Diseño de Sistemas Electrónicos Digitales

Sistema de grabación, tratamiento y reproducción de audio

Grupo 8.

Alejandro Gómez Molina

Luis Felipe Vélez Flores

Contenidos

[Bloque 1: Interfaz de audio 4](#_Toc533854079)

[Tarea 1.1: Generador de enables 4](#_Toc533854080)

[Tarea 1.3: Test del generador 4](#_Toc533854081)

[Tarea 1.6: Test del micrófono 5](#_Toc533854082)

[Tarea 1.11: Diagrama de bloques de la interfaz de audio 6](#_Toc533854083)

[Tarea 1.13: Test Interfaz completa 6](#_Toc533854084)

[Tarea 1.15: Test controlador 6](#_Toc533854085)

[Bloque 2: Filtro FIR 7](#_Toc533854086)

[Tarea 2.1: 7](#_Toc533854087)

[Tarea 2.2: Implementación de un filtro FIR de 5 etapas 7](#_Toc533854088)

[Planificación temporal 7](#_Toc533854089)

[Vinculación, análisis de tiempo de vida y asignación de registros 7](#_Toc533854090)

[Análisis de conexiones para la extracción de multiplexores: 8](#_Toc533854091)

[Implementación: 8](#_Toc533854092)

[Cronograma 9](#_Toc533854093)

[Tarea 2.3: Cuantificación de las señales: 10](#_Toc533854094)

[Tarea 2.6: Diagrama de estados del filtro FIR 10](#_Toc533854095)

[Bloque 3: Controlador y Memoria. 12](#_Toc533854096)

[Tarea 3.2: 12](#_Toc533854097)

[Tarea 3.3: 12](#_Toc533854098)

[Tarea 3.4: 12](#_Toc533854099)

[﷟HYPERLINK "bookmark://\_Toc533854100" 14](#_Toc533854100)

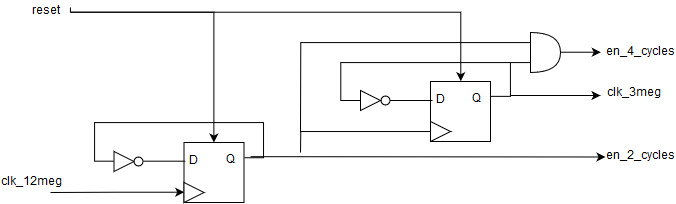
[﷟HYPERLINK "bookmark://\_Toc533854101" 14](#_Toc533854101)

[Utilización de led RGB: 15](#_Toc533854102)

# Bloque 1: Interfaz de audio

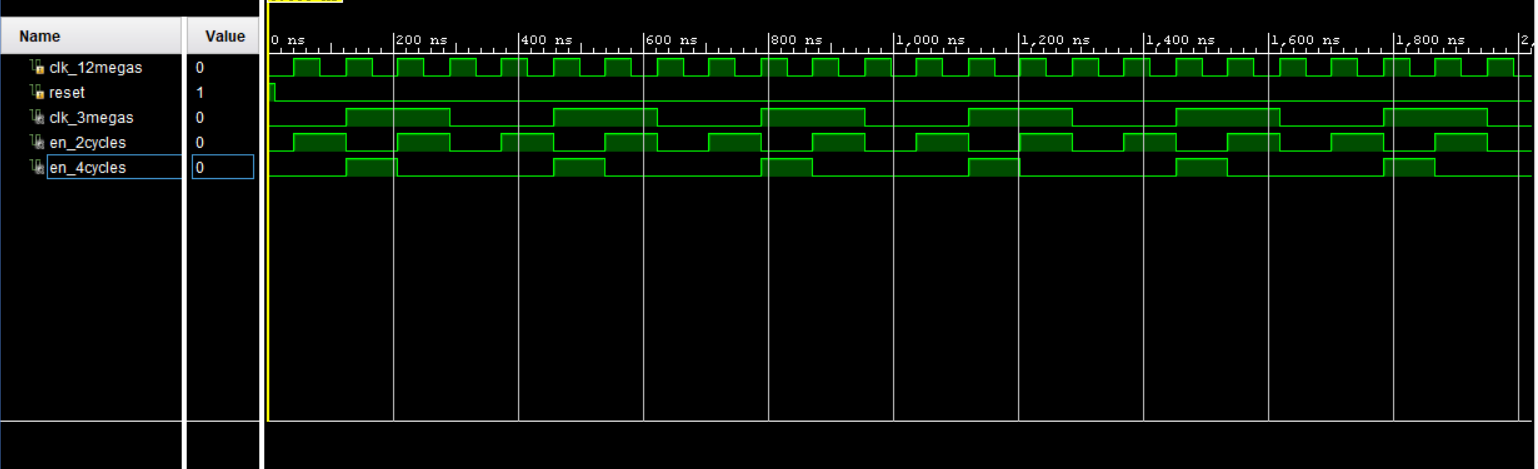
## Tarea 1.1: Generador de enables

El circuito capaz de generar la salida especificada es el siguiente:

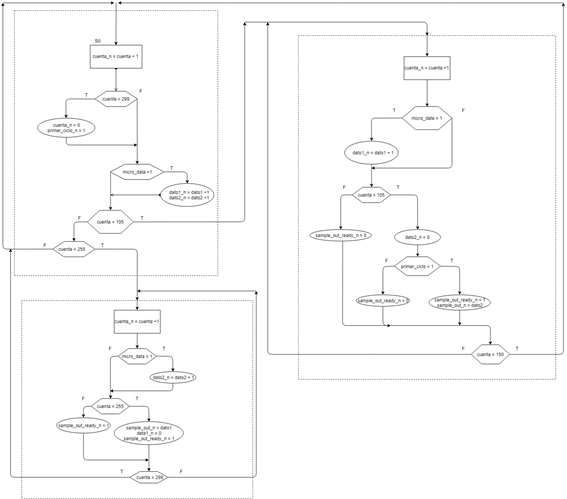


## Tarea 1.3: Test del generador

Una vez implementado, el resultado obtenido es el siguiente:



Tarea 1.4: Diagrama ASMD



Tarea 1.5:

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

-- Company: UPM

-- Engineer: Alejandro Gómez Molina & Luis Felipe Vélez

--

-- Create Date: 10/30/2018 05:59:03 PM

-- Design Name: FSMD Microphone

-- Module Name: FSMD\_microphone - Behavioral

-- Project Name: DSED -> Sistema de grabación, tratamiento y reproducción de audio.

-- Target Devices: Nexys 4 DDR

-- Tool Versions: Vivado 2017.2

-- Description: Para este módulo hemos empleado una arquitectura FSMD.

-- Dependencies: work.package\_dsed.all

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments: se puede utilizar un unico sumador por estado,reduciendo asi el

--uso de recursos.Para ello deberiamos añadir mas estados y sumar en un estado el contador

-- y en otro dato1 o dato2 segun corresponda

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

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use work.package\_dsed.all;

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;

architecture Behavioral of FSMD\_microphone is

--Types

type state\_t is (s0, s1, s2);

--Signals

signal state, state\_n : state\_t := s0;

signal src0\_data1, src0\_data2, src1\_data1, src1\_data2 : unsigned (sample\_size -1 downto 0) := (others => '0');

signal dato1, dato1\_n, dato2, dato2\_n : unsigned (sample\_size -1 downto 0) := (others => '0');

signal cuenta, cuenta\_n, src0\_cuenta, src1\_cuenta : unsigned (log2c(300)-1 downto 0) := (others => '0');

signal sample\_out\_n, sample\_out\_reg : STD\_LOGIC\_VECTOR (sample\_size -1 downto 0) := (others => '0');

signal primer\_ciclo, primer\_ciclo\_n : STD\_LOGIC := '0';

signal sample\_out\_ready\_n, sample\_out\_ready\_reg : STD\_LOGIC := '0';

begin

process(clk\_12megas, reset)

begin

--Asynchronus reset

if(reset = '1') then

--Load default values

sample\_out\_ready\_reg <= '0';

sample\_out\_reg <= (others=>'0');

primer\_ciclo <= '0';

dato1 <= (others=>'0');

dato2 <= (others=>'0');

cuenta <= (others => '0');

state <= s0;

elsif(rising\_edge(clk\_12megas)) then

sample\_out\_ready\_reg <= '0'; --Must to be active only one cycle!!

if( enable\_4\_cycles = '1') then

--Next state update

sample\_out\_ready\_reg <= sample\_out\_ready\_n;

sample\_out\_reg <= sample\_out\_n;

primer\_ciclo <= primer\_ciclo\_n;

dato1 <= dato1\_n;

dato2 <= dato2\_n;

state <= state\_n;

cuenta <= cuenta\_n;

end if;

end if;

end process;

--Next state logic

process( cuenta,state)

begin

case state is

when s0 =>

if cuenta = 105 then

state\_n <= s1;

elsif cuenta = 255 then

state\_n <= s2;

else

state\_n <= s0;

end if;

when s1 =>

if cuenta = 150 then

state\_n <= s0;

else

state\_n <= s1;

end if;

when s2 =>

if cuenta = 299 then

state\_n <= s0;

else

state\_n <= s2;

end if;

end case;

end process;

process(state, cuenta, dato1 ,dato2, micro\_data,enable\_4\_cycles)

begin

--Default values

sample\_out\_ready\_n <= sample\_out\_ready\_reg;

primer\_ciclo\_n <= primer\_ciclo;

