Diseño de Sistemas Electrónicos Digitales

Sistema de grabación, tratamiento y reproducción de audio

Versión 4.1

Pablo Ituero y Fernando García Redondo

Nombre y apellidos de los miembros del grupo:

Pablo Fernández Fernández
Santiago Romero Pomar

Departamento de Ingeniería Electrónica

Diseño de Sistemas Electrónicos Digitales 2017
Sistema de grabación, tratamiento y reproducción de audio

El bloque de filtrado está basado en el "Mini-project" "FIR-Filter Design" del curso IL2204 "DSP Design usign HDL" del "Royal Institute of Technology" de Estocolmo, Suecia.

Departamento de Ingeniería Electrónica
UNIVERSIDAD POLITÉCNICA DE MADRID
España
Teléfono: +34915495700 ext. 4207
E-mail: pituero@die.upm.es

Índice

1. Introducción

Este documento describe el proyecto final de la asignatura Diseño de Sistemas Electrónicos Digitales del plan de estudios GITST de la ETSI de Telecomunicación de la UPM.

El proyecto guía el diseño y la implementación de un sistema electrónico digital de complejidad media y está pensado para ocupar la segunda mitad de los de las horas de laboratorio totales de la asignatura.

En cuanto a los objetivos docentes, el proyecto pretende asentar y profundizar en las metodologías de diseño combinacional y secuencial, en el diseño de máquinas de estados finitos y de máquinas de estados finitos con ruta de datos asociadas, la metodología de síntesis de alto nivel y en cuestiones de temporización.

En concreto se propone un sistema que, utilizando las interfaces de entrada y salida de audio de la placa con la que han estado trabajando los alumnos en la primera mitad de las horas prácticas de laboratorio, adquiere, almacena, procesa y reproduce sonidos.

2. Especificaciones mínimas

El sistema tiene las siguientes especificaciones globales mínimas:

- El sistema se implementa sobre la placa Nexys 4DDR utilizando el flujo de trabajo de Vivado.
- El sistema recibe la información de audio del micrófono integrado en la placa.
- El sistema reproduce el audio a través de la salida mono de audio integrada en la placa.
- El sistema se controla mediante tres botones (BTNL, BTNC y BTNR) y dos switches (SW0 y SW1).
- El sistema funciona a una frecuencia de reloj de 12 MHz, que se genera a partir del reloj de 100 MHz disponible en la placa.
- El sistema tiene un reset global procedente del botón BTNU.
- El audio se muestrea y se reproduce a una frecuencia de 20 kHz.
- El sistema se describe en VHDL y la entidad global tiene la siguiente declaración:

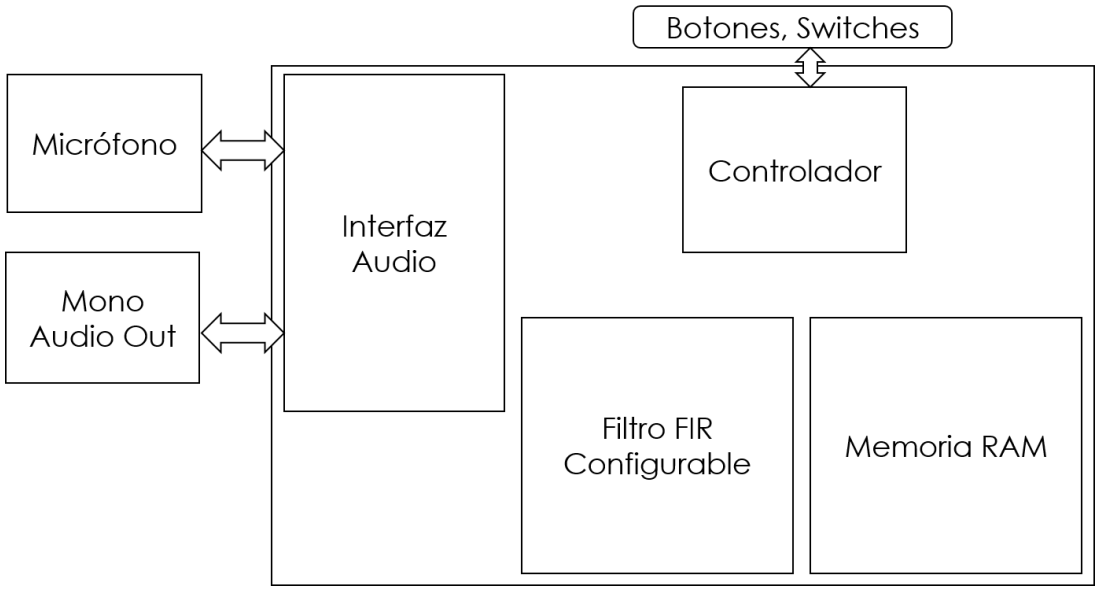
```
entity dsed_audio is
  Port (
    clk_100Mhz : in std_logic;
    reset: in std_logic;
    --Control ports
    BTNL: in STD_LOGIC;
    BTNC: in STD_LOGIC;
    BTNR: in STD_LOGIC;
```

```
    SW0:     in STD_LOGIC;
    SW1:     in STD_LOGIC;
    --To/From the microphone
    micro_clk : out STD_LOGIC;
    micro_data : in STD_LOGIC;
    micro_LR : out STD_LOGIC;
    --To/From the mini-jack
    jack_sd : out STD_LOGIC;
    jack_pwm : out STD_LOGIC
  );
end dsed_audio;
```

El funcionamiento es el siguiente:

- El sistema parte de una memoria vacía de audio.
- Cuando se presiona BTNL y durante el tiempo que éste esté presionado el sistema graba el audio.
- Cada vez que se vuelve a presionar BTNL el nuevo audio se añade al final del audio que estaba previamente grabado.
- Cuando se presiona BTNC se borra todo el audio grabado.
- Cuando se presiona BTNR el sistema debe realizar acciones en función de la información de los switches:
  - Si ambos switches están a '0' el sistema reproducirá todo el audio grabado.
  - Si SW0 está a '1' y SW1 está a '0' el sistema reproducirá el audio almacenado al revés.
  - Si SW0 está a '0' y SW1 está a '1' el sistema reproducirá el audio filtrado por un filtro paso bajo.
  - Si SW0 está a '1' y SW1 está a '1' el sistema reproducirá el audio filtrado por un filtro paso alto.

2.1. Descripción de la estructura del sistema



El sistema está compuesto por cuatro bloques principales:

- La interfaz de audio tiene dos funciones principales: En primer lugar es la encargada de la digitalización de las señales PDM provenientes del micrófono. Por otro lado es el bloque encargado de interactuar con la salida de audio proporcionándole una señal PWM. Este bloque se va a implementar siguiendo una metodología de máquina de estados finitos con ruta de datos asociada.
- El filtro FIR se encarga del procesado digital de la señal de audio para las funcionalidades de filtro paso alto y filtro paso bajo. Este bloque se va a implementar siguiendo la metodología de síntesis de alto nivel.
- La memoria RAM almacena las muestras de audio que se han grabado previamente. Este bloque se va generar con el asistente arquitectural de Vivado.
- El controlador es el encargado de orquestar toda la actuación del sistema proporcionando señales de control a los distintos bloques para ejecutar las órdenes del usuario del sistema. En este bloque el estilo de diseño es libre.

2.2. Uso de package

El código emplea un package propio denominado package_dsed. El objetivo es guardar todas las constantes y tipos de datos que vayan a ser utilizados por varios bloques en una librería común, de tal manera que el código se haga más legible, evitando el uso de expresiones numéricas que no se sabe de dónde aparecen.

A continuación se proporciona el código correspondiente a la declaración del package:

```
package package_dsed is
```

```
    constant sample_size: integer := 8;

end package_dsed;
```

La idea es ir añadiendo líneas a esta declaración en función de las necesidades del diseño. Para hacer uso de este package, hay que incluir la siguiente línea en la cabecera de nuestro código:

```
use work.package_dsed.all;
```

3. Planificación temporal

El proyecto está pensado para realizarse en parejas en la segunda mitad de las sesiones de laboratorio de la asignatura, esto equivale aproximadamente a 6 sesiones de 2-3 horas con el apoyo del profesor más otras 15 horas de trabajo no guiado.

El proyecto se ha dividido en hitos que se corresponden con los bloques principales de la estructura del sistema:

- Semanas 1-2. Interfaz de audio.
- Semanas 3-4. Filtro FIR.
- Semanas 5-6. Controlador y memoria.

Este documento está pensado como un cuaderno y guía de laboratorio donde ir apuntando las decisiones de diseño y los resultados de la implementación.

4. Evaluación

La evaluación del proyecto tendrá en cuenta los siguientes elementos:

- Una pequeña demostración en la que se mostrará el funcionamiento de la implementación en la placa, así como el resultado de distintos testbenches del sistema.
- La calidad de las decisiones de diseño reflejadas en la copia de cada pareja de este documento.
- La calidad del código VHDL. Es importante que cada pareja aplique criterios de buena legibilidad en su código (uso de comentarios explicativos, uso sistemático de prefijos y sufijos en nombres de puertos, señales y variables, etc).
- La calidad de las mejoras, que se evaluarán mediante los tres criterios anteriores.

5. Bloque 1: Interfaz de audio

La interfaz de audio va a tener dos funciones principales: Por un lado, en el momento de la grabación, va a recibir la información del micrófono, codificada en PDM y transformarla en una palabra digital de 8 bits a una frecuencia de muesteo de 20 kHz. Por otro lado, en el momento de la reproducción del audio, va a transformar una palabra de 8 bits en un pulso de anchura variable (codificación PWM) a una velocidad de 20000 muestras por segundo.

Tiene la siguiente declaración de entidad en VHDL:

```
entity audio_interface is
    Port ( clk_12megas : in STD_LOGIC;
           reset : in STD_LOGIC;
           --Recording ports
           --To/From the controller
           record_enable: in STD_LOGIC;
           sample_out: out STD_LOGIC_VECTOR (sample_size-1 downto 0);
           sample_out_ready: out STD_LOGIC;
           --To/From the microphone
           micro_clk : out STD_LOGIC;
           micro_data : in STD_LOGIC;
           micro_LR : out STD_LOGIC;
           --Playing ports
           --To/From the controller
           play_enable: in STD_LOGIC;
           sample_in: in std_logic_vector(sample_size-1 downto 0);
           sample_request: out std_logic;
           --To/From the mini-jack
           jack_sd : out STD_LOGIC;
           jack_pwm : out STD_LOGIC);
end audio_interface;
```

Descripción de cada puerto:

- clk_12megas: Reloj global de la arquitectura a 12 MHz.
- reset: Reset global del sistema.
- record_enable: Señal de control de la grabación. Cuando esté a '1', la digitalización de la información del micrófono funcionará.
- sample_out: Dato de 8 bits correspondiente a la señal digitalizada obtenida del micrófono.
- sample_out_ready: Señal de control que proporciona un pulso activo de un periodo de reloj de duración cada vez que se proporciona un nuevo dato digitalizado.
- micro_clk: Salida del reloj del micrófono. Un reloj de 3 MHz obtenido a partir de nuestro reloj de 12 MHz.
- micro_data: Entrada de la señal PDM proveniente del micrófono.

- micro_LR: Salida de control del micrófono que determina si las muestras se toman en el flanco de subida o de bajada del reloj. Dejaremos este valor estable en '1', correspondiente al flanco de subida.
- play_enable: Señal de control de la reproducción de audio. Cuando esté a '1', se procederá a la generación de la señal PWM hacia la salida de audio mono.
- sample_in: Dato de 8 bits correspondiente a la señal que hay que reproducir.
- sample_request: Señal de control que proporciona un pulso activo de un periodo de reloj de duración cada vez que se requiere un nuevo dato en sample_in.
- jack_sd: Información de control para los operacionales de la etapa de audio mono. Dejaremos este valor estable en '1'.
- jack_pwm: La señal PWM generada a partir del dato en sample_in.

A nivel estructural, en este bloque se pueden distinguir tres componentes:

- Interfaz del micrófono. Encargado de gestionar los "recording ports", excepto el micro_clk.
- Interfaz de la salida de audio. Encargado de gestionar los "playing ports".
- Generador de enables y salida de reloj del micrófono. Encargado de generar las señales enable que controlan las frecuencias de muestreo y de conversión de datos, así como la señal de reloj que alimenta al micrófono.

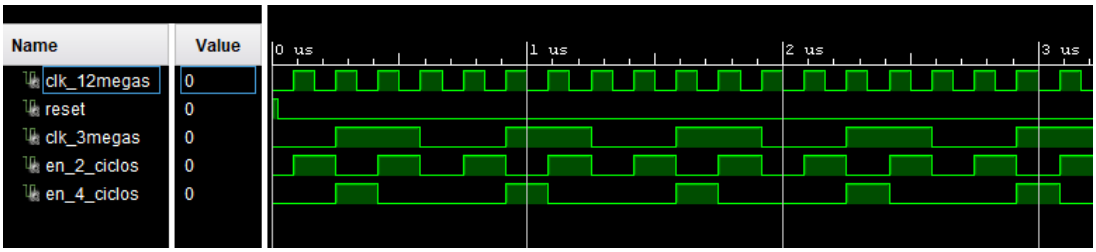
A continuación se describe cada uno de estos módulos y se guía el proceso de diseño.

5.1. Generador de enables y salida de reloj del micrófono

Este bloque recibe la señal del reloj global de 12 MHz y proporciona a sus salidas señales que van a ser utilizadas para temporizar el resto de módulos de la interfaz de audio. La entidad del módulo se define de la siguiente manera:

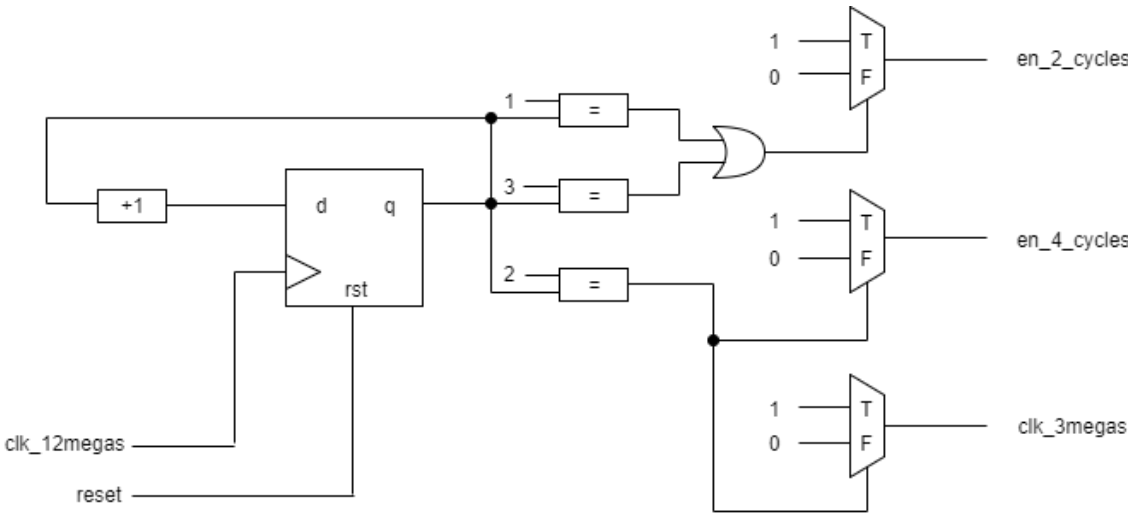
```
entity en_4_cycles is
    Port ( clk_12megas : in STD_LOGIC;
           reset : in STD_LOGIC;
           clk_3megas: out STD_LOGIC;
           en_2_cycles: out STD_LOGIC;
           en_4_cycles : out STD_LOGIC);
end en_4_cycles;
```

La salida clk_3megas es utilizada por el micrófono y es un reloj de 3 MHz con un duty cycle del 50 %. La salida en_2_cycles es utilizada por la interfaz de salida de audio y proporciona una señal activa durante un ciclo de reloj cada dos ciclos (equivalente a un reloj de 6 MHz). La salida en_4_cycles es utilizada por la interfaz del micrófono y proporciona una señal activa durante un ciclo de reloj cada cuatro ciclos. La siguiente figura muestra el funcionamiento esperado.



