0);
          F5 : in STD_LOGIC_VECTOR (8 downto 0);
          F6 : in STD_LOGIC_VECTOR (8 downto 0);
          F7 : in STD_LOGIC_VECTOR (8 downto 0);
          F8 : in STD_LOGIC_VECTOR (8 downto 0);
          F9 : in STD_LOGIC_VECTOR (8 downto 0);
          F10 : in STD_LOGIC_VECTOR (8 downto 0);
          F11 : in STD_LOGIC_VECTOR (8 downto 0);
          F12 : in STD_LOGIC_VECTOR (8 downto 0);
          F13 : in STD_LOGIC_VECTOR (8 downto 0);
          F14 : in STD_LOGIC_VECTOR (8 downto 0);
          F15 : in STD_LOGIC_VECTOR (8 downto 0);
          F16 : in STD_LOGIC_VECTOR (8 downto 0);
          F17 : in STD_LOGIC_VECTOR (8 downto 0);
          F18 : in STD_LOGIC_VECTOR (8 downto 0);
          F19 : in STD_LOGIC_VECTOR (8 downto 0);
          sal : out STD_LOGIC_VECTOR (8 downto 0)
          );
end mux1;
```

```
architecture Behavioral of mux1 is
signal dir_mem : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
begin
Process (CLK)
	begin
		if (CLK'event and CLK='1') then
			if (Qint="00000") then dir_mem <= Ff;
			elsif (Qint="00001") then dir_mem <= F1;
			elsif (Qint="00010") then dir_mem <= F2;
			elsif (Qint="00011") then dir_mem <= F3;
			elsif (Qint="00100") then dir_mem <= F4;
			elsif (Qint="00101") then dir_mem <= F5;
			elsif (Qint="00110") then dir_mem <= F6;
			elsif (Qint="00111") then dir_mem <= F7;
			elsif (Qint="01000") then dir_mem <= F8;
			elsif (Qint="01001") then dir_mem <= F9;
			elsif (Qint="01010") then dir_mem <= F10;
			elsif (Qint="01011") then dir_mem <= F11;
			elsif (Qint="01100") then dir_mem <= F12;
			elsif (Qint="01101") then dir_mem <= F13;
			elsif (Qint="01110") then dir_mem <= F14;
			elsif (Qint="01111") then dir_mem <= F15;
			elsif (Qint="10000") then dir_mem <= F16;
			elsif (Qint="10001") then dir_mem <= F17;
			elsif (Qint="10010") then dir_mem <= F18;
			elsif (Qint="10011") then dir_mem <= F19;
			elsif (Qint="10100") then dir_mem <= F19;
			end if;
		end if;
end process;
sal <= dir_mem;
end Behavioral;
```

Sen

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity sen is
   Port ( dir_mem : in STD_LOGIC_VECTOR (8 downto 0);
          sal : out integer
          );
end sen;
architecture Behavioral of sen is
          type ROM_sen is array (0 to 511) of integer;
          constant MEM_sen:ROM_sen:=(128,129,131,132,134,135,137,139,140,142,143,145,146,
                                     148,149,151,153,154,156,157,159,160,162,163,165,166,
                                     168,169,171,172,174,175,177,178,179,181,182,184,185,
                                     187,188,189,191,192,193,195,196,197,199,200,201,203,
                                     204,205,206,208,209,210,211,212,214,215,216,217,218,
                                     219,220,221,222,224,225,226,227,228,229,230,230,231,
                                     232,233,234,235,236,237,237,238,239,240,241,241,242,
                                     243,243,244,245,245,246,246,247,248,248,249,249,250,
                                     250,251,251,251,252,252,252,253,253,253,254,254,254,
                                     254,255,255,255,255,255,255,255,255,255,255,255,255,
                                     255,255,255,255,255,255,255,255,254,254,254,254,254,
```

```
                                253,253,253,252,252,252,251,251,250,250,249,249,248,248,
                                247, 247,246,246,245,244,244,243,242,242,241,240,239,239,
                                238,237,236,235,234,233,232,231,230,229,228,227,226,225,
                                224,223,222,221,220,219,218,216,215,214,213, 212,211,209,
                                208,207,206,205,203,202,201,199,198,197,195,194,193,191,
                                190,189,187,186,184,183,182,180,179,177,176,174,173,171,
                                170,168,167,165,164,162,161,159,158,156,155, 153,152,150,
                                149,147,146,144,142,141,139,138,136,135,133,131,130,128,
                                127,125,124,122,120,119,117,116,114,113,111,109,108,106,105,
                                103,102, 100,99,97,96,94,93,91,90,88,87,85,84,82,81,79,78,76,
                                75,73,72,71,69,68, 66,65,64,62,61,60,58,57,56,54,53,52,50,49,
                                48,47,46,44,43,42,41,40,39, 37,36,35,34,33,32,31,30,29,28,27,
                                26,25,24,23,22,21,21,20,19,18,17,16, 16,15,14,13,13,12,11,11,
                                10,9,9,8,8,7,7,6,6,5,5,4,4,3,3,3,2,2,2,1,1,1, 1,1,0,0,0,0,0,0,0,0,
                                0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,2,3,3,3,4,4,4, 5,5,6,6,7,7,8,9,
                                9,10,10,11,12,12,13,14,14,15,15,16,17,18,18,19,20,21, 22,23,
                                24,25,25,26,27,28,29,30,31,33,34,35,36,37,38,39,40,41,43,44,
                                45, 46,47,49,50,51,52,54,55,56,58, 59,60,62,63,64,66,67,68,70,
                                71,73,74,76, 77,78,80,81,83,84,86,87,89,90,92,93,95,96,98,99,
                                101,102,104,106,107,109, 110,112,113,115,116,118,120,121,
                                123,124,126,127);
begin
sal <= MEM_sen (CONV_INTEGER(dir_mem));
end Behavioral;
```

6.16.-Dmux

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DMUX is
   Port ( CLK : in  STD_LOGIC;                          -- Tiene que funcionar a 48000 * 21
          Qint : in STD_LOGIC_VECTOR (4 downto 0);
          sen : in integer;
          Bf : out STD_LOGIC_VECTOR (7 downto 0);
          B1 : out STD_LOGIC_VECTOR (7 downto 0);
          B2 : out STD_LOGIC_VECTOR (7 downto 0);
          B3 : out STD_LOGIC_VECTOR (7 downto 0);
          B4 : out STD_LOGIC_VECTOR (7 downto 0);
          B5 : out STD_LOGIC_VECTOR (7 downto 0);
          B6 : out STD_LOGIC_VECTOR (7 downto 0);
          B7 : out STD_LOGIC_VECTOR (7 downto 0);
          B8 : out STD_LOGIC_VECTOR (7 downto 0);
          B9 : out STD_LOGIC_VECTOR (7 downto 0);
          B10 : out STD_LOGIC_VECTOR (7 downto 0);
          B11 : out STD_LOGIC_VECTOR (7 downto 0);
          B12 : out STD_LOGIC_VECTOR (7 downto 0);
          B13 : out STD_LOGIC_VECTOR (7 downto 0);
          B14 : out STD_LOGIC_VECTOR (7 downto 0);
          B15 : out STD_LOGIC_VECTOR (7 downto 0);
          B16 : out STD_LOGIC_VECTOR (7 downto 0);
          B17 : out STD_LOGIC_VECTOR (7 downto 0);
          B18 : out STD_LOGIC_VECTOR (7 downto 0);
```

```
        B19 : out STD_LOGIC_VECTOR (7 downto 0)
        );
end DMUX;
architecture Behavioral of DMUX is
signal Bf_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B1_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B2_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B3_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B4_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B5_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B6_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B7_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B8_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B9_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B10_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B11_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B12_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B13_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B14_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B15_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B16_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B17_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B18_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B19_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
begin
Process (CLK)   -- Se mantenga dato durante un ciclo
        begin
                if (CLK'event and CLK='1') then
                        if (Qint="00001") then Bf_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00010") then B1_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00011") then B2_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00100") then B3_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00101") then B4_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00110") then B5_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="00111") then B6_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01000") then B7_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01001") then B8_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01010") then B9_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01011") then B10_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01100") then B11_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01101") then B12_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01110") then B13_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="01111") then B14_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10000") then B15_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10001") then B16_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10010") then B17_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10011") then B18_5 <= conv_std_logic_vector(sen,8);
                        elsif (Qint="10100") then B19_5 <= conv_std_logic_vector(sen,8);
                        end if;
                end if;
end process;
Bf <= Bf_5;
B1 <= B1_5;
B2 <= B2_5;
B3 <= B3_5;
B4 <= B4_5;
B5 <= B5_5;
B6 <= B6_5;
B7 <= B7_5;
B8 <= B8_5;
```

```
B9 <= B9_5;
B10 <= B10_5;
B11 <= B11_5;
B12 <= B12_5;
B13 <= B13_5;
B14 <= B14_5;
B15 <= B15_5;
B16 <= B16_5;
B17 <= B17_5;
B18 <= B18_5;
B19 <= B19_5;
end Behavioral;
```

6.17.-FF2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FF2 is
   Port ( CLK : in  STD_LOGIC;                          -- Tiene que funcionar a 48000*21
          Qint : in STD_LOGIC_VECTOR (4 downto 0);
          Bf_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B1_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B2_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B3_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B4_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B5_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B6_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B7_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B8_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B9_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B10_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B11_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B12_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B13_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B14_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B15_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B16_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B17_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B18_5 : in STD_LOGIC_VECTOR (7 downto 0);
          B19_5 : in STD_LOGIC_VECTOR (7 downto 0);
          Bf : out STD_LOGIC_VECTOR (7 downto 0);
          B1 : out STD_LOGIC_VECTOR (7 downto 0);
          B2 : out STD_LOGIC_VECTOR (7 downto 0);
          B3 : out STD_LOGIC_VECTOR (7 downto 0);
          B4 : out STD_LOGIC_VECTOR (7 downto 0);
          B5 : out STD_LOGIC_VECTOR (7 downto 0);
          B6 : out STD_LOGIC_VECTOR (7 downto 0);
          B7 : out STD_LOGIC_VECTOR (7 downto 0);
          B8 : out STD_LOGIC_VECTOR (7 downto 0);
          B9 : out STD_LOGIC_VECTOR (7 downto 0);
          B10 : out STD_LOGIC_VECTOR (7 downto 0);
          B11 : out STD_LOGIC_VECTOR (7 downto 0);
          B12 : out STD_LOGIC_VECTOR (7 downto 0);
          B13 : out STD_LOGIC_VECTOR (7 downto 0);
          B14 : out STD_LOGIC_VECTOR (7 downto 0);
          B15 : out STD_LOGIC_VECTOR (7 downto 0);
```

```
        B16 : out STD_LOGIC_VECTOR (7 downto 0);
        B17 : out STD_LOGIC_VECTOR (7 downto 0);
        B18 : out STD_LOGIC_VECTOR (7 downto 0);
        B19 : out STD_LOGIC_VECTOR (7 downto 0)
        );
end FF2;
architecture Behavioral of FF2 is
signal Bf_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B1_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B2_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B3_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B4_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B5_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B6_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B7_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B8_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B9_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B10_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B11_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B12_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B13_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B14_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B15_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B16_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B17_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B18_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal B19_FF : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
begin
Process (CLK)
        begin
                if (CLK'event and CLK='1') then
                        if (Qint="00000") then      -- almacena datos mientras son constantes
                                        Bf_FF <= Bf_5;
                                        B1_FF <= B1_5;
                                        B2_FF <= B2_5;
                                        B3_FF <= B3_5;
                                        B4_FF <= B4_5;
                                        B5_FF <= B5_5;
                                        B6_FF <= B6_5;
                                        B7_FF <= B7_5;
                                        B8_FF <= B8_5;
                                        B9_FF <= B9_5;
                                        B10_FF <= B10_5;
                                        B11_FF <= B11_5;
                                        B12_FF <= B12_5;
                                        B13_FF <= B13_5;
                                        B14_FF <= B14_5;
                                        B15_FF <= B15_5;
                                        B16_FF <= B16_5;
                                        B17_FF <= B17_5;
                                        B18_FF <= B18_5;
                                        B19_FF <= B19_5;
                        end if;
                end if;
end process;
Bf <= Bf_FF;
B1 <= B1_FF;
B2 <= B2_FF;
B3 <= B3_FF;
B4 <= B4_FF;
```

```
B5 <= B5_FF;
B6 <= B6_FF;
B7 <= B7_FF;
B8 <= B8_FF;
B9 <= B9_FF;
B10 <= B10_FF;
B11 <= B11_FF;
B12 <= B12_FF;
B13 <= B13_FF;
B14 <= B14_FF;
B15 <= B15_FF;
B16 <= B16_FF;
B17 <= B17_FF;
B18 <= B18_FF;
B19 <= B19_FF;

end Behavioral;
```

6.19.-Mult

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

entity mult is
   Port ( ampfund : in STD_LOGIC_VECTOR (7 downto 0);
          amp1 : in STD_LOGIC_VECTOR (7 downto 0);
          amp2 : in STD_LOGIC_VECTOR (7 downto 0);
          amp3 : in STD_LOGIC_VECTOR (7 downto 0);
          amp4 : in STD_LOGIC_VECTOR (7 downto 0);
          amp5 : in STD_LOGIC_VECTOR (7 downto 0);
          amp6 : in STD_LOGIC_VECTOR (7 downto 0);
          amp7 : in STD_LOGIC_VECTOR (7 downto 0);
          amp8 : in STD_LOGIC_VECTOR (7 downto 0);
          amp9 : in STD_LOGIC_VECTOR (7 downto 0);
          amp10 : in STD_LOGIC_VECTOR (7 downto 0);
          amp11 : in STD_LOGIC_VECTOR (7 downto 0);
          amp12 : in STD_LOGIC_VECTOR (7 downto 0);
          amp13 : in STD_LOGIC_VECTOR (7 downto 0);
          amp14 : in STD_LOGIC_VECTOR (7 downto 0);
          amp15 : in STD_LOGIC_VECTOR (7 downto 0);
          amp16 : in STD_LOGIC_VECTOR (7 downto 0);
          amp17 : in STD_LOGIC_VECTOR (7 downto 0);
          amp18 : in STD_LOGIC_VECTOR (7 downto 0);
          amp19 : in STD_LOGIC_VECTOR (7 downto 0);
          Bf_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B1_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B2_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B3_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B4_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B5_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B6_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B7_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B8_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B9_FF : in STD_LOGIC_VECTOR (7 downto 0);
          B10_FF : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        amp10 : in STD_LOGIC_VECTOR (7 downto 0);
        amp11 : in STD_LOGIC_VECTOR (7 downto 0);
        amp12 : in STD_LOGIC_VECTOR (7 downto 0);
        amp13 : in STD_LOGIC_VECTOR (7 downto 0);
        amp14 : in STD_LOGIC_VECTOR (7 downto 0);
        amp15 : in STD_LOGIC_VECTOR (7 downto 0);
        amp16 : in STD_LOGIC_VECTOR (7 downto 0);
        amp17 : in STD_LOGIC_VECTOR (7 downto 0);
        amp18 : in STD_LOGIC_VECTOR (7 downto 0);
        amp19 : in STD_LOGIC_VECTOR (7 downto 0);
        Bf_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B1_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B2_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B3_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B4_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B5_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B6_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B7_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B8_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B9_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B10_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B11_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B12_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B13_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B14_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B15_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B16_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B17_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B18_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B19_FF : in STD_LOGIC_VECTOR (7 downto 0);
        Bf_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B1_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B2_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B3_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B4_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B5_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B6_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B7_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B8_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B9_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B10_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B11_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B12_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B13_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B14_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B15_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B16_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B17_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B18_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B19_6 : out STD_LOGIC_VECTOR (15 downto 0)
        );
end mult;
architecture Behavioral of mult is
begin
                                Bf_6 <= Bf_FF * ampfund;
                                B1_6 <= B1_FF * amp1;
                                B2_6 <= B2_FF * amp2;
                                B3_6 <= B3_FF * amp3;
                                B4_6 <= B4_FF * amp4;
```

