# Diseño de Sistemas Electrónicos Digitales

**Sistema de grabacio´n, tratamiento y reproducci´on de audio**

Versi´on 4.2

Pablo Ituero y Fernando Garc´ıa Redondo

Nombre y apellidos de los miembros del grupo:

Carlos Gómez Rodríguez

Alejandro Ramos



*Departamento de Ingenier´ıa Electr´onica*

Disen˜o de Sistemas Electr´onicos Digitales 2019

*Sistema de grabaci´on, tratamiento y reproducci´on de audio*

El bloque de filtrado est´a 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 Ingenier´ıa Electr´onica*

UNIVERSIDAD POLITE´CNICA DE MADRID

Espan˜a

Tel´efono: +34915495700 ext. 4207 E-mail: [pitu](mailto:pituero@die.upm.es)[ero@die.upm.es](mailto:ero@die.upm.es)

# ´Indice

1. [Introducci´on](#_bookmark0) 1
2. [Especiftcaciones m´ınimas](#_bookmark1) 1
   1. [Descripci´on de la estructura del sistema](#_bookmark2) 3
   2. [Uso de package](#_bookmark3) 3
3. [Planiftcaci´on temporal](#_bookmark4) 4
4. [Evaluaci´on](#_bookmark5) 4
5. [Bloque 1: Interfaz de audio](#_bookmark6) 5
   1. [Generador de enables y salida de reloj del micr´ofono](#_bookmark7) 6
   2. [Interfaz del micr´ofono](#_bookmark8) 8
      1. [Esquema de muestreo](#_bookmark9) 8
      2. [Disen˜o e implementaci´on](#_bookmark10) 9
      3. [Estrategia de test](#_bookmark11) 12
   3. [Interfaz de la salida de audio](#_bookmark12) 12
      1. [Estrategia de test](#_bookmark13) 14
   4. [Integraci´on de la interfaz de audio](#_bookmark14) 14
      1. [Estrategia de test](#_bookmark15) 15
   5. [Implementaci´on f´ısica y test en placa](#_bookmark16) 15
6. [Bloque 2: Filtro FIR](#_bookmark17) 17
   1. [Filtros digitales](#_bookmark18) 17
   2. [Implementaci´on hardware de filtros FIR](#_bookmark19) 18
   3. [Sistemas basados en rutas de datos](#_bookmark20) 19
   4. [Metodolog´ıa de implementaci´on de algoritmos de procesado de sen˜al](#_bookmark21) 20
      1. [Asignaci´on](#_bookmark22) 20
      2. [Planificaci´on temporal](#_bookmark23) 21
      3. [Vinculaci´on](#_bookmark24) 21
      4. [Implementaci´on](#_bookmark25) 22
   5. [Notaci´on en punto fijo](#_bookmark26) 24
   6. [Especificaci´on del bloque](#_bookmark27) 26
   7. [Tareas pr´acticas](#_bookmark28) 27
      1. [Creaci´on de la ruta de datos](#_bookmark29) 27
      2. [Creaci´on del controlador](#_bookmark30) 32
      3. [Creaci´on de un testbench avanzado](#_bookmark31) 34
7. [Bloque 3: Controlador y Memoria](#_bookmark32) 36
   1. [Memoria RAM](#_bookmark33) 36
   2. [Conversi´on entre codificaciones](#_bookmark34) 38
   3. [Controlador y sistema global](#_bookmark35) 39
8. [Mejoras opcionales](#_bookmark36) 40
   1. [Control de volumen logar´ıtmico](#_bookmark37) 40
   2. [Eco y reverberaci´on configurables](#_bookmark38) 40
   3. [Informaci´on de segundos restantes](#_bookmark39) 41
   4. [Looper](#_bookmark40) 41
9. [Ap´endice 1](#_bookmark41) 42
10. [Ap´endice 2](#_bookmark42) 43

# Introduccio´n

Este documento describe el proyecto final de la asignatura Disen˜o de Sistemas Electr´onicos Digitales del plan de estudios GITST de la ETSI de Telecomunicaci´on de la UPM.

El proyecto gu´ıa el disen˜o y la implementaci´on de un sistema electr´onico digital de comple- jidad media y est´a 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 meto- dolog´ıas de disen˜o combinacional y secuencial, en el disen˜o de m´aquinas de estados finitos y de m´aquinas de estados finitos con ruta de datos asociadas, la metodolog´ıa de s´ıntesis de alto nivel y en cuestiones de temporizaci´on.

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 pr´acticas de laboratorio, adquiere, almacena, procesa y reproduce sonidos.

# Especificaciones m´ınimas

El sistema tiene las siguientes especificaciones globales m´ınimas:

El sistema se implementa sobre la placa Nexys 4DDR utilizando el flujo de trabajo de Vivado.

El sistema recibe la informaci´on de audio del micr´ofono integrado en la placa.

El sistema reproduce el audio a trav´es 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 bot´on 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 declaraci´on:

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 vac´ıa de audio.

Cuando se presiona BTNL y durante el tiempo que ´este est´e presionado el sistema graba el audio.

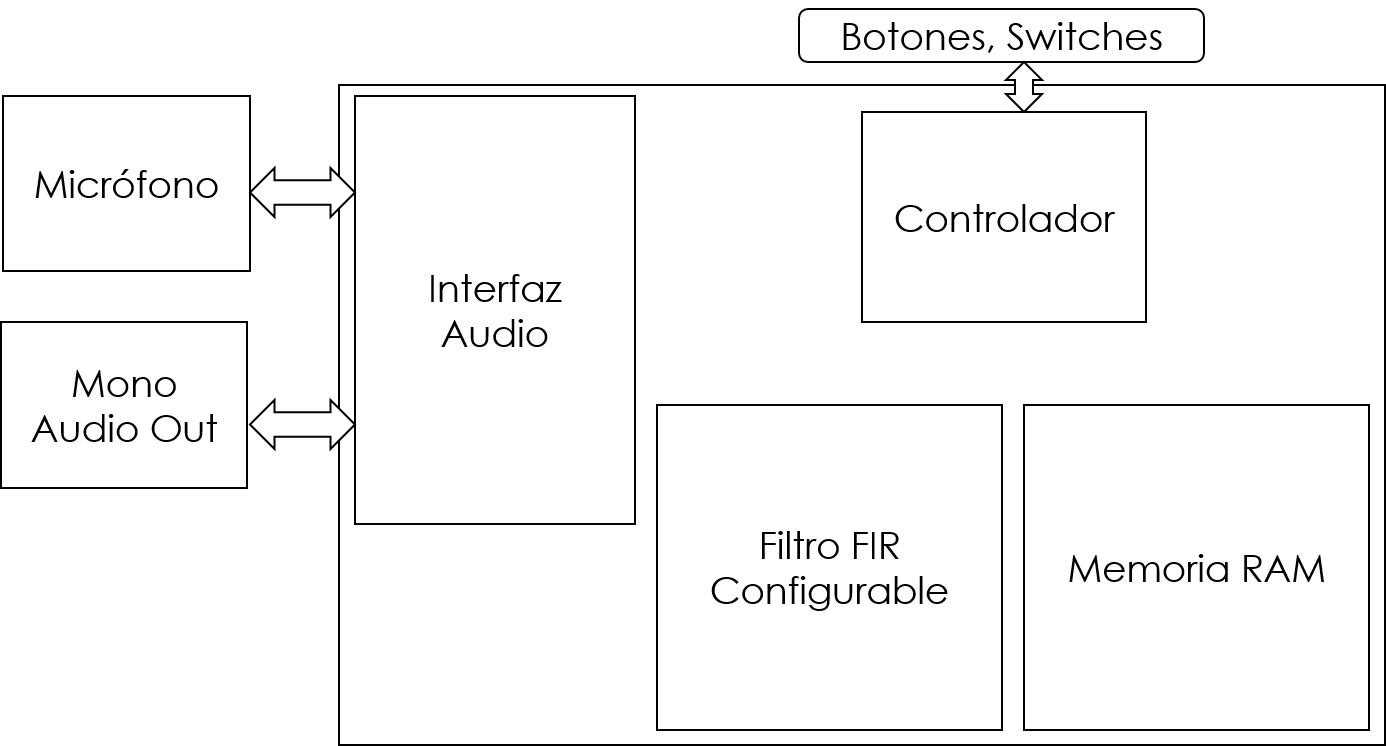
Cada vez que se vuelve a presionar BTNL el nuevo audio se an˜ade al final del audio que estaba previamente grabado.

Cuando se presiona BTNC se borra todo el audio grabado.

Cuando se presiona BTNR el sistema debe realizar acciones en funci´on de la informaci´on de los switches:

* + Si ambos switches est´an a ’0’ el sistema reproducir´a todo el audio grabado.
  + Si SW0 est´a a ’1’ y SW1 est´a a ’0’ el sistema reproducir´a el audio almacenado al rev´es.
  + Si SW0 est´a a ’0’ y SW1 est´a a ’1’ el sistema reproducir´a el audio filtrado por un filtro paso bajo.
  + Si SW0 est´a a ’1’ y SW1 est´a a ’1’ el sistema reproducir´a el audio filtrado por un filtro paso alto.