☞ **Tarea 1.1:**

Diseña un circuito que sea capaz de generar las señales especificadas. Dibuja el esquemático correspondiente a tu diseño en el espacio inferior.



☞ **Tarea 1.2:**
Implementa tu diseño en VHDL.

☞ **Tarea 1.3:**
Genera un testbench que verifique la funcionalidad de tu diseño.

5.2. Interfaz del micrófono

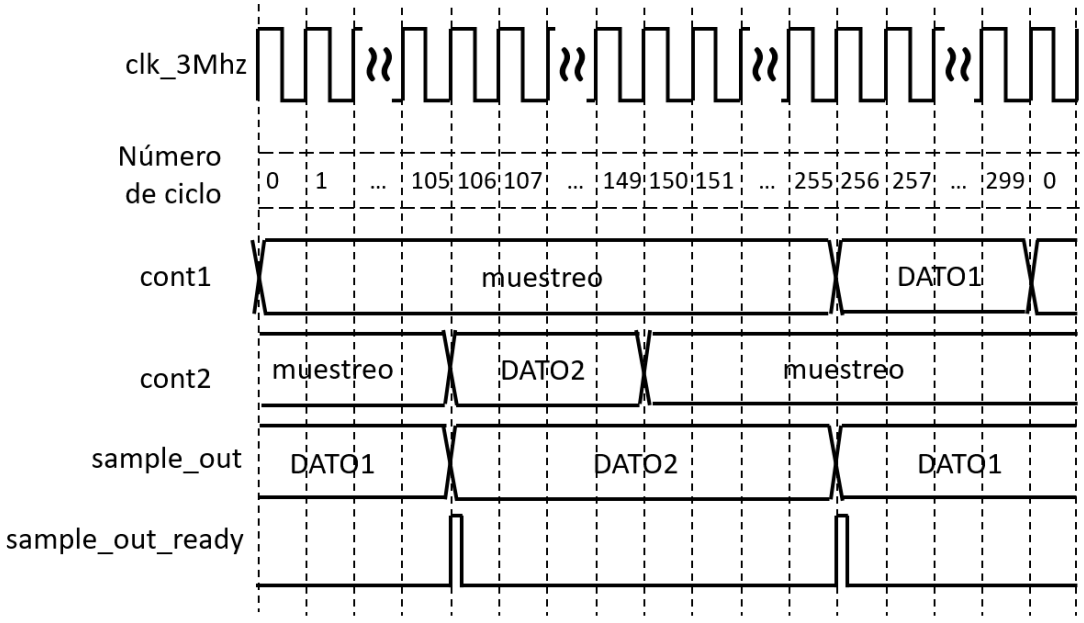
Para diseñar este módulo, es necesario conocer bien la interfaz que proporciona el micrófono omnidireccional de la placa Nexys4 DDR, la modulación de densidad de pulso (PDM) y la temporización de la interfaz del micrófono. Para ello se recomienda la lectura de la sección 15 del manual "Nexys4 DDR FPGA Board Reference Manual".

5.2.1. Esquema de muestreo

El sistema va a proporcionar una frecuencia de 3 MHz en micro_clk (333 ns de periodo) y necesita tomar una muestra cada 50 μs ($f_{sampling}$ = 20 kHz), es decir, una muestra cada 150 periodos de micro_clk. Se va a utilizar un esquema de dos contadores similar al explicado en la figura 28 del manual de la placa.

El sistema va a trabajar con datos de 8 bits, es decir va a contar el número de unos que hay a lo largo de 256 muestras.

La siguiente imagen muestra el esquema de muestreo del sistema:



Según se muestra en la figura, hay un contador general de módulo 300 que numera los ciclos del 0 al 299. Los 300 ciclos se corresponden con 100 μs, es decir el doble del periodo de muestreo. El primer contador muestrea entre los ciclos 0 y 255 y proporciona la señal digitalizada estable entre los ciclos 256 y 299. El segundo contador muestrea entre los ciclos 150 y 105 y proporciona

la señal digitalizada estable entre los ciclos 106 y 149. La salida sample_out en el ciclo 105 se actualiza con el valor digitalizado por el contador 2 y en el ciclo 256 con el valor digitalizado por el contador 1, es decir se actualiza cada 150 ciclos o cada 50 μs, la tasa de muestreo deseada. La salida sample_out_ready produce un pulso activo de un periodo del reloj general del sistema, de 12 MHz, cuando los nuevos datos digitalizados están disponibles.

5.2.2. Diseño e implementación

El diseño de la interfaz del micrófono se va a realizar empleando la metodología de FSMD: a partir de un pseudocódio se va a generar un gráfico ASMD, este gráfico servirá como base para la descripción del módulo VHDL. Se recomienda revisar los capítulos 11 y 12 del libro "RTL Hardware Design Using VHDL" del profesor Pong P. Chu y las diapositivas de clase para familiarizarse con esta metodología. La entidad del módulo se define de la siguiente manera:

```
entity FSMD_microphone is
    Port ( clk_12megas : in STD_LOGIC;
           reset : in STD_LOGIC;
           enable_4_cycles : in STD_LOGIC;
           micro_data : in STD_LOGIC;
           sample_out : out STD_LOGIC_VECTOR (sample_size-1 downto 0);
           sample_out_ready : out STD_LOGIC);
end FSMD_microphone;
```

La señal enable_4_cycles es una señal de enable que se activa durante un periodo de reloj cada cuatro periodos. Proviene del generador del enables y permite trabajar a la interfaz del micrófono de manera efectiva a una velocidad de 3 MHz. El resto de puertos ya se ha descrito anteriormente.

Desde el esquema de muestreo, podemos describir el algoritmo de digitalización mediante el siguiente pseudo-código:

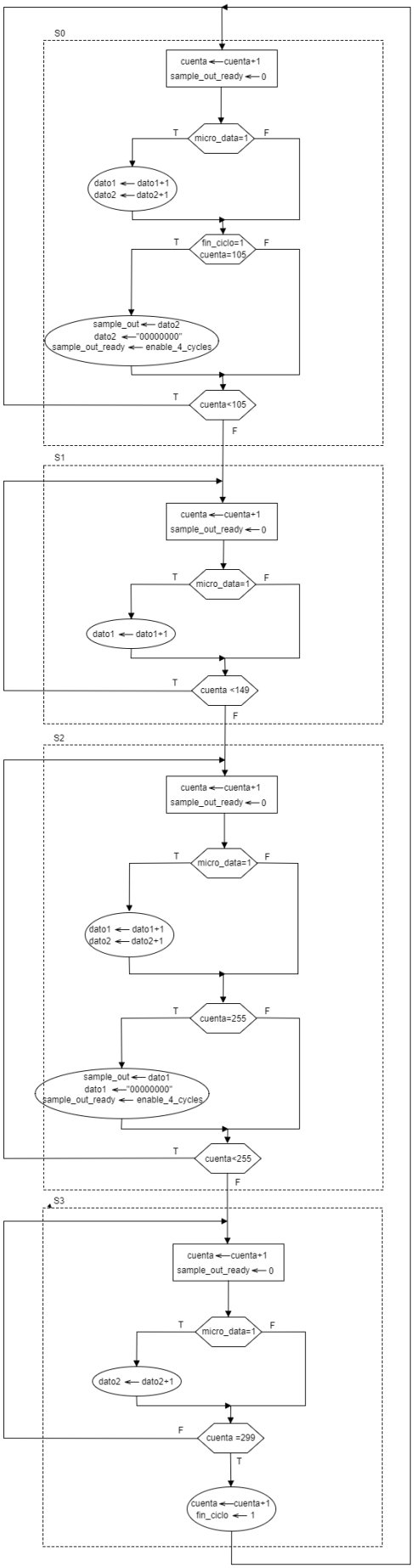
```
if (reset = 1)
    cuenta = 0
    dato1 = 0
    dato2 = 0
    primer_ciclo = 0
else
    if(0 <= cuenta <= 105 OR 150 <= cuenta <= 255)
       cuenta = cuenta + 1
       if(micro_data = 1)
          dato1 = dato1 + 1
          dato2 = dato2 + 1
    elsif(106 <= cuenta <= 149)
       cuenta = cuenta + 1
       if(micro_data = 1)
          dato1 = dato1 + 1
       if(primer_ciclo = 1 and cuenta = 106)
          sample_out = dato2
```

```
            dato2 = 0
            sample_out_ready = enable_4_cycles
        else
            sample_out_ready = 0
    else
        if(cuenta = 299)
            cuenta = 0
            primer_ciclo = 1
        else
            cuenta = cuenta + 1
        if(micro_data = 1)
            dato2 = dato2 + 1
        if(cuenta = 256)
            sample_out = dato1
            dato1 = 0
            sample_out_ready = enable_4_cycles
        else
            sample_out_ready = 0
```

Trata de comprender el pseudo-código, si tienes dificultades, contacta con el profesor de la asignatura.

Tarea 1.4:

Realiza el diagrama ASMD que describa la misma funcionalidad que el pseudo-código anterior.


S0
cuenta <= cuenta+1
sample_out_ready <= 0
T
micro_data=1
F
dato1 <= dato1+1
dato2 <= dato2+1
T
fin_ciclo=1
cuenta=105
F
sample_out <= dato2
dato2 <= "00000000"
sample_out_ready <= enable_4_cycles
T
cuenta<105
F
S1
cuenta <= cuenta+1
sample_out_ready <= 0
T
micro_data=1
F
dato1 <= dato1+1
T
cuenta <149
F
S2
cuenta <= cuenta+1
sample_out_ready <= 0
T
micro_data=1
F
dato1 <= dato1+1
dato2 <= dato2+1
T
cuenta=255
F
sample_out <= dato1
dato1 <= "00000000"
sample_out_ready <= enable_4_cycles
T
cuenta<255
F
S3
cuenta <= cuenta+1
sample_out_ready <= 0
T
micro_data=1
F
dato2 <= dato2+1
F
cuenta =299
T
cuenta <= cuenta+1
fin_ciclo <= 1

☞ **Tarea 1.5:**
Implementa el diagrama anterior en un fichero VHDL.

5.2.3. Estrategia de test

Antes de poder oír si el resultado de nuestra digitalización es correcto, necesitamos comprobar mediante testbenches que nuestro sistema funciona como esperamos. Podemos en un primer lugar comprobar que la máquina de estados recorre los estados como esperamos, generando las señales necesarias para reiniciar los contadores. Puedes instanciar el generador de enables en tu testbench para tener disponible la señal enable_4_cycles.

☞ **Tarea 1.6:**
Realiza un testbench que tenga la señal micro_data fija a '1' para comprobar que las transiciones de estados, la selección de los datos de salida y la activación de sample_out_ready se produce correctamente.

En una segunda fase podemos comprobar que la digitalización se hace correctamente, para ello podemos introducir señales pseudo-aleatorias en la señal micro_data y calcular el resultado esperado a mano. Podemos construir estas señales pseudo-aleatorias con sentencias del tipo:

```
a <= not a after 1300 ns;
b <= not b after 2100 ns;
c <= not c after 3700 ns;
micro_data <= a xor b xor c;
```

☞ **Tarea 1.7:**
Realiza un testbench que introduzca una señal pseudo-aleatoria en micro_data y comprueba que la digitalización se realiza correctamente.

5.3. Interfaz de la salida de audio

Para diseñar este módulo, es necesario conocer bien la interfaz que proporciona la salida de audio mono de la placa Nexys4 DDR y la modulación de anchura de pulso (PWM). Para ello se recomienda la lectura de la sección 16 del manual "Nexys4 DDR FPGA Board Reference Manual".

Como los datos se han muestreado a 20 kmuestras/s, la tasa de generación de pulsos tiene que ser la misma: un pulso cada 50 μs. Para la generación de la señal PWM se propone basarse en la sección 9.2.5 del libro "RTL Hardware Design Using VHDL" del profesor Pong P. Chu. En el esquema del libro, el valor de salida de un contador se compara con la palabra digital que se quiere convertir, si el valor del contador es más pequeño, la salida PWM se pone a '0' y a '1' en el caso contrario. En el caso de nuestro sistema, el contador va a aumentar su salida cada dos ciclos del reloj global, es decir, a una frecuencia de 6 MHz (166 ns de periodo). Es decir, que durante el periodo de muestreo de 50 μs, el contador va contar de 0 a 299. Como nuestro dato digital el de 8 bits, el máximo valor representado es 255, por lo tanto todos datos

que reproduzcamos van a tener una pequeña disminución de volumen por un factor de 255/300 = 0.85.

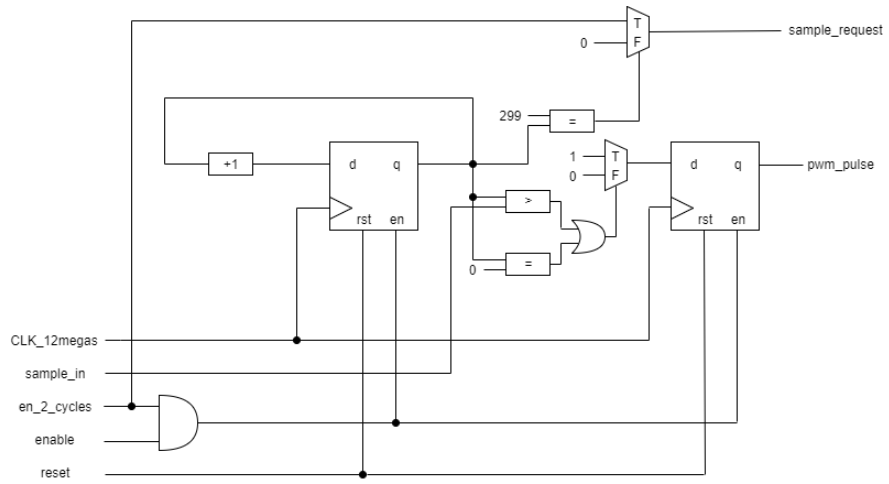
La entidad del bloque está declarada de la siguiente forma:

```
entity pwm is
   port(
      clk_12megas, in std_logic;
      reset, in std_logic;
      en_2_cycles: in std_logic;
      sample_in: in std_logic_vector(sample_size-1 downto 0);
      sample_request: out std_logic;
      pwm_pulse: out std_logic
   );
end pwm;
```

La señal en_2_cycles es una señal que se activa cada dos ciclos y es la que nos permite trabajar de manera efectiva a la mitad de la frecuencia. Cada vez que el módulo haya terminado una cuenta, esto es que haya llegado a 299 y vaya a pasar a 0, tiene que poner un pulso activo en sample_request de duración un sólo periodo del reloj general para solicitar una nueva muestra a la entrada.