```
                                        B5_6 <= B5_FF * amp5;
                                        B6_6 <= B6_FF * amp6;
                                        B7_6 <= B7_FF * amp7;
                                        B8_6 <= B8_FF * amp8;
                                        B9_6 <= B9_FF * amp9;
                                        B10_6 <= B10_FF * amp10;
                                        B11_6 <= B11_FF * amp11;
                                        B12_6 <= B12_FF * amp12;
                                        B13_6 <= B13_FF * amp13;
                                        B14_6 <= B14_FF * amp14;
                                        B15_6 <= B15_FF * amp15;
                                        B16_6 <= B16_FF * amp16;
                                        B17_6 <= B17_FF * amp17;
                                        B18_6 <= B18_FF * amp18;
                                        B19_6 <= B19_FF * amp19;
end Behavioral;
```

Offset

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity offset is
    Port ( ampfund : in STD_LOGIC_VECTOR (7 downto 0);
           amp1 : in STD_LOGIC_VECTOR (7 downto 0);
           amp2 : in STD_LOGIC_VECTOR (7 downto 0);
           amp3 : in STD_LOGIC_VECTOR (7 downto 0);
           amp4 : in STD_LOGIC_VECTOR (7 downto 0);
           amp5 : in STD_LOGIC_VECTOR (7 downto 0);
           amp6 : in STD_LOGIC_VECTOR (7 downto 0);
           amp7 : in STD_LOGIC_VECTOR (7 downto 0);
           amp8 : in STD_LOGIC_VECTOR (7 downto 0);
           amp9 : in STD_LOGIC_VECTOR (7 downto 0);
           amp10 : in STD_LOGIC_VECTOR (7 downto 0);
           amp11 : in STD_LOGIC_VECTOR (7 downto 0);
           amp12 : in STD_LOGIC_VECTOR (7 downto 0);
           amp13 : in STD_LOGIC_VECTOR (7 downto 0);
           amp14 : in STD_LOGIC_VECTOR (7 downto 0);
           amp15 : in STD_LOGIC_VECTOR (7 downto 0);
           amp16 : in STD_LOGIC_VECTOR (7 downto 0);
           amp17 : in STD_LOGIC_VECTOR (7 downto 0);
           amp18 : in STD_LOGIC_VECTOR (7 downto 0);
           amp19 : in STD_LOGIC_VECTOR (7 downto 0);
           Bf_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B1_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B2_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B3_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B4_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B5_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B6_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B7_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B8_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B9_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B10_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B11_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B12_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B13_6 : in STD_LOGIC_VECTOR (15 downto 0);
           B14_6 : in STD_LOGIC_VECTOR (15 downto 0);
```

```
        B15_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B16_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B17_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B18_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B19_6 : in STD_LOGIC_VECTOR (15 downto 0);
        Bf_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B1_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B2_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B3_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B4_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B5_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B6_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B7_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B8_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B9_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B10_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B11_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B12_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B13_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B14_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B15_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B16_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B17_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B18_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B19_7 : out STD_LOGIC_VECTOR (15 downto 0)
        );
end offset;

architecture Behavioral of offset is

signal max1 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal max2 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal cont : STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal med : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal divf :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div1 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div2 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div3 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div4 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div5 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div6 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div7 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div8 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div9 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div10 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div11 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div12 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div13 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div14 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div15 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div16 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div17 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div18 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal div19 :  STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
```

```
begin
        max1 <= conv_std_logic_vector(255,8);     -- cambiar dependiendo amplitud maxima mem
                                                     sen
        max2 <= conv_std_logic_vector(78,8);      -- 156/2 cambiar dependiendo de amplitud
                                                     maxima entre todos los instrumentos
        cont <= (max1 * max2);                    -- valor de offset
        med <= conv_std_logic_vector(128,8);      -- 255/2 aprox 128
        divf <= (ampfund * med);                  -- sacar punto medio señal
        div1 <= (amp1 * med);
        div2 <= (amp2 * med);
        div3 <= (amp3 * med);
        div4 <= (amp4 * med);
        div5 <= (amp5 * med);
        div6 <= (amp6 * med);
        div7 <= (amp7 * med);
        div8 <= (amp8 * med);
        div9 <= (amp9 * med);
        div10 <= (amp10 * med);
        div11 <= (amp11 * med);
        div12 <= (amp12 * med);
        div13 <= (amp13 * med);
        div14 <= (amp14 * med);
        div15 <= (amp15 * med);
        div16 <= (amp16 * med);
        div17 <= (amp17 * med);
        div18 <= (amp18 * med);
        div19 <= (amp19 * med);
        Bf_7 <= Bf_6 + cont - divf;               -- sumar diferencia
        B1_7 <= B1_6 + cont - div1;
        B2_7 <= B2_6 + cont - div2;
        B3_7 <= B3_6 + cont - div3;
        B4_7 <= B4_6 + cont - div4;
        B5_7 <= B5_6 + cont - div5;
        B6_7 <= B6_6 + cont - div6;
        B7_7 <= B7_6 + cont - div7;
        B8_7 <= B8_6 + cont - div8;
        B9_7 <= B9_6 + cont - div9;
        B10_7 <= B10_6 + cont - div10;
        B11_7 <= B11_6 + cont - div11;
        B12_7 <= B12_6 + cont - div12;
        B13_7 <= B13_6 + cont - div13;
        B14_7 <= B14_6 + cont - div14;
        B15_7 <= B15_6 + cont - div15;
        B16_7 <= B16_6 + cont - div16;
        B17_7 <= B17_6 + cont - div17;
        B18_7 <= B18_6 + cont - div18;
        B19_7 <= B19_6 + cont - div19;
end Behavioral;
```

6.21.-Suma

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity suma is
   Port ( Bf_7 : in STD_LOGIC_VECTOR (15 downto 0);
          B1_7 : in STD_LOGIC_VECTOR (15 downto 0);
          B2_7 : in STD_LOGIC_VECTOR (15 downto 0);
```

```
        B3_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B4_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B5_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B6_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B7_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B8_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B9_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B10_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B11_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B12_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B13_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B14_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B15_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B16_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B17_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B18_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B19_7 : in STD_LOGIC_VECTOR (15 downto 0);
        Sal : out STD_LOGIC_VECTOR (19 downto 0)
        );
end suma;
architecture Behavioral of suma is
begin
Sal <= "0000" & Bf_7 + B1_7 + B2_7 + B3_7 + B4_7 + B5_7 + B6_7 + B7_7 + B8_7 + B9_7 +
        B10_7 + B11_7 + B12_7 + B13_7  + B14_7 + B15_7 + B16_7 + B17_7 + B18_7 + B19_7;
end Behavioral;
```

Ajuste

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ajuste is
   Port ( Ent : in STD_LOGIC_VECTOR (19 downto 0);
        Sal1 : out STD_LOGIC_VECTOR (23 downto 0);
        Sal2 : out STD_LOGIC_VECTOR (23 downto 0)
        );
end ajuste;
architecture Behavioral of ajuste is
begin
Sal1 <= "0" & Ent & "000";          -- Dependiendo de los 0 que tengamos delante y detras variara el
                                    volumen
Sal2 <= "0" & Ent & "000";          -- otro canal
end Behavioral;
```

6.23.-Arp

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity arp is
   Port ( CLK : in  STD_LOGIC;                              -- Tiene que funcionar a 48000
        notas : in STD_LOGIC_VECTOR (11 downto 0);          -- Notas arpa
        des : in STD_LOGIC_VECTOR (11 downto 0);            -- Interruptores para desactivar notas
        Sal : out STD_LOGIC_VECTOR (11 downto 0)
        );
```

```
end arp;
architecture Behavioral of arp is
signal Salint : STD_LOGIC_VECTOR (11 downto 0):= (others=>'0');
begin
Process (CLK)
begin
      if (CLK'event and CLK='1') then
            if    (des(11)='1') then Salint(11) <= notas(11);
            else Salint(11)<='0';
            end if;
            if    (des(10)='1') then Salint(10) <= notas(10);
            else Salint(10)<='0';
            end if;
            if    (des(9)='1') then Salint(9) <= notas(9);
            else Salint(9)<='0';
            end if;
            if    (des(8)='1') then Salint(8) <= notas(8);
            else Salint(8)<='0';
            end if;
            if    (des(7)='1') then Salint(7) <= notas(7);
            else Salint(7)<='0';
            end if;
            if    (des(6)='1') then Salint(6) <= notas(6);
            else Salint(6)<='0';
            end if;
            if    (des(5)='1') then Salint(5) <= notas(5);
            else Salint(5)<='0';
            end if;
            if    (des(4)='1') then Salint(4) <= notas(4);
            else Salint(4)<='0';
            end if;
            if    (des(3)='1') then Salint(3) <= notas(3);
            else Salint(3)<='0';
            end if;
            if    (des(2)='1') then Salint(2) <= notas(2);
            else Salint(2)<='0';
            end if;
            if    (des(1)='1') then Salint(1) <= notas(1);
            else Salint(1)<='0';
            end if;
            if    (des(0)='1') then Salint(0) <= notas(0);
            else Salint(0)<='0';
            end if;
        end if;
end process;
Sal <= Salint;
end Behavioral;
```

Calc_notas

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity calc_notas is
   Port ( CLK : in  STD_LOGIC;                                  -- Tiene que funcionar a 48000
          notasent : in STD_LOGIC_VECTOR (11 downto 0);         -- En el arpa solo tenemos 12
                                                                notas de C a C - F/G
          nota1 : out STD_LOGIC_VECTOR (4 downto 0);
          nota2 : out STD_LOGIC_VECTOR (4 downto 0)
          );
```

```
end calc_notas;
architecture Behavioral of calc_notas is
signal i11 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i10 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i9 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i8 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i7 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i6 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i5 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i4 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i3 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i2 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i1 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal i0 : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal cant_notas : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal nota1int : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
signal nota2int : STD_LOGIC_VECTOR (4 downto 0):= (others=>'0');
begin
Process (CLK)
begin
	if notasent(11)='1' then i11<="00001"; else i11<="00000";end if;
	if notasent(10)='1' then i10<="00001"; else i10<="00000";end if;
	if notasent(9)='1' then i9<="00001"; else i9<="00000";end if;
	if notasent(8)='1' then i8<="00001"; else i8<="00000";end if;
	if notasent(7)='1' then i7<="00001"; else i7<="00000";end if;
	if notasent(6)='1' then i6<="00001"; else i6<="00000";end if;
	if notasent(5)='1' then i5<="00001"; else i5<="00000";end if;
	if notasent(4)='1' then i4<="00001"; else i4<="00000";end if;
	if notasent(3)='1' then i3<="00001"; else i3<="00000";end if;
	if notasent(2)='1' then i2<="00001"; else i2<="00000";end if;
	if notasent(1)='1' then i1<="00001"; else i1<="00000";end if;
	if notasent(0)='1' then i0<="00001"; else i0<="00000";end if;
end process;
cant_notas <= i11+i10+i9+i8+i7+i6+i5+i4+i3+i2+i1+i0;
Process (CLK)
begin
	if (CLK'event and CLK='1') then
		if	(cant_notas=0) then
			nota1int <= "00000";
			nota2int <= "00000";
		elsif (cant_notas=1) then
			if (notasent(11)='1') then
				nota1int <= "01110";
				nota2int <= "00000";
			elsif (notasent(10)='1') then
				nota1int <= "01101";
				nota2int <= "00000";
			elsif (notasent(9)='1') then
				nota1int <= "01100";
				nota2int <= "00000";
			elsif (notasent(8)='1') then
				nota1int <= "01011";
				nota2int <= "00000";
			elsif (notasent(7)='1') then
				nota1int <= "01010";
				nota2int <= "00000";
			elsif (notasent(6)='1') then
				nota1int <= "01001";
				nota2int <= "00000";
			elsif (notasent(5)='1') then
				nota1int <= "00111";
```

```
                    nota2int <= "00000";
                elsif (notasent(4)='1') then
                    nota1int <= "00110";
                    nota2int <= "00000";
                elsif (notasent(3)='1') then
                    nota1int <= "00101";
                    nota2int <= "00000";
                elsif (notasent(2)='1') then
                    nota1int <= "00100";
                    nota2int <= "00000";
                elsif (notasent(1)='1') then
                    nota1int <= "00011";
                    nota2int <= "00000";
                elsif (notasent(0)='1') then
                    nota1int <= "00010";
                    nota2int <= "00000";
                end if;
            elsif (cant_notas > 1) then
                if (notasent(11)='1') then
                    if notasent(0)='1' then
                        nota1int <= "01110";
                        nota2int <= "00010";
                    elsif notasent(1)='1' then
                            nota1int <= "01110";
                            nota2int <= "00011";
                    elsif notasent(2)='1' then
                            nota1int <= "01110";
                            nota2int <= "00100";
                    elsif notasent(3)='1' then
                            nota1int <= "01110";
                            nota2int <= "00101";
                    elsif notasent(4)='1' then
                            nota1int <= "01110";
                            nota2int <= "00110";
                    elsif notasent(5)='1' then
                            nota1int <= "01110";
                            nota2int <= "00111";
                    elsif notasent(6)='1' then
                            nota1int <= "01110";
                            nota2int <= "01001";
                    elsif notasent(7)='1' then
                            nota1int <= "01110";
                            nota2int <= "01010";
                    elsif notasent(8)='1' then
                            nota1int <= "01110";
                            nota2int <= "01011";
                    elsif notasent(9)='1' then
                            nota1int <= "01110";
                            nota2int <= "01100";
                    elsif notasent(10)='1' then
                            nota1int <= "01110";
                            nota2int <= "01101";
                    end if;
                elsif (notasent(10)='1') then
                    if notasent(0)='1' then
                        nota1int <= "01101";
                        nota2int <= "00010";
                    elsif notasent(1)='1' then
                            nota1int <= "01101";
                            nota2int <= "00011";
                    elsif notasent(2)='1' then
```

```
                                        nota1int <= "01101";
                                        nota2int <= "00100";
                        elsif notasent(3)='1' then
                                        nota1int <= "01101";
                                        nota2int <= "00101";
                        elsif notasent(4)='1' then
                                        nota1int <= "01101";
                                        nota2int <= "00110";
                        elsif notasent(5)='1' then
                                        nota1int <= "01101";
                                        nota2int <= "00111";
                        elsif notasent(6)='1' then
                                        nota1int <= "01101";
                                        nota2int <= "01001";
                        elsif notasent(7)='1' then
                                        nota1int <= "01101";
                                        nota2int <= "01010";
                        elsif notasent(8)='1' then
                                        nota1int <= "01101";
                                        nota2int <= "01011";
                        elsif notasent(9)='1' then
                                        nota1int <= "01101";
                                        nota2int <= "01100";
                        end if;
                elsif (notasent(9)='1') then
                        if notasent(0)='1' then
                                nota1int <= "01100";
                                nota2int <= "00010";
                        elsif notasent(1)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "00011";
                        elsif notasent(2)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "00100";
                        elsif notasent(3)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "00101";
                        elsif notasent(4)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "00110";
                        elsif notasent(5)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "00111";
                        elsif notasent(6)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "01001";
                        elsif notasent(7)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "01010";
                        elsif notasent(8)='1' then
                                        nota1int <= "01100";
                                        nota2int <= "01011";
                        end if;
                elsif (notasent(8)='1') then
                        if notasent(0)='1' then
                                nota1int <= "01011";
                                nota2int <= "00010";
                        elsif notasent(1)='1' then
                                        nota1int <= "01011";
                                        nota2int <= "00011";
                        elsif notasent(2)='1' then
```

```
                                        nota1int <= "01011";
                                        nota2int <= "00100";
                        elsif notasent(3)='1' then
                                        nota1int <= "01011";
                                        nota2int <= "00101";
                        elsif notasent(4)='1' then
                                        nota1int <= "01011";
                                        nota2int <= "00110";
                        elsif notasent(5)='1' then
                                        nota1int <= "01011";
                                        nota2int <= "00111";
                        elsif notasent(6)='1' then
                                        nota1int <= "01011";
                                        nota2int <= "01001";
                        elsif notasent(7)='1' then
                                        nota1int <= "01011";
                                        nota2int <= "01010";
                        end if;
                elsif (notasent(7)='1') then
                        if notasent(0)='1' then
                                nota1int <= "01010";
                                nota2int <= "00010";
                        elsif notasent(1)='1' then
                                        nota1int <= "01010";
                                        nota2int <= "00011";
                        elsif notasent(2)='1' then
                                        nota1int <= "01010";
                                        nota2int <= "00100";
                        elsif notasent(3)='1' then
                                        nota1int <= "01010";
                                        nota2int <= "00101";
                        elsif notasent(4)='1' then
                                        nota1int <= "01010";
                                        nota2int <= "00110";
                        elsif notasent(5)='1' then
                                        nota1int <= "01010";
                                        nota2int <= "00111";
                        elsif notasent(6)='1' then
                                        nota1int <= "01010";
                                        nota2int <= "01001";
                        end if;
                elsif (notasent(6)='1') then
                        if notasent(0)='1' then
                                nota1int <= "01001";
                                nota2int <= "00010";
                        elsif notasent(1)='1' then
                                        nota1int <= "01001";
                                        nota2int <= "00011";
                        elsif notasent(2)='1' then
                                        nota1int <= "01001";
                                        nota2int <= "00100";
                        elsif notasent(3)='1' then
                                        nota1int <= "01001";
                                        nota2int <= "00101";
                        elsif notasent(4)='1' then
                                        nota1int <= "01001";
                                        nota2int <= "00110";
                        elsif notasent(5)='1' then
                                        nota1int <= "01001";
                                        nota2int <= "00111";
                        end if;
```

```
                        elsif (notasent(5)='1') then
                                if notasent(0)='1' then
                                        nota1int <= "00111";
                                        nota2int <= "00010";
                                elsif notasent(1)='1' then
                                                nota1int <= "00111";
                                                nota2int <= "00011";
                                elsif notasent(2)='1' then
                                                nota1int <= "00111";
                                                nota2int <= "00100";
                                elsif notasent(3)='1' then
                                                nota1int <= "00111";
                                                nota2int <= "00101";
                                elsif notasent(4)='1' then
                                                nota1int <= "00111";
                                                nota2int <= "00110";
                                end if;
                        elsif (notasent(4)='1') then
                                if notasent(0)='1' then
                                        nota1int <= "00110";
                                        nota2int <= "00010";
                                elsif notasent(1)='1' then
                                                nota1int <= "00110";
                                                nota2int <= "00011";
                                elsif notasent(2)='1' then
                                                nota1int <= "00110";
                                                nota2int <= "00100";
                                elsif notasent(3)='1' then
                                                nota1int <= "00110";
                                                nota2int <= "00101";
                                end if;
                        elsif (notasent(3)='1') then
                                if notasent(0)='1' then
                                        nota1int <= "00101";
                                        nota2int <= "00010";
                                elsif notasent(1)='1' then
                                                nota1int <= "00101";
                                                nota2int <= "00011";
                                elsif notasent(2)='1' then
                                                nota1int <= "00101";
                                                nota2int <= "00100";
                                end if;
                        elsif (notasent(2)='1') then
                                if notasent(0)='1' then
                                        nota1int <= "00100";
                                        nota2int <= "00010";
                                elsif notasent(1)='1' then
                                                nota1int <= "00100";
                                                nota2int <= "00011";
                                end if;
                        elsif (notasent(1)='1') then
                                        nota1int <= "00100";
                                        nota2int <= "00010";
                        end if;
                end if;
        end if;
end process;
nota1 <= nota1int;
nota2 <= nota2int;
end Behavioral;
```

