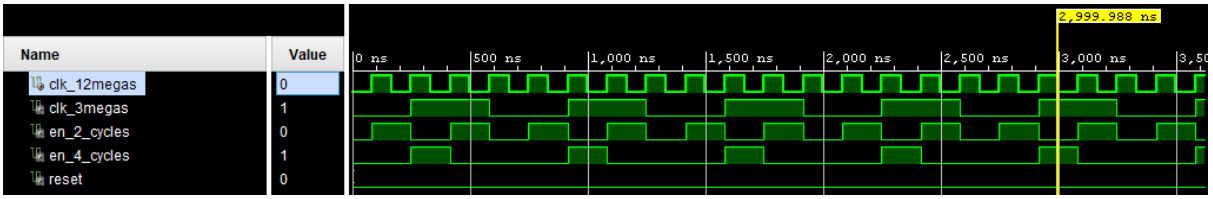
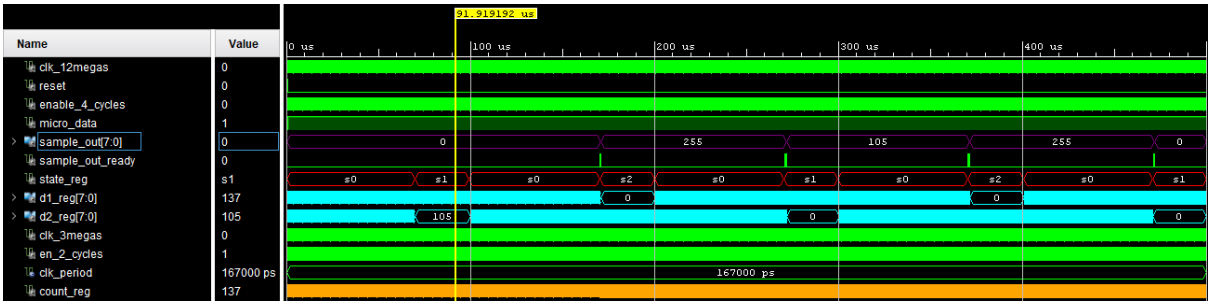
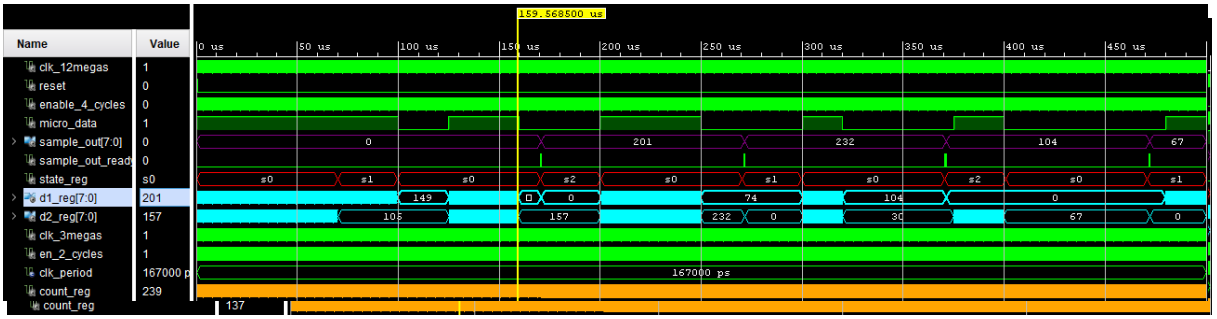
Testbenches ——— Tareas 1.3, 1.6, 1.7, 1.10, 1.13, 1.15