☞ **Tarea 1.8:**

Modifica el diseño propuesto en el libro del Dr. Chu para que sea compatible con las especificaciones de nuestro sistema. Es decir, que el contador cuente de 0 a 299 e incluya reset y enable y que el circuito produzca la salida sample_request en el momento apropiado y con la duración apropiada. Dibuja el esquemático de tu diseño en la parte inferior.



☞ **Tarea 1.9:**

Implementa el esquemático anterior en un fichero VHDL.

5.3.1. Estrategia de test

Antes de probar el módulo en la placa, necesitamos asegurarnos mediante simulaciones de que el funcionamiento es el esperado. En concreto, tenemos que comprobar que el contador cuente cada dos ciclos, que la cuenta se reinicia al llegar a 299 y que la señales pwm_pulse y sample_request son las esperadas. Para ello podemos podemos verificar el funcionamiento introduciendo los valores extremos ("0000 0000" y "1111 1111") y algún valor aleatorio intermedio. Puedes instanciar el generador de enables en tu testbench para tener disponible la señal enable_2_cycles.

☞ **Tarea 1.10:**

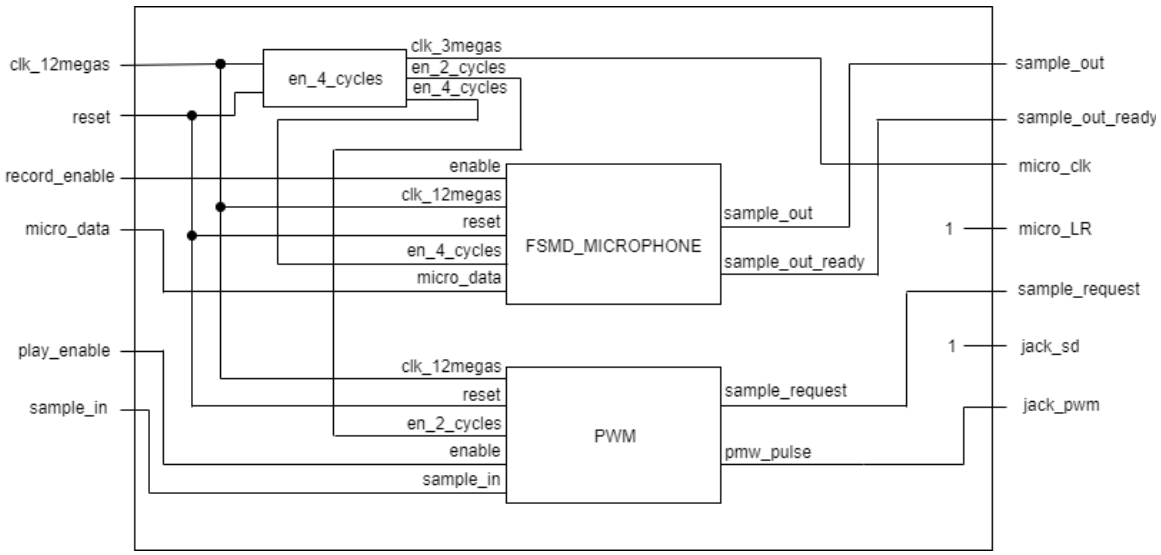
Realiza un testbench que verifique el funcionamiento de la interfaz de la salida de audio.

5.4. Integración de la interfaz de audio

Una vez implementados los tres módulos anteriores, el siguiente paso del diseño consiste en la instanciación de cada uno de estos bloques en un único módulo que formará la interfaz de audio completa.

☞ **Tarea 1.11:**

Dibuja el esquemático a nivel bloque de la interfaz de audio completa. Ten en cuenta que record_enable y play_enable especifican cuándo están activas las interfaces del micrófono y de la salida de audio, respectivamente. Del mismo modo, tienes que asignar un '1' tanto a micro_LR, como a jack_sd.



☞ **Tarea 1.12:**
Implementa en VHDL la interfaz de audio completa. Para ello utiliza un estilo de código estructural que instancie cada uno de los tres bloques desarrollados anteriormente.

5.4.1. Estrategia de test

Para verificar la funcionalidad del bloque completo, puedes reutilizar los estímulos empleados en los testbenches anteriores.

☞ **Tarea 1.13:**
Diseña un testbench que verifique la funcionalidad de la interfaz de audio completa.

5.5. Implementación física y test en placa

Una vez verificada la funcionalidad del bloque completo mediante un testbench, vas a realizar la implementación física del bloque en la placa. Para poder comprobar que el sistema efectivamente digitaliza bien la señal del micrófono y transforma bien la palabra digital en un pulso PWM, el test en placa va a consistir en hacer que lo que se grabe por el micrófono salga por la salida de audio.

Para lograr esto, necesitas un pequeño controlador que genere la señal de 12 MHz a partir del reloj de 100 MHz de la placa, mantena record_enable y play_enable a '1' y que introduzca la palabra digitalizada por la interfaz del micrófono (sample_out) en la interfaz de la salida de audio (sample_in). No necesitan ningún tipo de sincronización especial, mientras que sample_in esté estable durante 50 μs (no importa si cambia cuando el contador se pone a 0 o no). Este controlador tiene la siguiente declaración de entidad:

```
entity controlador is
  Port (
    clk_100Mhz : in std_logic;
    reset: in std_logic;
    --To/From the microphone
    micro_clk : out STD_LOGIC;
    micro_data : in STD_LOGIC;
    micro_LR : out STD_LOGIC;
    --To/From the mini-jack
    jack_sd : out STD_LOGIC;
    jack_pwm : out STD_LOGIC
  );
end controlador;
```

☞ **Tarea 1.14:**
Implementa en VHDL el controlador. Utiliza estilo estructural. Necesitas emplear el asistente arquitectural de Vivado para generar el reloj de 12 MHz a partir del rejol de 100 MHz.

☞ **Tarea 1.15:**

Diseña un testbench que verifique la funcionalidad del controlador. Puedes utilizar los estímulos empleados en testbenches anteriores.

☞ **Tarea 1.16:**

Escribe el fichero de restricciones .xdc. Sintetiza y realiza la implementación física del diseño. Genera el fichero .bit y programa la FPGA. Comprueba el funcionamiento correcto conectando unos auriculares a la salida de audio de la placa. Si la salida de audio es muy ruidosa, puedes jugar con el valor de micro_LR y ponerlo a '0' en lugar de a '1'.

Guarda en un fichero de texto todos los mensajes de warning que hayan aparecido en el proceso anterior.

Avisa al profesor para comprobar el funcionamiento.

Puedes aprovechar para incluir pequeñas funcionalidades adicionales al controlador. Por ejemplo, puedes hacer un indicador de intensidad de señal grabada conectando 8 LEDs a la palabra digitalizada. También puedes hacer que el sistema sólo funcione cuando se aprieta uno de los botones.

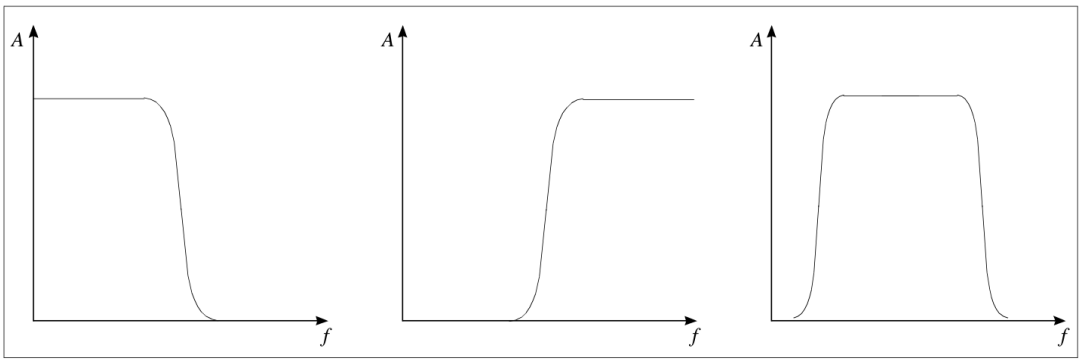
6. Bloque 2: Filtro FIR

Este bloque implementa un filtro FIR empleando VHDL y una arquitectura basada en una ruta de datos controlada por una máquina de estados finitos. El diseño de este bloque profundiza en conceptos de cuantificación, implementación de algoritmos, arquitecturas segmentadas y test-benches de complejidad media.

Crédito: Este bloque está basado en el "Mini-project" "FIR-Filter Design" del curso IL2204 "DSP Design usign HDL" del "Royal Institute of Technology" de Estocolmo, Suecia.

6.1. Filtros digitales

Hablamos de filtros en el contexto de procesado digital de señales para referirnos a circuitos que permiten pasar señales dentro de un determinado rango de frecuencias, mientras que bloquean a las señales que caigan fuera de este rango. El rango de frecuencias que el filtro permite pasar se conoce como banda de paso y el rango de frecuencias que no puede pasar se conoce como banda prohibida. Se pueden clasificar los filtros con respecto a la posición de las bandas de paso y prohibida: Tenemos filtros paso bajo, paso alto y paso banda, sus funciones de transferencia se ilustran en la siguiente figura.



Una manera común de expresar la relación de la salida del filtro $y[n]$, donde n es el número de muestra, en función de la entrada $x[n]$ de un filtro, es la siguiente:

$$y[n] = \sum_{m=0}^{N-1} \alpha[m]x[n-m] - \sum_{m=1}^{N-1} \beta[m]y[n-m] \tag{1}$$

El primer término de la ecuación relaciona la salida del filtro con entradas pasadas. El segundo término relaciona la salida del filtro con salidas pasadas. Podemos distinguir dos tipos de filtros:

- Filtros de repuesta al impulso infinita (IIR) en los que están presentes los dos términos de la ecuación 1. También son conocidos como filtros de respuesta infinita, ya que el segundo término describe un comportamiento recursivo (la salida es función de la salida).
- Filtros de respuesta al impulso finita (FIR) en los que sólo está presente el primer término de la ecuación 1. La señal de salida de un filtro FIR es una suma ponderada de la señal de entrada. La respuesta al impulso de un filtro FIR tiene una longitud de N datos de salida.

Podemos destacar las siguientes propiedades de los filtros FIR:

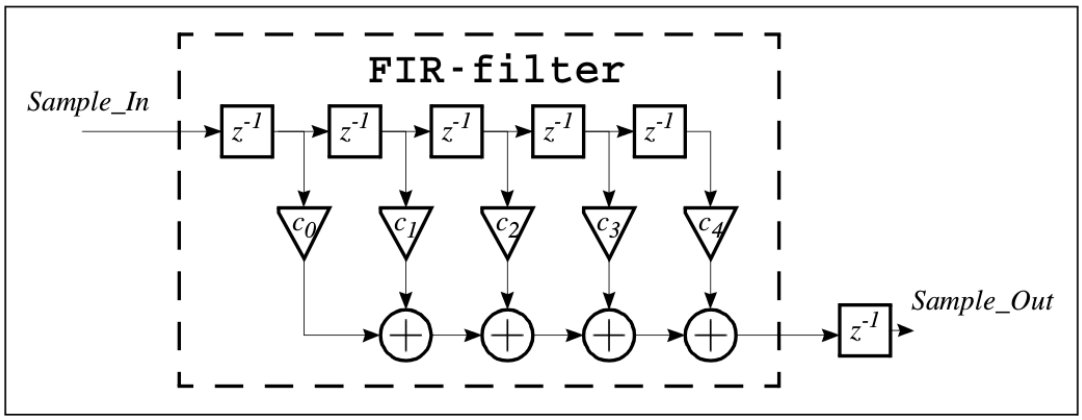
- La respuesta al impulso unidad de un filtro FIR es la secuencia de sus coeficientes $\alpha[m]$. Esto nos proporciona una manera sencilla de comprobar el funcionamiento del filtro. Lo único que tenemos que hacer es introducir una delta unidad (x(0)=1, x(n)=0 para n distinto de 0, también conocida como delta de Dirac) y obtendremos la secuencia de los coeficientes.
- Si todos los coeficientes se multiplican por una constante, se cambia la ganancia del filtro, no sus características.
- La ganancia en continua del filtro equivale a la suma de todos sus coeficientes.

6.2. Implementación hardware de filtros FIR

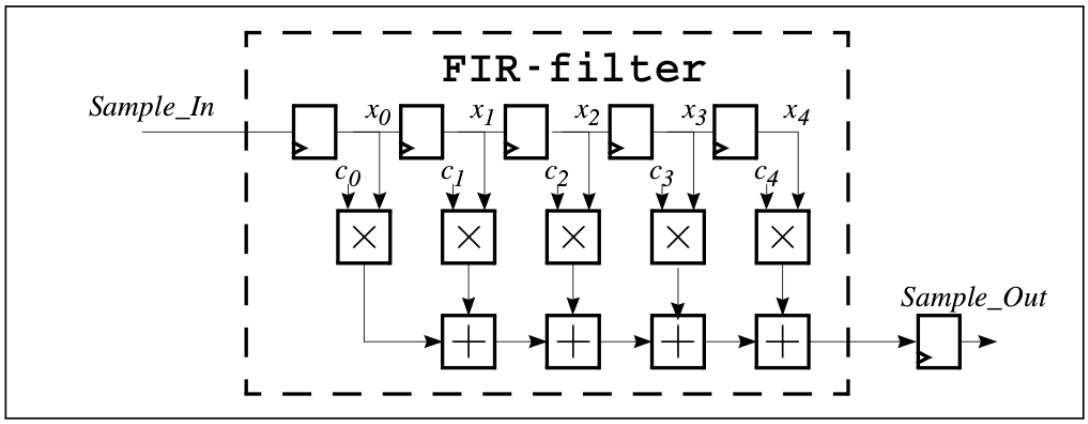
La transformada Z de un filtro FIR viene dada por la siguiente expresión

$$H_{FIR}(z) = \sum_{n=0}^{N-1} c_n z^{-n} \tag{2}$$

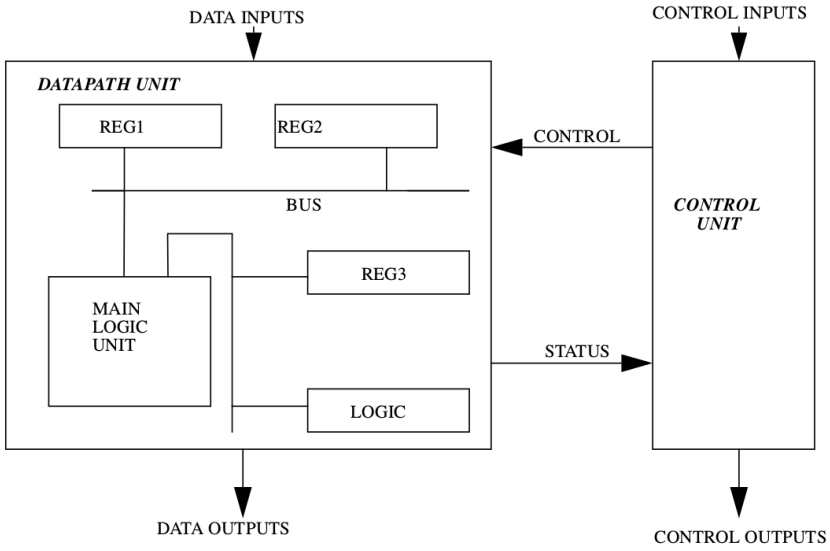
Donde z^{-n} en la ecuación corresponde a líneas de retardo y c_n son los coeficientes que ponderan las muestras de entrada, equivalentes a las $\alpha[m]$ de la ecuación 1. En la ecuación anterior N es el orden del filtro, que también se corresponde como el número de etapas del filtro. El diagrama del flujo de señal de un filtro FIR de 5 etapas se muestra en la siguiente figura. Observa que la última línea de retardo (el registro a la salida) no forma parte de la ecuación anterior.