Sumfin

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity sumfin is
   Port ( Ent1 : in STD_LOGIC_VECTOR (23 downto 0);
          Ent1_2 : in STD_LOGIC_VECTOR (23 downto 0);
          Ent2 : in STD_LOGIC_VECTOR (23 downto 0);
          Ent2_2 : in STD_LOGIC_VECTOR (23 downto 0);
          Sal1 : out STD_LOGIC_VECTOR (23 downto 0);
          Sal2 : out STD_LOGIC_VECTOR (23 downto 0)
          );
end sumfin;
architecture Behavioral of sumfin is
begin
Sal1 <= Ent1 + Ent2;
Sal2 <= Ent1_2 + Ent2_2;
end Behavioral;
```

Invertir

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity invertir is
   Port ( CLK : in  STD_LOGIC;
                notas : in STD_LOGIC_VECTOR (11 downto 0);
          notas_inv : out STD_LOGIC_VECTOR (11 downto 0)                                    );
end invertir;
architecture Behavioral of invertir is
signal Salint : STD_LOGIC_VECTOR (11 downto 0):= (others=>'0');
begin
Process (CLK)
begin
          if (CLK'event and CLK='1') then
                    if          (notas(11)='0') then Salint(11) <= '1';
                    else Salint(11)<='0';
                    end if;
                    if          (notas(10)='0') then Salint(10) <='1';
                    else Salint(10)<='0';
                    end if;
                    if          (notas(9)='0') then Salint(9) <= '1';
                    else Salint(9)<='0';
                    end if;
                    if          (notas(8)='0') then Salint(8) <= '1';
                    else Salint(8)<='0';
                    end if;
                    if          (notas(7)='0') then Salint(7) <= '1';
                    else Salint(7)<='0';
                    end if;
                    if          (notas(6)='0') then Salint(6) <= '1';
                    else Salint(6)<='0';
                    end if;
                    if          (notas(5)='0') then Salint(5) <= '1';
                    else Salint(5)<='0';
                    end if;
```

```
                    if          (notas(4)='0') then Salint(4) <= '1';
                    else Salint(4)<='0';
                    end if;
                    if          (notas(3)='0') then Salint(3) <= '1';
                    else Salint(3)<='0';
                    end if;
                    if          (notas(2)='0') then Salint(2) <= '1';
                    else Salint(2)<='0';
                    end if;
                    if          (notas(1)='0') then Salint(1) <= '1';
                    else Salint(1)<='0';
                    end if;
                    if          (notas(0)='0') then Salint(0) <= '1';
                    else Salint(0)<='0';
                    end if;
          end if;
end process;
notas_inv <= Salint;
end Behavioral;
```

Detección

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity deteccion is
   Port ( CLK : in  STD_LOGIC;                                                notasent : in
STD_LOGIC_VECTOR (11 downto 0);                                          en : in STD_LOGIC;
                    des : out STD_LOGIC_VECTOR (11 downto 0)
                    );
end deteccion;
architecture Behavioral of deteccion is
signal Salint : STD_LOGIC_VECTOR (11 downto 0):= (others=>'0');
begin
Process (CLK)
begin
          if (CLK'event and CLK='1') then
                    if (en='0')then
                              if          (notasent(11)='1') then Salint(11) <= '0';
                              else Salint(11)<='1';
                              end if;
                              if          (notasent(10)='1') then Salint(10) <= '0';
                              else Salint(10)<='1';
                              end if;
                              if          (notasent(9)='1') then Salint(9) <= '0';
                              else Salint(9)<='1';
                              end if;
                              if          (notasent(8)='1') then Salint(8) <= '0';
                              else Salint(8)<='1';
                              end if;
                              if          (notasent(7)='1') then Salint(7) <= '0';
                              else Salint(7)<='1';
                              end if;
                              if          (notasent(6)='1') then Salint(6) <= '0';
                              else Salint(6)<='1';
                              end if;
                              if          (notasent(5)='1') then Salint(5) <= '0';
                              else Salint(5)<='1';
                              end if;
```

```
                        if      (notasent(4)='1') then Salint(4) <= '0';
                        else Salint(11)<='1';
                        end if;
                        if      (notasent(3)='1') then Salint(3) <= '0';
                        else Salint(3)<='1';
                        end if;
                        if      (notasent(2)='1') then Salint(2) <= '0';
                        else Salint(2)<='1';
                        end if;
                        if      (notasent(1)='1') then Salint(1) <= '0';
                        else Salint(1)<='1';
                        end if;
                        if      (notasent(0)='1') then Salint(0) <= '0';
                        else Salint(0)<='1';
                        end if;
                else Salint <= Salint;
                end if;
        end if;
end process;
des <= Salint;
end Behavioral;
```

Selección

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

entity seleccion is
   Port ( sel : in std_logic;
                clk : in std_logic;
                notas_arp :in STD_LOGIC_VECTOR (11 downto 0);
                notas_int :in STD_LOGIC_VECTOR (11 downto 0); -- notas
                                                                interruptor
                notas_sel: out STD_LOGIC_VECTOR (11 downto 0)
                );
end seleccion;
architecture Behavioral of seleccion is
signal inter : STD_LOGIC_VECTOR (11 downto 0);
begin
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                if (sel='1') then inter <= notas_arp;
                else inter <= notas_int;
                end if;
        end if;
end process;
notas_sel <= inter;
end Behavioral;
```

LED2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity LED2 is
   Port ( notas_sel : in  STD_LOGIC_VECTOR (11 downto 0);
          leds : out  STD_LOGIC_VECTOR (11 downto 0));
end LED2;
architecture Behavioral of LED2 is
begin
leds <= notas_sel;
end Behavioral;
```

LED3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity LED3 is
   Port ( INIT : IN std_logic;
          CLK : IN std_logic;
          W_EN : IN std_logic;
          RST : in STD_LOGIC;
          sel : IN std_logic;                                          --modo
          num_inst : in STD_LOGIC_VECTOR (1 downto 0);
          LED_config : out STD_LOGIC_VECTOR (2 downto 0);
          LED_modo : out std_logic;
          LED_inst : out STD_LOGIC_VECTOR (1 downto 0));
end LED3;
architecture Behavioral of LED3 is
signal inter : STD_LOGIC_VECTOR (2 downto 0);
signal inter2 : STD_LOGIC_VECTOR (2 downto 0);
begin
LED_modo <= sel;
LED_inst <= num_inst;
inter <= INIT & W_EN & RST;
Process (CLK)
begin
          if (CLK'event and CLK='1') then
                    if (inter="000") then inter2 <= "101";
                    elsif (inter="001") then inter2 <= "000";
                    elsif (inter="010") then inter2 <= "111";
                    elsif (inter="011") then inter2 <= "000";
                    elsif (inter="100") then inter2 <= "001";
                    elsif (inter="101") then inter2 <= "000";
                    elsif (inter="110") then inter2 <= "011";
                    elsif (inter="111") then inter2 <= "000";
                    end if;
          end if;
end process;
LED_config <= inter2;
end Behavioral;
```

Proyecto

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library work;
use work.componentes.ALL;

entity proyecto is
	Port (	CLK : in  STD_LOGIC;                              -- reloj a 50MHz
		notas : in STD_LOGIC_VECTOR (11 downto 0);
		notas_int : in STD_LOGIC_VECTOR (11 downto 0);
		num_inst : in STD_LOGIC_VECTOR (1 downto 0);
		en : in STD_LOGIC;
		sel : IN std_logic;
		RST : in STD_LOGIC;
		INIT : IN std_logic;
		W_EN : IN std_logic;
		LED_config : out STD_LOGIC_VECTOR (2 downto 0);
		LED_modo : out std_logic;
		LED_inst : out STD_LOGIC_VECTOR (1 downto 0);
		--sim : out STD_LOGIC_VECTOR (23 downto 0);----SOLO SIMULACIONES----
		sal_LED : out  STD_LOGIC_VECTOR (11 downto 0);
		pulse_48KHz : OUT std_logic;                      -- sample sync pulse
		AUD_MCLK : OUT std_logic;                         -- codec master clock input
		AUD_BCLK : OUT std_logic;                         -- digital audio bit clock
		AUD_DACDAT : OUT std_logic;                       -- DAC data lines
		AUD_DACLRCK : OUT std_logic;                      -- DAC data left/right select
		I2C_SDAT : OUT std_logic;                         -- serial interface data line
		I2C_SCLK : OUT std_logic                          -- serial interface clock
		);
end proyecto;
architecture Behavioral of proyecto is
signal CLK2 : STD_LOGIC;
signal CLK3 : STD_LOGIC;
signal CLK4 : STD_LOGIC;
signal des : STD_LOGIC_VECTOR (11 downto 0);
signal notas_inv : STD_LOGIC_VECTOR (11 downto 0);
signal Sintaud : STD_LOGIC_VECTOR (23 downto 0);
signal Sintaud2 : STD_LOGIC_VECTOR (23 downto 0);
signal f1 : STD_LOGIC_VECTOR (8 downto 0);
signal f2 : STD_LOGIC_VECTOR (8 downto 0);
signal f3 : STD_LOGIC_VECTOR (8 downto 0);
signal f4 : STD_LOGIC_VECTOR (8 downto 0);
signal f5 : STD_LOGIC_VECTOR (8 downto 0);
signal f6 : STD_LOGIC_VECTOR (8 downto 0);
signal f7 : STD_LOGIC_VECTOR (8 downto 0);
signal f8 : STD_LOGIC_VECTOR (8 downto 0);
signal f9 : STD_LOGIC_VECTOR (8 downto 0);
signal f10 : STD_LOGIC_VECTOR (8 downto 0);
signal f11 : STD_LOGIC_VECTOR (8 downto 0);
signal f12 : STD_LOGIC_VECTOR (8 downto 0);
signal f13 : STD_LOGIC_VECTOR (8 downto 0);
signal f14 : STD_LOGIC_VECTOR (8 downto 0);
signal f15 : STD_LOGIC_VECTOR (8 downto 0);
signal f16 : STD_LOGIC_VECTOR (8 downto 0);
signal f17 : STD_LOGIC_VECTOR (8 downto 0);
signal f18 : STD_LOGIC_VECTOR (8 downto 0);
signal f19 : STD_LOGIC_VECTOR (8 downto 0);
```

```
signal af : STD_LOGIC_VECTOR (7 downto 0);
signal a1 : STD_LOGIC_VECTOR (7 downto 0);
signal a2 : STD_LOGIC_VECTOR (7 downto 0);
signal a3 : STD_LOGIC_VECTOR (7 downto 0);
signal a4 : STD_LOGIC_VECTOR (7 downto 0);
signal a5 : STD_LOGIC_VECTOR (7 downto 0);
signal a6 : STD_LOGIC_VECTOR (7 downto 0);
signal a7 : STD_LOGIC_VECTOR (7 downto 0);
signal a8 : STD_LOGIC_VECTOR (7 downto 0);
signal a9 : STD_LOGIC_VECTOR (7 downto 0);
signal a10 : STD_LOGIC_VECTOR (7 downto 0);
signal a11 : STD_LOGIC_VECTOR (7 downto 0);
signal a12 : STD_LOGIC_VECTOR (7 downto 0);
signal a13 : STD_LOGIC_VECTOR (7 downto 0);
signal a14 : STD_LOGIC_VECTOR (7 downto 0);
signal a15 : STD_LOGIC_VECTOR (7 downto 0);
signal a16 : STD_LOGIC_VECTOR (7 downto 0);
signal a17 : STD_LOGIC_VECTOR (7 downto 0);
signal a18 : STD_LOGIC_VECTOR (7 downto 0);
signal a19 : STD_LOGIC_VECTOR (7 downto 0);
signal cont : STD_LOGIC_VECTOR (4 downto 0);
signal ince : integer;
signal inc : STD_LOGIC_VECTOR (17 downto 0);
signal armf : STD_LOGIC_VECTOR (19 downto 0);
signal arm1 : STD_LOGIC_VECTOR (19 downto 0);
signal arm2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm3 : STD_LOGIC_VECTOR (19 downto 0);
signal arm4 : STD_LOGIC_VECTOR (19 downto 0);
signal arm5 : STD_LOGIC_VECTOR (19 downto 0);
signal arm6 : STD_LOGIC_VECTOR (19 downto 0);
signal arm7 : STD_LOGIC_VECTOR (19 downto 0);
signal arm8 : STD_LOGIC_VECTOR (19 downto 0);
signal arm9 : STD_LOGIC_VECTOR (19 downto 0);
signal arm10 : STD_LOGIC_VECTOR (19 downto 0);
signal arm11 : STD_LOGIC_VECTOR (19 downto 0);
signal arm12 : STD_LOGIC_VECTOR (19 downto 0);
signal arm13 : STD_LOGIC_VECTOR (19 downto 0);
signal arm14 : STD_LOGIC_VECTOR (19 downto 0);
signal arm15 : STD_LOGIC_VECTOR (19 downto 0);
signal arm16 : STD_LOGIC_VECTOR (19 downto 0);
signal arm17 : STD_LOGIC_VECTOR (19 downto 0);
signal arm18 : STD_LOGIC_VECTOR (19 downto 0);
signal arm19 : STD_LOGIC_VECTOR (19 downto 0);
signal Bf : STD_LOGIC_VECTOR (19 downto 0);
signal B1 : STD_LOGIC_VECTOR (19 downto 0);
signal B2 : STD_LOGIC_VECTOR (19 downto 0);
signal B3 : STD_LOGIC_VECTOR (19 downto 0);
signal B4 : STD_LOGIC_VECTOR (19 downto 0);
signal B5 : STD_LOGIC_VECTOR (19 downto 0);
signal B6 : STD_LOGIC_VECTOR (19 downto 0);
signal B7 : STD_LOGIC_VECTOR (19 downto 0);
signal B8 : STD_LOGIC_VECTOR (19 downto 0);
signal B9 : STD_LOGIC_VECTOR (19 downto 0);
signal B10 : STD_LOGIC_VECTOR (19 downto 0);
signal B11 : STD_LOGIC_VECTOR (19 downto 0);
signal B12 : STD_LOGIC_VECTOR (19 downto 0);
signal B13 : STD_LOGIC_VECTOR (19 downto 0);
signal B14 : STD_LOGIC_VECTOR (19 downto 0);
signal B15 : STD_LOGIC_VECTOR (19 downto 0);
signal B16 : STD_LOGIC_VECTOR (19 downto 0);
signal B17 : STD_LOGIC_VECTOR (19 downto 0);
```