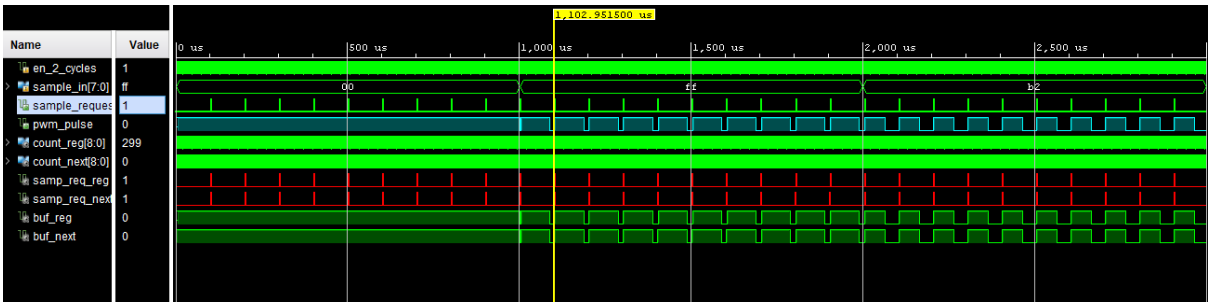
en_4_cycles_tb.vhd



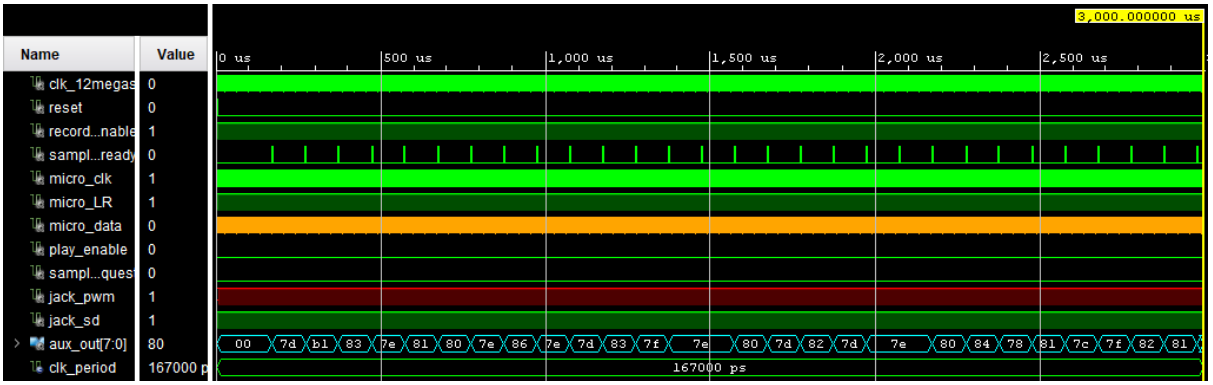
fsmd_microphone_tb.vhd (fixed micro_data)



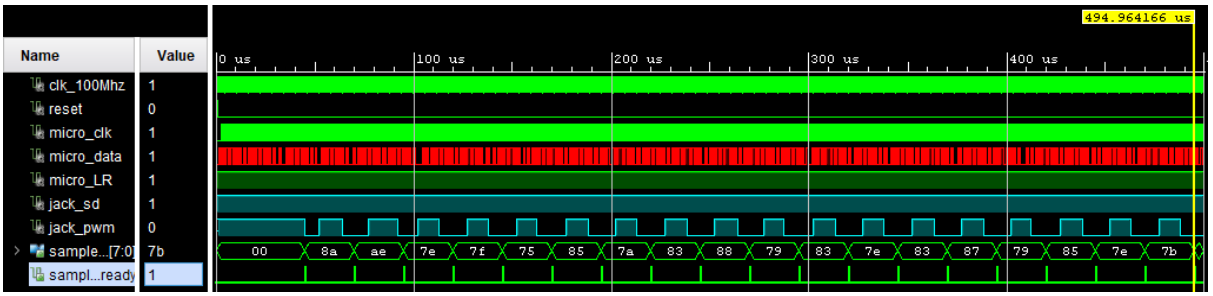
fsmd_microphone_tb.vhd (pseudo-random micro_data)