Las z^{-n}, correspondientes a los retardos, se implementan empleando registros. Para aplicar los coeficientes a cada muestra se emplean multiplicadores. Las sumas parciales se hacen a través de sumadores. La siguiente figura muestra la implementación directa de un filtro FIR de 5 etapas.



6.3. Sistemas basados en rutas de datos



La mayoría de los sistemas digitales se basa en una ruta de datos y un controlador (normalmente en forma de una máquina de estados finitos). La ruta de datos consiste en unidades de almacenamiento (como flipflops, registros y memorias) y unidades combinacionales (como ALUs, multiplexores, desplazadores, comparadores, etc.) conectados mediante buses (líneas de varias interconexiones). Los valores guardados en las unidades de almacenamiento se leen después del flanco de subida (o de bajada) de reloj, se procesan en las unidades combinacionales y se guardan en las unidades de almacenamiento en el siguiente flanco de reloj. Por otro lado, el controlador proporciona información de control y recibe información de estado de la ruta de datos. Normalmente se implementa a través de una máquina de estados finitos controlada por entradas y salidas del mundo exterior. La primera etapa en el diseño de un sistema basado en una ruta de datos es la descripción algorítmica de la especificación funcional de un sistema digital. A partir de esta descripción algorítmica se construye la ruta de datos.

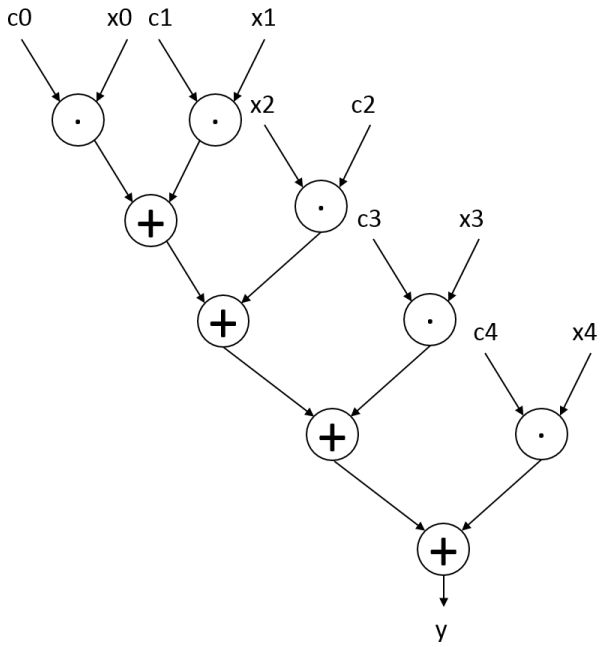
6.4. Metodología de implementación de algoritmos de procesado de señal

El proceso general de implementación de algoritmos para procesado digital de señales se suele dividir en las siguientes etapas:

- Asignación
- Planificación temporal
- Vinculación
- Implementación

El proceso no es estrictamente lineal, sino que puede que tengamos que retroceder alguna etapa para realizar alguna optimización concreta, o puede que podamos tomar decisiones sobre etapas posteriores sin haber finalizado algún paso intermedio.

Para ilustrar cada una de estas etapas, vamos a emplear el caso del filtro FIR de cinco etapas visto en las secciones anteriores que se puede representar por el siguiente grafo o diagrama de flujo.



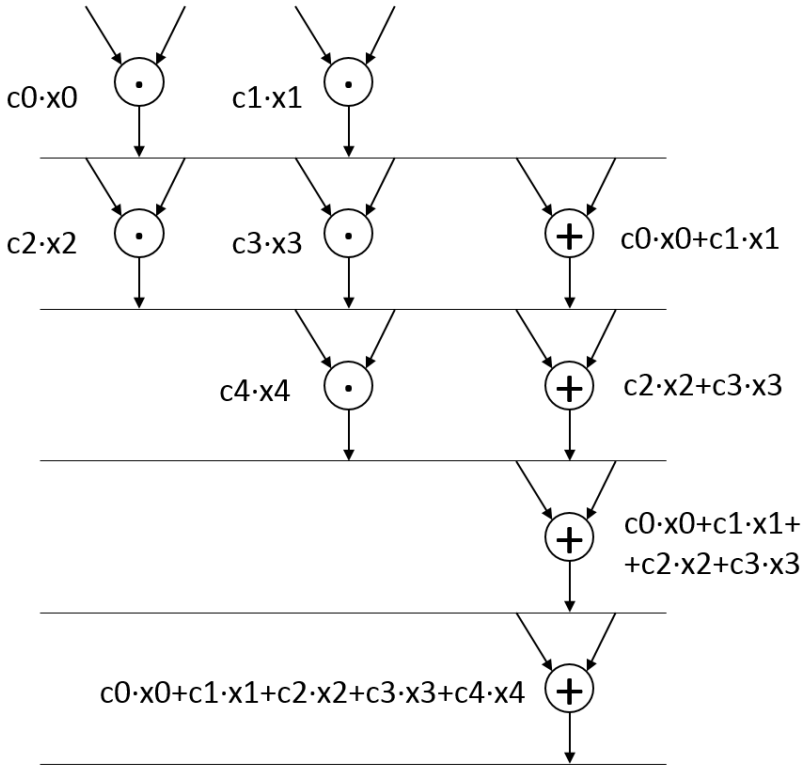
6.4.1. Asignación

Designamos asignación a la selección del número y tipo de unidades hardware que van a implementar las funciones descritas por el diagrama de flujo del algoritmo. En el caso del filtro FIR de cinco etapas, podríamos decidir implementarlo con un multiplicador y un sumador, dos multiplicadores y un sumador, dos medios multiplicadores y dos sumadores, etc. En función del número y tipo de unidades que se escojan se obtendrán unos valores distintos de área, throughput y latencia, con lo cual hay que tener en cuenta las restricciones del sistema para tomar esta decisión.

6.4.2. Planificación temporal

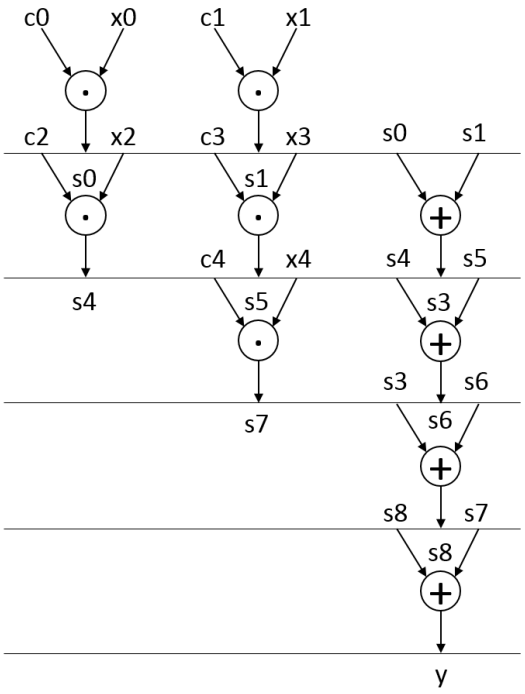
Durante esta fase, se distribuyen las operaciones del diagrama de flujo en el tiempo teniendo en cuenta que no se pueden exceder el número de unidades disponibles. En este momento se puede jugar con las propiedades distributibas de los operandos suma y multiplicación para reordenar el diagrama de flujo y obtener estructuras más interesantes desde el punto de vista hardware.

Fijando una asignación de dos multiplicadores y un sumador, la siguiente figura muestra la planificación temporal del diagrama de flujo del filtro FIR. Cada línea horizontal separa las operaciones que se producen durante un ciclo de control del siguiente. En este caso, para reducir la latencia del algoritmo, en lugar de hacer $((c0 \cdot x0 + c1 \cdot x1) + (c2 \cdot x2) + (c3 \cdot x3))$, que desaprovecharía la opción de utilizar dos multiplicadores simultáneamente, se ha propuesto $((c0 \cdot x0 + c1 \cdot x1) + (c2 \cdot x2 + c3 \cdot x3))$.



6.4.3. Vinculación

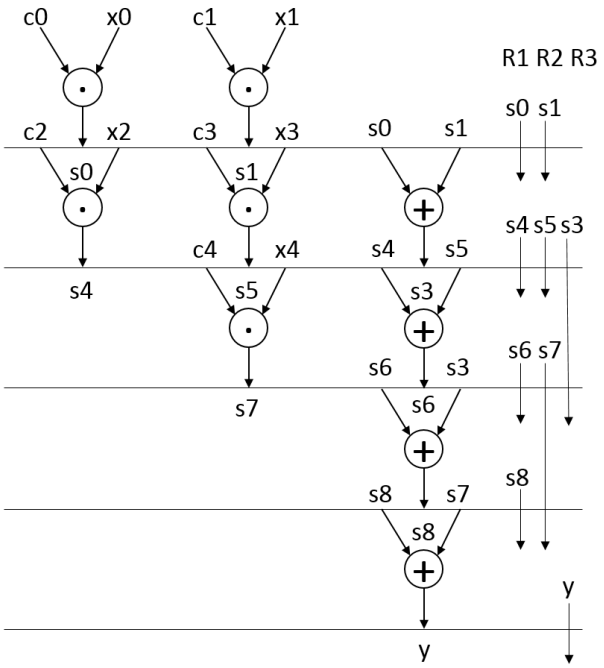
Cada unidad concreta se asocia a una operación concreta en cada unidad de tiempo. También se especifica a qué puerto concreto se conecta cada operando. Las señales internas se nombran para facilitar las procedimientos de las siguientes fases. La siguiente figura muestra el resultado de esta fase para el filtro FIR.



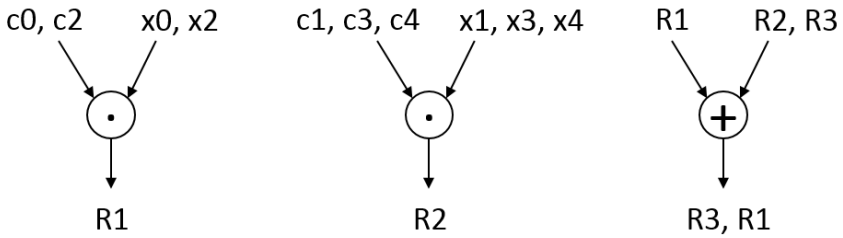
6.4.4. Implementación

Esta es la fase final del diseño y es en la que se obtiene como resultado una estructura hardware que podemos implementar, por ejemplo, en una FPGA. Esta etapa se puede subdividir en distintas tareas: Análisis del tiempo de vida de las variables, minimización del número de registros y multiplexores y creación de la estructura hardware.

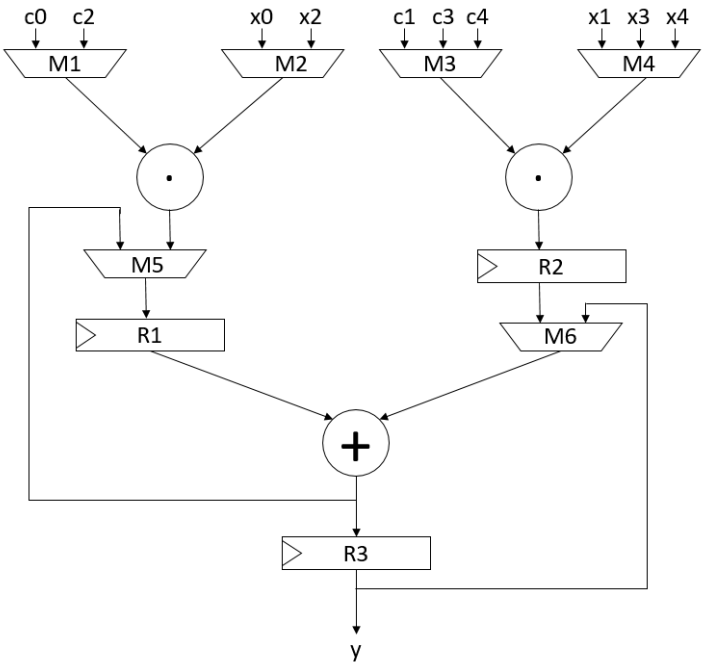
En el análisis del tiempo de vida de las variables se estudia la necesidad de almacenamiento interno (registros) de la estructura que vamos a generar. A partir del diagrama generado después de la vinculación, podemos analizar los momentos en que se producen las señales internas y los momentos en que se consumen, obteniendo así su tiempo de vida. En la siguiente figura se ha representado mediante una flecha el tiempo de vida de las distintas variables. El máximo número de flechas paralelas, tres en el ejemplo, establece el número de registros necesarios. Una vez establecido el número de registros necesarios, R1, R2 y R3, podemos asignar cada variable a un registro para cada momento. En el ejemplo, en el registro R0 se irán guardando s0, s4, s6 y s8 según se desenvuelve el algoritmo.



Cada entrada de cada operador y registro recibe distintos datos a lo largo del tiempo, es por ello que tenemos que asignar multiplexores que controlen qué entrada se conecta en cada momento. La siguiente figura estudia las posibles conexiones que pueden necesitar los módulos aritméticos del filtro FIR.



Finalmente, con toda la información que hemos ido ganando en las distintas etapas, ya podemos realizar una estructura hardware que sea capaz de realizar el algoritmo de procesado digital con las unidades que hemos elegido en la etapa de asignación. La siguiente estructura muestra la implementación del filtro FIR de cinco etapas con dos multiplicadores y un sumador.