```
signal B18 : STD_LOGIC_VECTOR (19 downto 0);
signal B19 : STD_LOGIC_VECTOR (19 downto 0);
signal Bf_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B1_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B2_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B3_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B4_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B5_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B6_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B7_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B8_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B9_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B10_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B11_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B12_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B13_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B14_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B15_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B16_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B17_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B18_3 : STD_LOGIC_VECTOR (8 downto 0);
signal B19_3 : STD_LOGIC_VECTOR (8 downto 0);
signal Bf_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B1_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B2_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B3_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B4_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B5_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B6_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B7_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B8_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B9_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B10_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B11_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B12_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B13_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B14_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B15_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B16_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B17_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B18_4 : STD_LOGIC_VECTOR (8 downto 0);
signal B19_4 : STD_LOGIC_VECTOR (8 downto 0);
signal dir_mem : STD_LOGIC_VECTOR (8 downto 0);
signal sensal : integer;
signal Bf_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B1_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B2_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B3_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B4_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B5_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B6_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B7_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B8_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B9_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B10_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B11_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B12_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B13_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B14_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B15_5 : STD_LOGIC_VECTOR (7 downto 0);
signal B16_5 : STD_LOGIC_VECTOR (7 downto 0);
```

```
signal B16_6 : STD_LOGIC_VECTOR (15 downto 0);
signal B17_6 : STD_LOGIC_VECTOR (15 downto 0);
signal B18_6 : STD_LOGIC_VECTOR (15 downto 0);
signal B19_6 : STD_LOGIC_VECTOR (15 downto 0);
signal Bf_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B1_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B2_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B3_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B4_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B5_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B6_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B7_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B8_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B9_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B10_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B11_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B12_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B13_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B14_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B15_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B16_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B17_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B18_7 : STD_LOGIC_VECTOR (15 downto 0);
signal B19_7 : STD_LOGIC_VECTOR (15 downto 0);
signal Salsum : STD_LOGIC_VECTOR (19 downto 0);
signal notasent : STD_LOGIC_VECTOR (11 downto 0);
signal notas_sel : STD_LOGIC_VECTOR (11 downto 0);
signal nota1 : STD_LOGIC_VECTOR (4 downto 0);
signal nota2 : STD_LOGIC_VECTOR (4 downto 0);
signal Sal1 : STD_LOGIC_VECTOR (23 downto 0);
signal Sal2 : STD_LOGIC_VECTOR (23 downto 0);
signal Sintaud_2 : STD_LOGIC_VECTOR (23 downto 0);
signal Sintaud2_2 : STD_LOGIC_VECTOR (23 downto 0);
signal f1_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f2_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f5_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f6_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f7_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f8_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f9_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f10_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f11_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f12_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f13_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f14_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f15_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f16_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f17_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f18_2 : STD_LOGIC_VECTOR (8 downto 0);
signal f19_2 : STD_LOGIC_VECTOR (8 downto 0);
signal af_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a1_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a2_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a3_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a4_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a6_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a7_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a8_2 : STD_LOGIC_VECTOR (7 downto 0);
```

```
signal a9_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a10_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a11_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a12_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a13_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a14_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a15_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a16_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a17_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a18_2 : STD_LOGIC_VECTOR (7 downto 0);
signal a19_2 : STD_LOGIC_VECTOR (7 downto 0);
signal ince_2 : integer;
signal inc_2 : STD_LOGIC_VECTOR (17 downto 0);
signal armf_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm1_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm2_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm3_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm4_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm5_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm6_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm7_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm8_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm9_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm10_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm11_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm12_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm13_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm14_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm15_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm16_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm17_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm18_2 : STD_LOGIC_VECTOR (19 downto 0);
signal arm19_2 : STD_LOGIC_VECTOR (19 downto 0);
signal Bf_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B1_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B2_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B3_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B4_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B5_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B6_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B7_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B8_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B9_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B10_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B11_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B12_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B13_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B14_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B15_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B16_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B17_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B18_2 : STD_LOGIC_VECTOR (19 downto 0);
signal B19_2 : STD_LOGIC_VECTOR (19 downto 0);
signal Bf_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B1_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B2_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B3_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B4_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B5_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B6_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B7_3_2 : STD_LOGIC_VECTOR (8 downto 0);
```

```
signal B8_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B9_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B10_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B11_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B12_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B13_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B14_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B15_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B16_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B17_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B18_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B19_3_2 : STD_LOGIC_VECTOR (8 downto 0);
signal Bf_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B1_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B2_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B3_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B4_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B5_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B6_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B7_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B8_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B9_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B10_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B11_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B12_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B13_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B14_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B15_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B16_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B17_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B18_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal B19_4_2 : STD_LOGIC_VECTOR (8 downto 0);
signal dir_mem_2 : STD_LOGIC_VECTOR (8 downto 0);
signal sensal_2 : integer;
signal Bf_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B1_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B2_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B3_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B4_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B5_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B6_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B7_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B8_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B9_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B10_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B11_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B12_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B13_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B14_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B15_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B16_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B17_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B18_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B19_5_2 : STD_LOGIC_VECTOR (7 downto 0);
signal Bf_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B1_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B2_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B3_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B4_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B5_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B6_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
```

```
signal B7_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B8_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B9_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B10_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B11_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B12_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B13_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B14_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B15_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B16_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B17_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B18_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal B19_FF_2 : STD_LOGIC_VECTOR (7 downto 0);
signal Bf_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B1_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B2_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B3_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B4_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B5_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B6_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B7_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B8_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B9_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B10_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B11_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B12_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B13_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B14_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B15_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B16_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B17_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B18_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B19_6_2 : STD_LOGIC_VECTOR (15 downto 0);
signal Bf_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B1_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B2_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B3_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B4_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B5_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B6_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B7_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B8_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B9_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B10_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B11_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B12_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B13_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B14_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B15_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B16_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B17_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B18_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal B19_7_2 : STD_LOGIC_VECTOR (15 downto 0);
signal Salsum_2 : STD_LOGIC_VECTOR (19 downto 0);
begin
U1 : divisor1 Port map (CLK, RST, CLK2);--divisor aprox 48000Hz
U2 : divisor2 Port map (CLK, RST, CLK3);--divisor 25MHz
U3 : divisor3 Port map (CLK, RST, CLK4);--divisor 1MHz
U4 : g00_audio_interface  PORT map(Sal1 , Sal2, CLK3, RST,INIT, W_EN,    --interfaz de audio
                                   pulse_48KHz, AUD_MCLK, AUD_BCLK,
                                   AUD_DACDAT, AUD_DACLRCK, I2C_SDAT,
```

```
--U5 : LED Port map (nota1, sal_LED);
U5 : LED2 Port map ( notas_sel,sal_LED );
U6 : fase Port map(CLK2,nota1,num_inst,f1,f2,f3,f4,f5,f6,f7,f8,
f9,f10,f11,f12,f13,f14,f15,f16,f17,f18,f19);
U6 : fase Port map (CLK2,nota1,num_inst,f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,
                    f11,f12,f13,f14,f15,f16,f17,f18,f19);
U7 : amplitud Port map (CLK2,nota1,num_inst,af,a1,a2,a3,a4,a5,a6,a7,a8,
                        a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19);
U8 : control Port map (CLK4,RST,cont);                          -- señales de control
U9 : incrementos Port map (CLK2,nota1,num_inst,ince,inc);
U10 : arm Port map (CLK2,nota1,num_inst,RST,ince,inc,armf,arm1,arm2,       -- Calculo armonicos
                    arm3,arm4,arm5,arm6,arm7,arm8,arm9,arm10,arm11,arm12,
                    arm13,arm14,arm15,arm16,arm17,arm18,arm19);
U11 : acum Port map (CLK2,RST,armf,arm1,arm2,arm3,arm4,arm5,arm6,arm7,-- acumulador
                     arm8,arm9,arm10,arm11,arm12,arm13,arm14,arm15,arm16,
                     arm17,arm18,arm19,Bf,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,
                     B11,B12,B13,B14,B15,B16,B17,B18,B19);
U12 : sumfas Port map (f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,          -- suma fase
                       f16,f17,f18,f19,Bf,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,
                       B11,B12,B13,B14,B15,B16,B17,B18,B19,Bf_3,B1_3,B2_3,
                       B3_3,B4_3,B5_3,B6_3,B7_3,B8_3,B9_3,B10_3,B11_3,B12_3,
                       B13_3,B14_3,B15_3,B16_3,B17_3,B18_3,B19_3);
U13 : FF1 Port map (CLK2,Bf_3,B1_3,B2_3,B3_3,B4_3,B5_3,B6_3,B7_3,B8_3,B9_3,
                    B10_3,B11_3,B12_3,B13_3,B14_3,B15_3,B16_3,B17_3,B18_3,
                    B19_3,Bf_4,B1_4,B2_4,B3_4,B4_4,B5_4,B6_4,B7_4,B8_4,B9_4,
                    B10_4,B11_4,B12_4,B13_4,B14_4,B15_4,B16_4,B17_4,B18_4,
                    B19_4);
U14 : mux1 Port map (CLK4,cont,Bf_4,B1_4,B2_4,B3_4,B4_4,B5_4,B6_4,B7_4,B8_4,
                     B9_4,B10_4,B11_4,B12_4,B13_4,B14_4,B15_4,B16_4,B17_4,
                     B18_4,B19_4,dir_mem);
U15 : sen Port map (dir_mem,sensal);
U16 : DMUX Port map (CLK4,cont,sensal,Bf_5,B1_5,B2_5,B3_5,B4_5,B5_5,B6_5,
                     B7_5,B8_5,B9_5,B10_5,B11_5,B12_5,B13_5,B14_5,B15_5,
                     B16_5,B17_5,B18_5,B19_5);
U17 : FF2 Port map (CLK4,cont,Bf_5,B1_5,B2_5,B3_5,B4_5,B5_5,B6_5,B7_5,B8_5,
                    B9_5,B10_5,B11_5,B12_5,B13_5,B14_5,B15_5,B16_5,B17_5,
                    B18_5,B19_5,Bf_FF,B1_FF,B2_FF,B3_FF,B4_FF,B5_FF,B6_FF,
                    B7_FF,B8_FF,B9_FF,B10_FF,B11_FF,B12_FF,B13_FF,B14_FF,
                    B15_FF,B16_FF,B17_FF,B18_FF,B19_FF);
U18 : mult Port map (af,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,    --mult por la amplitud
                     a16,a17,a18,a19,Bf_FF,B1_FF,B2_FF,B3_FF,B4_FF,B5_FF,
                     B6_FF,B7_FF,B8_FF,B9_FF,B10_FF,B11_FF,B12_FF,B13_FF,
                     B14_FF,B15_FF,B16_FF,B17_FF,B18_FF,B19_FF,Bf_6,B1_6,
                     B2_6,B3_6,B4_6,B5_6,B6_6,B7_6,B8_6,B9_6,B10_6,B11_6,
                     B12_6,B13_6,B14_6,B15_6,B16_6,B17_6,B18_6,B19_6);
U19 : offset Port map (af,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,
                       a15,a16,a17,a18,a19,Bf_6,B1_6,B2_6,B3_6,B4_6,B5_6,
                       B6_6,B7_6,B8_6,B9_6,B10_6,B11_6,B12_6,B13_6,B14_6,
                       B15_6,B16_6,B17_6,B18_6,B19_6,Bf_7,B1_7,B2_7,B3_7,
                       B4_7,B5_7,B6_7,B7_7,B8_7,B9_7,B10_7,B11_7,B12_7,
                       B13_7,B14_7,B15_7,B16_7,B17_7,B18_7,B19_7);
U20 : suma Port map (Bf_7,B1_7,B2_7,B3_7,B4_7,B5_7,B6_7,B7_7,B8_7,B9_7,B10_7,
                     B11_7,B12_7,B13_7,B14_7,B15_7,B16_7,B17_7,B18_7,B19_7,
                     Salsum);
U21 : ajuste Port map (Salsum,Sintaud,Sintaud2);                    -- ajuste del volumen
U42:  invertir Port map ( CLK2,notas,notas_inv);
U40 : deteccion Port map (CLK2,notas_inv,en,des);                   -- detectar fundidos
U22 : arp Port map (CLK2,notas_inv,des,notasent);                   -- desactivar senyales
U41 : seleccion Port map (sel,clk2,notasent,notas_int,notas_sel);   -- seleccionar notas
                                                                    arp o notas interupt
U23 : calc_notas Port map (CLK2,notas_sel,nota1,nota2);             -- Convertir a notas
```

```
U24 : fase Port map (CLK2,nota2,num_inst,f1_2,f2_2,f3_2,f4_2,f5_2,f6_2,f7_2,
                     f8_2,f9_2,f10_2,f11_2,f12_2,f13_2,f14_2,f15_2,f16_2,
                     f17_2,f18_2,f19_2);
U25 : amplitud Port map (CLK2,nota2,num_inst,af_2,a1_2,a2_2,a3_2,a4_2,a5_2,
                         a6_2,a7_2,a8_2,a9_2,a10_2,a11_2,a12_2,a13_2,a14_2,
                         a15_2,a16_2,a17_2,a18_2,a19_2);
U26 : incrementos Port map (CLK2,nota2,num_inst,ince_2,inc_2);
U27 : arm Port map (CLK2,nota2,num_inst,RST,ince_2,inc_2,armf_2,arm1_2,
                    arm2_2,arm3_2,arm4_2,arm5_2,arm6_2,arm7_2,arm8_2,
                    arm9_2,arm10_2,arm11_2,arm12_2,arm13_2,arm14_2,arm15_2,
                    arm16_2,arm17_2,arm18_2,arm19_2);
U28 : acum Port map (CLK2,RST,armf_2,arm1_2,arm2_2,arm3_2,arm4_2,arm5_2,
                     arm6_2,arm7_2,arm8_2,arm9_2,arm10_2,arm11_2,arm12_2,
                     arm13_2,arm14_2,arm15_2,arm16_2,arm17_2,arm18_2,arm19_2,
                     Bf_2,B1_2,B2_2,B3_2,B4_2,B5_2,B6_2,B7_2,B8_2,B9_2,B10_2,
                     B11_2,B12_2,B13_2,B14_2,B15_2,B16_2,B17_2,B18_2,B19_2);
U29 : sumfas Port map (f1_2,f2_2,f3_2,f4_2,f5_2,f6_2,f7_2,f8_2,f9_2,f10_2,
                       f11_2,f12_2,f13_2,f14_2,f15_2,f16_2,f17_2,f18_2,f19_2,
                       Bf_2,B1_2,B2_2,B3_2,B4_2,B5_2,B6_2,B7_2,B8_2,B9_2,B10_2,
                       B11_2,B12_2,B13_2,B14_2,B15_2,B16_2,B17_2,B18_2,B19_2,
                       Bf_3_2,B1_3_2,B2_3_2,B3_3_2,B4_3_2,B5_3_2,B6_3_2,B7_3_2,
                       B8_3_2,B9_3_2,B10_3_2,B11_3_2,B12_3_2,B13_3_2,B14_3_2,
                       B15_3_2,B16_3_2,B17_3_2,B18_3_2,B19_3_2);
U30 : FF1 Port map (CLK2,Bf_3_2,B1_3_2,B2_3_2,B3_3_2,B4_3_2,B5_3_2,B6_3_2,
                    B7_3_2,B8_3_2,B9_3_2,B10_3_2,B11_3_2,B12_3_2,B13_3_2,
                    B14_3_2,B15_3_2,B16_3_2,B17_3_2,B18_3_2,B19_3_2,Bf_4_2,
                    B1_4_2,B2_4_2,B3_4_2,B4_4_2,B5_4_2,B6_4_2,B7_4_2,B8_4_2,
                    B9_4_2,B10_4_2,B11_4_2,B12_4_2,B13_4_2,B14_4_2,B15_4_2,
                    B16_4_2,B17_4_2,B18_4_2,B19_4_2);
U31 : mux1 Port map (CLK4,cont,Bf_4_2,B1_4_2,B2_4_2,B3_4_2,B4_4_2,B5_4_2,
                     B6_4_2,B7_4_2,B8_4_2,B9_4_2,B10_4_2,B11_4_2,B12_4_2,
                     B13_4_2,B14_4_2,B15_4_2,B16_4_2,B17_4_2,B18_4_2,B19_4_2,
                     dir_mem_2);
U32 : sen Port map (dir_mem_2,sensal_2);
U33 : DMUX Port map (CLK4,cont,sensal_2,Bf_5_2,B1_5_2,B2_5_2,B3_5_2,B4_5_2,
                     B5_5_2,B6_5_2,B7_5_2,B8_5_2,B9_5_2,B10_5_2,B11_5_2,
                     B12_5_2,B13_5_2,B14_5_2,B15_5_2,B16_5_2,B17_5_2,B18_5_2,
                     B19_5_2);
U34 : FF2 Port map (CLK4,cont,Bf_5_2,B1_5_2,B2_5_2,B3_5_2,B4_5_2,B5_5_2,
                    B6_5_2,B7_5_2,B8_5_2,B9_5_2,B10_5_2,B11_5_2,B12_5_2,
                    B13_5_2,B14_5_2,B15_5_2,B16_5_2,B17_5_2,B18_5_2,B19_5_2,
                    Bf_FF_2,B1_FF_2,B2_FF_2,B3_FF_2,B4_FF_2,B5_FF_2,B6_FF_2,
                    B7_FF_2,B8_FF_2,B9_FF_2,B10_FF_2,B11_FF_2,B12_FF_2,
                    B13_FF_2,B14_FF_2,B15_FF_2,B16_FF_2,B17_FF_2,B18_FF_2,
                    B19_FF_2);
U35 : mult Port map (af_2,a1_2,a2_2,a3_2,a4_2,a5_2,a6_2,a7_2,a8_2,a9_2,a10_2,
                     a11_2,a12_2,a13_2,a14_2,a15_2,a16_2,a17_2,a18_2,a19_2,
                     Bf_FF_2,B1_FF_2,B2_FF_2,B3_FF_2,B4_FF_2,B5_FF_2,B6_FF_2,
                     B7_FF_2,B8_FF_2,B9_FF_2,B10_FF_2,B11_FF_2,B12_FF_2,
                     B13_FF_2,B14_FF_2,B15_FF_2,B16_FF_2,B17_FF_2,B18_FF_2,
                     B19_FF_2,Bf_6_2,B1_6_2,B2_6_2,B3_6_2,B4_6_2,B5_6_2,B6_6_2,
                     B7_6_2,B8_6_2,B9_6_2,B10_6_2,B11_6_2,B12_6_2,B13_6_2,
                     B14_6_2,B15_6_2,B16_6_2,B17_6_2,B18_6_2,B19_6_2);
U36 : offset Port map (af_2,a1_2,a2_2,a3_2,a4_2,a5_2,a6_2,a7_2,a8_2,a9_2,a10_2,
                       a11_2,a12_2,a13_2,a14_2,a15_2,a16_2,a17_2,a18_2,a19_2,
                       Bf_6_2,B1_6_2,B2_6_2,B3_6_2,B4_6_2,B5_6_2,B6_6_2,B7_6_2,
                       B8_6_2,B9_6_2,B10_6_2,B11_6_2,B12_6_2,B13_6_2,B14_6_2,
                       B15_6_2,B16_6_2,B17_6_2,B18_6_2,B19_6_2,Bf_7_2,B1_7_2,
                       B2_7_2,B3_7_2,B4_7_2,B5_7_2,B6_7_2,B7_7_2,B8_7_2,B9_7_2,
                       B10_7_2,B11_7_2,B12_7_2,B13_7_2,B14_7_2,B15_7_2,B16_7_2,
                       B17_7_2,B18_7_2,B19_7_2);
```