pwm_tb.vhd

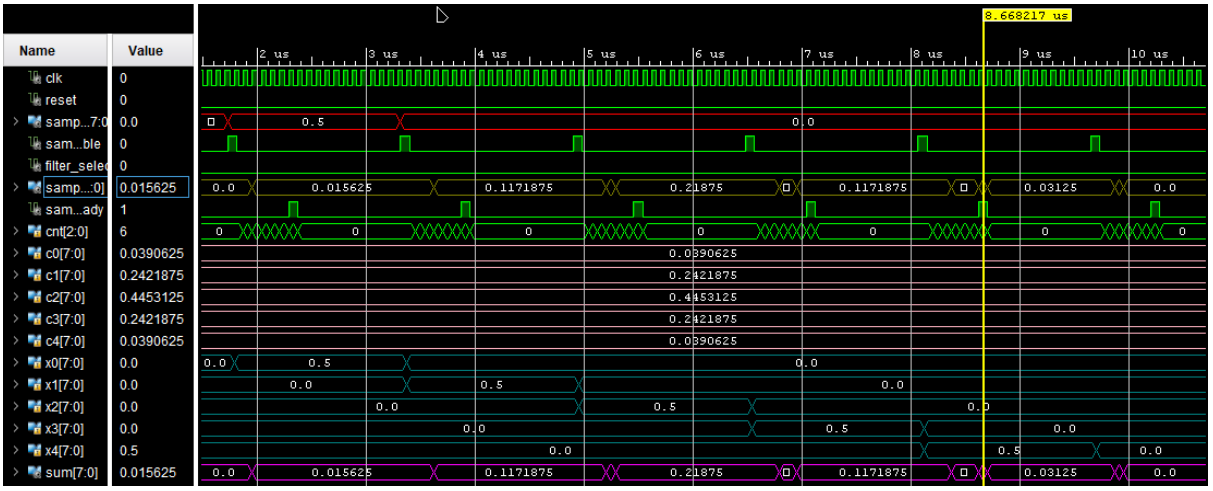


audio_interface_tb.vhd

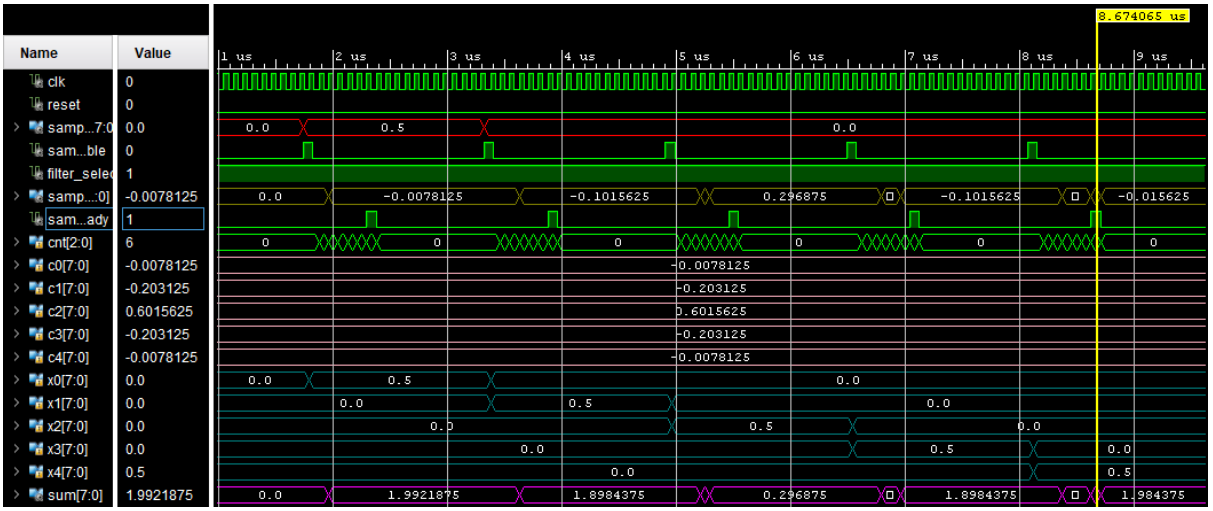


controlador_tb.vhd

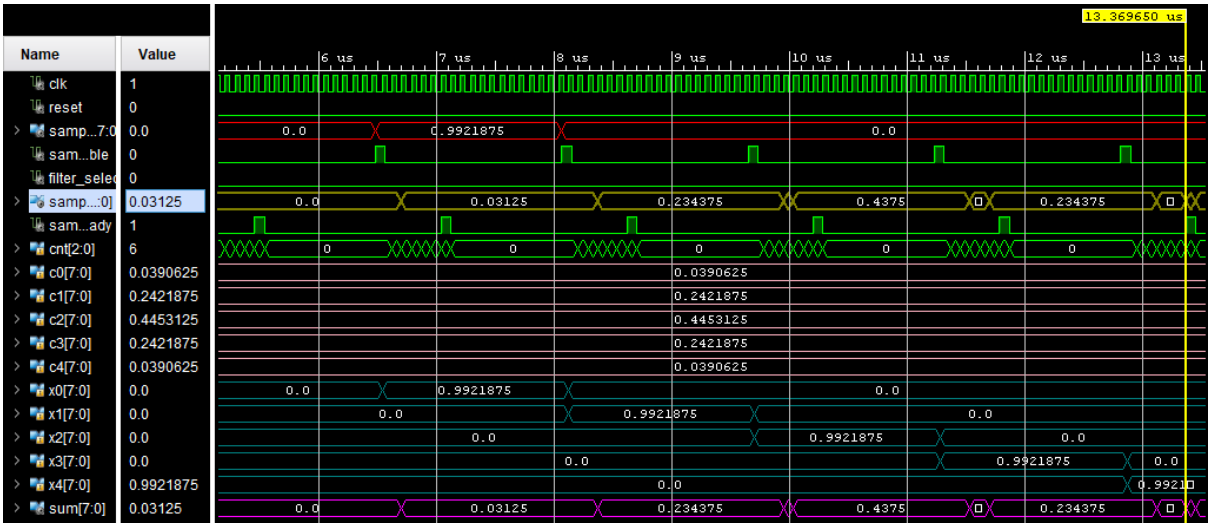
Testbenches ——— Tareas 2.5, 2.9, 2.10