sample\_out\_n <= sample\_out\_reg;

src0\_data1 <= dato1;

src1\_data1 <= (others => '0');

src0\_data2 <= dato2;

src1\_data2 <= (others => '0');

src0\_cuenta <= cuenta;

src1\_cuenta <= (others => '0');

case state\_n is

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

when s0 =>

src0\_cuenta <= cuenta;

src1\_cuenta <= to\_unsigned( 1 ,src1\_cuenta'length);

if cuenta = 299 then

src0\_cuenta <= (others => '0');

src1\_cuenta <= (others => '0');

primer\_ciclo\_n <= '1';

end if;

if(micro\_data = '1') then

src0\_data1 <= dato1;

src1\_data1 <= to\_unsigned( 1 ,src1\_data1'length);

src0\_data2 <= dato2;

src1\_data2 <= to\_unsigned( 1 ,src1\_data2'length);

end if;

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

when s1 =>

src0\_cuenta <= cuenta;

src1\_cuenta <= to\_unsigned( 1 ,src1\_cuenta'length);

if(micro\_data = '1') then

src0\_data1 <= dato1;

src1\_data1 <= to\_unsigned( 1 ,src1\_data1'length);

end if;

if(cuenta = 105) then

src0\_data2 <= (others => '0');

src1\_data2 <= (others => '0');

if(primer\_ciclo = '1') then

sample\_out\_ready\_n <= '1';

sample\_out\_n <= STD\_LOGIC\_VECTOR(dato2);

else

sample\_out\_ready\_n <= '0';

end if;

else

sample\_out\_ready\_n <= '0';

end if;

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

when s2 =>

if(micro\_data = '1') then

src0\_data2 <= dato2;

src1\_data2 <= to\_unsigned( 1 ,src1\_data2'length) ;

end if;

if(cuenta = 255) then

sample\_out\_n <= STD\_LOGIC\_VECTOR(dato1);

src0\_data1 <= (others => '0');

src1\_data1 <= (others => '0');

sample\_out\_ready\_n <= '1';

else

sample\_out\_ready\_n <= '0';

end if;

src0\_cuenta <= cuenta;

src1\_cuenta <= to\_unsigned( 1 ,src1\_cuenta'length);

end case;

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

dato1\_n <= src0\_data1 + src1\_data1;

dato2\_n <= src0\_data2 + src1\_data2;

cuenta\_n <= src0\_cuenta + src1\_cuenta;

end process;

--Output logic

sample\_out\_ready <= sample\_out\_ready\_reg;

sample\_out <= sample\_out\_reg;

end Behavioral;

## Tarea 1.6: Test del micrófono

library IEEE;

use IEEE.Std\_logic\_1164.all;

use IEEE.Numeric\_Std.all;

use work.package\_dsed.all;

entity FSMD\_microphone\_tb is

end;

architecture bench of FSMD\_microphone\_tb is

component en\_4\_cycles

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 component;

component 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 component;

signal clk\_12megas: STD\_LOGIC;

signal reset: STD\_LOGIC;

signal clk\_3megas: STD\_LOGIC;

signal en\_2\_cycles: STD\_LOGIC;

signal ena\_4\_cycles: STD\_LOGIC ;

signal micro\_data: STD\_LOGIC;

signal sample\_out : STD\_LOGIC\_VECTOR (sample\_size -1 downto 0);

signal sample\_out\_ready : STD\_LOGIC;

constant clock\_period: time := 83 ns;

signal stop\_the\_clock: boolean;

begin

uut: en\_4\_cycles port map ( clk\_12megas => clk\_12megas,

reset => reset,

clk\_3megas => clk\_3megas,

en\_2\_cycles => en\_2\_cycles,

en\_4\_cycles => ena\_4\_cycles );

uut1: FSMD\_microphone port map ( clk\_12megas => clk\_12megas,

reset => reset,

enable\_4\_cycles => ena\_4\_cycles,

micro\_data => micro\_data,

sample\_out => sample\_out,

sample\_out\_ready => sample\_out\_ready);

stimulus: process

begin

reset <= '1';

micro\_data <= '1';

wait for 160 ns;

reset <= '0';

wait;

end process;

clocking: process

begin

while not stop\_the\_clock loop

clk\_12megas <= '0', '1' after clock\_period / 2;

wait for clock\_period;