```
U37 : suma Port map (Bf_7_2,B1_7_2,B2_7_2,B3_7_2,B4_7_2,B5_7_2,B6_7_2,B7_7_2,
                     B8_7_2,B9_7_2,B10_7_2,B11_7_2,B12_7_2,B13_7_2,B14_7_2,
                     B15_7_2,B16_7_2,B17_7_2,B18_7_2,B19_7_2,Salsum_2);
U38 : ajuste Port map (Salsum_2,Sintaud_2,Sintaud2_2);
U39 : sumfin Port map (Sintaud,Sintaud2,Sintaud_2,Sintaud2_2,Sal1,Sal2);
sim <= Sal1;       -- para las simulaciones
U43 : LED3 Port map ( INIT,CLK, W_EN , RST ,sel, num_inst,LED_config,LED_modo,LED_inst);
end Behavioral;
```

Componentes

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
package componentes is
component divisor1 is
        Port ( CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        salida : out STD_LOGIC
         );
end component;
component divisor2 is
        Port ( CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        salida : out STD_LOGIC);
end component;
component divisor3 is
        Port ( CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        salida : out STD_LOGIC);
end component;
component g00_audio_interface IS
        PORT(  LDATA, RDAT: IN std_logic_vector(23 downto 0); --parallel external data inputs
        clk, rst, INIT, W_EN : IN std_logic; -- clk should be 24MHz
        pulse_48KHz : OUT std_logic; -- sample sync pulse
        AUD_MCLK : OUT std_logic; -- codec master clock input
        AUD_BCLK : OUT std_logic; -- digital audio bit clock
        AUD_DACDAT : OUT std_logic; -- DAC data lines
        AUD_DACLRCK : OUT std_logic; -- DAC data left/right select
        I2C_SDAT : OUT std_logic; -- serial interface data line
        I2C_SCLK : OUT std_logic  -- serial interface clock
        );
END component;
component LED is
        Port ( entrada : in  STD_LOGIC_VECTOR (4 downto 0);
        salida : out  STD_LOGIC_VECTOR (17 downto 0));
end component;
component fase is
        Port (     CLK : in  STD_LOGIC;
        num_nota : in STD_LOGIC_VECTOR (4 downto 0);
        num_inst : in STD_LOGIC_VECTOR (1 downto 0);
        f1 : out STD_LOGIC_VECTOR (8 downto 0);
        f2 : out STD_LOGIC_VECTOR (8 downto 0);
        f3 : out STD_LOGIC_VECTOR (8 downto 0);
        f4 : out STD_LOGIC_VECTOR (8 downto 0);
        f5 : out STD_LOGIC_VECTOR (8 downto 0);
        f6 : out STD_LOGIC_VECTOR (8 downto 0);
        f7 : out STD_LOGIC_VECTOR (8 downto 0);
        f8 : out STD_LOGIC_VECTOR (8 downto 0);
        f9 : out STD_LOGIC_VECTOR (8 downto 0);
        f10 : out STD_LOGIC_VECTOR (8 downto 0);
```

```
        f11 : out STD_LOGIC_VECTOR (8 downto 0);
        f12 : out STD_LOGIC_VECTOR (8 downto 0);
        f13 : out STD_LOGIC_VECTOR (8 downto 0);
        f14 : out STD_LOGIC_VECTOR (8 downto 0);
        f15 : out STD_LOGIC_VECTOR (8 downto 0);
        f16 : out STD_LOGIC_VECTOR (8 downto 0);
        f17 : out STD_LOGIC_VECTOR (8 downto 0);
        f18 : out STD_LOGIC_VECTOR (8 downto 0);
        f19 : out STD_LOGIC_VECTOR (8 downto 0));
end component;
component amplitud is
        Port ( CLK : in  STD_LOGIC;
        num_nota : in STD_LOGIC_VECTOR (4 downto 0);
        num_inst : in STD_LOGIC_VECTOR (1 downto 0);
        af : out STD_LOGIC_VECTOR (7 downto 0);
        a1 : out STD_LOGIC_VECTOR (7 downto 0);
        a2 : out STD_LOGIC_VECTOR (7 downto 0);
        a3 : out STD_LOGIC_VECTOR (7 downto 0);
        a4 : out STD_LOGIC_VECTOR (7 downto 0);
        a5 : out STD_LOGIC_VECTOR (7 downto 0);
        a6 : out STD_LOGIC_VECTOR (7 downto 0);
        a7 : out STD_LOGIC_VECTOR (7 downto 0);
        a8 : out STD_LOGIC_VECTOR (7 downto 0);
        a9 : out STD_LOGIC_VECTOR (7 downto 0);
        a10 : out STD_LOGIC_VECTOR (7 downto 0);
        a11 : out STD_LOGIC_VECTOR (7 downto 0);
        a12 : out STD_LOGIC_VECTOR (7 downto 0);
        a13 : out STD_LOGIC_VECTOR (7 downto 0);
        a14 : out STD_LOGIC_VECTOR (7 downto 0);
        a15 : out STD_LOGIC_VECTOR (7 downto 0);
        a16 : out STD_LOGIC_VECTOR (7 downto 0);
        a17 : out STD_LOGIC_VECTOR (7 downto 0);
        a18 : out STD_LOGIC_VECTOR (7 downto 0);
        a19 : out STD_LOGIC_VECTOR (7 downto 0));
end component;
component control is
        Port (     CLK : in STD_LOGIC;
        RST : in STD_LOGIC;
        cont : out STD_LOGIC_VECTOR (4 downto 0));
end component;
component incrementos is
        Port (     CLK : in  STD_LOGIC;
        num_nota : in STD_LOGIC_VECTOR (4 downto 0);
        num_inst : in STD_LOGIC_VECTOR (1 downto 0);
        Sale : out integer;
        Sal : out STD_LOGIC_VECTOR (17 downto 0));
end component;
component arm is
        Port (     CLK : in  STD_LOGIC;
        num_nota : in STD_LOGIC_VECTOR (4 downto 0);
        num_inst : in STD_LOGIC_VECTOR (1 downto 0);
        RST : in STD_LOGIC;
        INCe : in integer;
        INC : in STD_LOGIC_VECTOR (17 downto 0);
        Bf : out STD_LOGIC_VECTOR (19 downto 0);
        B1 : out STD_LOGIC_VECTOR (19 downto 0);
        B2 : out STD_LOGIC_VECTOR (19 downto 0);
        B3 : out STD_LOGIC_VECTOR (19 downto 0);
        B4 : out STD_LOGIC_VECTOR (19 downto 0);
        B5 : out STD_LOGIC_VECTOR (19 downto 0);
        B6 : out STD_LOGIC_VECTOR (19 downto 0);
```

```
        B7 : out STD_LOGIC_VECTOR (19 downto 0);
        B8 : out STD_LOGIC_VECTOR (19 downto 0);
        B9 : out STD_LOGIC_VECTOR (19 downto 0);
        B10 : out STD_LOGIC_VECTOR (19 downto 0);
        B11 : out STD_LOGIC_VECTOR (19 downto 0);
        B12 : out STD_LOGIC_VECTOR (19 downto 0);
        B13 : out STD_LOGIC_VECTOR (19 downto 0);
        B14 : out STD_LOGIC_VECTOR (19 downto 0);
        B15 : out STD_LOGIC_VECTOR (19 downto 0);
        B16 : out STD_LOGIC_VECTOR (19 downto 0);
        B17 : out STD_LOGIC_VECTOR (19 downto 0);
        B18 : out STD_LOGIC_VECTOR (19 downto 0);
        B19 : out STD_LOGIC_VECTOR (19 downto 0));
end component;
component acum is
        Port ( CLK : in  STD_LOGIC;
        RST : in STD_LOGIC;
        Bf_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B1_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B2_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B3_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B4_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B5_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B6_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B7_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B8_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B9_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B10_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B11_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B12_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B13_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B14_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B15_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B16_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B17_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B18_1 : in STD_LOGIC_VECTOR (19 downto 0);
        B19_1 : in STD_LOGIC_VECTOR (19 downto 0);
        Bf : out STD_LOGIC_VECTOR (19 downto 0);
        B1 : out STD_LOGIC_VECTOR (19 downto 0);
        B2 : out STD_LOGIC_VECTOR (19 downto 0);
        B3 : out STD_LOGIC_VECTOR (19 downto 0);
        B4 : out STD_LOGIC_VECTOR (19 downto 0);
        B5 : out STD_LOGIC_VECTOR (19 downto 0);
        B6 : out STD_LOGIC_VECTOR (19 downto 0);
        B7 : out STD_LOGIC_VECTOR (19 downto 0);
        B8 : out STD_LOGIC_VECTOR (19 downto 0);
        B9 : out STD_LOGIC_VECTOR (19 downto 0);
        B10 : out STD_LOGIC_VECTOR (19 downto 0);
        B11 : out STD_LOGIC_VECTOR (19 downto 0);
        B12 : out STD_LOGIC_VECTOR (19 downto 0);
        B13 : out STD_LOGIC_VECTOR (19 downto 0);
        B14 : out STD_LOGIC_VECTOR (19 downto 0);
        B15 : out STD_LOGIC_VECTOR (19 downto 0);
        B16 : out STD_LOGIC_VECTOR (19 downto 0);
        B17 : out STD_LOGIC_VECTOR (19 downto 0);
        B17 : out STD_LOGIC_VECTOR (19 downto 0);
        B18 : out STD_LOGIC_VECTOR (19 downto 0);
        B19 : out STD_LOGIC_VECTOR (19 downto 0)
        );
end component;
```

```
component sumfas is
        Port (     fase1 : in STD_LOGIC_VECTOR (8 downto 0);
        fase2 : in STD_LOGIC_VECTOR (8 downto 0);
        fase3 : in STD_LOGIC_VECTOR (8 downto 0);
        fase4 : in STD_LOGIC_VECTOR (8 downto 0);
        fase5 : in STD_LOGIC_VECTOR (8 downto 0);
        fase6 : in STD_LOGIC_VECTOR (8 downto 0);
        fase7 : in STD_LOGIC_VECTOR (8 downto 0);
        fase8 : in STD_LOGIC_VECTOR (8 downto 0);
        fase9 : in STD_LOGIC_VECTOR (8 downto 0);
        fase10 : in STD_LOGIC_VECTOR (8 downto 0);
        fase11 : in STD_LOGIC_VECTOR (8 downto 0);
        fase12 : in STD_LOGIC_VECTOR (8 downto 0);
        fase13 : in STD_LOGIC_VECTOR (8 downto 0);
        fase14 : in STD_LOGIC_VECTOR (8 downto 0);
        fase15 : in STD_LOGIC_VECTOR (8 downto 0);
        fase16 : in STD_LOGIC_VECTOR (8 downto 0);
        fase17 : in STD_LOGIC_VECTOR (8 downto 0);
        fase18 : in STD_LOGIC_VECTOR (8 downto 0);
        fase19 : in STD_LOGIC_VECTOR (8 downto 0);
        Bf_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B1_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B2_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B3_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B4_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B5_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B6_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B7_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B8_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B9_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B10_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B11_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B12_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B13_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B14_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B15_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B16_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B17_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B18_2 : in STD_LOGIC_VECTOR (19 downto 0);
        B19_2 : in STD_LOGIC_VECTOR (19 downto 0);
        Bf_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B1_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B2_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B3_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B4_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B5_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B6_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B7_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B8_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B9_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B10_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B11_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B12_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B13_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B14_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B15_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B16_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B17_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B18_3 : out STD_LOGIC_VECTOR (8 downto 0);
        B19_3 : out STD_LOGIC_VECTOR (8 downto 0));
end component;
```

```
component FF1 is
	Port (	CLK : in  STD_LOGIC;
	Bf_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B1_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B2_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B3_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B4_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B5_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B6_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B7_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B8_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B9_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B10_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B11_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B12_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B13_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B14_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B15_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B16_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B17_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B18_3 : in STD_LOGIC_VECTOR (8 downto 0);
	B19_3 : in STD_LOGIC_VECTOR (8 downto 0);
	Bf : out STD_LOGIC_VECTOR (8 downto 0);
	B1 : out STD_LOGIC_VECTOR (8 downto 0);
	B2 : out STD_LOGIC_VECTOR (8 downto 0);
	B3 : out STD_LOGIC_VECTOR (8 downto 0);
	B4 : out STD_LOGIC_VECTOR (8 downto 0);
	B5 : out STD_LOGIC_VECTOR (8 downto 0);
	B6 : out STD_LOGIC_VECTOR (8 downto 0);
	B7 : out STD_LOGIC_VECTOR (8 downto 0);
	B8 : out STD_LOGIC_VECTOR (8 downto 0);
	B9 : out STD_LOGIC_VECTOR (8 downto 0);
	B10 : out STD_LOGIC_VECTOR (8 downto 0);
	B11 : out STD_LOGIC_VECTOR (8 downto 0);
	B12 : out STD_LOGIC_VECTOR (8 downto 0);
	B13 : out STD_LOGIC_VECTOR (8 downto 0);
	B14 : out STD_LOGIC_VECTOR (8 downto 0);
	B15 : out STD_LOGIC_VECTOR (8 downto 0);
	B16 : out STD_LOGIC_VECTOR (8 downto 0);
	B17 : out STD_LOGIC_VECTOR (8 downto 0);
	B18 : out STD_LOGIC_VECTOR (8 downto 0);
	B19 : out STD_LOGIC_VECTOR (8 downto 0)
	);
end component;
component mux1 is
	Port (	CLK : in  STD_LOGIC;
	Qint : in STD_LOGIC_VECTOR (4 downto 0);
	Ff : in STD_LOGIC_VECTOR (8 downto 0);
	F1 : in STD_LOGIC_VECTOR (8 downto 0);
	F2 : in STD_LOGIC_VECTOR (8 downto 0);
	F3 : in STD_LOGIC_VECTOR (8 downto 0);
	F4 : in STD_LOGIC_VECTOR (8 downto 0);
	F5 : in STD_LOGIC_VECTOR (8 downto 0);
	F6 : in STD_LOGIC_VECTOR (8 downto 0);
	F7 : in STD_LOGIC_VECTOR (8 downto 0);
	F8 : in STD_LOGIC_VECTOR (8 downto 0);
	F9 : in STD_LOGIC_VECTOR (8 downto 0);
	F10 : in STD_LOGIC_VECTOR (8 downto 0);
	F11 : in STD_LOGIC_VECTOR (8 downto 0);
	F12 : in STD_LOGIC_VECTOR (8 downto 0);
	F13 : in STD_LOGIC_VECTOR (8 downto 0);
```

```
        F14 : in STD_LOGIC_VECTOR (8 downto 0);
        F15 : in STD_LOGIC_VECTOR (8 downto 0);
        F16 : in STD_LOGIC_VECTOR (8 downto 0);
        F17 : in STD_LOGIC_VECTOR (8 downto 0);
        F18 : in STD_LOGIC_VECTOR (8 downto 0);
        F19 : in STD_LOGIC_VECTOR (8 downto 0);
        sal : out STD_LOGIC_VECTOR (8 downto 0)
        );
end component;
component sen is
        Port ( dir_mem : in STD_LOGIC_VECTOR (8 downto 0);
        sal : out integer
         );
end component;
component DMUX is
        Port (     CLK : in  STD_LOGIC;
        Qint : in STD_LOGIC_VECTOR (4 downto 0);
        sen : in integer;
        Bf : out STD_LOGIC_VECTOR (7 downto 0);
        B1 : out STD_LOGIC_VECTOR (7 downto 0);
        B2 : out STD_LOGIC_VECTOR (7 downto 0);
        B3 : out STD_LOGIC_VECTOR (7 downto 0);
        B4 : out STD_LOGIC_VECTOR (7 downto 0);
        B5 : out STD_LOGIC_VECTOR (7 downto 0);
        B6 : out STD_LOGIC_VECTOR (7 downto 0);
        B7 : out STD_LOGIC_VECTOR (7 downto 0);
        B8 : out STD_LOGIC_VECTOR (7 downto 0);
        B9 : out STD_LOGIC_VECTOR (7 downto 0);
        B10 : out STD_LOGIC_VECTOR (7 downto 0);
        B11 : out STD_LOGIC_VECTOR (7 downto 0);
        B12 : out STD_LOGIC_VECTOR (7 downto 0);
        B13 : out STD_LOGIC_VECTOR (7 downto 0);
        B14 : out STD_LOGIC_VECTOR (7 downto 0);
        B15 : out STD_LOGIC_VECTOR (7 downto 0);
        B16 : out STD_LOGIC_VECTOR (7 downto 0);
        B17 : out STD_LOGIC_VECTOR (7 downto 0);
        B18 : out STD_LOGIC_VECTOR (7 downto 0);
        B19 : out STD_LOGIC_VECTOR (7 downto 0)
        );
end component;
component FF2 is
        Port (     CLK : in  STD_LOGIC;
        Qint : in STD_LOGIC_VECTOR (4 downto 0);
        Bf_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B1_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B2_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B3_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B4_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B5_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B6_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B7_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B8_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B9_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B10_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B11_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B12_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B13_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B14_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B15_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B16_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B17_5 : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        B18_5 : in STD_LOGIC_VECTOR (7 downto 0);
        B19_5 : in STD_LOGIC_VECTOR (7 downto 0);
        Bf : out STD_LOGIC_VECTOR (7 downto 0);
        B1 : out STD_LOGIC_VECTOR (7 downto 0);
        B2 : out STD_LOGIC_VECTOR (7 downto 0);
        B3 : out STD_LOGIC_VECTOR (7 downto 0);
        B4 : out STD_LOGIC_VECTOR (7 downto 0);
        B5 : out STD_LOGIC_VECTOR (7 downto 0);
        B6 : out STD_LOGIC_VECTOR (7 downto 0);
        B7 : out STD_LOGIC_VECTOR (7 downto 0);
        B8 : out STD_LOGIC_VECTOR (7 downto 0);
        B9 : out STD_LOGIC_VECTOR (7 downto 0);
        B10 : out STD_LOGIC_VECTOR (7 downto 0);
        B11 : out STD_LOGIC_VECTOR (7 downto 0);
        B12 : out STD_LOGIC_VECTOR (7 downto 0);
        B13 : out STD_LOGIC_VECTOR (7 downto 0);
        B14 : out STD_LOGIC_VECTOR (7 downto 0);
        B15 : out STD_LOGIC_VECTOR (7 downto 0);
        B16 : out STD_LOGIC_VECTOR (7 downto 0);
        B17 : out STD_LOGIC_VECTOR (7 downto 0);
        B18 : out STD_LOGIC_VECTOR (7 downto 0);
        B19 : out STD_LOGIC_VECTOR (7 downto 0)
        );
end component;
component mult is
        Port (     ampfund : in STD_LOGIC_VECTOR (7 downto 0);
        amp1 : in STD_LOGIC_VECTOR (7 downto 0);
        amp2 : in STD_LOGIC_VECTOR (7 downto 0);
        amp3 : in STD_LOGIC_VECTOR (7 downto 0);
        amp4 : in STD_LOGIC_VECTOR (7 downto 0);
        amp5 : in STD_LOGIC_VECTOR (7 downto 0);
        amp6 : in STD_LOGIC_VECTOR (7 downto 0);
        amp7 : in STD_LOGIC_VECTOR (7 downto 0);
        amp8 : in STD_LOGIC_VECTOR (7 downto 0);
        amp9 : in STD_LOGIC_VECTOR (7 downto 0);
        amp10 : in STD_LOGIC_VECTOR (7 downto 0);
        amp11 : in STD_LOGIC_VECTOR (7 downto 0);
        amp12 : in STD_LOGIC_VECTOR (7 downto 0);
        amp13 : in STD_LOGIC_VECTOR (7 downto 0);
        amp14 : in STD_LOGIC_VECTOR (7 downto 0);
        amp15 : in STD_LOGIC_VECTOR (7 downto 0);
        amp16 : in STD_LOGIC_VECTOR (7 downto 0);
        amp17 : in STD_LOGIC_VECTOR (7 downto 0);
        amp18 : in STD_LOGIC_VECTOR (7 downto 0);
        amp19 : in STD_LOGIC_VECTOR (7 downto 0);
        Bf_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B1_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B2_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B3_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B4_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B5_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B6_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B7_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B8_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B9_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B10_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B11_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B12_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B13_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B14_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B15_FF : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        B16_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B17_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B18_FF : in STD_LOGIC_VECTOR (7 downto 0);
        B19_FF : in STD_LOGIC_VECTOR (7 downto 0);
        Bf_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B1_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B2_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B3_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B4_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B5_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B6_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B7_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B8_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B9_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B10_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B11_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B12_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B13_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B14_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B15_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B16_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B17_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B18_6 : out STD_LOGIC_VECTOR (15 downto 0);
        B19_6 : out STD_LOGIC_VECTOR (15 downto 0)
        );
end component;
component offset is
        Port (     ampfund : in STD_LOGIC_VECTOR (7 downto 0);
        amp1 : in STD_LOGIC_VECTOR (7 downto 0);
        amp2 : in STD_LOGIC_VECTOR (7 downto 0);
        amp3 : in STD_LOGIC_VECTOR (7 downto 0);
        amp4 : in STD_LOGIC_VECTOR (7 downto 0);
        amp5 : in STD_LOGIC_VECTOR (7 downto 0);
        amp6 : in STD_LOGIC_VECTOR (7 downto 0);
        amp7 : in STD_LOGIC_VECTOR (7 downto 0);
        amp8 : in STD_LOGIC_VECTOR (7 downto 0);
        amp9 : in STD_LOGIC_VECTOR (7 downto 0);
        amp10 : in STD_LOGIC_VECTOR (7 downto 0);
        amp11 : in STD_LOGIC_VECTOR (7 downto 0);
        amp12 : in STD_LOGIC_VECTOR (7 downto 0);
        amp13 : in STD_LOGIC_VECTOR (7 downto 0);
        amp14 : in STD_LOGIC_VECTOR (7 downto 0);
        amp15 : in STD_LOGIC_VECTOR (7 downto 0);
        amp16 : in STD_LOGIC_VECTOR (7 downto 0);
        amp17 : in STD_LOGIC_VECTOR (7 downto 0);
        amp18 : in STD_LOGIC_VECTOR (7 downto 0);
        amp19 : in STD_LOGIC_VECTOR (7 downto 0);
        Bf_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B1_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B2_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B3_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B4_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B5_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B6_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B7_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B8_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B9_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B10_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B11_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B12_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B13_6 : in STD_LOGIC_VECTOR (15 downto 0);
```

```
        B14_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B15_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B16_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B17_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B18_6 : in STD_LOGIC_VECTOR (15 downto 0);
        B19_6 : in STD_LOGIC_VECTOR (15 downto 0);
        Bf_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B1_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B2_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B3_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B4_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B5_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B6_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B7_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B8_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B9_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B10_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B11_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B12_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B13_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B14_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B15_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B16_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B17_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B18_7 : out STD_LOGIC_VECTOR (15 downto 0);
        B19_7 : out STD_LOGIC_VECTOR (15 downto 0)
        );
end component;
component suma is
        Port (     Bf_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B1_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B2_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B3_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B4_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B5_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B6_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B7_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B8_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B9_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B10_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B11_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B12_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B13_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B14_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B15_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B16_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B17_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B18_7 : in STD_LOGIC_VECTOR (15 downto 0);
        B19_7 : in STD_LOGIC_VECTOR (15 downto 0);
        Sal : out STD_LOGIC_VECTOR (19 downto 0)
        );
end component;
component ajuste is
        Port (     Ent : in STD_LOGIC_VECTOR (19 downto 0);
        Sal1 : out STD_LOGIC_VECTOR (23 downto 0);
        Sal2 : out STD_LOGIC_VECTOR (23 downto 0)
        );
end component;
component arp is
        Port (     CLK : in  STD_LOGIC;
        notas : in STD_LOGIC_VECTOR (11 downto 0);
```

```
        des : in STD_LOGIC_VECTOR (11 downto 0);
        Sal : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component calc_notas is
        Port (     CLK : in  STD_LOGIC;
        notasent : in STD_LOGIC_VECTOR (11 downto 0);
        nota1 : out STD_LOGIC_VECTOR (4 downto 0);
        nota2 : out STD_LOGIC_VECTOR (4 downto 0));
end component;
component sumfin is
        Port (     Ent1 : in STD_LOGIC_VECTOR (23 downto 0);
        Ent1_2 : in STD_LOGIC_VECTOR (23 downto 0);
        Ent2 : in STD_LOGIC_VECTOR (23 downto 0);
        Ent2_2 : in STD_LOGIC_VECTOR (23 downto 0);
        Sal1 : out STD_LOGIC_VECTOR (23 downto 0);
        Sal2 : out STD_LOGIC_VECTOR (23 downto 0));
end component;
component seleccion is
        Port (     sel : in std_logic;
        clk : in std_logic;
        notas_arp :in STD_LOGIC_VECTOR (11 downto 0);
        notas_int :in STD_LOGIC_VECTOR (11 downto 0);
        notas_sel: out STD_LOGIC_VECTOR (11 downto 0));
end component;
component deteccion is
        Port (     CLK : in  STD_LOGIC;
        notasent : in STD_LOGIC_VECTOR (11 downto 0);
        en : in STD_LOGIC;
        des : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component invertir is
        Port (     CLK : in  STD_LOGIC;
        notas : in STD_LOGIC_VECTOR (11 downto 0);
        notas_inv : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component LED2 is
        Port ( notas_sel : in  STD_LOGIC_VECTOR (11 downto 0);
        leds : out  STD_LOGIC_VECTOR (11 downto 0));
end component;
component LED3 is
        Port (     INIT : IN std_logic;
        CLK : IN std_logic;
        W_EN : IN std_logic;
        RST : in STD_LOGIC;
        sel : IN std_logic;
    num_inst : in STD_LOGIC_VECTOR (1 downto 0);
        LED_config : out STD_LOGIC_VECTOR (2 downto 0);
        LED_modo : out std_logic;
        LED_inst : out STD_LOGIC_VECTOR (1 downto 0));
end component;
end componentes;
```