lp → fir_filter_tb.vhd (entrada +0.5)

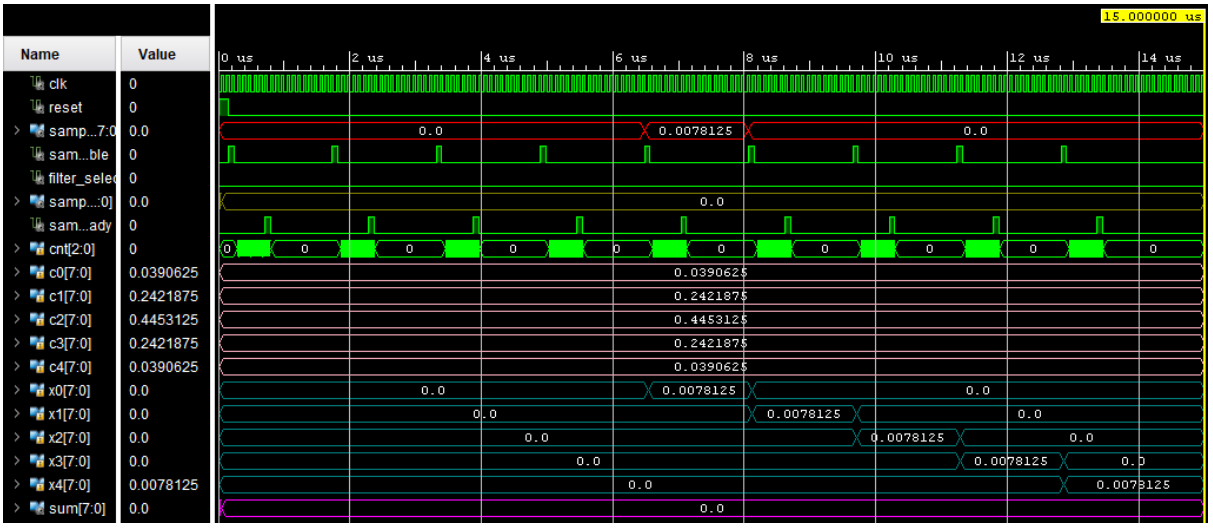


hp → fir_filter_tb.vhd (entrada +0.5)



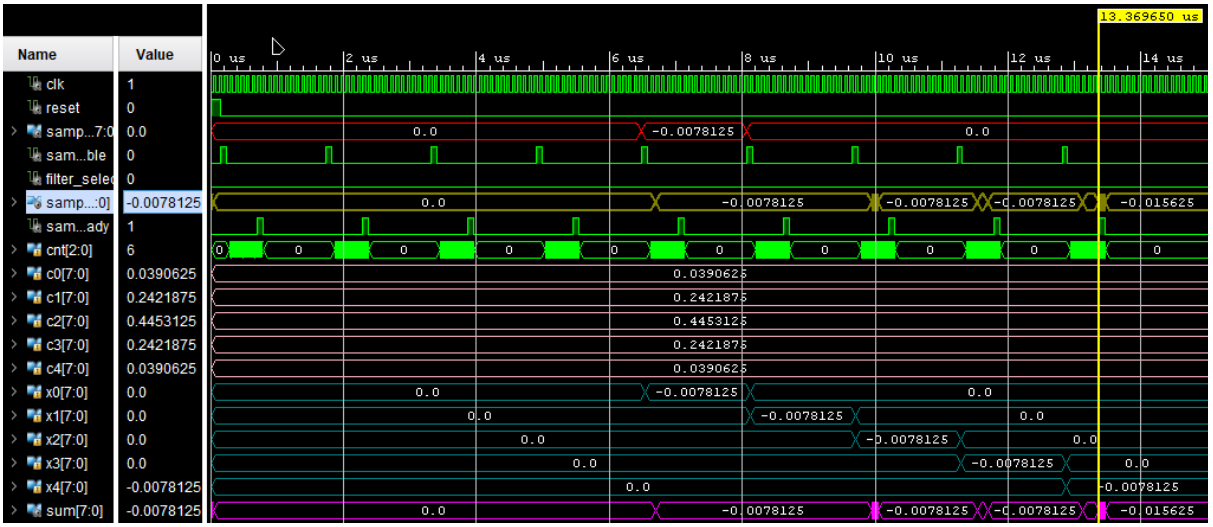
lp → fir_filter_tb.vhd (0, 0, 0, 0, 01111111, 0, 0, 0, 0)