## Descripci´on de la estructura del sistema



El sistema est´a compuesto por cuatro bloques principales:

La interfaz de audio tiene dos funciones principales: En primer lugar es la encargada de la digitalizaci´on de las sen˜ales PDM provenientes del micr´ofono. Por otro lado es el bloque encargado de interactuar con la salida de audio proporcion´andole una sen˜al PWM. Este bloque se va a implementar siguiendo una metodolog´ıa de m´aquina de estados finitos con ruta de datos asociada.

El filtro FIR se encarga del procesado digital de la sen˜al de audio para las funcionalida- des de filtro paso alto y filtro paso bajo. Este bloque se va a implementar siguiendo la metodolog´ıa 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 ´ordenes del usuario del sistema. En este bloque el estilo de diseño es libre.

## 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 comu´n, de tal manera que el código se haga m´as legible, evitando el uso de expresiones numéricas que no se sabe de donde 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 l´ıneas a esta declaración en funci´on 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;

# 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.

# Evaluación

La evaluación del proyecto tendr´a 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.

# 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 muestreo 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 funcionar´á.

sample out: Dato de 8 bits (número positivo sin signo) 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 duraci´on cada vez que se requiere un nuevo dato en sample in.

jack sd: Informaci´on de control para los operacionales de la etapa de audio mono. Dejare- mos este valor estable en ’1’.

jack pwm: La sen˜al PWM generada a partir del dato en sample in.

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

Interfaz del micr´ofono. 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 micr´ofono. Encargado de generar las sen˜ales enable que controlan las frecuencias de muestreo y de conversi´on de datos, as´ı como la sen˜al de reloj que alimenta al micr´ofono.

A continuaci´on se describe cada uno de estos m´odulos y se gu´ıa el proceso de disen˜o.

## Generador de enables y salida de reloj del micr´ofono

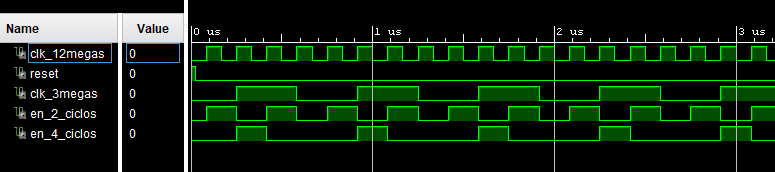
Este bloque recibe la sen˜al del reloj global de 12 MHz y proporciona a sus salidas sen˜ales que van a ser utilizadas para temporizar el resto de m´odulos de la interfaz de audio. La entidad del m´odulo se define de la siguiente manera:

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 micr´ofono 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 sen˜al 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 micr´ofono y proporciona una sen˜al activa durante un ciclo de reloj cada cuatro ciclos. La siguiente figura muestra el funcionamiento esperado.



R

**Tarea 1.1:**

Disen˜a un circuito que sea capaz de generar las sen˜ales especificadas. Dibuja el es- quem´atico correspondiente a tu disen˜o en el espacio inferior.

R

**Tarea 1.2:**

Implementa tu disen˜o en VHDL.

R

**Tarea 1.3:**

Genera un testbench que verifique la funcionalidad de tu disen˜o.

## Interfaz del micr´ofono

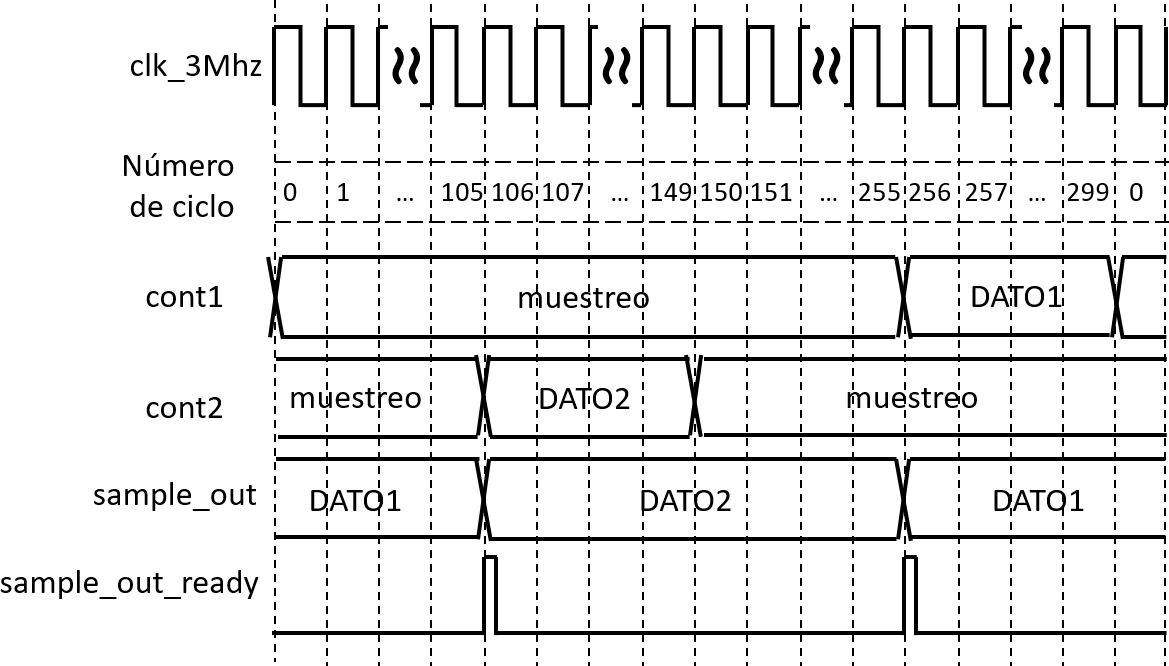
Para disen˜ar este m´odulo, es necesario conocer bien la interfaz que proporciona el micr´ofono omnidireccional de la placa Nexys4 DDR, la modulaci´on de densidad de pulso (PDM) y la temporizaci´on de la interfaz del micr´ofono. Para ello se recomienda la lectura de la secci´on 15 del manual “Nexys4 DDR FPGA Board Reference Manual”.

### 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 *µ*s (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 nu´mero de unos que hay a lo largo de 256 muestras.

La siguiente imagen muestra el esquema de muestreo del sistema:



Segu´n se muestra en la figura, hay un contador general de m´odulo 300 que numera los ciclos del 0 al 299. Los 300 ciclos se corresponden con 100 *µ*s, es decir el doble del periodo de muestreo. El primer contador muestrea entre los ciclos 0 y 255 y proporciona la sen˜al digitalizada estable entre los ciclos 256 y 299. El segundo contador muestrea entre los ciclos 150 y 105 y proporciona

la sen˜al 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 *µ*s, 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 est´an disponibles.

### Disen˜o e implementaci´on

El disen˜o de la interfaz del micr´ofono se va a realizar empleando la metodolog´ıa de FSMD: a partir de un pseudoc´odio se va a generar un gr´afico ASMD, este gr´afico servir´a como base para la descripci´on del m´odulo VHDL. Se recomienda revisar los cap´ıtulos 11 y 12 del libro “RTL Hardware Design Using VHDL” del profesor Pong P. Chu y las diapositivas de clase para familiarizarse con esta metodolog´ıa. La entidad del m´odulo 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 sen˜al enable 4 cycles es una sen˜al 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 micr´ofono 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 digitalizaci´on mediante el siguiente pseudo-c´odigo:

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-c´odigo, 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-c´odigo anterior.

R

R

**Tarea 1.5:**

Implementa el diagrama anterior en un fichero VHDL.

### Estrategia de test

Antes de poder o´ır si el resultado de nuestra digitalizaci´on es correcto, necesitamos comprobar mediante testbenches que nuestro sistema funciona como esperamos. Podemos en un primer lugar comprobar que la m´aquina de estados recorre los estados como esperamos, generando las sen˜ales necesarias para reiniciar los contadores. Puedes instanciar el generador de enables en tu testbench para tener disponible la sen˜al enable 4 cycles.

**Tarea 1.6:**

Realiza un testbench que tenga la sen˜al micro data fija a ’1’ para comprobar que las transiciones de estados, la selecci´on de los datos de salida y la activaci´on de sam- ple out ready se produce correctamente.

R

En una segunda fase podemos comprobar que la digitalizaci´on se hace correctamente, para ello podemos introducir sen˜ales pseudo-aleatorias en la sen˜al micro data y calcular el resultado esperado a mano. Podemos construir estas sen˜ales 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;

R

**Tarea 1.7:**

Realiza un testbench que introduzca una sen˜al pseudo-aleatoria en micro data y com- prueba que la digitalizaci´on se realiza correctamente.