6.28.-Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.std_logic_textio.all;
use std.textio.all;
ENTITY proyecto_tb IS
```

```
END proyecto_tb;
ARCHITECTURE behavior OF proyecto_tb IS
        file vect_out: text open WRITE_MODE is
                "archivo_escritura.txt";
 -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT proyecto
   PORT(        CLK : in  STD_LOGIC;
                notas : in STD_LOGIC_VECTOR (11 downto 0);
                des : in STD_LOGIC_VECTOR (11 downto 0);
                num_inst : in STD_LOGIC_VECTOR (1 downto 0);
                RST : in STD_LOGIC;
                INIT : IN std_logic;
                W_EN : IN std_logic;
                sim : out STD_LOGIC_VECTOR (23 downto 0);
                sal_LED : out  STD_LOGIC_VECTOR (17 downto 0);
                pulse_48KHz : OUT std_logic;
                AUD_MCLK : OUT std_logic;
                AUD_BCLK : OUT std_logic;
                AUD_DACDAT : OUT std_logic;
                AUD_DACLRCK : OUT std_logic;
                I2C_SDAT : OUT std_logic;
                I2C_SCLK : OUT std_logic   );
   END COMPONENT;
     --Inputs
  signal CLK : std_logic := '0';
  signal notas : std_logic_vector(11 downto 0) := "100000000001"; -- introducir notas que toco
  signal des : std_logic_vector(11 downto 0) := "000000000000";    --introducir que quiero desactivar
  signal num_inst : std_logic_vector(1 downto 0) := "10";
  signal RST : std_logic := '1';
  signal INIT : std_logic := '0';
  signal W_EN : std_logic := '0';
      --Outputs
  signal sim : STD_LOGIC_VECTOR (23 downto 0);
  signal sal_LED : std_logic_vector(17 downto 0);
  signal pulse_48KHz : std_logic;
  signal AUD_MCLK : std_logic;
  signal AUD_BCLK : std_logic;
  signal AUD_DACDAT : std_logic;
  signal AUD_DACLRCK : std_logic;
  signal I2C_SDAT : std_logic;
  signal I2C_SCLK : std_logic;
  signal Salent : integer;
  signal i : STD_LOGIC_VECTOR (31 downto 0):= (others=>'0');
  signal CLK2 : std_logic := '0';
  signal Qint : STD_LOGIC_VECTOR (10 downto 0):= (others=>'0');
BEGIN
       -- Instantiate the Unit Under Test (UUT)
  uut: proyecto PORT MAP (
        CLK => CLK,
        notas => notas,
        des => des,
        num_inst => num_inst,
        RST => RST,
        INIT => INIT,
        W_EN => W_EN,
        sim => sim,
        sal_LED => sal_LED,
        pulse_48KHz => pulse_48KHz,
        AUD_MCLK => AUD_MCLK,
        AUD_BCLK => AUD_BCLK,
        AUD_DACDAT => AUD_DACDAT,
```