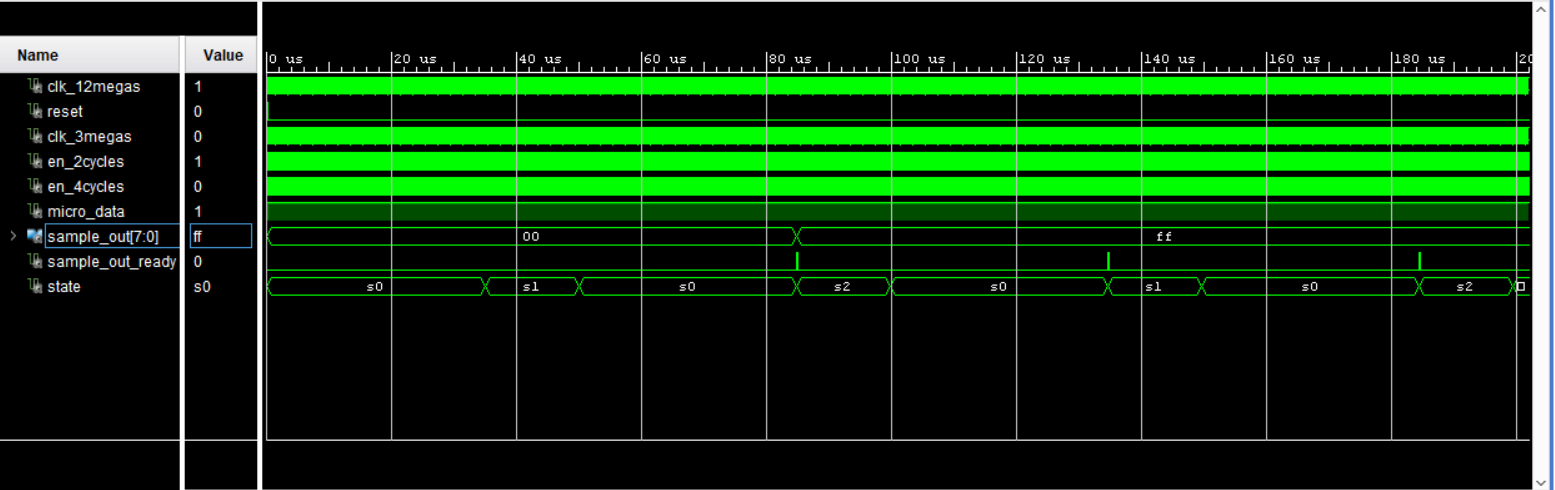
end loop;

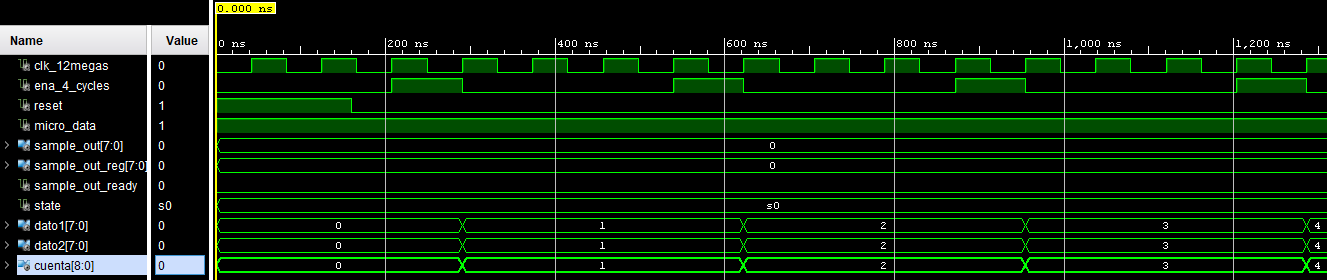
wait;

end process;

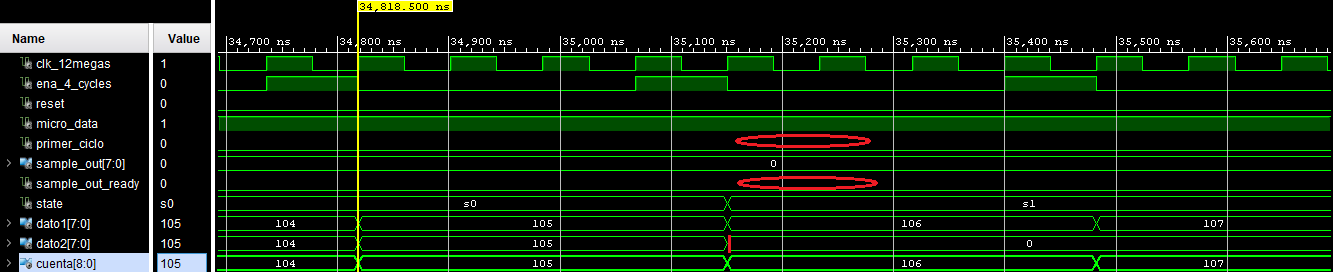
end;

Como podemos ver en la siguiente figura, el comportamiento es el esperado.

A continuación, presentamos los resultados del tb en las partes cruciales: cambio de estado, reinicio de contador, cambio entre dato1 y dato2.