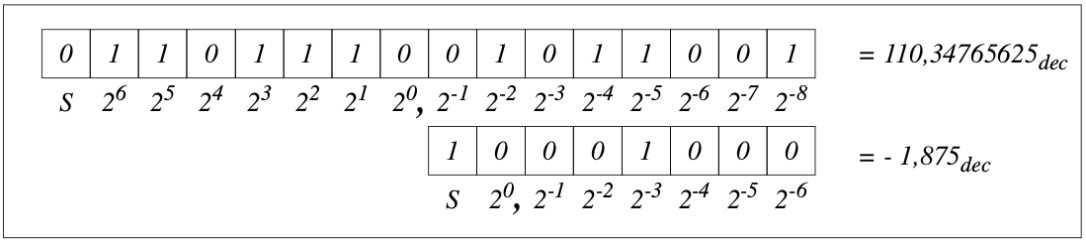
Hasta aquí llegan los pasos para la creación de la ruta de datos, como es lógico, se espera que un controlador externo se encargue de proporcionar las señales de control de los multiplexores y los load de los registros en el momento apropiado. Para implementar el controlador, viene muy bien un cronograma que especifique en cada instante de tiempo qué señal tiene que dejar pasar cada multiplexor y qué señal tiene que tener guardada cada registro. La siguiente figura muestra el cronograma para el ejemplo del filtro FIR. En cada instante se especifica qué señal tiene que dejar pasar cada multiplexor y la señal de control asociada entre paréntesis, así como los datos guardados en los registros para cada instante de tiempo.

	t1	t2	t3	t4	t5	t6
M1	C0 (0)	C2 (1)				
M2	X0 (0)	X2 (1)				
M3	c1 (00)	c3 (01)	c4 (10)			
M4	x1 (00)	x3 (01)	x4 (10)			
M5	s0 (1)	s4 (1)	s6 (0)	s8 (0)		
M6		s1 (0)	s5 (0)	s3 (1)	s7 (0)	
R1		s0	s4	s6	s8	
R2		s1	s5	s7	s7	
R3			s3	s3		y

6.5. Notación en punto fijo

En este bloque vamos a trabajar con números representados en notación en punto fijo. Se puede establecer una analogía entre los números decimales en punto fijo y los binarios en punto

fijo. Un número binario en punto fijo tiene el siguiente aspecto: "10001110.101". El punto de un número binario en punto fijo se pone a la derecha de cifra que representa 2^0. Las cifras a la izquierda del punto se interpretan como un binario sin punto y las cifras a la derecha representan la parte decimal o fraccionaria. La siguiente figura muestra dos ejemplos de notación en punto fijo empleando complemento a dos. Observa la figura y asegúrate de entender el procedimiento de conversión entre binario en punto fijo y base 10.



Para este bloque vamos a utilizar la siguiente notación para representar los tamaños de números en punto fijo: $< a.b >$ donde a es el número de bits que representan la parte entera y b es el número de bits necesarios para representar la parte decimal. Por ejemplo el número superior de la anterior figura necesita 16 bits, y su tamaño estaría representado por $< 8{,}8 >$. Con esta notación nos es útil para establecer los tamaños resultantes de las operaciones de multiplicación y suma: Multiplicación: Si $< 3{,}4 >$ se multiplica con $< 5{,}6 >$ el resultado será $< 3{+}5{,}4{+}6 >=< 8{,}10 >$. Si a la salida tenemos que proporcionar un dato de 10 bits, truncamos el resultado y obtenemos $< 8{,}2 >$ (o $< 7{,}3 >$ si sabemos que el resultado final de acumular estos números no va a producir un desbordamiento). Suma: Si $< 3{,}4 >$ se suma a $< 5{,}6 >$ el resultado será $< max(3,5)+1 \; . \; max(4,6) >=< 6{,}6 >$. El +1 en la parte entera tiene en cuenta posibles desbordamientos (acarreos de salida).

☞ **Tarea 2.1:**

¿Cuál es el rango de un número de 8 bits en punto fijo $< 1{,}7 >$ en complemento a dos? ¿Cuál es su precisión?

RANGO:

<1,7> { 1 parte entera; 7 parte decimal

Así:

1,0000000 = -1+0 = -1

0,1111111 = +0,9921875 = 127/128

El rango será: [-1,127/128]

PRECISIÓN:

La precisión será:

1 LSB = 2^{-7} = 1/128 = $7{,}8125 \cdot 10^{-3}$

6.6. Especificación del bloque

En este bloque vamos a implementar un filtro FIR de 5 etapas, cuya interfaz se describe en la siguiente entidad de VHDL:

```
entity fir_filter is
    Port ( clk : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Sample_In : in  signed (sample_size-1 downto 0);
           Sample_In_enable : in  STD_LOGIC;
           filter_select: in STD_LOGIC; --0 lowpass, 1 highpass
           Sample_Out : out  signed (sample_size-1 downto 0);
           Sample_Out_ready : out STD_LOGIC);
end fir_filter;
```

- Tanto Sample_In, Sample_Out, como los coeficientes del filtro se tienen que codificar con palabras de 8 bits en complemento a dos $< 1{,}7 >$.
- Sample_In_enable es una entrada de control que informa de cuándo se ha actualizado el valor de Sample_In con un pulso activo durante un ciclo de reloj.
- filter_select es una entrada de control que selecciona el filtrado que se va a realizar: '0' para el filtro de paso bajo, '1' para el filtro de paso alto.
- Sample_Out_ready es una salida de control que informa de cuándo se ha actualizado el valor de Sample_Out con un pulso activo durante un ciclo de reloj.
- El filtro paso bajo utiliza los siguientes coeficientes: c0 = c4 = +0.039, c1 = c3 = +0.2422, c2 = +0.4453.
- El filtro paso alto utiliza los siguientes coeficientes: c0 = c4 = -0.0078, c1 = c3 = -0.2031, c2 = +0.6015.
- La implementación del filtro emplea una estructura de ruta de datos controlada por un controlador.
- La ruta de datos sólo emplea dos medios multiplicadores y un sumador.

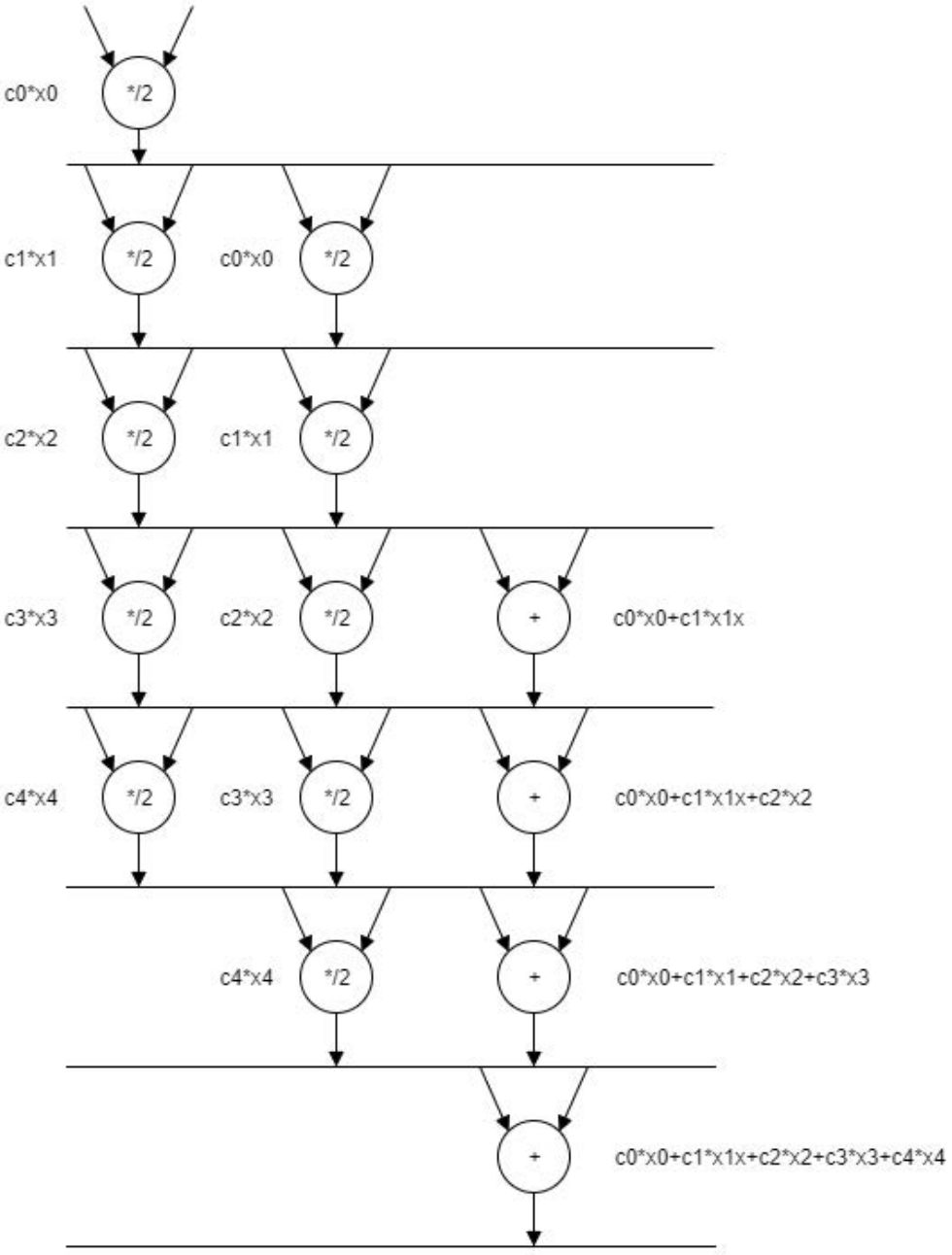
6.7. Tareas prácticas

6.7.1. Creación de la ruta de datos

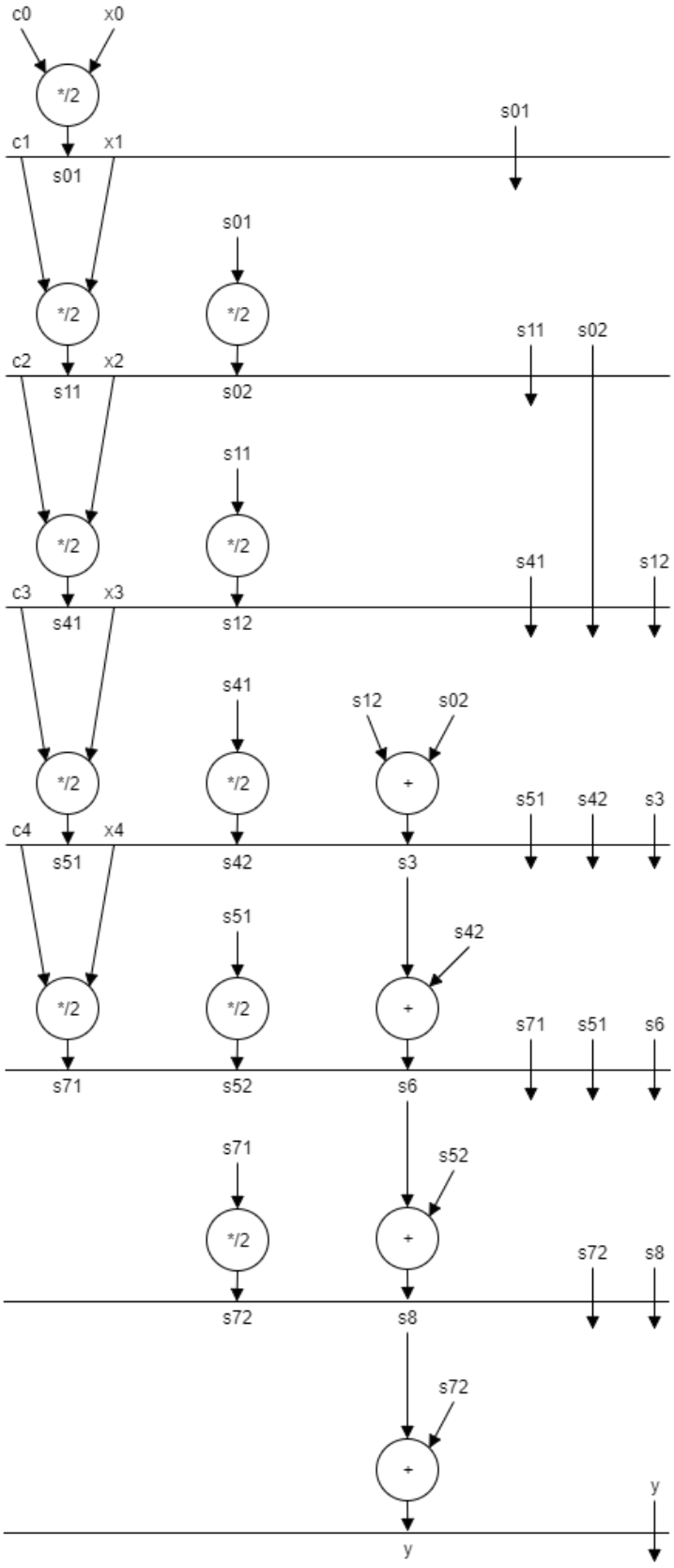
Tarea 2.2:

☞ Realiza sobre papel la implementación de un filtro FIR de 5 etapas con la siguiente asignación: Dos medios multiplicadores y un sumador. Realiza todos los pasos descritos en la sección 6.4.

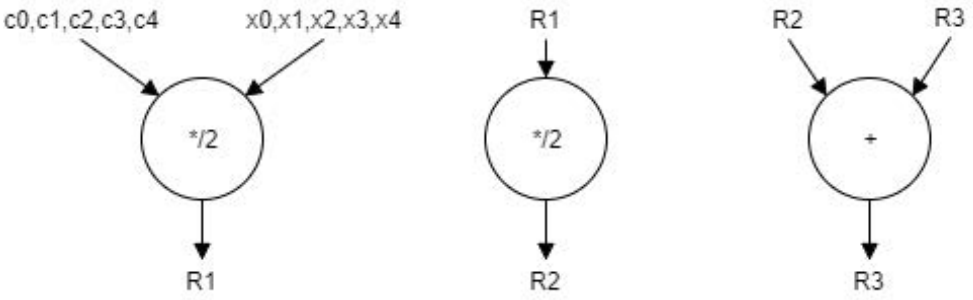
☞ **Planificación temporal:**



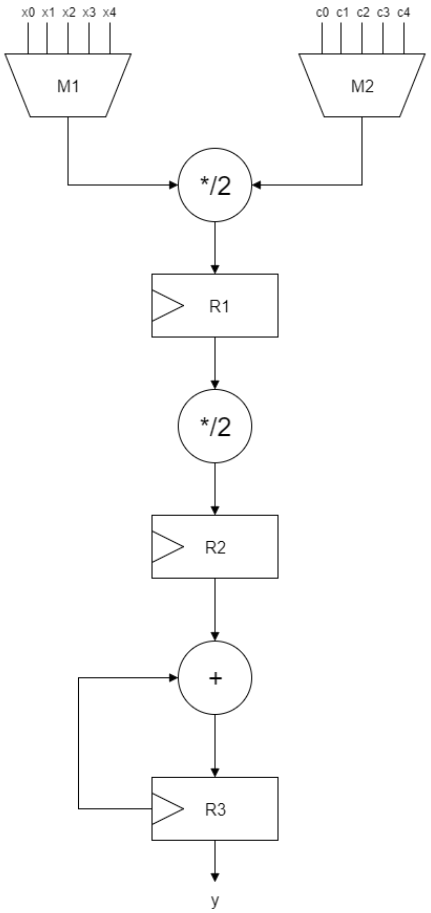
☞ **Vinculación, análisis de tiempo de vida de variables y asignación de registros:**



☞ **Análisis de conexiones para la extracción de multiplexores:**



☞ **Implementación:**



☞ **Cronograma:**

	t1	t2	t3	t4	t5	t6	t7	t8
M1	c0 (000)	c1 (001)	c2 (010)	c3 (011)	c4 (100)			
M2	x0 (000)	x1 (001)	x2 (010)	x3 (011)	x4 (100)			
R1		s01	s11	s41	s51	s71		
R2			s02	s02	s42	s52	s72	
R3				s12	s3	s6	s8	y

Tarea 2.3:

☞ Realiza un análisis de cuantificación de las señales, de manera que especifiques cuántos bits vas a emplear en cada señal y en qué posición va a estar el punto decimal. Emplea la notación y las metodologías presentadas en clase.