```
        AUD_DACLRCK => AUD_DACLRCK,
        I2C_SDAT => I2C_SDAT,
        I2C_SCLK => I2C_SCLK
      );
CLK <= not CLK after 5 ns;
RST <= '0' after 10 ns;
process (CLK)
        begin
                if (CLK'event and CLK='1') then
                        if Qint = 1041 then Qint <= "00000000000"; CLK2 <= '1';-- CLK a 48000
                        else Qint <= Qint+1; CLK2 <= '0';
                                -- Qint contador reloj señales control MUX
                        end if;
                end if;
end process;
process (CLK2)
        variable buf_in,buf_out:line;
        begin
        if (CLK2'event and CLK2='1') then
                Salent <= conv_integer(sim);
                i <= i+1;                               -- saber cuantos numeros estoy sacando
                WRITE(buf_out,Salent);
                WRITELINE(vect_out,buf_out);
        end if;
end process;
END;
```

Tono

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity tono is
   Port ( CLK : in  STD_LOGIC;
        RST : in STD_LOGIC;
        Sal : out STD_LOGIC_VECTOR (16 downto 0)
        );
end tono;
architecture Behavioral of tono is
type ROM_sen is array (0 to 511) of integer;
        constant
MEM_sen:ROM_sen:=( 0,1,3,4,6,7,9,11,12,14,15,17,18,20,21,23,25,26,28,29,31,32,34,35,37,38,40,
                        41,43,44,46,47,49,50,51,53,54,56,57,59,60,61,63,64,65,67,68,69,71,72,73,
                        75,76,77,78,80,81,82,83,84,86,87,88,89,90,91,92,93,94,96,97,98,99,100,
                        101,102,102,103,104,105,106,107,108,109,109,110,111,112,113,113,114,
                        115,115,116,117,117,118,118,119,120,120,121,121,122,122,123,123,123,
                        124,124,124,125,125,125,126,126,126,126,127,127,127,127,127,127,127,
                        127,127,127,127,127,127,127,127,127,127,127,127,127,126,126,126,126,
                        126,125,125,125,124,124,124,123,123,122,122,121,121,120,120,119,119,
                        118,118,117,116,116,115,114,114,113,112,111,111,110,109,108,107,106,
                        106,105,104,103,102,101,100,99,98,97,96,95,94,93,92,91,90,88,87,86,85,
                        84,83,81,80,79,78,77,75,74,73,71,70,69,67,66,65,63,62,61,59,58,56,55,54,
                        52,51,49,48,46,45,43,42,40,39,37,36,34,33,31,30,28,27,25,24,22,21,19,18,
                        16,14,13,11,10,8,7,5,3,2,0,-1,-3,-4,-6,-8,-9,-11,-12,-14,-15,-17,-19,-20,-22,
                        -23,-25,-26,-28,-29,-31,-32,-34,-35,-37,-38,-40,-41,-43,-44,-46,-47,-49,-50,
                        -52,-53,-55,-56,-57,-59,-60,-62,-63,-64,-66,-67,-68,-70,-71,-72,-74,-75,-76,
                        -78,-79,-80,-81,-82,-84,-85,-86,-87,-88,-89,-91,-92,-93,-94,-95,-96,-97,-98,
                        -99,-100,-101,-102,-103,-104,-105,-106,-107,-107,-108,-109,-110,-111,
                        -112,-112,-113,-114,-115,-115,-116,-117,-117,-118,-119,-119,-120,-120,
                        -121,-121,-122,-122,-123,-123,-124,-124,-125,-125,-125,-126,-126,-126,
```

```
                        -127,-127,-127,-127,-127,-128,-128,-128,-128,-128,-128,-128,-128,-128,
                        -128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-128,-127,-127,-127,
                        -127,-126,-126,-126,-125,-125,-125,-124,-124,-124,-123,-123,-122,-122,
                        -121,-121,-120,-119,-119,-118,-118,-117,-116,-116,-115,-114,-114,-113,
                        -112,-111,-110,-110,-109,-108,-107,-106,-105,-104,-103,-103,-102,-101,
                        -100,-99,-98,-97,-95,-94,-93,-92,-91,-90,-89,-88,-87,-85,-84,-83,-82,-81,
                        -79,-78,-77,-76,-74,-73,-72,-70,-69,-68,-66,-65,-64,-62,-61,-60,-58,-57,-55,
                        -54,-52,-51,-50,-48,-47,-45,-44,-42,-41,-39,-38,-36,-35,-33,-32,-30,-29,-27,
                        -26,-24,-22,-21,-19,-18,-16,-15,-13,-12,-10,-8,-7,-5,-4,-2,-1);
-------------------------------------------Signals-----------------------------------------------
signal INCe : integer:=0;
signal INC : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal fase1e : integer:=0;
signal fase1 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal ampfunde : integer:=0;
signal ampfund : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal Bf_1 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_2 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_3 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_4 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal Bf_4mem : STD_LOGIC_VECTOR (9 downto 0):= (others=>'0');
signal senf : integer:=0;
 signal Bf_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal Bf_6 : STD_LOGIC_VECTOR (15  downto 0):= (others=>'0');
signal Bf_sal : integer:=0;
signal Bsum : integer:=0;
signal Bf_1sum2 : STD_LOGIC_VECTOR (23 downto 0):= (others=>'0');
signal Bf_antsal : STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal Bf_pos : STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal Bf_i : integer:=0;
signal pos : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal neg : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
-----------------------------------------------------------------------------------------------
begin
        INCe <= 8080;
        INC <= conv_std_logic_vector(INCe,18);
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                ----------------- RESET -------------------------
                if (RST='1') then
                                Bf_1 <= (others => '0');
                                Bf_2 <= (others => '0');
                elsif (RST='0') then
                                ---------------------------------------------------------amplitudes
                                ampfunde <= 100;
                                ampfund <= conv_std_logic_vector(ampfunde,8);
                                ----------------- Primer registro ---------------
                                Bf_1 <= INC;        -- frecuencia fundamental
                                ------------------- Acumulador --------------------
                                Bf_2 <= Bf_2+Bf_1;
                end if;
        end if;
end Process;
                                ---------------- suma de la fase ------------------
                                Bf_3 <= Bf_2;
                                ----------------- truncado para memoria L bits------------
                                Bf_4 <= Bf_3 (17 downto 9);
                                Bf_4mem <= "0" & Bf_4;
                                Bf_i <= MEM_sen (CONV_INTEGER(Bf_4mem));
```

```
                                    Bf_5 <= conv_std_logic_vector(Bf_i,8);
                                    Bf_6 <= Bf_5 * ampfund;
Bf_antsal <= Bf_6;
Bf_pos <= conv_std_logic_vector(13500,16); -- poder convertir a integer para simular
Sal <= (Bf_antsal(15) & Bf_antsal) + (Bf_pos(15) & Bf_pos);
end Behavioral;
```

Mejora 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity mejora1 is
   Port ( CLK : in  STD_LOGIC;
          RST : in STD_LOGIC;
          Sal : out STD_LOGIC_VECTOR (11 downto 0)
          Sal : out STD_LOGIC_VECTOR (16 downto 0)
          );
end mejora1;
architecture Behavioral of mejora1 is
type ROM_sen is array (0 to 255) of integer;
constant MEM_sen:ROM_sen:=(   0,1,3,4,6,7,9,11,12,14,15,17,18,20,21,23,25,26,28,29,31,32,34,35,
                              37,38,40,41,43,44,46,47,49,50,51,53,54,56,57,59,60,61,63,64,65,
                              67,68,69,71,72,73,75,76,77,78,80,81,82,83,84,86,87,88,89,90,91,
                              92,93,94,96,97,98,99,100,101,102,102,103,104,105,106,107,108,
                              109,109,110,111,112,113,113,114,115,115,116,117,117,118,118,
                              119,120,120,121,121,122,122,123,123,123,124,124,124,125,125,
                              125,126,126,126,126,127,127,127,127,127,127,127,127,127,127,
                              127,127,127,127,127,127,127,127,127,127,126,126,126,126,126,
                              125,125,125,124,124,124,123,123,122,122,121,121,120,120,119,
                              119,118,118,117,116,116,115,114,114,113,112,111,111,110,109,
                              108,107,106,106,105,104,103,102,101,100,99,98,97,96,95,94,93,
                              92,91,90,88,87,86,85,84,83,81,80,79,78,77,75,74,73,71,70,69,67,
                              66,65,63,62,61,59,58,56,55,54,52,51,49,48,46,45,43,42,40,39,37,
                              36,34,33,31,30,28,27,25,24,22,21,19,18,16,14,13,11,10,8,7,5,3,2,
                              1);
-------------------------------------------Signals-----------------------------------------------
signal INCe : integer:=0;
signal INC : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal fase1e : integer:=0;
signal fase1 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal ampfunde : integer:=0;
signal ampfund : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal Bf_1 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_2 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_3 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_4 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal Bf_4mem : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal senf : integer:=0;
signal Bf_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal Bf_6 : STD_LOGIC_VECTOR (15  downto 0):= (others=>'0');
signal Bf_sal : integer:=0;
signal Bsum : integer:=0;
signal Bf_1sum2 : STD_LOGIC_VECTOR (23 downto 0):= (others=>'0');
signal Bf_antsal : STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal Bf_pos : STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal Bf_i : integer:=0;
```

```
signal pos : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal neg : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
-------------------------------------------------------------------------------------------
begin
        INCe <= 8080;
        INC <= conv_std_logic_vector(INCe,18);
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                ----------------- RESET -------------------------
                if (RST='1') then
                                Bf_1 <= (others => '0');
                                Bf_2 <= (others => '0');
                elsif (RST='0') then
                                ----------------------------------------------------------amplitudes
                                ampfunde <= 100;
                                ampfund <= conv_std_logic_vector(ampfunde,8);
                                ----------------- Primer registro ---------------
                                Bf_1 <= INC;        -- frecuencia fundamental
                                ------------------- Acumulador ---------------------
                                Bf_2 <= Bf_2+Bf_1;
                end if;
        end if;
end Process;
                                ---------------- suma de la fase ------------------
                                Bf_3 <= Bf_2;
                                ----------------- truncado para memoria L bits------------
                                Bf_4 <= Bf_3 (17 downto 9);
                                Bf_4mem <= "0" & Bf_4(7 downto 0);
                                ------------sacar valores mem sen con direcciones bits anteriores---

Process (CLK)
        begin
                if (Bf_4(8)='0') then Bf_i <= MEM_sen (CONV_INTEGER(Bf_4mem));
                else Bf_i <= -MEM_sen (CONV_INTEGER(Bf_4mem));
                end if;
end process;
                Bf_5 <= conv_std_logic_vector(Bf_i,8);
                Bf_6 <= Bf_5 * ampfund;
Bf_antsal <= Bf_6;
Bf_pos <= conv_std_logic_vector(13500,16); -- poder convertir a integer para simular
Sal <= (Bf_antsal(15) & Bf_antsal) + (Bf_pos(15) & Bf_pos);
end Behavioral;
```

Mejora 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
entity mejora2 is
   Port ( CLK : in  STD_LOGIC;
          RST : in STD_LOGIC;
          Sal : out STD_LOGIC_VECTOR (16 downto 0)
          );
end mejora2;
architecture Behavioral of mejora2 is
type ROM_sen is array (0 to 127) of integer;
constant MEM_sen:ROM_sen:=(    0,1,3,4,6,7,9,11,12,14,15,17,18,20,21,23,25,26,28,29,31,32,34,
```

```
                                    35, 37,38,40,41,43,44,46,47,49,50,51,53,54,56,57,59,60,61,63,64,
                                    65,67,68,69,71,72,73,75,76,77,78,80,81,82,83,84,86,87,88,89,90,
                                    91,92,93,94,96,97,98,99,100,101,102,102,103,104,105,106,107,
                                    108,109,109,110,111,112,113,113,114,115,115,116,117,117,118,
                                    118,119,120,120,121,121,122,122,123,123,123,124,124,124,125,
                                    125,125,126,126,126,126,127,127,127,127,127,127,127,127,127,
                                    127);
-----------------------------------------Signals-----------------------------------------
signal INCe : integer:=0;
signal INC : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal fase1e : integer:=0;
signal fase1 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal ampfunde : integer:=0;
signal ampfund : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal Bf_1 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_2 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_3 : STD_LOGIC_VECTOR (17 downto 0):= (others=>'0');
signal Bf_4 : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal Bf_4mem : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal senf : integer:=0;
signal Bf_5 : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
signal Bf_6 : STD_LOGIC_VECTOR (15  downto 0):= (others=>'0');
signal Bf_sal : integer:=0;
signal Bsum : integer:=0;
signal Bf_1sum2 : STD_LOGIC_VECTOR (23 downto 0):= (others=>'0');
signal Bf_antsal : STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal Bf_pos : STD_LOGIC_VECTOR (15 downto 0):= (others=>'0');
signal Bf_i : integer:=0;
signal pos : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal neg : STD_LOGIC_VECTOR (8 downto 0):= (others=>'0');
signal bf_res : STD_LOGIC_VECTOR (7 downto 0):= (others=>'0');
begin
        INCe <= 8080;
        INC <= conv_std_logic_vector(INCe,18);
Process (CLK)
begin
        if (CLK'event and CLK='1') then
                ---------- RESET -------------------------
                if (RST='1') then
                                    Bf_1 <= (others => '0');
                                    Bf_2 <= (others => '0');
                elsif (RST='0')
                                    ampfunde <= 100;
                                    ampfund <= conv_std_logic_vector(ampfunde,8);
                                    ---------- Primer registro ---------------
                                    Bf_1 <= INC;        -- frecuencia fundamental
                                    ------------------- Acumulador ---------------------
                                    Bf_2 <= Bf_2+Bf_1;
                end if;
        end if;
end Process;
                                    ---------- suma de la fase -------------------
                                    Bf_3 <= Bf_2;
                                    ---------- truncado para memoria L bits------------
                                    Bf_4 <= Bf_3 (17 downto 9);
                                    Bf_4mem <= "0" & Bf_4(6 downto 0);
Process (CLK)
        begin
                if (Bf_4(8)='0') then
                        if Bf_4(7)='0' then Bf_i <= MEM_sen (CONV_INTEGER(Bf_4mem));
```

```
                        else Bf_i <= MEM_sen (CONV_INTEGER(Bf_res));
                        end if;
                elsif  (Bf_4(8)='1') then
                        if Bf_4(7)='0' then Bf_i <= -MEM_sen (CONV_INTEGER(Bf_4mem));
                        else Bf_i <= -MEM_sen (CONV_INTEGER(Bf_res));
                        end if;
        end if;
end process;
Bf_5 <= conv_std_logic_vector(Bf_i,8);
Bf_6 <= Bf_5 * ampfund;
Bf_antsal <= Bf_6;
Bf_pos <= conv_std_logic_vector(13500,16); -- poder convertir a integer para simular
Sal <= (Bf_antsal(15) & Bf_antsal) + (Bf_pos(15) & Bf_pos);
end Behavioral;
```

Anexo 3: Código Matlab utilizado en la simulación de la síntesis de frecuencia.

C4 Oboe:

```
function c4_oboe
    fs=48000;
    Ts=1/fs;

    fundHz=261.626;
    a_freqHz=9941.78;

    a_n1_p =-1.898;
    a_n1_a =2931.55;
    a_n2_p =-1.427;
    a_n2_a =749.7;
    a_n3_p =-2.018;
    a_n3_a =2679.48;
    a_n4_p = -1.442;
    a_n4_a = 4936.02;
    a_n5_p = -1.666;
    a_n5_a = 5374;
    a_n6_p = -2.94;
    a_n6_a = 2510.78;
    a_n7_p = 2.653;
    a_n7_a = 307.69;
    a_n8_p = -0.539;
    a_n8_a = 442.5;
    a_n9_p = -0.461;
    a_n9_a = 739.29;
    a_n10_p = -0.158;
    a_n10_a = 533.49;
    a_n11_p = 1.196;
    a_n11_a = 461.5;
    a_n12_p = 1.036;
    a_n12_a = 497.54;
    a_n13_p = 0.515;
    a_n13_a = 304.65;
    a_n14_p = 0.372;
    a_n14_a = 24.51;
    a_n15_p = 0.261;
    a_n15_a = 48.97;
    a_n16_p = -0.36;
    a_n16_a = 14.76;
    a_n17_p = -1.328;
    a_n17_a = 81.97;
    a_n18_p = -0.321;
    a_n18_a = 46.85;
    a_n19_p = -1.236;
    a_n19_a = 15.9;
    t_fin=5; %5 segundos
    n_max=t_fin*fs;
    t=[1:n_max]*Ts;
senyal=((a_freqHz*sin(2*pi*t*fundHz))+(a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))+(a_n2_a*sin(2*
pi*t*3*fundHz+a_n2_p))+(a_n3_a*sin(2*pi*t*4*fundHz+a_n3_p))+(a_n4_a*sin(2*pi*t*5*fundHz+
a_n4_p))+(a_n5_a*sin(2*pi*t*6*fundHz+a_n5_p))+(a_n6_a*sin(2*pi*t*7*fundHz+a_n6_p))+
(a_n7_a*sin(2*pi*t*8*fundHz+a_n7_p))+(a_n8_a*sin(2*pi*t*9*fundHz+a_n8_p))+(a_n9_a*sin(2*
pi*t*10*fundHz+a_n9_p))+(a_n10_a*sin(2*pi*t*11*fundHz+a_n10_p))+(a_n11_a*sin(2*pi*t*12*
```

```
fundHz+a_n11_p))+(a_n12_a*sin(2*pi*t*13*fundHz+a_n12_p))+(a_n13_a*sin(2*pi*t*14*fundHz+
a_n13_p))+(a_n14_a*sin(2*pi*t*15*fundHz+a_n14_p))+(a_n15_a*sin(2*pi*t*16*fundHz+
a_n15_p))+(a_n16_a*sin(2*pi*t*17*fundHz+a_n16_p))+(a_n17_a*sin(2*pi*t*18*fundHz+
a_n17_p))+(a_n18_a*sin(2*pi*t*19*fundHz+a_n18_p))+(a_n19_a*sin(2*pi*t*20*fundHz+
a_n19_p)))*10^-4.5;
   %senyal NO puede pasar de +-1
   %senyal = (a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))

   hold on
   plot(t,senyal)
   AXIS([0 0.01 -1 1])
   sound(senyal,fs)
```