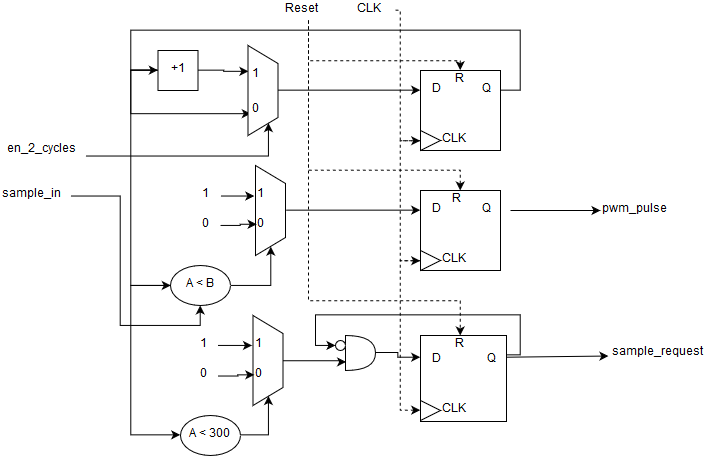


En la siguiente captura observamos que sample\_out\_ready no se pone a 1 pues primer\_ciclo tampoco lo está. Además, reiniciamos dato2.

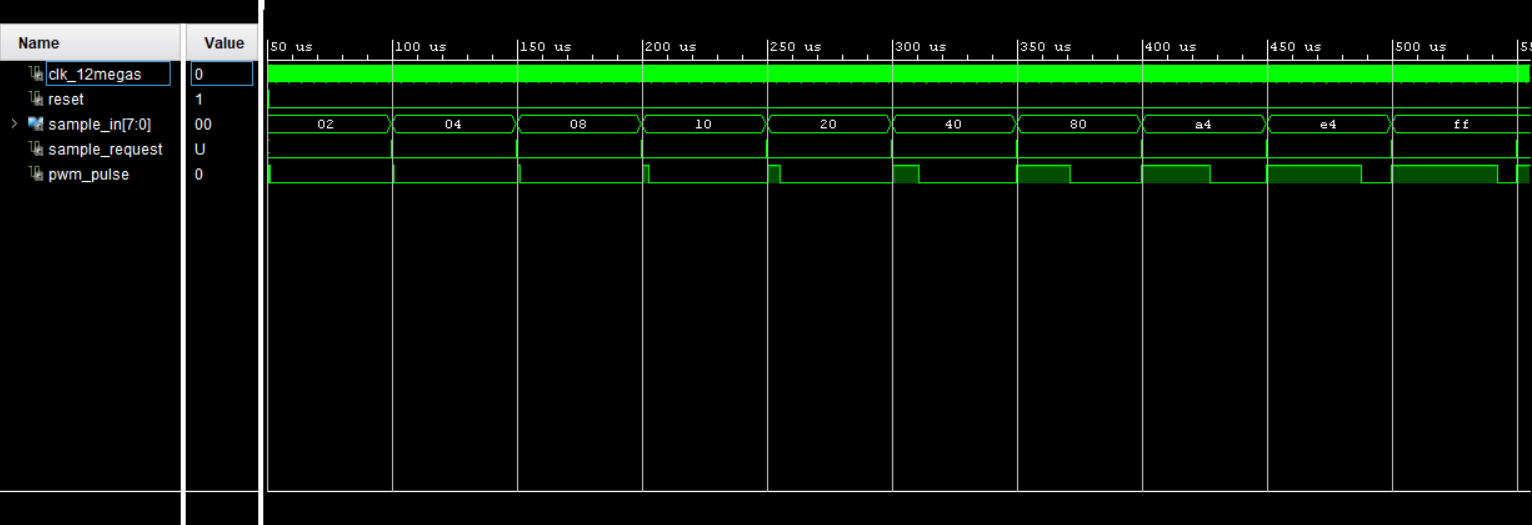


Tarea 1.7: Test aleatorio del micrófono

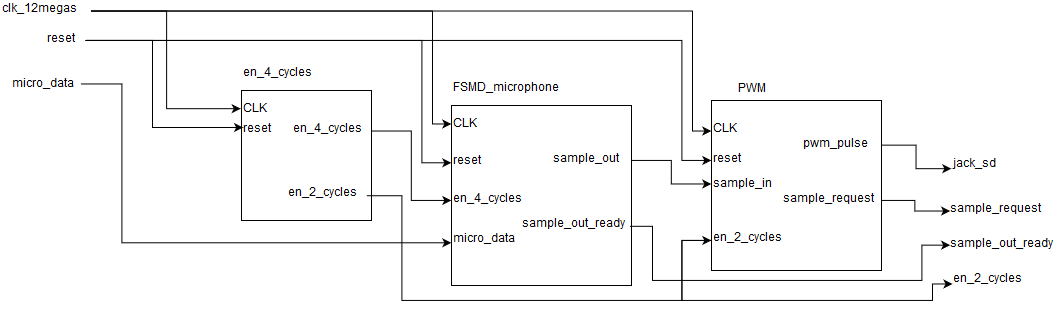
Tarea 1.8: Esquemático del PWM



Tarea 1.10: Test del PWM



## Tarea 1.11: Diagrama de bloques de la interfaz de audio



## Tarea 1.13: Test Interfaz completa

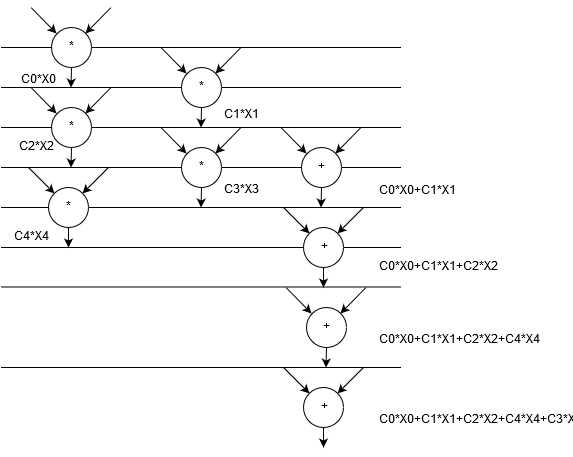
## Tarea 1.15: Test controlador

# Bloque 2: Filtro FIR

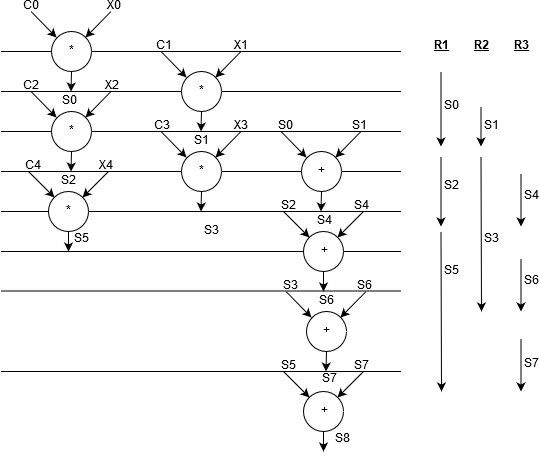