hp → fir_filter_tb.vhd (0, 0, 0, 0, 01111111, 0, 0, 0, 0)

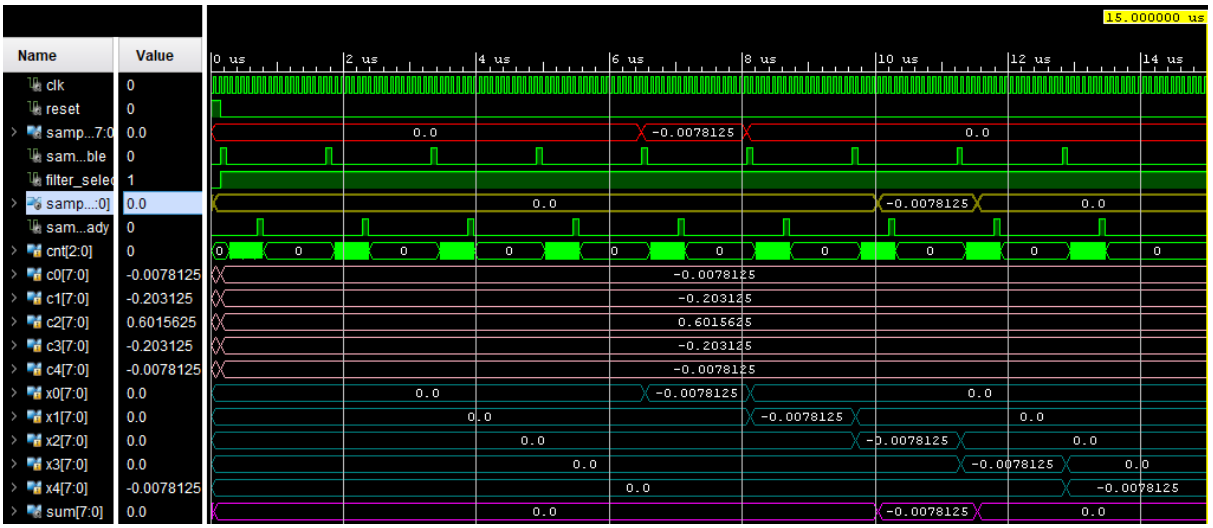


lp → fir_filter_tb.vhd (0, 0, 0, 0, 00000001, 0, 0, 0, 0)