## Interfaz de la salida de audio

Para disen˜ar este m´odulo, es necesario conocer bien la interfaz que proporciona la salida de audio mono de la placa Nexys4 DDR y la modulaci´on de anchura de pulso (PWM). Para ello se recomienda la lectura de la secci´on 16 del manual “Nexys4 DDR FPGA Board Reference Manual”.

Como los datos se han muestreado a 20 kmuestras/s, la tasa de generaci´on de pulsos tiene que ser la misma: un pulso cada 50 *µ*s. Para la generaci´on de la sen˜al PWM se propone basarse en la secci´on 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 m´as pequen˜o, 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 *µ*s, el contador va contar de 0 a 299. Como nuestro dato digital el de 8 bits, el m´aximo valor representado es 255, por lo tanto todos datos

que reproduzcamos van a tener una pequen˜a disminuci´on de volumen por un factor de 255/300

= 0.85.

La entidad del bloque est´a 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 sen˜al en 2 cycles es una sen˜al 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 m´odulo 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 duraci´on un s´olo periodo del reloj general para solicitar una nueva muestra a la entrada.

**Tarea 1.8:**

Modifica el disen˜o 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 duraci´on apropiada. Dibuja el esquem´atico de tu disen˜o en la parte inferior.

R

R

**Tarea 1.9:**

Implementa el esquem´atico anterior en un fichero VHDL.

### Estrategia de test

Antes de probar el m´odulo 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 sen˜ales 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 algu´n valor aleatorio inter- medio. Puedes instanciar el generador de enables en tu testbench para tener disponible la sen˜al enable 2 cycles.

**Tarea 1.10:**

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

R

## Integraci´on de la interfaz de audio

Una vez implementados los tres m´odulos anteriores, el siguiente paso del disen˜o consiste en la instanciaci´on de cada uno de estos bloques en un u´nico m´odulo que formar´a la interfaz de audio completa.

**Tarea 1.11:**

Dibuja el esquem´atico a nivel bloque de la interfaz de audio completa. Ten en cuenta que record enable y play enable especifican cu´ando est´an activas las interfaces del micr´ofono y de la salida de audio, respectivamente. Del mismo modo, tienes que asignar un ’1’ tanto a micro LR, como a jack sd.

R

R

**Tarea 1.12:**

Implementa en VHDL la interfaz de audio completa. Para ello utiliza un estilo de c´odigo estructural que instancie cada uno de los tres bloques desarrollados anteriormente.

### Estrategia de test

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

**Tarea 1.13:**

Disen˜a un testbench que verifique la funcionalidad de la interfaz de audio completa.

R

## Implementaci´on f´ısica y test en placa

Una vez verificada la funcionalidad del bloque completo mediante un testbench, vas a realizar la implementaci´on f´ısica del bloque en la placa. Para poder comprobar que el sistema efectiva- mente digitaliza bien la sen˜al del micr´ofono 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 micr´ofono salga por la salida de audio.

Para lograr esto, necesitas un pequen˜o controlador que genere la sen˜al de 12 MHz a partir del reloj de 100 MHz de la placa, mantenga record enable y play enable a ’1’ y que introduzca la palabra digitalizada por la interfaz del micr´ofono (sample out) en la interfaz de la salida de audio (sample in). No necesitan ningu´n tipo de sincronizaci´on especial, mientras que sample in est´e estable durante 50 *µ*s (no importa si cambia cuando el contador se pone a 0 o no). Este controlador tiene la siguiente declaraci´on 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.

R

R

**Tarea 1.15:**

Disen˜a un testbench que verifique la funcionalidad del controlador. Puedes utilizar los est´ımulos empleados en testbenches anteriores.

R

**Tarea 1.16:**

Escribe el fichero de restricciones .xdc. Sintetiza y realiza la implementaci´on f´ısica del disen˜o. 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 pequen˜as funcionalidades adicionales al controlador. Por ejemplo, puedes hacer un indicador de intensidad de sen˜al grabada conectando 8 LEDs a la palabra digitalizada. Tambi´en puedes hacer que el sistema s´olo funcione cuando se aprieta uno de los botones.

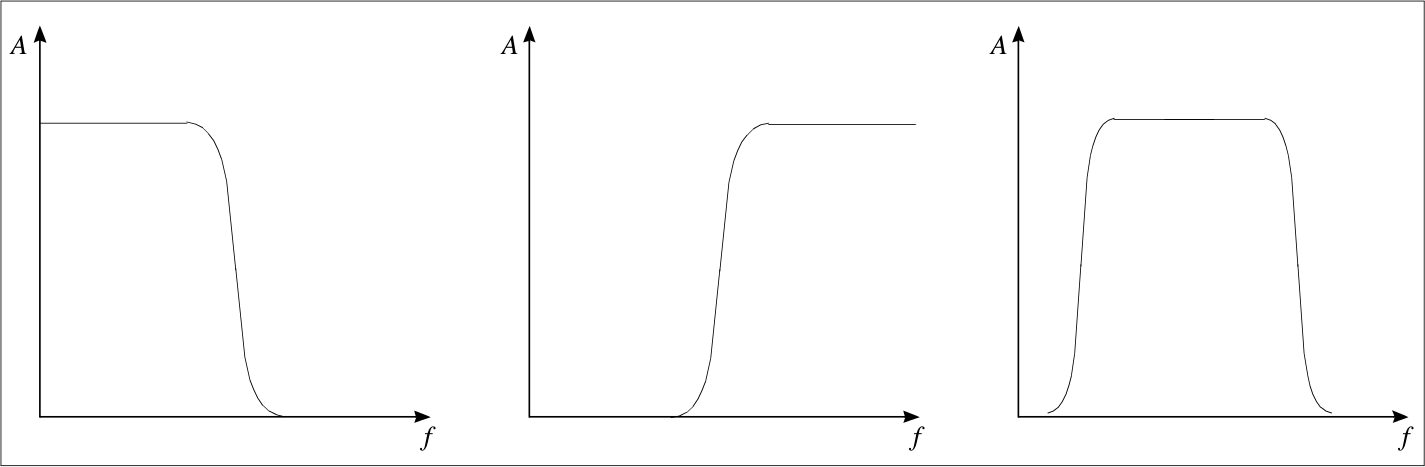
# Bloque 2: Filtro FIR

Este bloque implementa un filtro FIR empleando VHDL y una arquitectura basada en una ruta de datos controlada por una m´aquina de estados finitos. El disen˜o de este bloque profundiza en conceptos de cuantificaci´on, implementaci´on de algoritmos, arquitecturas segmentadas y test- benches de complejidad media.

*Cr´edito*: Este bloque est´a basado en el “Mini-project”“FIR-Filter Design” del curso IL2204 “DSP Design usign HDL” del “Royal Institute of Technology” de Estocolmo, Suecia.

## Filtros digitales

Hablamos de filtros en el contexto de procesado digital de sen˜ales para referirnos a circuitos que permiten pasar sen˜ales dentro de un determinado rango de frecuencias, mientras que blo- quean a las sen˜ales 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 rechazada. Se pueden clasificar los filtros con respecto a la posici´on de las bandas de paso y rechazada: Tenemos filtros paso bajo, paso alto y paso banda, sus funciones de transferencia se ilustran en la siguiente figura.



Una manera comu´n de expresar la relaci´on de la salida del filtro *y*[*n*], donde *n* es el nu´mero de muestra, en funci´on de la entrada *x*[*n*] de un filtro, es la siguiente:

*N−*1 *N−*1

*y*[*n*] =

Σ *α*[*m*]*x*[*n* − *m*] − Σ *β*[*m*]*y*[*n* − *m*] (1)

*m*=0

*m*=1

El primer t´ermino de la ecuaci´on relaciona la salida del filtro con entradas pasadas. El segundo t´ermino relaciona la salida del filtro con salidas pasadas. Podemos distinguir dos tipos de filtros:

Filtros de repuesta al impulso infinita (IIR) en los que est´an presentes los dos t´erminos de la ecuaci´on 1. Tambi´en son conocidos como filtros de respuesta infinita, ya que el segundo t´ermino describe un comportamiento recursivo (la salida es funci´on de la salida).

Filtros de respuesta al impulso finita (FIR) en los que s´olo est´a presente el primer t´ermino de la ecuaci´on 1. La sen˜al de salida de un filtro FIR es una suma ponderada de la sen˜al 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 *α*[*m*]. Esto nos proporciona una manera sencilla de comprobar el funcionamiento del filtro. Lo u´nico que tenemos que hacer es introducir una delta unidad (x(0)=1, x(n)=0 para n distinto de 0, tambi´en 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 caracter´ısticas.

La ganancia en continua del filtro equivale a la suma de todos sus coeficientes.