## Tarea 2.1:

## Tarea 2.2: Implementación de un filtro FIR de 5 etapas

### Planificación temporal



### Vinculación, análisis de tiempo de vida y asignación de registros

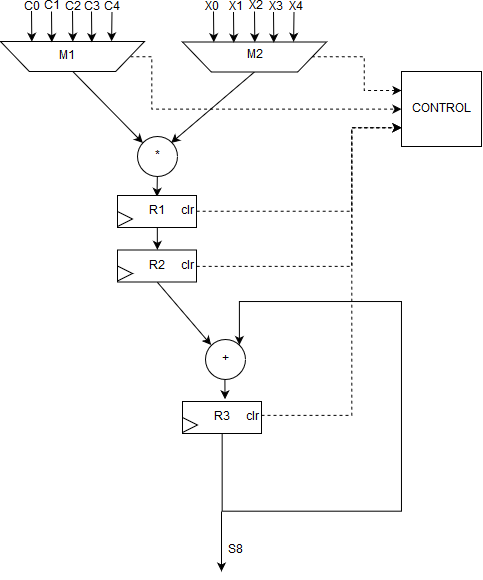


### Análisis de conexiones para la extracción de multiplexores:

Teniendo en cuenta que usaremos un solo multiplicador completo segmentado, el análisis de conexiones es el siguiente:



### Implementación:



### Cronograma

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 |
| C\_M12 | "000" | "001" | "010" | "011" | "100" |  |  |  |
| O\_M1 | C0 | C1 | C2 | C3 | C4 |  |  |  |
| O\_M2 | X0 | X1 | X2 | X3 | X4 |  |  |  |
| R1 | 0 | S0 | S1 | S2 | S3 | S5 |  |  |
| R2 | 0 | 0 | S0 | S1 | S2 | S3 | S5 |  |
| R3 | 0 | 0 | 0 | S0 | S4 | S6 | S7 | S8 |

Donde:

**C\_M12** es la señal de control de los multiplexores M1 y M2

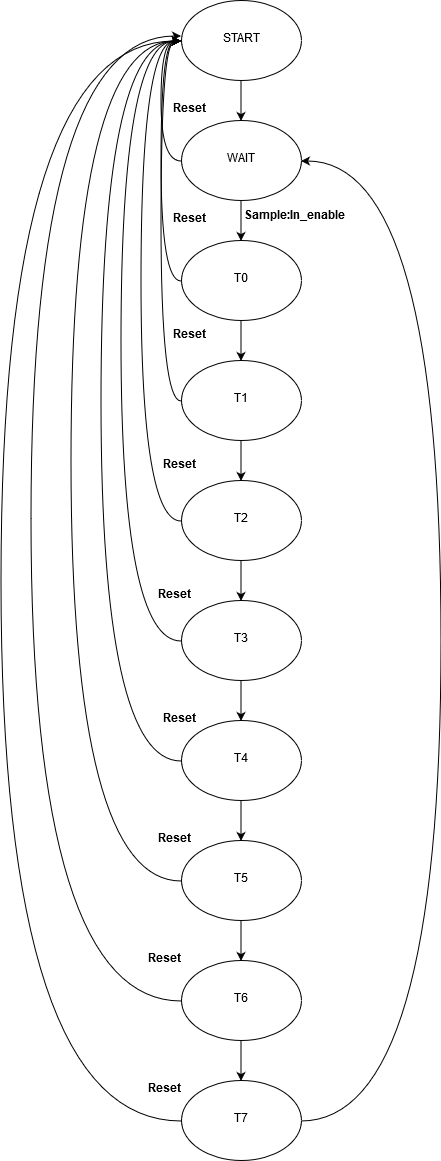
**O\_M1 y O\_M2** son las salidas de los multiplexores M1 y M2 respectivamente

**R1, R2 y R3** son los valores de los registros

## Tarea 2.3: Cuantificación de las señales:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Formato | Tamaño | Explicación |
| S0 | <2,14> | 16 | Resultado de multiplicar dos números (Cx, Xx) con formato <1,7> |
| S1 |
| S2 |
| S3 |
| S5 |
| S4 | <3,14> | 17 | Resultado de sumar  A<2,14> + B<2,14> |
| S6 | <4,14> | 18 | Resultado de sumar  A<2,14> + B<3,14> |
| S7 | <5,14> | 19 | Resultado de sumar  A<2,14> + B<4,14> |
| S8 | <6,14> | 20 | Resultado de sumar  A<2,14> + B<5,14> |

## Tarea 2.6: Diagrama de estados del filtro FIR



# Bloque 3: Controlador y Memoria.

## Tarea 3.2:

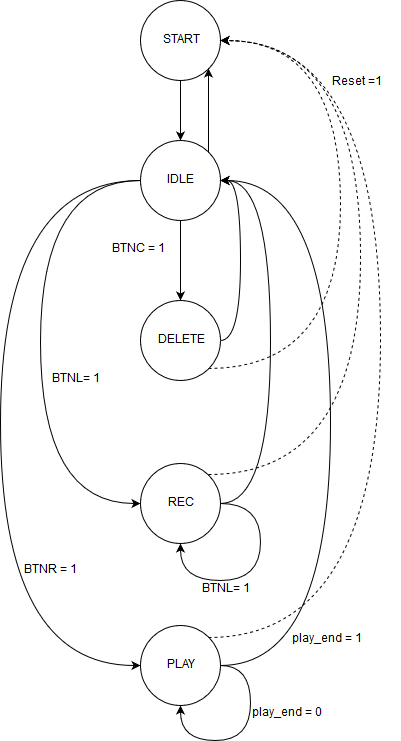
La memoria tiene una capacidad de 524288 bytes. Teniendo en cuenta que muestreamos a 20.000 muestras/segundo y 1 byte/muestra:

## Tarea 3.3:

Para hacer dicha transformación, solo será necesario negar el bit más significativo.

## Tarea 3.4:

El diagrama de estados del controlador es el siguiente:



**START:** Entra en este estado solo tras un reset. Reinicia los registros y señales de control.

**IDLE:** Se encuentra en este estado a la espera de que algún botón sea pulsado.

**REC:** Mientras el botón BTNL se encuentre pulsado, grabará los datos en la memoria RAM. Para esta acción hace uso del registro last\_b, el cual apunta a la última dirección grabada.

**PLAY:** Cuando se pulsa el botón BTNR, se reproducirá todo el audio almacenado. El estado de SW0 y SW1 definirán el orden de reproducción. Para este estado usamos el registro cur\_b, el cual apunta a la última dirección reproducida.