C5 Oboe:

```
function c5_oboe
   fs=48000;
   Ts=1/fs;

   fundHz=523.251;
   a_freqHz=8895.26;
   a_n1_p =0.31;
   a_n1_a =1316.94;
   a_n2_p =3.117;
   a_n2_a =3380.0;
   a_n3_p =-0.097;
   a_n3_a =578.76;
   a_n4_p = -1.368;
   a_n4_a = 1539.72;
   a_n5_p = -0.604;
   a_n5_a = 264.51;
   a_n6_p = -0.734;
   a_n6_a = 1579.59;
   a_n7_p = 0.876;
   a_n7_a = 88.45;
   a_n8_p = 2.915;
   a_n8_a = 299.15;
   a_n9_p = -2.582;
   a_n9_a = 132.4;
   a_n10_p = -1.535;
   a_n10_a = 101.05;
   a_n11_p = -2.769;
   a_n11_a = 11.85;
   a_n12_p = -2.397;
   a_n12_a = 24.1;
   a_n13_p = 2.159;
   a_n13_a = 9.18;
   a_n14_p = 2.712;
   a_n14_a = 18.93;
   a_n15_p = -0.949;
   a_n15_a = 16.78;
   a_n16_p = -2.007;
   a_n16_a = 20.85;
   a_n17_p = -2.237;
   a_n17_a = 6.02;
   a_n18_p = 1.225;
   a_n18_a = 15.3;
   a_n19_p = -0.013;
   a_n19_a = 9.94;
```

```
    t_fin=5; %5 segundos
    n_max=t_fin*fs;
    t=[1:n_max]*Ts;

senyal=((a_freqHz*sin(2*pi*t*fundHz))+(a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))+(a_n2_a*sin(2*
pi*t*3*fundHz+a_n2_p))+(a_n3_a*sin(2*pi*t*4*fundHz+a_n3_p))+(a_n4_a*sin(2*pi*t*5*fundHz+
a_n4_p))+(a_n5_a*sin(2*pi*t*6*fundHz+a_n5_p))+(a_n6_a*sin(2*pi*t*7*fundHz+a_n6_p))+
(a_n7_a*sin(2*pi*t*8*fundHz+a_n7_p))+(a_n8_a*sin(2*pi*t*9*fundHz+a_n8_p))+(a_n9_a*sin(2*
pi*t*10*fundHz+a_n9_p))+(a_n10_a*sin(2*pi*t*11*fundHz+a_n10_p))+(a_n11_a*sin(2*pi*t*12*
fundHz+a_n11_p))+(a_n12_a*sin(2*pi*t*13*fundHz+a_n12_p))+(a_n13_a*sin(2*pi*t*14*fundHz+
a_n13_p))+(a_n14_a*sin(2*pi*t*15*fundHz+a_n14_p))+(a_n15_a*sin(2*pi*t*16*fundHz+
a_n15_p))+(a_n16_a*sin(2*pi*t*17*fundHz+a_n16_p))+(a_n17_a*sin(2*pi*t*18*fundHz+
a_n17_p))+(a_n18_a*sin(2*pi*t*19*fundHz+a_n18_p))+(a_n19_a*sin(2*pi*t*20*fundHz+
a_n19_p)))*10^-4.5;
    %senyal NO puede pasar de +-1
    %senyal = (a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))

    hold on
    plot(t,senyal)
    AXIS([0 0.005 -1 1])

    sound(senyal,fs)
```

C5 Violín:

```
function c5_violin_muted_vibrato
    fs=48000;
    Ts=1/fs;
    fundHz=523.251;
    a_freqHz=8895.26;
    a_n1_p =-2.507;
    a_n1_a =2385.3;
    a_n2_p =-1.103;
    a_n2_a =2565;
    a_n3_p =-2.522;
    a_n3_a =769.53;
    a_n4_p = -0.014;
    a_n4_a = 771.31;
    a_n5_p = 1.634;
    a_n5_a = 145.77;
    a_n6_p = -2.855;
    a_n6_a = 201.23;
    a_n7_p = -0.725;
    a_n7_a = 52.14;
    a_n8_p = 3.028;
    a_n8_a = 36.36;
    a_n9_p = -2.751;
    a_n9_a = 27.21;
    a_n10_p = -1.848;
    a_n10_a = 10.54;
    a_n11_p = -2.0;
    a_n11_a = 46.17;
    a_n12_p = 0.302;
    a_n12_a = 74.87;
    a_n13_p = -1.655;
    a_n13_a = 89.83;
    a_n14_p = 2.644;
```

```
    a_n14_a = 56.35;
    a_n15_p = -2.03;
    a_n15_a = 62.88;
    a_n16_p = -0.749;
    a_n16_a = 30.95;
    a_n17_p = -1.492;
    a_n17_a = 27.66;
    a_n18_p = 1.336;
    a_n18_a = 18.03;
    a_n19_p = -2.694;
    a_n19_a = 12.78;
    t_fin=5; %5 segundos
    n_max=t_fin*fs;
    t=[1:n_max]*Ts;
senyal=((a_freqHz*sin(2*pi*t*fundHz))+(a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))+(a_n2_a*sin(2*
pi*t*3*fundHz+a_n2_p))+(a_n3_a*sin(2*pi*t*4*fundHz+a_n3_p))+(a_n4_a*sin(2*pi*t*5*fundHz+
a_n4_p))+(a_n5_a*sin(2*pi*t*6*fundHz+a_n5_p))+(a_n6_a*sin(2*pi*t*7*fundHz+a_n6_p))+
(a_n7_a*sin(2*pi*t*8*fundHz+a_n7_p))+(a_n8_a*sin(2*pi*t*9*fundHz+a_n8_p))+(a_n9_a*sin(2*
pi*t*10*fundHz+a_n9_p))+(a_n10_a*sin(2*pi*t*11*fundHz+a_n10_p))+(a_n11_a*sin(2*pi*t*12*
fundHz+a_n11_p))+(a_n12_a*sin(2*pi*t*13*fundHz+a_n12_p))+(a_n13_a*sin(2*pi*t*14*fundHz+
a_n13_p))+(a_n14_a*sin(2*pi*t*15*fundHz+a_n14_p))+(a_n15_a*sin(2*pi*t*16*fundHz+
a_n15_p))+(a_n16_a*sin(2*pi*t*17*fundHz+a_n16_p))+(a_n17_a*sin(2*pi*t*18*fundHz+
a_n17_p))+(a_n18_a*sin(2*pi*t*19*fundHz+a_n18_p))+(a_n19_a*sin(2*pi*t*20*fundHz+
a_n19_p)))*10^-4.5;
    hold on
    plot(t,senyal,'b')
    AXIS([0 0.005 -1 1])
   sound(senyal,fs)
```

C6 Violín:

```
 function c6_violin_muted_vibrato
    fs=48000;
    Ts=1/fs;
    fundHz=1046.5;
    a_freqHz=8372.26;
    a_n1_p =-1.521;
    a_n1_a =1920;
    a_n2_p =2.295;
    a_n2_a =475.93;
    a_n3_p =-2.698;
    a_n3_a =119.21;
    a_n4_p = -0.741;
    a_n4_a = 60.5;
    a_n5_p = -2.987;
    a_n5_a = 134.65;
    a_n6_p = -2.794;
    a_n6_a = 123.03;
    a_n7_p = -2.427;
    a_n7_a = 46.17;
    a_n8_p = -0.768;
    a_n8_a = 24.5;
    a_n9_p = -1.378;
    a_n9_a = 58.51;
    a_n10_p = 0;
    a_n10_a = 0;
    a_n11_p = 0;
    a_n11_a = 0;
    a_n12_p = 0;
```

```
    a_n12_a = 0;
    a_n13_p = 0;
    a_n13_a = 0;
    a_n14_p = 0;
    a_n14_a = 0;
    a_n15_p = 0;
    a_n15_a = 0;
    a_n16_p = 0;
    a_n16_a = 0;
    a_n17_p = 0;
    a_n17_a = 0;
    a_n18_p = 0;
    a_n18_a = 0;
    a_n19_p = 0;
    a_n19_a = 0;
    t_fin=5; %5 segundos
    n_max=t_fin*fs;
    t=[1:n_max]*Ts;
senyal=((a_freqHz*sin(2*pi*t*fundHz))+(a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))+(a_n2_a*sin(2*
pi*t*3*fundHz+a_n2_p))+(a_n3_a*sin(2*pi*t*4*fundHz+a_n3_p))+(a_n4_a*sin(2*pi*t*5*fundHz+
a_n4_p))+(a_n5_a*sin(2*pi*t*6*fundHz+a_n5_p))+(a_n6_a*sin(2*pi*t*7*fundHz+a_n6_p))+
(a_n7_a*sin(2*pi*t*8*fundHz+a_n7_p))+(a_n8_a*sin(2*pi*t*9*fundHz+a_n8_p))+(a_n9_a*sin(2*
pi*t*10*fundHz+a_n9_p))+(a_n10_a*sin(2*pi*t*11*fundHz+a_n10_p))+(a_n11_a*sin(2*pi*t*12*
fundHz+a_n11_p))+(a_n12_a*sin(2*pi*t*13*fundHz+a_n12_p))+(a_n13_a*sin(2*pi*t*14*fundHz+
a_n13_p))+(a_n14_a*sin(2*pi*t*15*fundHz+a_n14_p))+(a_n15_a*sin(2*pi*t*16*fundHz+
a_n15_p))+(a_n16_a*sin(2*pi*t*17*fundHz+a_n16_p))+(a_n17_a*sin(2*pi*t*18*fundHz+
a_n17_p))+(a_n18_a*sin(2*pi*t*19*fundHz+a_n18_p))+(a_n19_a*sin(2*pi*t*20*fundHz+
a_n19_p)))*10^-4.5;
    hold on
    plot(t,senyal,'b')
    AXIS([0 0.005 -1 1])
    sound(senyal,fs)
```

C3 Tuba

```
function c3_tuba
    fs=48000;
    Ts=1/fs;
    fundHz=130.81;
    a_freqHz=9941.78;
    a_n1_p =-2.368;
    a_n1_a =2925.42;
    a_n2_p =0.313;
    a_n2_a =3488;
    a_n3_p =-2.778;
    a_n3_a =2024.17;
    a_n4_p = 0.326;
    a_n4_a = 816.78;
    a_n5_p = -1.637;
    a_n5_a = 161.46;
    a_n6_p = -0.736;
    a_n6_a = 504.68;
    a_n7_p = 2.266;
    a_n7_a = 363.4;
    a_n8_p = 3.036;
    a_n8_a = 53.73;
    a_n9_p = 1.747;
    a_n9_a = 73.84;
    a_n10_p = -0.852;
    a_n10_a = 28.47;
```

```
    a_n11_p = 1.984;
    a_n11_a = 27.72;
    a_n12_p = -1.552;
    a_n12_a = 23.72;
    a_n13_p = 1.253;
    a_n13_a = 26.67;
    a_n14_p = -1.121;
    a_n14_a = 8.95;
    a_n15_p = -0.904;
    a_n15_a = 3.84;
    a_n16_p = 0.515;
    a_n16_a = 3.83;
    a_n17_p = -2.001;
    a_n17_a = 5.61;
    a_n18_p = -1.481;
    a_n18_a = 2.53;
    a_n19_p = -0.638;
    a_n19_a = 4.73;
    t_fin=5; %5 segundos
    n_max=t_fin*fs;
    t=[1:n_max]*Ts;
senyal=((a_freqHz*sin(2*pi*t*fundHz))+(a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))+(a_n2_a*sin(2*
pi*t*3*fundHz+a_n2_p))+(a_n3_a*sin(2*pi*t*4*fundHz+a_n3_p))+(a_n4_a*sin(2*pi*t*5*fundHz+
a_n4_p))+(a_n5_a*sin(2*pi*t*6*fundHz+a_n5_p))+(a_n6_a*sin(2*pi*t*7*fundHz+a_n6_p))+
(a_n7_a*sin(2*pi*t*8*fundHz+a_n7_p))+(a_n8_a*sin(2*pi*t*9*fundHz+a_n8_p))+(a_n9_a*sin(2*
pi*t*10*fundHz+a_n9_p))+(a_n10_a*sin(2*pi*t*11*fundHz+a_n10_p))+(a_n11_a*sin(2*pi*t*12*
fundHz+a_n11_p))+(a_n12_a*sin(2*pi*t*13*fundHz+a_n12_p))+(a_n13_a*sin(2*pi*t*14*fundHz+
a_n13_p))+(a_n14_a*sin(2*pi*t*15*fundHz+a_n14_p))+(a_n15_a*sin(2*pi*t*16*fundHz+
a_n15_p))+(a_n16_a*sin(2*pi*t*17*fundHz+a_n16_p))+(a_n17_a*sin(2*pi*t*18*fundHz+
a_n17_p))+(a_n18_a*sin(2*pi*t*19*fundHz+a_n18_p))+(a_n19_a*sin(2*pi*t*20*fundHz+
a_n19_p)))*10^-4.5;
    hold on
    plot(t,senyal,'b')
    AXIS([0 0.020 -1 1])
    sound(senyal,fs)
```

C4 Tuba

```
function c4_tuba
    fs=48000;
    Ts=1/fs;
    fundHz=261.63;
    a_freqHz=6802.27;
    a_n1_p =1.411;
    a_n1_a =2043;
    a_n2_p =2.288;
    a_n2_a =254.34;
    a_n3_p =1.824;
    a_n3_a =128.07;
    a_n4_p = 1.594;
    a_n4_a = 14.84;
    a_n5_p = 2.702;
    a_n6_p = -1.07;
    a_n6_a = 3.91;
    a_n7_p = 0.982;
    a_n7_a = 1.45;
    a_n8_p = 0.744;
    a_n8_a = 0.51;
    a_n9_p = -2.245;
    a_n9_a = 0.93;
```

```
    a_n10_p = -0.202;
    a_n10_a = 0.43;
    a_n11_p = 0.467;
    a_n11_a = 0.35;
    a_n12_p = 0.002;
    a_n12_a = 0.13;
    a_n13_p = 2.266;
    a_n13_a = 0.13;
    a_n14_p = -2.07;
    a_n14_a = 0.21;
    a_n15_p = -1.267;
    a_n15_a = 0.14;
    a_n16_p = -2.481;
    a_n16_a = 0.3;
    a_n17_p = 1.169;
    a_n17_a = 0.12;
    a_n18_p = 2.629;
    a_n18_a = 0.18;
    a_n19_p = 2.079;
    a_n19_a = 0.13;
    t_fin=5; %5 segundos
    n_max=t_fin*fs;
    t=[1:n_max]*Ts;
senyal=((a_freqHz*sin(2*pi*t*fundHz))+(a_n1_a*sin(2*pi*t*2*fundHz+a_n1_p))+(a_n2_a*sin(2*p
i*t*3*fundHz+a_n2_p))+(a_n3_a*sin(2*pi*t*4*fundHz+a_n3_p))+(a_n4_a*sin(2*pi*t*5*fundHz+
a_n4_p))+(a_n5_a*sin(2*pi*t*6*fundHz+a_n5_p))+(a_n6_a*sin(2*pi*t*7*fundHz+a_n6_p))+
(a_n7_a*sin(2*pi*t*8*fundHz+a_n7_p))+(a_n8_a*sin(2*pi*t*9*fundHz+a_n8_p))+(a_n9_a*sin(2*
pi*t*10*fundHz+a_n9_p))+(a_n10_a*sin(2*pi*t*11*fundHz+a_n10_p))+(a_n11_a*sin(2*pi*t*12*
fundHz+a_n11_p))+(a_n12_a*sin(2*pi*t*13*fundHz+a_n12_p))+(a_n13_a*sin(2*pi*t*14*fundHz+
a_n13_p))+(a_n14_a*sin(2*pi*t*15*fundHz+a_n14_p))+(a_n15_a*sin(2*pi*t*16*fundHz+
a_n15_p))+(a_n16_a*sin(2*pi*t*17*fundHz+a_n16_p))+(a_n17_a*sin(2*pi*t*18*fundHz+
a_n17_p))+(a_n18_a*sin(2*pi*t*19*fundHz+a_n18_p))+(a_n19_a*sin(2*pi*t*20*fundHz+
a_n19_p)))*10^-4.5;
    hold on
    plot(t,senyal,'b')
    AXIS([0 0.010 -1 1])
    sound(senyal,fs)
```