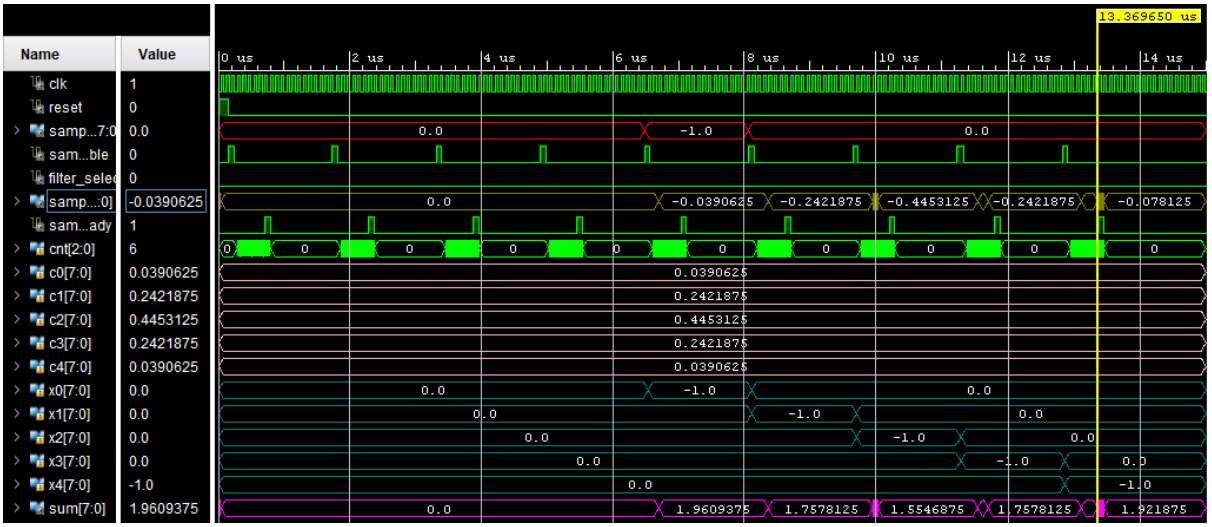
hp → fir_filter_tb.vhd (0, 0, 0, 0, 00000001, 0, 0, 0, 0)



lp → fir_filter_tb.vhd (0, 0, 0, 0, 11111111, 0, 0, 0, 0)

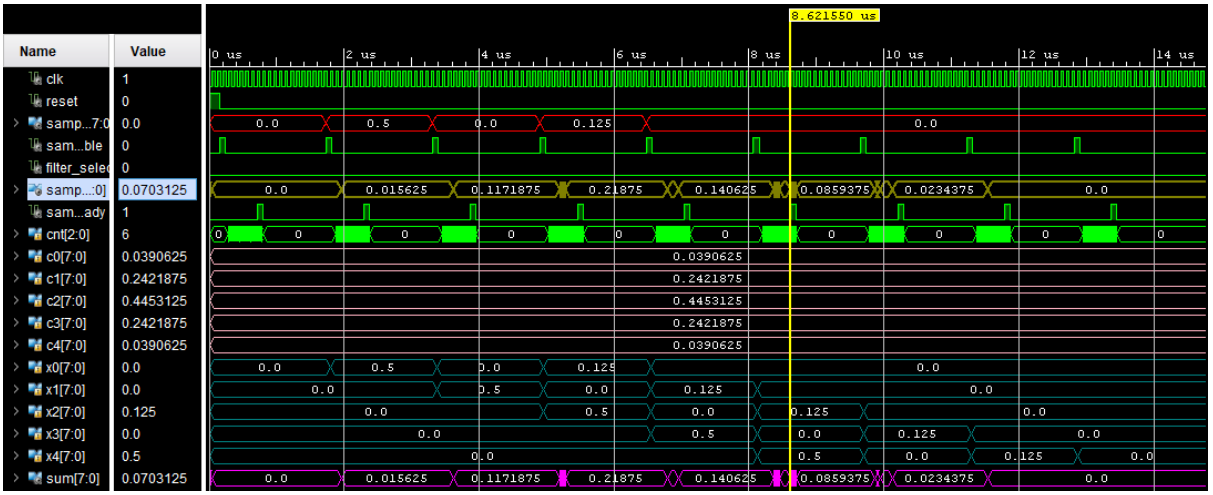


hp → fir_filter_tb.vhd (0, 0, 0, 0, 11111111, 0, 0, 0, 0)