## Implementaci´on hardware de filtros FIR

La transformada Z de un filtro FIR viene dada por la siguiente expresi´on

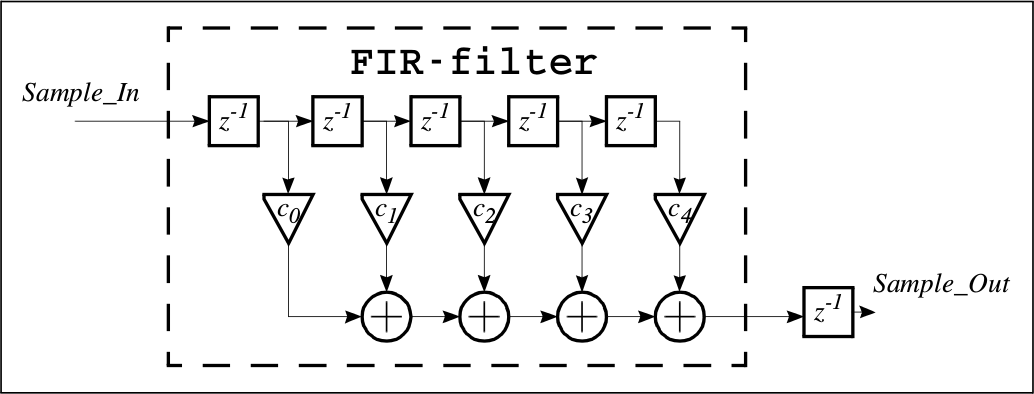
*N−*1

Σ

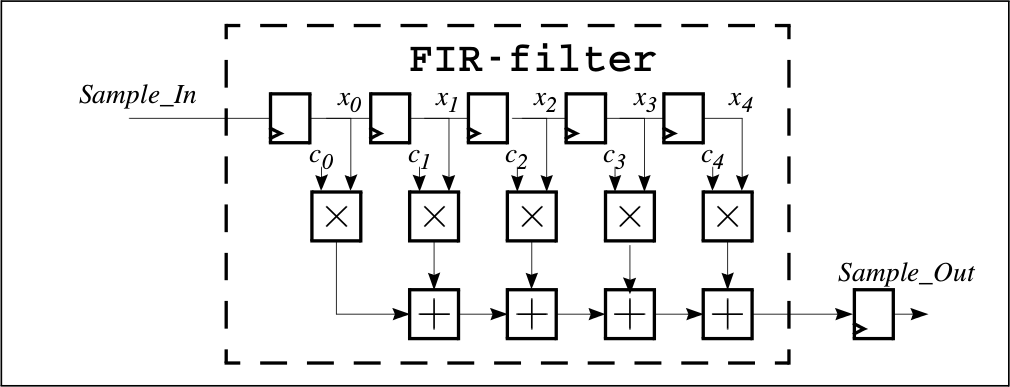
*HF IR*(*z*) = *cnz−n* (2)

*n*=0

Donde *z−n* en la ecuaci´on corresponde a l´ıneas de retardo y *cn* son los coeficientes que ponderan las muestras de entrada, equivalentes a las *α*[*m*] de la ecuaci´on 1. En la ecuaci´on anterior *N* es el orden del filtro, que tambi´en se corresponde como el nu´mero de etapas del filtro. El diagrama del flujo de sen˜al de un filtro FIR de 5 etapas se muestra en la siguiente figura. Observa que la u´ltima l´ınea de retardo (el registro a la salida) no forma parte de la ecuaci´on 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 trav´es de sumadores. La siguiente figura muestra la implementaci´on directa de un filtro FIR de 5 etapas.



## Sistemas basados en rutas de datos

La mayor´ıa de los sistemas digitales se basa en una ruta de datos y un controlador (nor- malmente en forma de una m´aquina 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 (l´ıneas de varias interconexiones). Los valores guardados en las unidades de almacenamiento se leen des- pu´es 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 informaci´on de control y recibe informaci´on de estado de la ruta de datos. Normalmente se implementa a trav´es de una m´aquina de estados finitos controlada por entradas y salidas del mundo exterior. La primera etapa en el disen˜o de un sistema basado en una ruta de datos es la descripci´on algor´ıtmica de la especificaci´on funcional de un sistema digital. A partir de esta descripci´on algor´ıtmica se construye la ruta de datos.

## Metodolog´ıa de implementaci´on de algoritmos de procesado de sen˜al

El proceso general de implementaci´on de algoritmos para procesado digital de sen˜ales se suele dividir en las siguientes etapas:

Asignaci´on

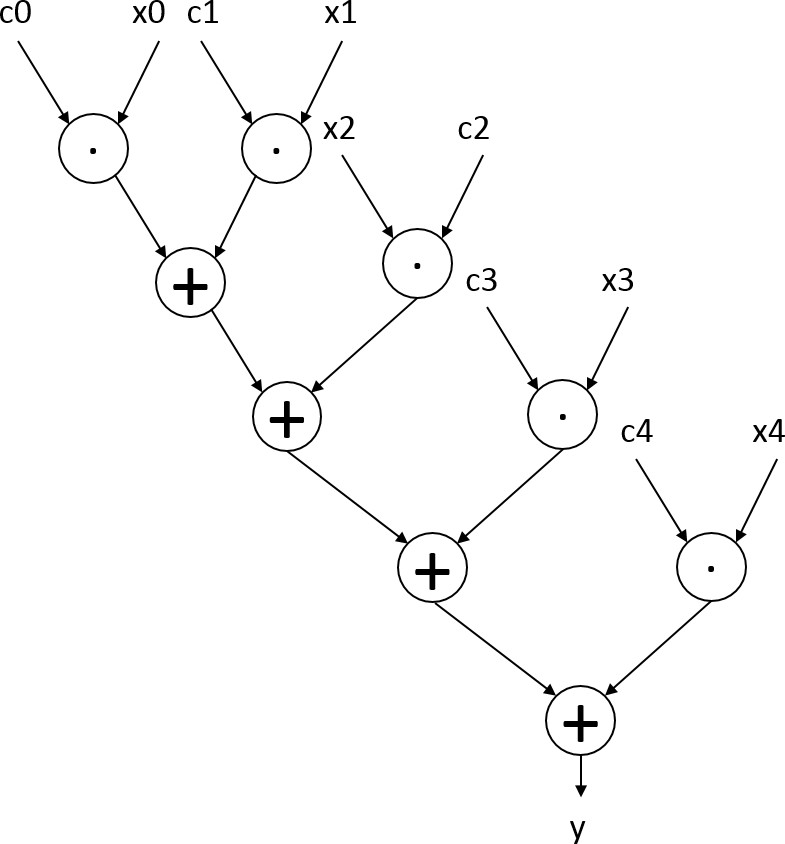
Planificaci´on temporal

Vinculaci´on

Implementaci´on

El proceso no es estrictamente lineal, sino que puede que tengamos que retroceder alguna etapa para realizar alguna optimizaci´on concreta, o puede que podamos tomar decisiones sobre etapas posteriores sin haber finalizado algu´n 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.



### Asignaci´on

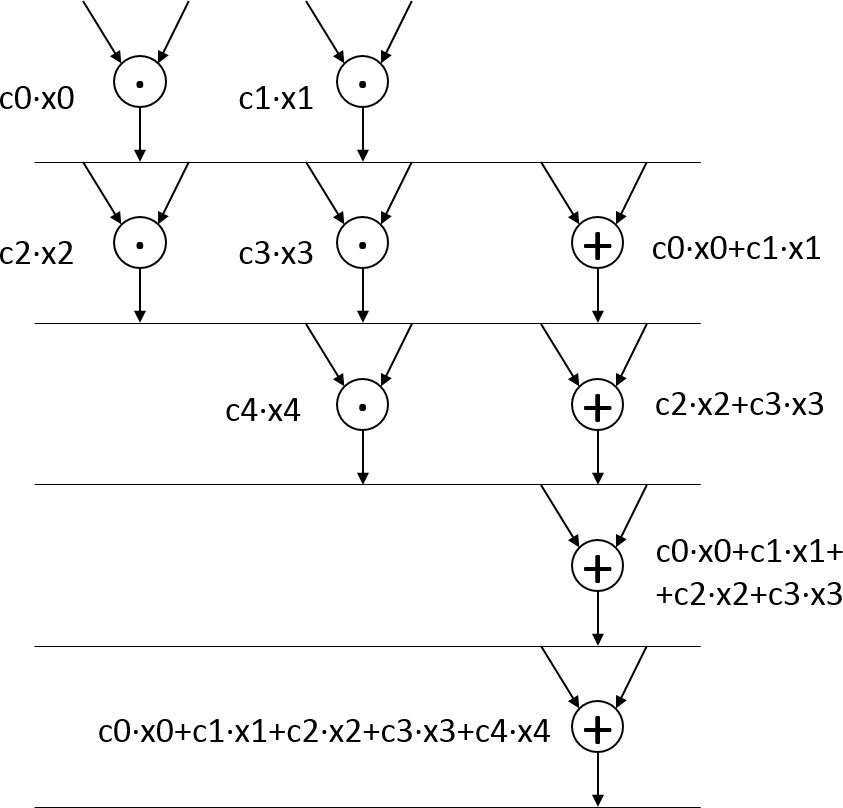
Designamos asignaci´on a la selecci´on del nu´mero y tipo de unidades hardware que van a implementar las funciones descritas por el diagrama de flujo del algoritmo. En el caso del filtro FIR de cinco etapas, podr´ıamos decidir implementarlo con un multiplicador y un sumador, dos multiplicadores y un sumador, dos medios multiplicadores y dos sumadores, etc. En funci´on del nu´mero y tipo de unidades que se escojan se obtendr´an unos valores distintos de ´area, throughput y latencia, con lo cual hay que tener en cuenta las restricciones del sistema para tomar esta decisi´on.