**DELETE:** Cuando se pulsa el botón BTNC, el registro last\_b pasa a apuntar a la primera dirección de memoria.

La lógica de estado siguiente es:

**Process(**state**,** BTNL**,** BTNC**,** BTNR**,** cur\_b**,** last\_b**)**

**begin**

state\_n **<=** state**;**

state\_out **<=** "000"**;**

**case** state **is**

**when** START **=>**

state\_out **<=** "001"**;**

state\_n **<=** IDLE**;**

**when** IDLE **=>**

state\_out **<=** "010"**;**

**if(** BTNL **=** '1'**)** **then**

state\_n **<=** REC**;**

**elsif(** BTNC **=** '1'**)** **then**

state\_n **<=** DELETE**;**

**elsif(** BTNR **=** '1'**)** **then**

state\_n **<=** PLAY**;**

**end** **if;**

**when** REC **=>**

state\_out **<=** "011"**;**

**if(**BTNL **=** '1'**)** **then**

state\_n **<=** REC**;**

**else**

state\_n **<=** IDLE**;**

**end** **if;**

**when** PLAY **=>**

state\_out **<=** "100"**;**

**if(** **((NOT** SW1**)AND** SW0**)** **=** '1'**)** **then**

**if(**cur\_b **=** 0**)** **then**

state\_n **<=** IDLE**;**

**end** **if;**

**else**

**if(**cur\_b **=** last\_b**)** **then**

state\_n **<=** IDLE**;**

**end** **if;**

**end** **if;**

**when** DELETE **=>**

state\_out **<=** "101"**;**

state\_n **<=** IDLE**;**

**end** **case;**

**end** **process;**

# Mejoras opcionales:

## Información de segundos restantes.

El código relativo a esta mejora está en el modulo seconds2segments. Utilizamos 4 de los 8 diplays para mostrar los segundos restantes en la memoria y los segundos se la reproducción. En este módulo podemos distinguir tres partes:

**Control de los displays:**

Para la activación de los ánodos utilizamos un contador de 15 bits que se incrementa en una unidad cada flanco de reloj de 12Mhz. Los dos bits más significativos del contador definen la salida de la siguiente manera:

-- Anodes out!

**with** counter**(**14 **downto** 13**)**

**select** an\_reg **<=** "11111110" **when** "00"**,** --Using only 4

"11111101" **when** "01"**,**

"11111011" **when** "10"**,**

"11110111" **when** "11"**,**

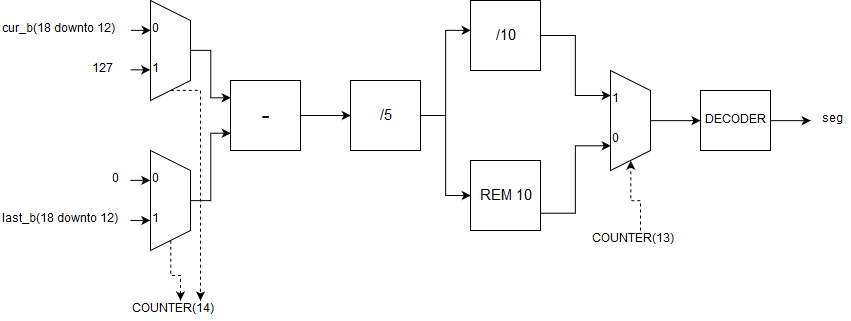
"11111111" **when** **others;**

Utilizando esta implementación, obtenemos una frecuencia de refresco:

De manera similar, usamos estos bits para actualizar las salidas de los displays.

**Control de datos:**

Para determinar los segundos restantes, utilizamos los registros last\_b y cur\_b. El datapath del módulo es el siguiente:



Para evitar utilizar divisores de 19 bits (área muy grande), hemos hecho aproximación usando los 7 bits más significativos. El funcionamiento es sencillo:

Sabemos que un segundo equivale a 20K muestras. Los 7 últimos bits la dirección serán múltiplos de , además 4096\*5 = 20480 ≈ 20000. Por tanto, si dividimos el valor de los 7 bits mas significativos entre 5, tendremos el valor aproximado en segundos. El error total cometido será

Aunque el error es un poco alto, de esta manera reducimos el área en más de un 50%

**Alarma:**

El modulo cuenta con una señal de alarma la cual se activa cuando quedan 3 segundos de grabación o cuando la memoria está llena. Está señal es usada por el controlador en el estado REC: Cuando está activa, la entrada del PWM aumenta en una unidad cada flanco de reloj, obteniendo una señal de diente de sierra.

## Utilización de led RGB:

En esta mejora utilizamos uno de los leds RGB para indicar la cantidad de memoria disponible. El led hace un degradado desde verde (memoria vacía) hasta rojo. Para ello, dejamos el valor de azul a 0 y variamos el rojo y el verde de la manera mostrada en la figura.

|  |  |
| --- | --- |
| Evolución de los valores RGB | Gama de colores obtenidos |

Para conseguir el degradado, utilizamos los 9 bits mas significativos de la dirección de la siguiente manera:

**if(**addrA**(**18**)** **=** '0'**)** **then**

G\_reg\_n **<=** **(others** **=>** '1'**);**

R\_reg\_n **<=** addrA**(**17 **downto** 10**);**

**else**

R\_reg\_n **<=** **(others** **=>** '1'**);**

G\_reg\_n **<=** **NOT** addrA**(**17 **downto** 10**);**

**end** **if;**