PASO BAJO

- c0 = c4 = +0,039 $\longrightarrow$ $2^{-5}+2^{-8}+2^{-9}+2^{-10}+2^{-11}+2^{-12}+2^{-13}+2^{-14} \sim 2^{-5}+2^{-7}$ = 0,03906 = **0,0000101**
- c1 = c3 = +0,2422 $\longrightarrow$ $2^{-3}+2^{-4}+2^{-5}+2^{-6}+2^{-7}+2^{-14} \sim 2^{-3}+2^{-4}+2^{-5}+2^{-6}+2^{-7}$ = 0,2421875 = **0,0011111**
- c2 = +0,4453 $\longrightarrow$ $2^{-2}+2^{-3}+2^{-4}+2^{-7}$ = 0,4453125 = **0,0111001**

PASO ALTO

- c0 = c4 = -0,078 $\longrightarrow$ $-1+2^{-1}+2^{-2}+2^{-3}+2^{-4}+2^{-5}+2^{-6}+2^{-7}$ = -0,0078125 = **1,1111111**
- c1 = c3 = -0,2031 $\longrightarrow$ $-1+2^{-1}+2^{-2}+2^{-5}+2^{-6}$ = -0,203125 = **1,1100110**
- c2 = +0,6015 $\longrightarrow$ $2^{-1}+2^{-4}+2^{-5}+2^{-7}$ = 0,6015625 = **0,1001101**

Tarea 2.4:

☞ Escribe modelos VHDL para todos los componentes que se emplean en tu ruta de datos. Los operadores aritméticos (suma y multiplicación) no necesitan describirse en un estilo estructural. Para conseguir el medio multiplicador, añade un registro extra a la salida de un multiplicador completo, de esta forma se comportará como un multiplicador perfectamente segmentado. La herramienta de síntesis se encargará de mover este registro al interior del multiplicador de manera óptima. Comprueba el funcionamiento correcto de tus componentes creando los correspondientes testbenches.

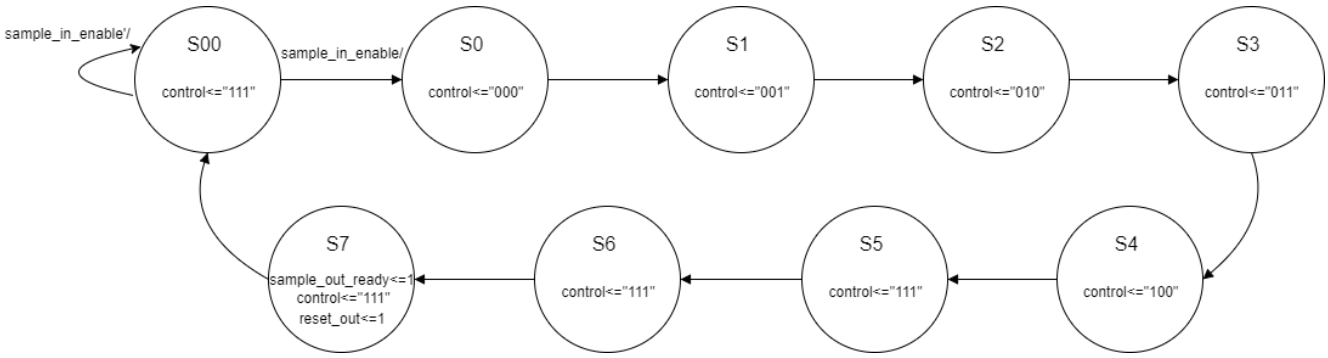
Tarea 2.5:

☞ Escribe un modelo estructural de la ruta de datos completa en VHDL. Crea un testbench para la ruta de datos.

6.7.2. Creación del controlador

☞ **Tarea 2.6:**

Dibuja el diagrama de estados que describe el controlador de tu filtro FIR. Sigue la metodología de implementación de máquinas de estados finitos vista en clase.



☞ **Tarea 2.7:**
Escribe el código VHDL correspondiente a tu controlador.

☞ **Tarea 2.8:**
Escribe el código VHDL estructural correspondiente a tu filtro completo.

☞ **Tarea 2.9:**
Escribe el código VHDL correspondiente a un testbench que introduzca un único impulso a la entrada de valor +0.5. ¿Qué secuencia esperas en Sample_Out si en Sample_In la secuencia es (0, 0, 0, 0, X, 0, 0, 0, 0)? Considera que X es el mayor/menor número positivo/negativo (cuatro casos) que se puede representar. Ten en cuenta la cuantificación de tus señales. Asegúrate de que tu diseño proporciona los valores esperados.

Impulso único de valor +0,5

0,5*C0(bajo) = 0,5*C4(bajo)	10000000*00000101 = 0**000 0001 0**100 0000
0,5*C0(alto) = 0,5*C4(alto)	10000000*11111111 = 1**111 1111 1**100 0000
0,5*C1(bajo) = 0,5*C3(bajo)	10000000*00011111 = 0**000 0111 1**100 0000
0,5*C1(alto) = 0,5*C3(alto)	10000000*11100110 = 1**111 1001 1**000 0000
0,5*C2(bajo)	10000000*00111001 = 0**000 1110 0**100 0000
0,5*C2(alto)	10000000*01001101 = 0**001 0011 0**100 0000

Positivo Mayor

X*C0(bajo) = X*C4(bajo)	01111111*00000101 = 0**000 0010 0**111 1011
X*C0(alto) = X*C4(alto)	01111111*11111111 = 1**111 1111 1**000 0001
X*C1(bajo) = X*C3(bajo)	01111111*00011111 = 0**000 1111 0**110 0001
X*C1(alto) = X*C3(alto)	01111111*11100110 = 1**111 0011 0**001 1010
X*C2(bajo)	01111111*00111001 = 0**001 1100 0**100 0111
X*C2(alto)	01111111*01001101 = 0**010 0110 0**011 0011

Positivo Menor

X*C0(bajo) = X*C4(bajo)	00000001*00000101 = 0**000 0000 0**000 0101
X*C0(alto) = X*C4(alto)	00000001*11111111 = 1**111 1111 1**111 1111
X*C1(bajo) = X*C3(bajo)	00000001*00011111 = 0**000 0000 0**001 1111
X*C1(alto) = X*C3(alto)	00000001*11100110 = 1**111 1111 1**110 0110
X*C2(bajo)	00000001*00111001 = 0**000 0000 0**011 1001
X*C2(alto)	00000001*01001101 = 0**000 0000 0**100 1101

Negativo Mayor

X*C0(bajo) = X*C4(bajo)	11111111*00000101 = 1**111 1111 1**111 1011
X*C0(alto) = X*C4(alto)	11111111*11111111 = 0**000 0000 0**000 0001
X*C1(bajo) = X*C3(bajo)	11111111*00011111 = 1**111 1111 1**110 0001
X*C1(alto) = X*C3(alto)	11111111*11100110 = 0**000 0000 0**001 1010
X*C2(bajo)	11111111*00111001 = 1**111 1111 1**100 0111
X*C2(alto)	11111111*01001101 = 1**111 1111 1**011 0011

Negativo Menor

X*C0(bajo) = X*C4(bajo)	10000000*00000101 = 1**111 1101 1**000 0000
X*C0(alto) = X*C4(alto)	10000000*11111111 = 0**000 0000 1**000 0000
X*C1(bajo) = X*C3(bajo)	10000000*00011111 = 1**111 0000 1**000 0000
X*C1(alto) = X*C3(alto)	10000000*11100110 = 0**000 1101 0**000 0000
X*C2(bajo)	10000000*00111001 = 1**110 0011 1**000 0000
X*C2(alto)	10000000*01001101 = 1**101 1001 1**000 0000

Tarea 2.10:

☞ Escribe el código VHDL correspondiente a un testbench que introduzca en la entrada la siguiente secuencia (0, 0.5, 0, 0.125, 0, 0, 0, 0, ...). ¿Qué secuencia esperas en Sample_Out para esta secuencia de entrada? Asegúrate de que tu diseño proporciona los valores esperados.

0,5*C0(bajo) = 0,5*C4(bajo)	10000000*00000101 = 0**000 0001 0**100 0000
0,5*C0(alto) = 0,5*C4(alto)	10000000*11111111 = 1**111 1111 1**100 0000
0,5*C1(bajo) = 0,5*C3(bajo)	10000000*00011111 = 0**000 0111 1**100 0000
0,5*C1(alto) = 0,5*C3(alto)	10000000*11100110 = 1**111 1001 1**000 0000
0,5*C2(bajo)	10000000*00111001 = 0**000 1110 0**100 0000
0,5*C2(alto)	10000000*01001101 = 0**001 0011 0**100 0000

0,125*C0(bajo) = 0,125*C4(bajo)	01111111*00000101 = 0**000 0000 0**101 0000
0,125*C0(alto) = 0,125*C4(alto)	01111111*11111111 = 1**111 1111 1**111 0000
0,125*C1(bajo) = 0,125*C3(bajo)	01111111*00011111 = 0**000 0001 1**111 0000
0,125*C1(alto) = 0,125*C3(alto)	01111111*11100110 = 1**111 1110 0**110 0000
0,125*C2(bajo)	01111111*00111001 = 0**000 0011 1**001 0000
0,125*C2(alto)	01111111*01001101 = 0**000 0100 1**101 0000

6.7.3. Creación de un testbench avanzado

Si tuviéramos que hacer pruebas más exhaustivas para comprobar el funcionamiento de nuestro filtro, el empleo de los testbenches que hemos usado hasta ahora dejaría de ser práctico, principalmente por dos razones:

- Cada vez que necesitamos probar una secuencia de entrada nueva, tenemos que modificar y recompilar el testbench.
- Para comprobar que la secuencia de salida que obtenemos es la correcta, tenemos que haberla calculado a mano previamente y, después de ejecutar el testbench, tenemos que comprobar uno a uno que los datos son correctos.

Para solucionar estos problemas podemos echar mano de las opciones de escritura y lectura de ficheros de VHDL. Los ficheros a los que se accede desde un código VHDL pueden o bien contener datos de un único tipo (std_logic_vector, integer, singed, etc.) o bien pueden ser de tipo TEXT. Nosotros nos vamos a ocupar únicamente de este último tipo de archivos, los que contienen datos de tipo TEXT, que son ficheros ASCII. Los ficheros de tipo TEXT se leen línea a línea empleando READLINE, que guarda la línea que corresponda en una señal o variable de tipo LINE. Los fichero de tipo TEXT se escriben línea a línea empleando WRITELINE. Para extraer los datos contenidos en una señal/variable de tipo LINE se emplea READ, y la operación opuesta (escritura en un tipo LINE) se hace a través de WRITE. Tanto READ como WRITE funcionan con bit, bit_vector, boolean, character, integer, real, string y time.

Matlab escribe y lee los datos en notación decimal, legible por nosotros. Por lo tanto tenemos que convertir los datos del fichero que leamos desde entero (`integer`) a binario (`signed`). Para ello vamos a emplear la siguiente función del paquete `IEEE.NUMERIC_STD`:

- `my_signed <= to_signed(int_number,8)` donde `int_number` es el `integer` que queremos convertir, 8 es el número de bits que queremos que tenga nuestro vector y `my_signed` es el `signed` al que se le asigna la conversión.

De la misma forma, para escribir en el fichero, tenemos que convertir nuestros datos, que son `signed` a `integer`. Para ello emplearemos la siguiente función:

- `my_int <= to_integer(signed_number)` donde `signed_number` es el `signed` que queremos convertir, y `my_int` es el `integer` al que se le asigna la conversión.

A la hora de hacer la conversión entre `signed` y `integer` y viceversa, el programa no es capaz de trabajar con la representación $< 1,7 >$ o con cualquier otra en punto fijo, siempre va a considerar la notación $< 8,0 >$, es decir, sin coma decimal. Esto implica que los datos del fichero estarán entre +127 y -128.

En el apéndice 1 puedes encontrar un ejemplo de testbench que lee de un fichero TEXT para que te hagas una idea de cómo funciona.

Tarea 2.11:

Escribe el código VHDL correspondiente a un testbench que lea las muestras de un fichero y escriba los resultados en otro.

☞

- Llama al fichero que contiene los datos de entrada "sample_in.dat"; llama al fichero de salida "sample_out.dat".
- Rellena el fichero de entrada con una secuencia que introduzca un único impulso. Cada línea del fichero se corresponde con un dato.

Tarea 2.12:

Utiliza el fichero "haha.wav" en Matlab para crear un fichero de entrada para tu testbench. En el apéndice 2 se detallan las secuencias que tienes que emplear en Matlab para:

☞

- Cargar un fichero de audio en Matlab y crear otro fichero con el formato necesario para tu testbench.
- Utilizar la función *filter* de Matlab para obtener la respuesta de un filtro FIR con precisión real.
- Cargar y escuchar la salida que ha producido tu testbench en Matlab.

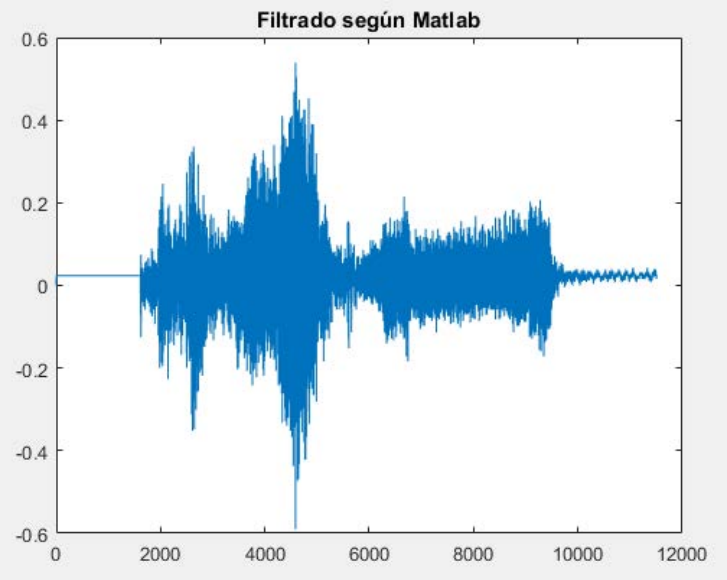
Tarea 2.13:

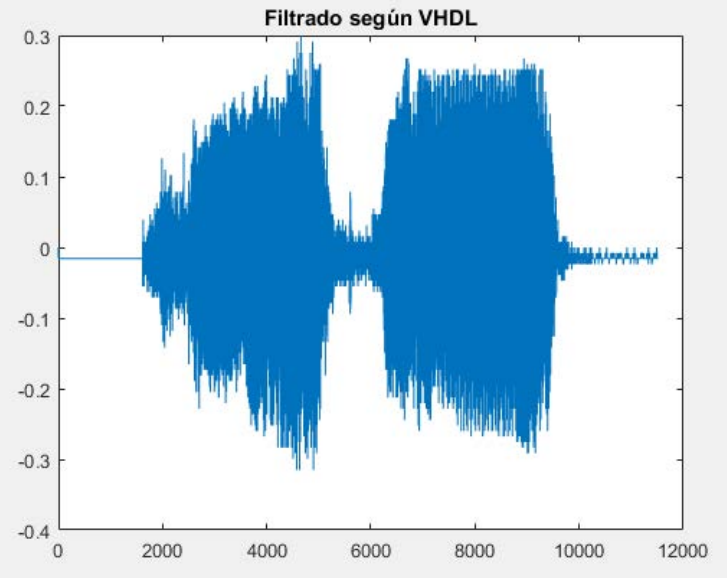
☞ Emplea tu testbench para procesar el fichero de entrada que te ha proporcionado Matlab. Escribe los datos filtrados en el fichero "sample_out.dat", que importarás posteriormente en Matlab.