### Planiftcaci´on temporal

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

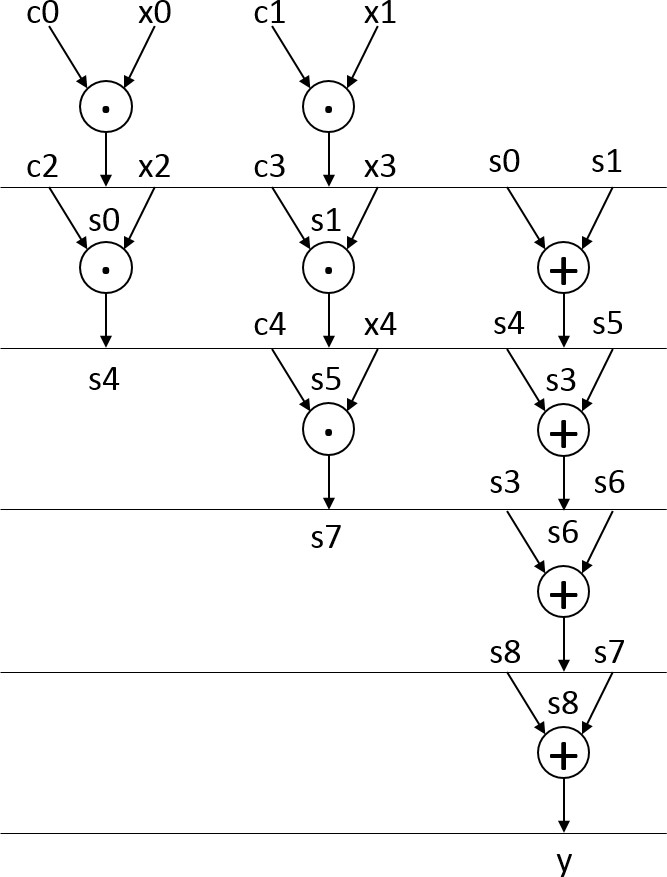
Fijando una asignaci´on de dos multiplicadores y un sumador, la siguiente figura muestra la planificaci´on temporal del diagrama de flujo del filtro FIR. Cada l´ınea horizontal separa

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 ((*c*0 · *x*0 + *c*1 · *x*1) + (*c*2 · *x*2) + (*c*3 · *x*3)), que desaprovechar´ıa la opci´on de utilizar dos multiplicadores simult´aneamente, se ha propuesto ((*c*0 · *x*0 + *c*1 · *x*1) + (*c*2 · *x*2 + *c*3 · *x*3)).



### Vinculaci´on

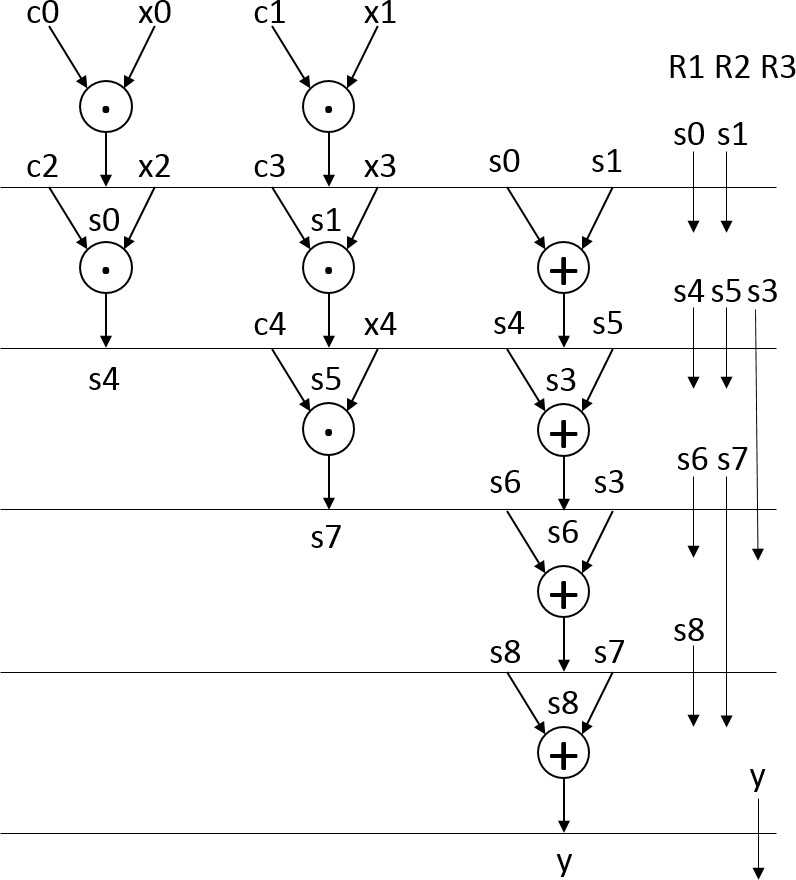
Cada unidad concreta se asocia a una operaci´on concreta en cada unidad de tiempo. Tambi´en se especifica a qu´e puerto concreto se conecta cada operando. Las sen˜ales internas se nombran para facilitar las procedimientos de las siguientes fases. La siguiente figura muestra el resultado de esta fase para el filtro FIR.



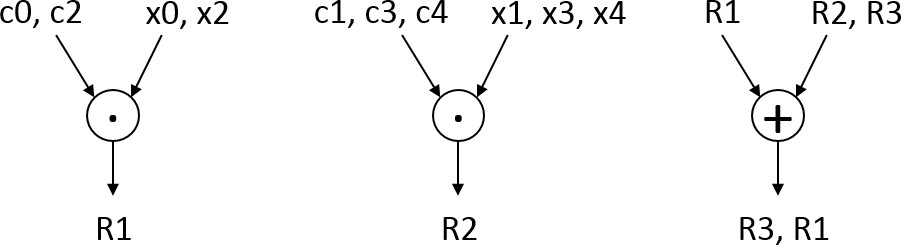
### Implementaci´on

Esta es la fase final del disen˜o 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: An´alisis del tiempo de vida de las variables, minimizaci´on del nu´mero de registros y multiplexores y creaci´on de la estructura hardware.

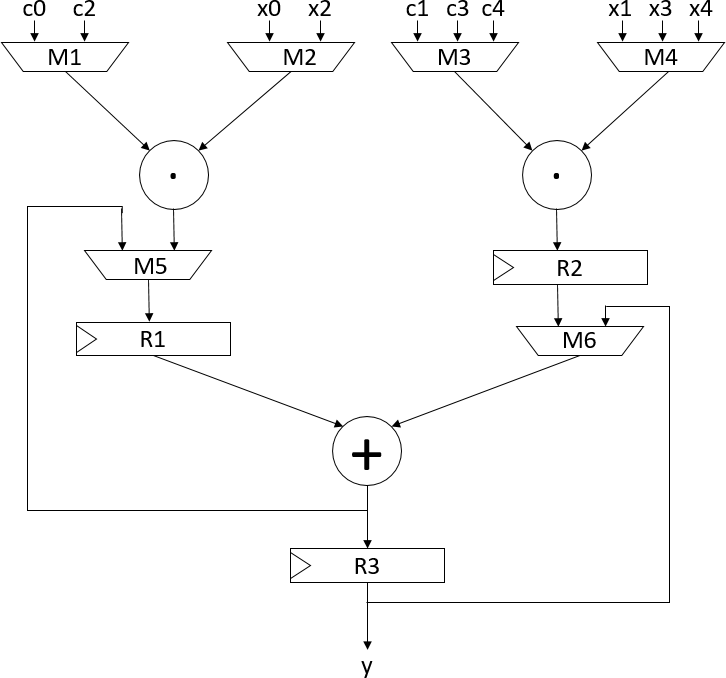
En el an´alisis del tiempo de vida de las variables se estudia la necesidad de almacenamiento interno (registros) de la estructura que vamos a generar. A partir del diagrama generado despu´es de la vinculaci´on, podemos analizar los momentos en que se producen las sen˜ales internas y los momentos en que se consumen, obteniendo as´ı su tiempo de vida. En la siguiente figura se ha representado mediante una flecha el tiempo de vida de las distintas variables. El m´aximo nu´mero de flechas paralelas, tres en el ejemplo, establece el nu´mero de registros necesarios. Una vez establecido el nu´mero 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 ir´an guardando s0, s4, s6 y s8 segu´n se desenvuelve el algoritmo.