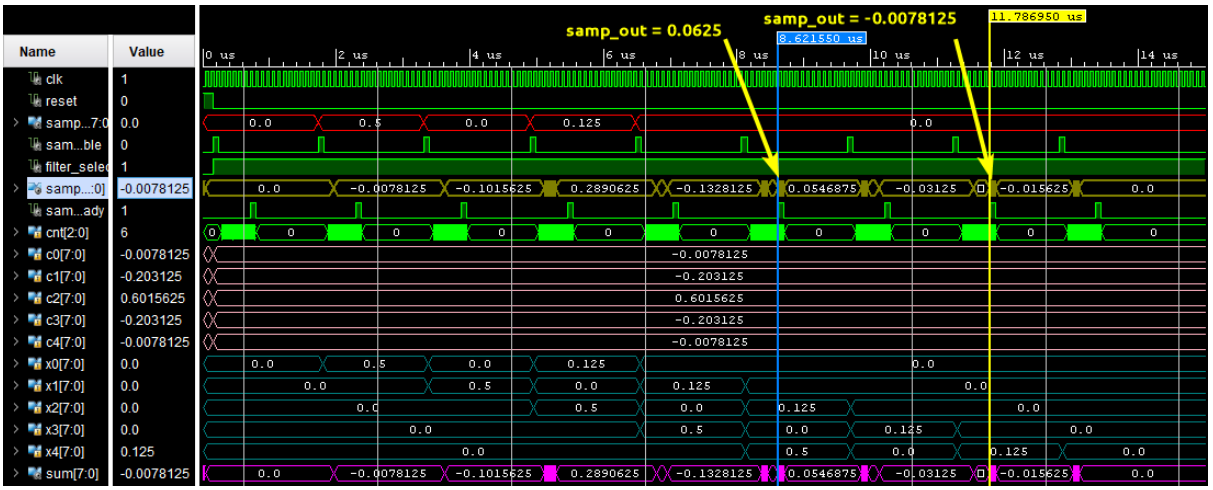


lp → fir_filter_tb.vhd (0, 0, 0, 0, 10000000, 0, 0, 0, 0)

hp → fir_filter_tb.vhd (0, 0, 0, 0, 10000000, 0, 0, 0, 0)



lp → fir_filter_tb.vhd (0, 0.5, 0, 0.125, 0, 0, 0, 0, …)



hp → fir_filter_tb.vhd (0, 0.5, 0, 0.125, 0, 0, 0, 0, …)

Se ha tenido en cuenta que que en la representación <1.7>:

Signo \ Magnitud (val. abs)	Mínima	Máxima
Positivos	`00000001 →  0.0078125`	`01111111 →  0.9921875`
Negativos	`11111111 → -0.0078125`	`10000000 → -1.0000000`

Mejoras adicionales

En el proyecto DSED AudioBox se han realizado las siguientes mejoras al sistema principal:

1. Aviso luminoso de memoria RAM casi llena y llena completamente
2. Muestra de tiempo actual de grabación almacenado, y tiempo restante de reproducción en el estado *play*
3. Muestra del estado actual del sistema
4. Control y visualización del volumen de la salida de audio

Avisos de capacidad de la Memoria RAM

Mediante esta mejora, el usuario obtiene un aviso inicial cuando la memoria RAM está al 90% de su capacidad y también cuando se ha llegado al 100% de su capacidad.

Para ello, se necesita actuar según el contenido grabado actualmente en la memoria RAM. Esto es posible realizarlo desde el sistema principal implementado en *main.vhd*. La máquina FSMD de control del sistema almacena en todo momento en un puntero la dirección de memoria donde se ha almacenado la última muestra grabada. Emplearemos este registro y sencillas comparaciones para obtener el resultado del estado de la memoria deseado.

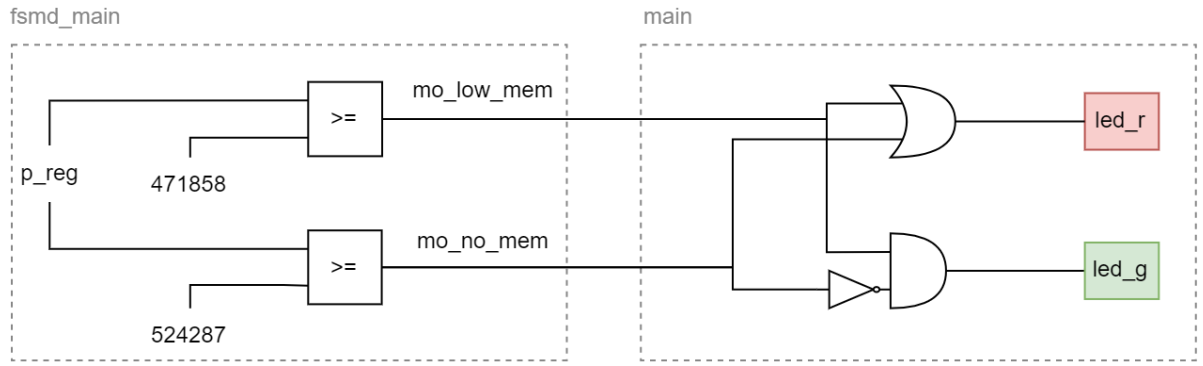
Para comenzar, se parte de que la memoria almacena en su totalidad 524287 palabras. La memoria se escribe secuencialmente desde posiciones bajas a altas. Para obtener el instante cuando la memoria se ha llenado un 90% se calcula la palabra que representa el 90% de las palabras totales en la memoria:

$$\text{floor}(0.9 \cdot 524287) = 471858$$

Es decir, si el registro del puntero contiene la dirección 471858 o superior, estamos en el último 10% de la memoria. De la misma manera, si el registro del puntero contiene la dirección 524287 o superior, habremos acabado la memoria.