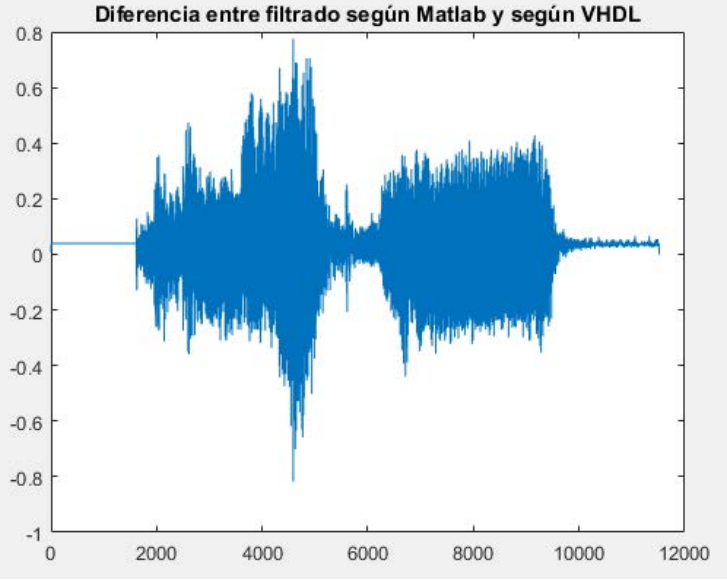
Tarea 2.14:

Importa tu fichero "sample_out.dat" en Matlab. Compara los resultados del testbench con los valores con precisión real proporcionados por la función *filter* de Matlab. Haz

☞ un gráfico del error de tus resultados (resta tus resultados de los datos con precisión real).
Utiliza la función *sound* de Matlab para escuchar la forma de onda original. Compárala después con tu sonido filtrado y observa si hay alguna diferencia entre el filtro con precisión real y tu filtro.


Filtrado según Matlab
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
2000
4000
6000
8000
10000
12000


Filtrado según VHDL
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
2000
4000
6000
8000
10000
12000


Diferencia entre filtrado según Matlab y según VHDL
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2000
4000
6000
8000
10000
12000

7. Bloque 3: Controlador y Memoria

En este tercer bloque se va a terminar de dar forma al sistema de audio. En concreto, se va a añadir al diseño un core IP que va a encapsular una memoria RAM donde almacenar las muestras de audio, se va a diseñar un controlador que orqueste todas las operaciones del sistema y se va a integrar todos los elementos.

El diseño del controlador está completamente abierto y se proporciona al estudiante la posibilidad de practicar las habilidades adquiridas a lo largo del curso, diseñando el sistema desde cero teniendo en cuenta el conjunto de especificaciones que lo definen.

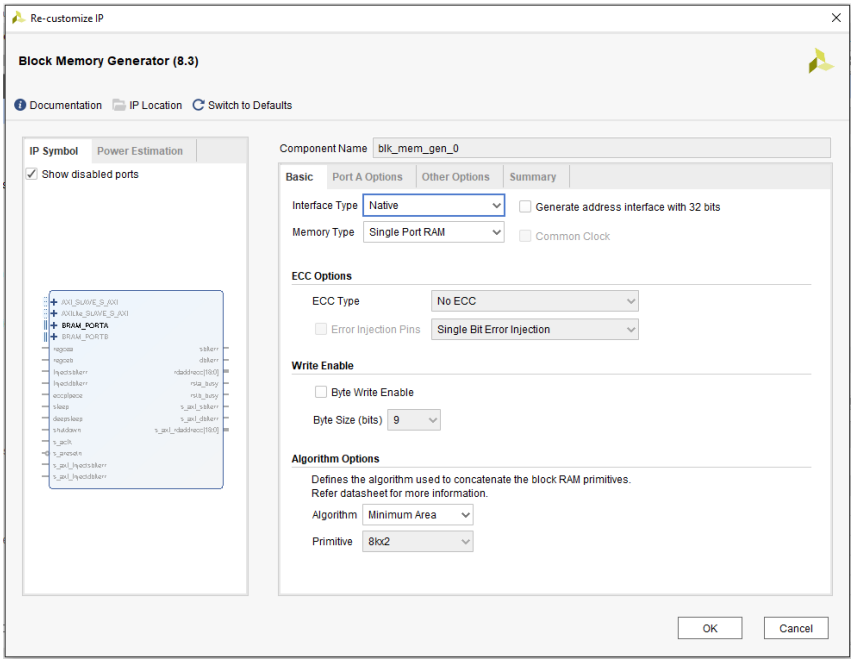
7.1. Memoria RAM

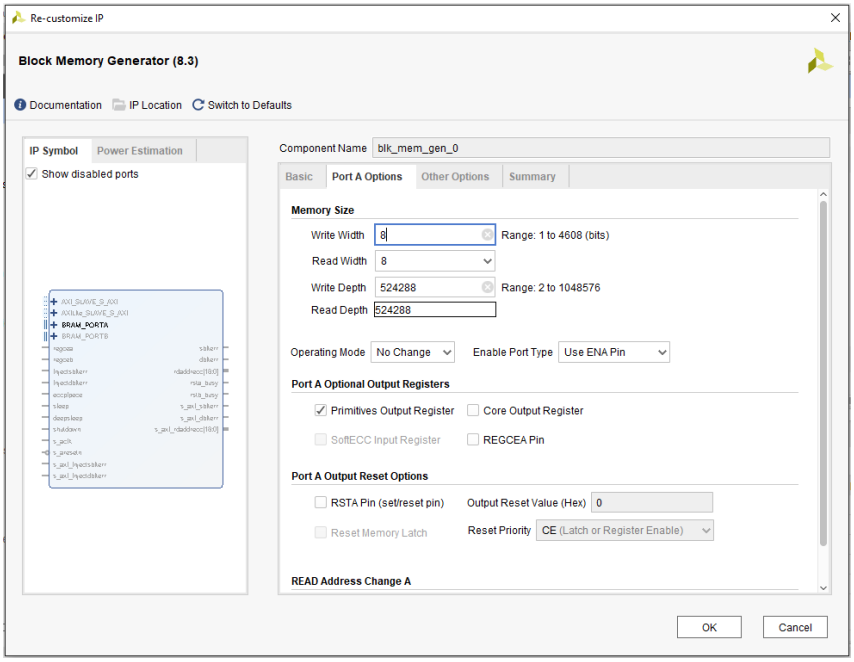
Las muestras grabadas se van a almacenar en una memoria RAM. Las características de la memoria son las siguientes:

- Tamaño de cada palabra almacenada: 8 bits.
- Tamaño del bus de direcciones: 19 bits.
- Número de palabras almacenadas. $2^{19} = 524288$.
- Tamaño en bits de la memoria: $2^{19} * 8 = 4194304$.

Para instanciar esta memoria en nuestro diseño se va a emplear el "IP Catalog" del Vivado, donde hay que buscar la siguiente opción: "Memories and Storage Elements", "RAMs and ROMs and BRAM", Block Memory Generator.

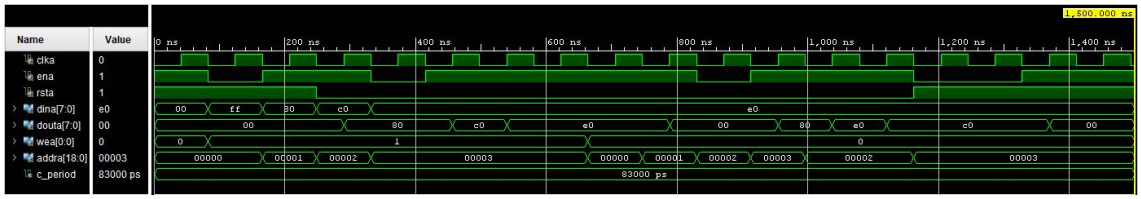
La siguientes dos imágenes muestran la manera de configurar el asistente para encapsular la memoria que se requiere.

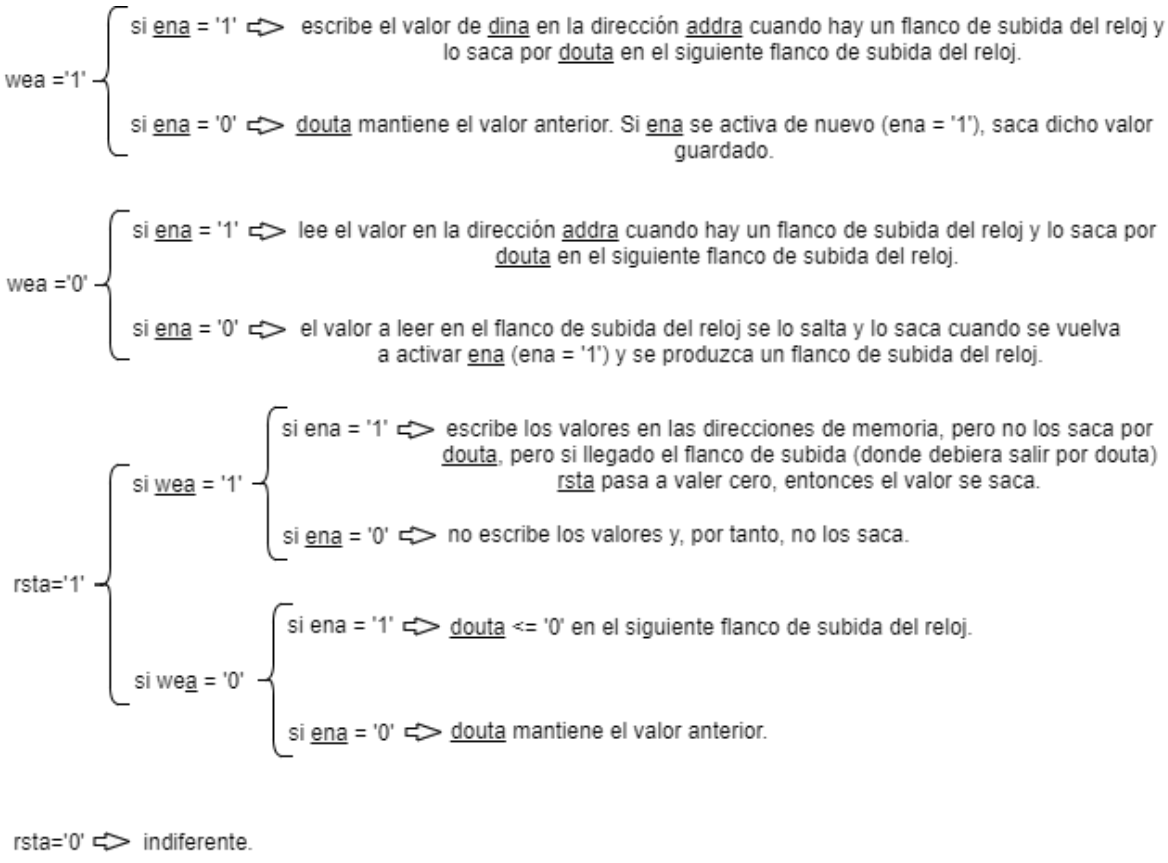




Tarea 3.1:

☞ Crea un testbench que trabaje a la frecuencia del sistema para comprobar y entender el funcionamiento de la memoria encapsulada. Anota a continuación la función de cada puerto y la temporización de la memoria (un pequeño cronograma que incluya una escritura y una lectura).





☞ **Tarea 3.2:**
Determina cuánto tiempo de grabación va a poder estar almacenado en la memoria.

Se produce un Sample_out_ready del micrófono (interfaz de audio) cada 150 ciclos de 3MHz.
Respecto al tiempo que conlleva esta operación:

$$150 * (1/3*10^{6}) = 5*10^{-5} \text{ segundos}$$

Así, si la RAM tiene una capicidad de almacenaje de 524288 palabras:

$$524288*(5*10^{-5}) = 26.2144 \text{ segundos}$$

Tiempo de grabación máximo en la memoria = 26.2144 segundos

7.2. Conversión entre codificaciones

En el Bloque 1 se obtienen datos de la interfaz del micrófono que utilizan una codificación binaria de 8 bits. Esta misma codificación se emplea en la interfaz de salida de audio. En el bloque 2, sin embargo, el filtro FIR necesita utilizar una representación en complemento a dos para poder usar coeficientes tanto negativos como positivos. Además, los datos están normalizados, de tal manera que el máximo valor con el que trabaja es 1, con lo que emplea un esquema de cuantificación $< 1{,}7 >$ en complemento a dos.

Sin embargo, estos dos bloques van a estar trabajando con las mismas muestras (las obtenidas por el micrófono) y, por lo tanto, tenemos que buscar un mecanismo para convertir entre una codificación y otra.

La primera transformación que necesitamos hacer es convertir muestras en notación binaria a muestras en notación en complemento a dos y viceversa. La representación binaria tiene un rango de $[0 \rightarrow 2^8 - 1]$ mientras que la representación en complemento a dos tiene un rango de $[-2^7 \rightarrow 2^7 - 1]$, la correspondencia de las muestras de una representación a otra es directa, es decir al elemento menor de una representación le corresponde el elemento menor de la otra (00000000_b se corresponde con 10000000_{2c}), el siguiente con el siguiente y así sucesivamente hasta el mayor (11111111_2 se corresponde con 01111111_{2c}).

Tarea 3.3:

☞ Determina qué operación es necesaria para hacer la transformación de las muestras de binaria a complemento a dos y de complemento a dos a binaria.

Es necesario realizar dos operaciones:

- Una que nos tranforme de muestras binarias a muestras en complemento a dos.
- Una que nos transforme de muestras en complemento a dos a muestras binarias.

Ya que cada elemento de una muestra binaria tiene su valor correspondiente directo sobre una muestra en complemento a dos y viceversa, realizando una serie de ejemplos hayamos la operación necesaria para dicha transformación:

	binario (b)		complemento a dos (Ca2)
Respecto a los valores máximos:	1111 1111	⟷	0111 1111
	1111 1110	⟷	0111 1110
	1111 1110	⟷	0111 1101
Respecto a los valores mínimos:	0000 0000	⟷	1000 0000
	0000 0001	⟷	1000 0001

```
b   ⟶ Ca2  ⟹  Ca2 <= (not b(7)) & b(6 downto 0);
Ca2 ⟶ b    ⟹  b <= (not Ca2(7)) & Ca2(6 downto 0);
```

La siguiente transformación consiste en pasar de un esquema $< 8{,}0 >$ a un esquema $< 1{,}7 >$. Pero, ¿es realmente necesario realizar algún cambio en nuestros datos? El lugar donde colocamos el punto decimal es una abstracción que nos permite calcular el valor en base 10 equivalente y con él ajustar el valor de los coeficientes del filtro, pero el hardware se va a ser idéntico sin importar el lugar del punto decimal. Por lo tanto, esta segunda transformación no es necesaria, la primera sí.