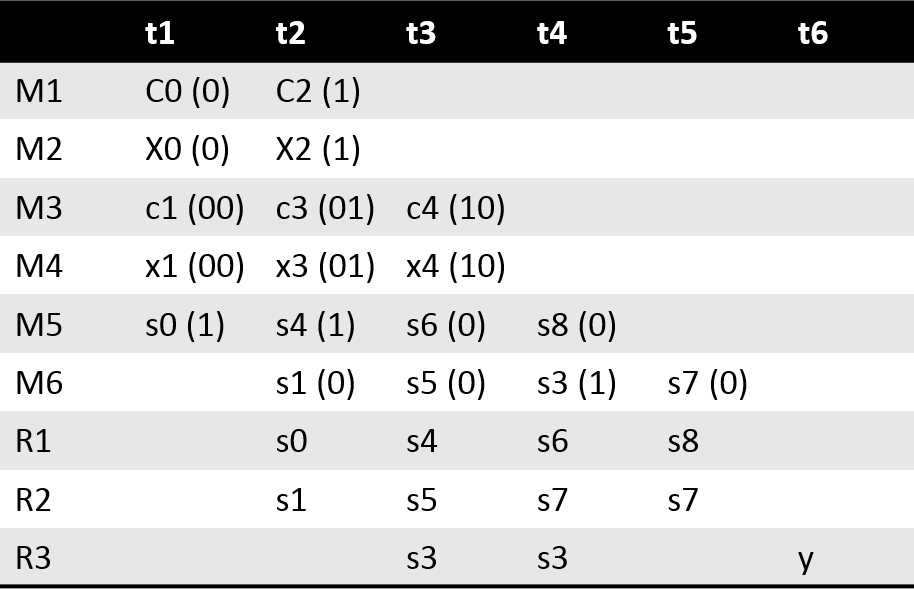
Cada entrada de cada operador y registro recibe distintos datos a lo largo del tiempo, es por ello que tenemos que asignar multiplexores que controlen qu´e entrada se conecta en cada momento. La siguiente figura estudia las posibles conexiones que pueden necesitar los m´odulos aritm´eticos del filtro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



Finalmente, con toda la informaci´on 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 asignaci´on. La siguiente estructura muestra la implementaci´on del filtro FIR de cinco etapas con dos multiplicadores y un sumador.



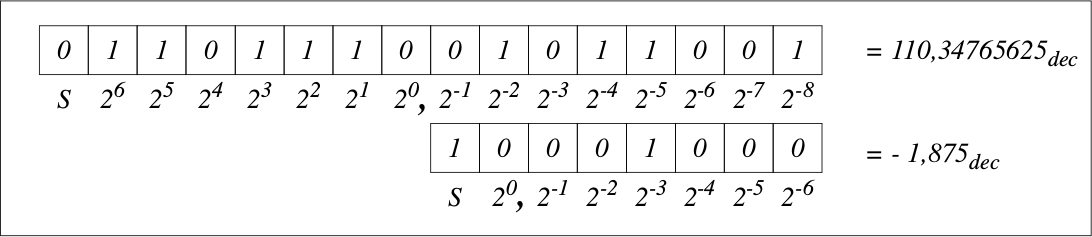
Hasta aqu´ı llegan los pasos para la creaci´on de la ruta de datos, como es l´ogico, se espera que un controlador externo se encargue de proporcionar las sen˜ales de control de los multiplexores y los load de los registros en el momento apropiado. Para implementar el controlador, viene muy bien un cronograma que especifique en cada instante de tiempo qu´e sen˜al tiene que dejar pasar cada multiplexor y qu´e sen˜al tiene que tener guardada cada registro. La siguiente figura muestra el cronograma para el ejemplo del filtro FIR. En cada instante se especifica qu´e sen˜al tiene que dejar pasar cada multiplexor y la sen˜al de control asociada entre par´entesis, as´ı como los datos guardados en los registros para cada instante de tiempo.



## Notaci´on en punto fijo

En este bloque vamos a trabajar con nu´meros representados en notaci´on en punto fijo. Se puede establecer una analog´ıa entre los nu´meros decimales en punto fijo y los binarios en punto

fijo. Un nu´mero binario en punto fijo tiene el siguiente aspecto: “10001110.101”. El punto de un nu´mero binario en punto fijo se pone a la derecha de cifra que representa 20. 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 notaci´on en punto fijo empleando complemento a dos. Observa la figura y asegu´rate de entender el procedimiento de conversi´on entre binario en punto fijo y base 10.



Para este bloque vamos a utilizar la siguiente notaci´on para representar los taman˜os de nu´meros en punto fijo: *< a.b >* donde a es el nu´mero de bits que representan la parte entera y b es el nu´mero de bits necesarios para representar la parte decimal. Por ejemplo el nu´mero superior de la anterior figura necesita 16 bits, y su taman˜o estar´ıa representado por *<* 8*,*8 *>*.

Con esta notaci´on nos es u´til para establecer los taman˜os resultantes de las operaciones de

multiplicaci´on y suma: Multiplicaci´on: Si *<* 3*,*4 *>* se multiplica con *<* 5*,*6 *>* el resultado ser´a

*<* 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 nu´meros no va a producir un desbordamiento). Suma: Si *<* 3*,*4 *>* se suma a *<* 5*,*6 *>* el resultado ser´a *< 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:**

¿Cu´al es el rango de un nu´mero de 8 bits en punto fijo *<* 1*,*7 *>* en complemento a dos?

¿Cu´al es su precisi´on?

R

## Especificaci´on 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 cu´ando 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 cu´ando 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 implementaci´on del filtro emplea una estructura de ruta de datos controlada por un controlador.

La ruta de datos s´olo emplea un multiplicador en pipeline y un sumador.

## Tareas pr´acticas

### Creaci´on de la ruta de datos

**Tarea 2.2:**

Realiza sobre papel la implementaci´on de un filtro FIR de 5 etapas con la siguiente asignaci´on: Dos medios multiplicadores y un sumador. Realiza todos los pasos descritos en la secci´on [6.4](#_bookmark21).

R

### R Planiftcaci´on temporal:

R

**Vinculaci´on, an´alisis de tiempo de vida de variables y asignaci´on de regis- tros:**

### R An´alisis de conexiones para la extracci´on de multiplexores:

R **Implementaci´on:**

R **Cronograma:**

R

**Tarea 2.3:**

Realiza un an´alisis de cuantificaci´on de las sen˜ales, de manera que especifiques cu´antos bits vas a emplear en cada sen˜al y en qu´e posici´on va a estar el punto decimal. Emplea la notaci´on y las metodolog´ıas presentadas en clase.

R

**Tarea 2.4:**

Escribe modelos VHDL para todos los componentes que se emplean en tu ruta de da- tos. Los operadores aritm´eticos (suma y multiplicaci´on) no necesitan describirse en un estilo estructural. Para conseguir el medio multiplicador, an˜ade un registro extra a la salida de un multiplicador completo, de esta forma se comportar´a como un multiplica- dor perfectamente segmentado. La herramienta de s´ıntesis se encargar´a de mover este registro al interior del multiplicador de manera ´optima. Comprueba el funcionamiento correcto de tus componentes creando los correspondientes testbenches.

R

**Tarea 2.5:**

Escribe un modelo estructural de la ruta de datos completa en VHDL. Crea un test- bench para la ruta de datos.

### Creaci´on del controlador

**Tarea 2.6:**

Dibuja el diagrama de estados que describe el controlador de tu filtro FIR. Sigue la metodolog´ıa de implementaci´on de m´aquinas de estados finitos vista en clase.

R

R

**Tarea 2.7:**

Escribe el c´odigo VHDL correspondiente a tu controlador.

R

**Tarea 2.8:**

Escribe el c´odigo VHDL estructural correspondiente a tu filtro completo.

R

**Tarea 2.9:**

Escribe el c´odigo VHDL correspondiente a un testbench que introduzca un u´nico impul- so a la entrada de valor +0.5. ¿Qu´e 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 nu´mero posi- tivo/negativo (cuatro casos) que se puede representar. Ten en cuenta la cuantificaci´on de tus sen˜ales. Asegu´rate de que tu disen˜o proporciona los valores esperados.

R

**Tarea 2.10:**

Escribe el c´odigo VHDL correspondiente a un testbench que introduzca en la entrada la siguiente secuencia (0, 0.5, 0, 0.125, 0, 0, 0, 0, ...). ¿Qu´e secuencia esperas en Sam- ple Out para esta secuencia de entrada? Asegu´rate de que tu disen˜o proporciona los valores esperados.

### Creaci´on de un testbench avanzado

Si tuvi´eramos que hacer pruebas m´as exhaustivas para comprobar el funcionamiento de nuestro filtro, el empleo de los testbenches que hemos usado hasta ahora dejar´ıa de ser pr´actico, 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, despu´es 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 c´odigo VHDL pueden o bien contener datos de un u´nico tipo (std logic vector, integer, singed, etc.) o bien pueden ser de tipo TEXT. Nosotros nos vamos a ocupar u´nicamente de este u´ltimo tipo de archivos, los que contienen datos de tipo TEXT, que son ficheros ASCII. Los ficheros de tipo TEXT se leen l´ınea a l´ınea empleando READLINE, que guarda la l´ınea que corresponda en una sen˜al o variable de tipo LINE. Los fichero de tipo TEXT se escriben l´ınea a l´ınea empleando WRITELINE. Para extraer los datos contenidos en una sen˜al/variable de tipo LINE se emplea READ, y la operaci´on opuesta (escritura en un tipo LINE) se hace a trav´es 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 notaci´on 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 funci´on del paquete IEEE.NUMERIC\_STD:

my\_signed <= to\_signed(int\_number,8) donde int\_number es el integer que querer- mos convertir, 8 es el nu´mero de bits que queremos que tenga nuestro vector y my\_signed es el signed al que se le asigna la conversi´on.

De la misma forma, para escribir en el fichero, tenemos que convertir nuestros datos, que son

signed a integer. Para ello emplearemos la siguiente funci´on:

my\_int <= to\_integer(signed\_number) donde signed\_number es el signed que querer- mos convertir, y my\_int es el integer al que se le asigna la conversi´on.

A la hora de hacer la conversi´on entre signed y integer y viceversa, el programa no es capaz de trabajar con la representaci´on *<* 1*,*7 *>* o con cualquier otra en punto fijo, siempre va a considerar la notaci´on *<* 8*,*0 *>*, es decir, sin coma decimal. Esto implica que los datos del fichero estar´an entre +127 y -128.

En el ap´endice 1 puedes encontrar un ejemplo de testbench que lee de un fichero TEXT para que te hagas una idea de c´omo funciona.

R

**Tarea 2.11:**

Escribe el c´odigo 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 u´nico impulso. Cada l´ınea del fichero se corresponde con un dato.

R

**Tarea 2.12:**

Utiliza el fichero “haha.wav” en Matlab para crear un fichero de entrada para tu testbench. En el ap´endice 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 funci´on *filter* de Matlab para obtener la respuesta de un filtro FIR con precisi´on real.

Cargar y escuchar la salida que ha producido tu testbench en Matlab.

R

**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 importar´as pos- teriormente en Matlab.

R

**Tarea 2.14:**

Importa tu fichero “sample out.dat” en Matlab. Compara los resultados del testbench con los valores con precisi´on real proporcionados por la funci´on *filter* de Matlab. Haz un gr´afico del error de tus resultados (resta tus resultados de los datos con precisi´on real).

Utiliza la funci´on *sound* de Matlab para escuchar la forma de onda original. Comp´arala despu´es con tu sonido filtrado y observa si hay alguna diferencia entre el filtro con precisi´on real y tu filtro.

# 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 an˜adir al disen˜o un core IP que va a encapsular una memoria RAM donde almacenar las muestras de audio, se va a disen˜ar un controlador que orqueste todas las operaciones del sistema y se va a integrar todos los elementos.

El disen˜o del controlador est´a completamente abierto y se proporciona al estudiante la po- sibilidad de practicar las habilidades adquiridas a lo largo del curso, disen˜ando el sistema desde cero teniendo en cuenta el conjunto de especificaciones que lo definen.

## Memoria RAM

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

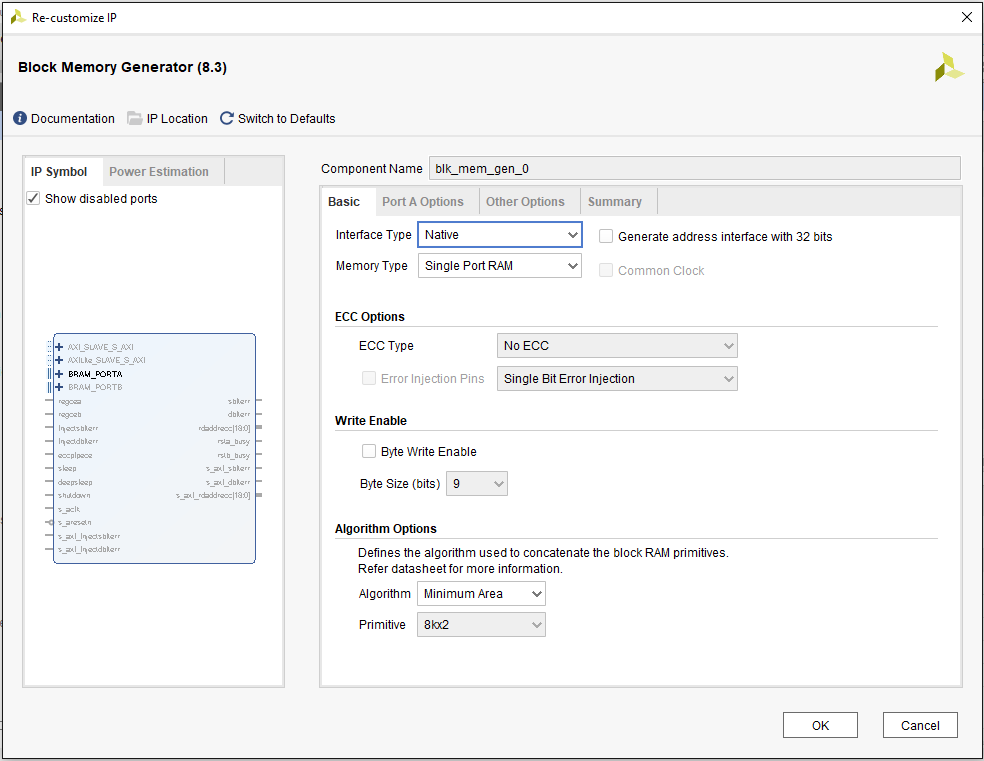
Taman˜o de cada palabra almacenada: 8 bits.

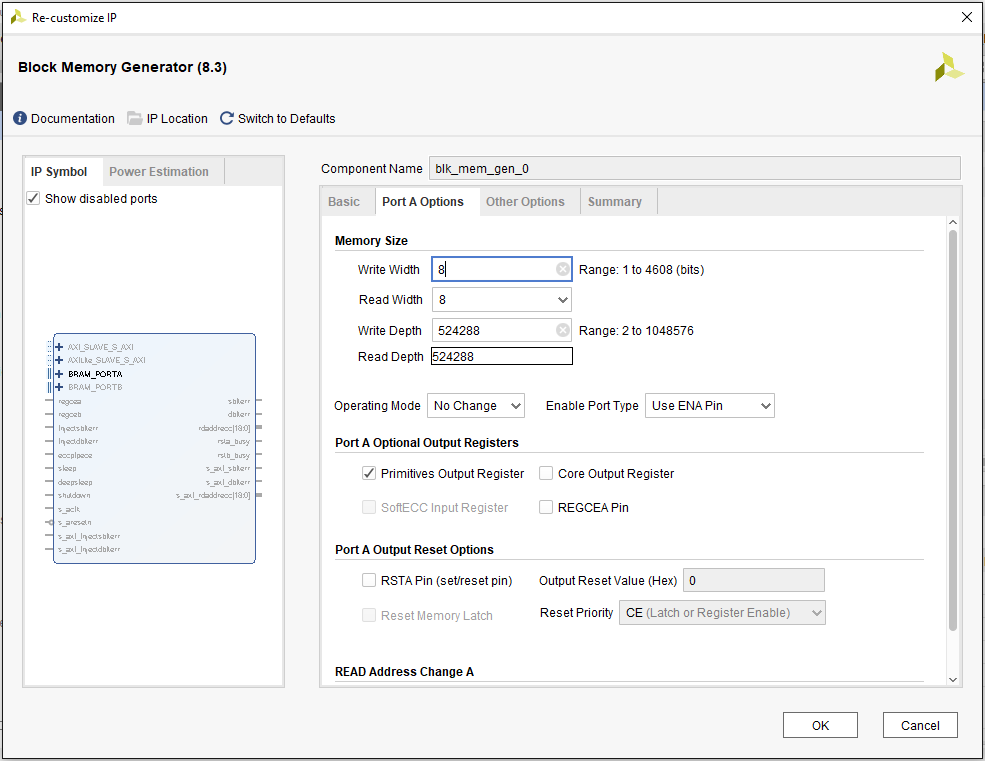
Taman˜o del bus de direcciones: 19 bits.