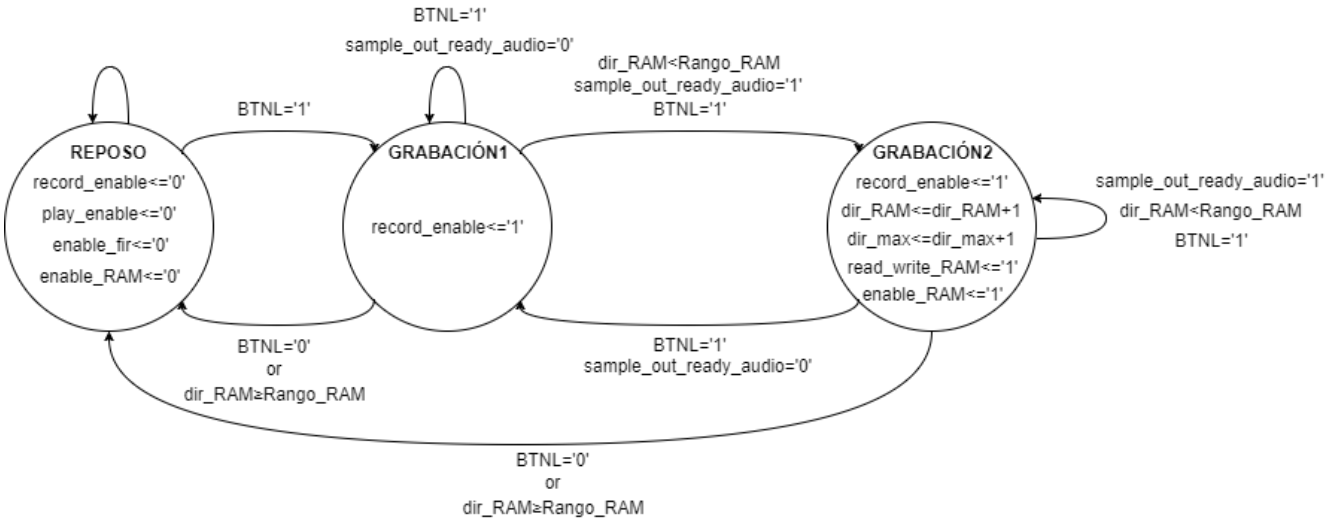
7.3. Controlador y sistema global

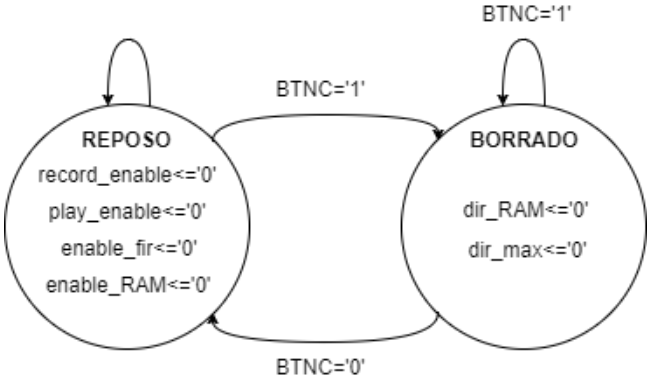
El controlador es el único módulo del proyecto donde el proceso de diseño no está guiado y es la última pieza del puzzle para construir el sistema global. A partir de las especificaciones dadas en la sección 2 y conociendo las funcionalidades y las interfaces de los distintos bloques, decide cómo vas a implementar la funcionalidad del controlador. En concreto, decide que estilo de diseño vas a utilizar (máquina de estados, máquina de estados con ruta de datos asociada, un sistema con distintos elementos secuenciales, etc.), divide el diseño de tal manera que lo puedas ir implementando de manera progresiva añadiendo funciones nuevas cuando estén aseguradas las anteriores, planifica cómo vas a comprobar el funcionamiento de cada especificación y haz una planificación temporal.

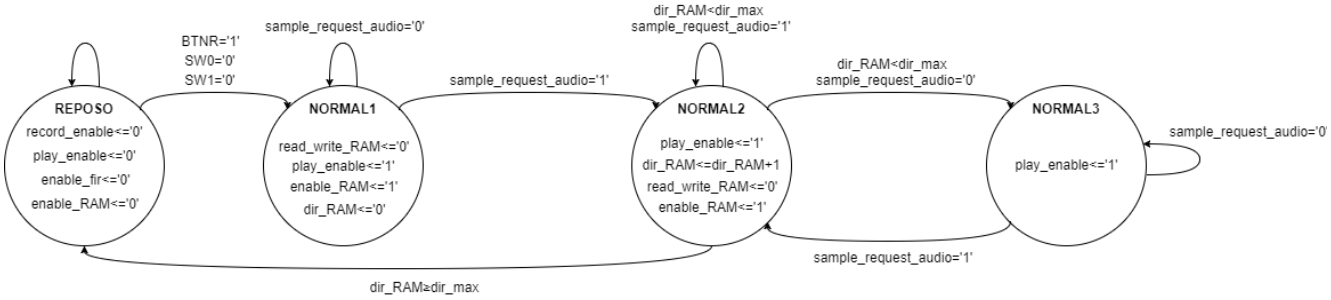
☞

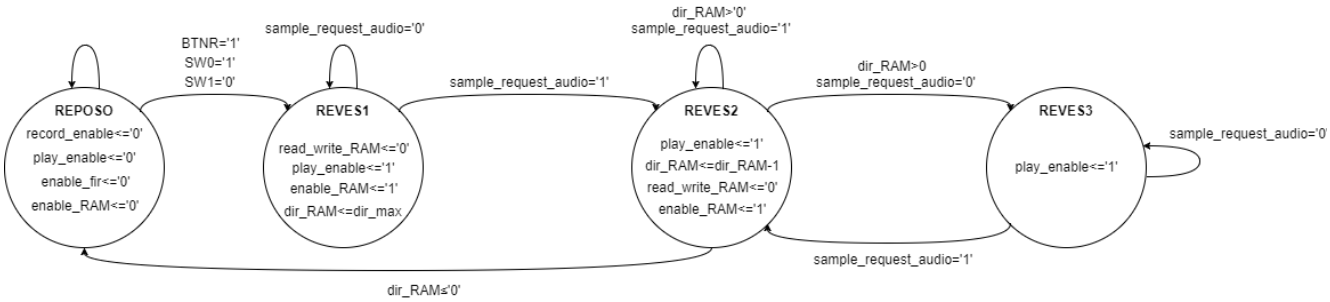
Tarea 3.4:

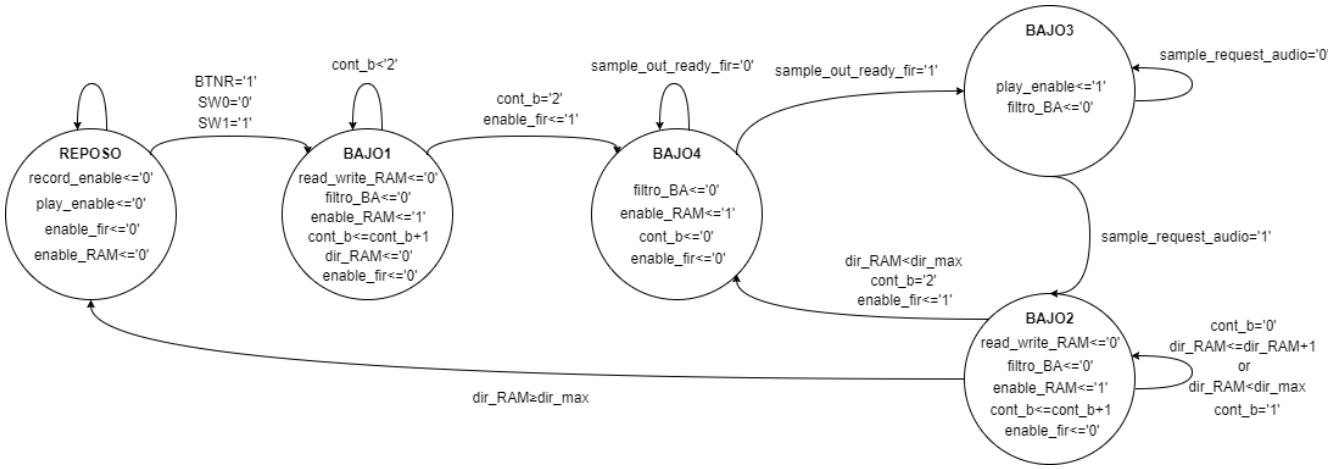
Añade a continuación todas las hojas necesarias para describir tu diseño y la planificación para llevarlo a cabo. Puedes incluir diagramas esquemáticos, cronogramas explicativos, un plan de pruebas, una planificación temporal y todo lo que consideres necesario para explicar las decisiones que has tomado.
Avisa al profesor antes de comenzar con el diseño para obtener el visto bueno.

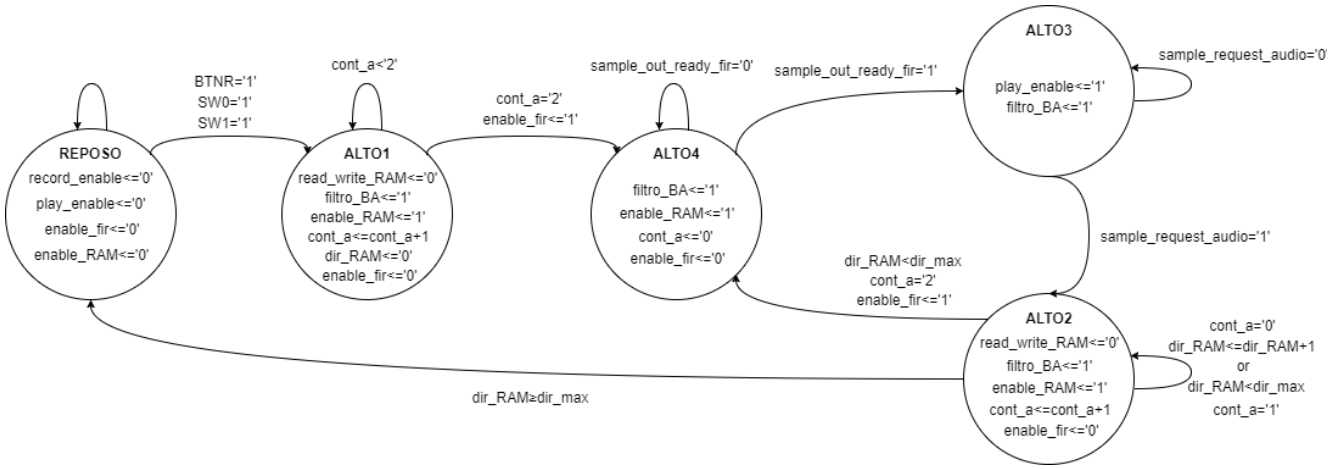



BTNC='1'
BTNC='1'
REPOSO
record_enable<='0'
play_enable<='0'
enable_fir<='0'
enable_RAM<='0'
BORRADO
dir_RAM<='0'
dir_max<='0'
BTNC='0'


BTNR='1'
SW0='0'
SW1='0'
sample_request_audio='0'
sample_request_audio='1'
dir_RAM<dir_max
sample_request_audio='1'
dir_RAM<dir_max
sample_request_audio='0'
sample_request_audio='0'
REPOSO
record_enable<='0'
play_enable<='0'
enable_fir<='0'
enable_RAM<='0'
NORMAL1
read_write_RAM<='0'
play_enable<='1'
enable_RAM<='1'
dir_RAM<='0'
NORMAL2
play_enable<='1'
dir_RAM<=dir_RAM+1
read_write_RAM<='0'
enable_RAM<='1'
NORMAL3
play_enable<='1'
sample_request_audio='1'
dir_RAM≥dir_max


BTNR='1'
SW0='1'
SW1='0'
sample_request_audio='0'
sample_request_audio='1'
dir_RAM>'0'
sample_request_audio='1'
dir_RAM>0
sample_request_audio='0'
sample_request_audio='0'
REPOSO
record_enable<='0'
play_enable<='0'
enable_fir<='0'
enable_RAM<='0'
REVES1
read_write_RAM<='0'
play_enable<='1'
enable_RAM<='1'
dir_RAM<=dir_max
REVES2
play_enable<='1'
dir_RAM<=dir_RAM-1
read_write_RAM<='0'
enable_RAM<='1'
REVES3
play_enable<='1'
sample_request_audio='1'
dir_RAM≤'0'





☞ **Tarea 3.5:**

Avisa al profesor cuando tengas todas las especificaciones mínimas del sistema global cubiertas.

☞ **Tarea 3.6:**

Sube al Moodle un fichero .vhd con todas las fuentes de tu proyecto.

9. Apéndice 1

Ejemplo de testbench que lee de un fichero de texto. Recuerda meter la ruta completa del fichero que quieras leer o escribir.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
use std.textio.all;

ENTITY ejemplo_tb IS
END ejemplo_tb;

ARCHITECTURE behavior OF ejemplo_tb IS
   -- Clock signal declaration
   signal clk : std_logic := '1';

   -- Declaration of the reading signal
   signal Sample_In : signed(7 downto 0) := (others => '0');

   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

-- Clock statement
   clk <= not clk after clk_period/2;

read_process : PROCESS (clk)
  FILE in_file : text OPEN read_mode IS "sample_in.dat";
  VARIABLE in_line : line;
  VARIABLE in_int : integer;
  VARIABLE in_read_ok : BOOLEAN;
BEGIN
if (clk'event and clk = '1') then
  if NOT endfile(in_file) then
    ReadLine(in_file,in_line);
    Read(in_line, in_int, in_read_ok);
    sample_in <= to_signed(in_int, 8); -- 8 = the bit width
  else
    assert false report "Simulation Finished" severity failure;
  end if;
end if;
end process;

END;
```

10. Apéndice 2

Comandos Matlab que se emplean en el bloque 2.

Carga de un fichero .wav en Matlab y creación de otro fichero con el formato que emplea el testbench.

```
[data, fs] = audioread('haha.wav');
file = fopen('sample_in.dat','w');
fprintf(file, '%d\n', round(data.*127));
```

Utilización de la función *filter* de Matlab para obtener la respuesta de un filtro FIR con precisión real.

```
test = filter([0.5, -0.5, 1.0, -0.5, 0.5],[1, 0, 0, 0, 0], data);
sound(test);
```

Recuerda modificar este comando con los coeficientes de los filtros que vayas a utilizar.

Carga y escucha de un fichero con el formato de salida del testbench. Atención, es importante no olvidar la división por 127 para no destruir los altavoces.

```
vhdlout=load('sample_out.dat')/127;
sound(vhdlout);
```