Nu´mero de palabras almacenadas. 219 = 524288. Taman˜o en bits de la memoria: 219 ∗ 8 = 4194304.

Para instanciar esta memoria en nuestro disen˜o se va a emplear el “IP Catalog” del Vivado, donde hay que buscar la siguiente opci´on: “Memories and Storage Elements”, “RAMs and ROMs and BRAM”, Block Memory Generator.

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





R

**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 continuaci´on la funci´on de cada puerto y la temporizaci´on de la memoria (un pequen˜o cronograma que incluya una escritura y una lectura).

R

**Tarea 3.2:**

Determina cu´anto tiempo de grabaci´on va a poder estar almacenado en la memoria.

## Conversi´on entre codificaciones

En el Bloque 1 se obtienen datos de la interfaz del micr´ofono que utilizan una codificaci´on binaria de 8 bits. Esta misma codificaci´on se emplea en la interfaz de salida de audio. En el bloque 2, sin embargo, el filtro FIR necesita utilizar una representaci´on en complemento a dos para poder usar coeficientes tanto negativos como positivos. Adem´as, los datos est´an normalizados, de tal manera que el m´aximo valor con el que trabaja es 1, con lo que emplea un esquema de cuantificaci´on *<* 1*,*7 *>* en complemento a dos.

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

La primera transformaci´on que necesitamos hacer es convertir muestras en notaci´on binaria a muestras en notaci´on en complemento a dos y viceversa. La representaci´on binaria tiene un

rango de [0 → 28 − 1] mientras que la representaci´on en complemento a dos tiene un rango de [−27 → 27 − 1], la correspondencia de las muestras de una representaci´on a otra es directa, es decir al elemento menor de una representaci´on le corresponde el elemento menor de la otra

(00000000*b* se corresponde con 100000002*c*), el siguiente con el siguiente y as´ı sucesivamente hasta el mayor (111111112 se corresponde con 011111112*c*).

**Tarea 3.3:**

Determina qu´e operaci´on es necesaria para hacer la transformaci´on de las muestras de binaria a complemento a dos y de complemento a dos a binaria.

R

La siguiente transformaci´on consiste en pasar de un esquema *<* 8*,*0 *>* a un esquema *<* 1*,*7 *>*. Pero, ¿es realmente necesario realizar algu´n cambio en nuestros datos? El lugar donde colocamos el punto decimal es una abstracci´on que nos permite calcular el valor en base 10 equivalente y con ´el ajustar el valor de los coeficientes del filtro, pero el hardware se va a ser id´entico sin importar el lugar del punto decimal. Por lo tanto, esta segunda transformaci´on no es necesaria, la primera s´ı.

## Controlador y sistema global

El controlador es el u´nico m´odulo del proyecto donde el proceso de disen˜o no est´a guiado y es la u´ltima pieza del puzzle para construir el sistema global. A partir de las especificaciones dadas en la secci´on [2](#_bookmark1) y conociendo las funcionalidades y las interfaces de los distintos bloques, decide c´omo vas a implementar la funcionalidad del controlador. En concreto, decide que estilo de disen˜o vas a utilizar (m´aquina de estados, m´aquina de estados con ruta de datos asociada, un sistema con distintos elementos secuenciales, etc.), divide el disen˜o de tal manera que lo puedas ir implementando de manera progresiva an˜adiendo funciones nuevas cuando est´en aseguradas las anteriores, planifica c´omo vas a comprobar el funcionamiento de cada especificaci´on y haz una planificaci´on temporal.

**Tarea 3.4:**

An˜ade a continuaci´on todas las hojas necesarias para describir tu disen˜o y la pla- nificaci´on para llevarlo a cabo. Puedes incluir diagramas esquem´aticos, cronogramas explicativos, un plan de pruebas, una planificaci´on temporal y todo lo que consideres necesario para explicar las decisiones que has tomado.

Avisa al profesor antes de comenzar con el disen˜o para obtener el visto bueno.

R

R

**Tarea 3.5:**

Avisa al profesor cuando tengas todas las especificaciones m´ınimas del sistema global cubiertas.

R

**Tarea 3.6:**

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

# Mejoras opcionales

Una vez conseguidas las especificaciones b´asicas, los alumnos interesados pueden ampliar el disen˜o con mejoras. Las mejoras pueden ser propuestas por los propios alumnos o basarse en la lista que se proporciona a continuaci´on. Las mejoras se evaluar´an en funci´on de su nivel de dificultad, sumar´an a la nota general del proyecto y se tendr´an en cuenta para la asignaci´on de las matr´ıculas de honor del curso.

## Control de volumen logar´ıtmico

Especificaciones m´ınimas:

Al sistema se le incorporan dos botones de control, uno correspondiente a subir volumen y otro correspondiente a bajar volumen.

El sistema muestra en todo momento el nivel de volumen en dos displays de 7 segmentos.

El sistema tiene 21 niveles de volumen, el 0 correspondiente a silencio, el 10 correspondiente al volumen original y el 20 correspondiente a 8 veces el volumen original.

El control de volumen es logar´ıtmico para adaptarse a la percepci´on del o´ıdo humano. En concreto el factor por el que se multiplica el volumen vendr´a dado por la siguiente f´ormula:

*nivel*

*factor* = 7 10 − 1

7 − 1

(3)

donde nivel es el nivel de volumen actual.

El control de volumen es independiente al resto del sistema y puede funcionar en cualquier momento, por ejemplo mientras se est´a reproduciendo o grabando.

## Eco y reverberaci´on configurables

Especificaciones m´ınimas:

Al sistema se le incorporan dos botones de control, uno correspondiente a aumentar el eco/reverberaci´on y otro correspondiente a reducirlo.

El sistema muestra en todo momento el nivel de eco/reverberaci´on en dos displays de 7 segmentos.

El sistema tiene 21 niveles de eco/reverberaci´on, el 0 correspondiente a la ausencia de efecto y cada nivel va aumentando 100*µ*s, de tal manera que el retardo de la segunda muestra viene dado por:

*retardo* = *nivel* ∗ 100*µs* (4)

La segunda muestra ser´a igual a la primera con una reducci´on de volumen del 25 %;

## Informaci´on de segundos restantes

Especificaciones m´ınimas:

El sistema muestra en dos displays de 7 segmentos la cantidad de segundos restantes en la memoria de grabaci´on.

En los u´ltimos tres segundos se emite un pitido breve (250*µ*s) en la salida de audio y cuando la memoria est´a vac´ıa se emite el pitido constantemente.

## Looper

Especificaciones m´ınimas:

El sistema se le incorpora un switch de control para encender el modo looper y un led de salida a modo de metr´onomo.

En el modo looper, al pulsar el bot´on de reproducci´on:

* + - El sistema primero reproduce cuatro pitidos breves (250*µ*s) separados 1s para avisar del comienzo y marcar el ritmo inicial. A la vez que suenan los pitidos breves, se activar´a el led metr´onomo.
    - Seguidamente el sistema reproducir´a todo lo almacenado en la memoria de principio a fin en bucle, es decir, cuando llegue a la direcci´on final volver´a a la inicial inde- finidamente. Durante el tiempo de la reproducci´on el led metr´onomo se encender´a brevemente cada segundo sincronizado con el principio de la grabaci´on.
    - El bucle se detiene cuanto se pulse de nuevo el bot´on de reproducci´on.
    - Si la memoria est´a vac´ıa, sonar´an los cuatro pitidos iniciales y el sistema se quedar´a sin reproducir nada, pero con los leds parpadeando hasta que se vuelva a pulsar el bot´on de reproducci´on.

En el modo looper, el bot´on de grabaci´on s´olo tiene efecto mientras se est´a reproduciendo. La funcionalidad es la siguiente:

* + - Al pulsar el bot´on lo que se registre en el micr´ofono se mezclar´a con lo que est´a sonando en ese momento (el valor digitalizado se suma al valor almacenado en la memoria y se guarda el resultado de la suma en la misma posici´on de memoria).
    - Si se mantiene el bot´on de grabaci´on pulsado al llegar el final de la grabaci´on anterior: Durante el tiempo que est´e el bot´on pulsado se detiene la reproducci´on, los leds siguen parpadeando y lo que se recoja del micr´ofono se guarda en posiciones nuevas de la memoria. Al soltar el bot´on se reanuda la reproducci´on en bucle volviendo al principio.

**Tarea 3.7 (Opcional):**

An˜ade a continuaci´on todas las hojas necesarias para describir las mejoras que hayas implementado.

R

# Ap´endice 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;

# Ap´endice 2

Comandos Matlab que se emplean en el bloque 2.

Carga de un fichero .wav en Matlab y creaci´on 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));

Utilizaci´on de la funci´on *filter* de Matlab para obtener la respuesta de un filtro FIR con precisi´on 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. Atenci´on, es importante no olvidar la divisi´on por 127 para no destruir los altavoces.

vhdlout=load(’sample\_out.dat’)/127; sound(vhdlout);