Para avisar al usuario se utiliza el LED **LD16 RGB**, emitiendo un color amarillo cuando la memoria está a punto de acabarse y rojo cuando no queda más espacio.

Con esto en cuenta, se añade la siguiente lógica de salida a la FSMD de control del sistema *main_fsmd.vhd* y *main.vhd*:



Tiempo de grabación y reproducción

Para esta mejora se han llevado a cabo muchas adiciones y reestructuraciones importantes del sistema original. Esta mejora permite visualizar en 4 dígitos de la pantalla

de 7 segmentos de la placa, el tiempo de grabación y reproducción en segundos y centésimas de segundo.

Su implementación se ha dividido en varias partes:

- Contabilización del tiempo de grabación y de reproducción en la FSMD de control del sistema
- Traducción de los valores binarios de la cuenta a valores BCD para su manejo sencillo con el display
- Traducción de los dígitos individuales de BCD a entradas al display de 7 segmentos.

Los dos últimos apartados se tratarán en un apartado específico que habla del trabajo con el display. Este apartado se centra en la contabilización del tiempo de grabación y de reproducción en FSMD Main.

Para la contabilización del tiempo se ha creado un proceso nuevo llamado *time* que gobierna el funcionamiento de unos contadores que irán contabilizando el tiempo transcurrido.

En primer lugar, es necesaria la creación de un contador de segundos y de centésimas de segundo para almacenar por un lado el tiempo de grabación almacenado y el tiempo actual de reproducción. Para mejorar la utilización de área y explorar las capacidades del *IP Catalog* de Xilinx visto en clase, se ha preferido hacer uso de *Binary Counters* proporcionados por el fabricante y emplear el uso de slices DSP48.

Además, la frecuencia de operación de este proceso será la misma que la de la FSMD Main, 12 MHz. Para que hayan transcurrido 0.01 s, en el proceso habrán transcurrido 120000 ciclos, así podremos saber cuando contabilizar una centésima de segundo.

Entonces, se han creado dos IP de contadores: un componente contador up/down de módulo 99 y otro up/down de módulo 120000. Ambos contadores cuentan con las siguientes características habilitadas:

- Ports enabled
    - CLK: Clock
    - SCLR: Synchronous Clear
    - CE: Clock Enable
    - LOAD: Load Active
    - UP: Up Active
    - L: Load In Port
    - Q: Count Out Port
- Synthesis
    - Fabric: DSP48
    - SCLR override CE
    - Forward latency: 1
    - Count limit: none
    - Count direction: UPDOWN

El contador de módulo 99 tiene un ancho de puertos de 7 bits y el de 120000 de 16 bits. Se instancian 4 contadores mod 99 (cseg y seg para reproducción y grabación) y un contador mod 120000. Al haberse configurado como up/down, el contador no puede limitar automáticamente el conteo, por lo que se realizará externamente por el proceso controlador poniendo el contador a cero con SCLR o LOAD como sea conveniente.

Los contadores son free-running, lo que quiere decir que siempre que el contador tenga CE activo, contará. A los relojes se les suministra el mismo reloj de 12 MHz del sistema.

Se crean los registros correspondientes para almacenar las señales SCLR, CE, LOAD, UP, L, Q y permitir que el sistema continúe operando síncronamente sin situaciones de metaestabilidad entre señales. En total se han empleado 30 registros para su control.

El proceso opera en base al siguiente pseudocódigo:

```
loop: registers <= ‘0’;
        all counters count up <= '1';
            switch state:
                  case idle:
                        if going into play state:
                              load into play_sec from rec_sec;
                              load into play_csec from rec_csec;
                              do load play counters;
                  case clr:
                        clear all counters;
                  case rec:
                        cycles_count_clock_enable <= '1';
                        if cycles_count is 120000:
                              clear cycles_count;
                              rec_cseg_count_clock_enable <= '1';
                              if rec_cseg_count is 99;
                                    clear rec_cseg_count;
                                    rec_seg_count_clock_enable <= '1';
                  case play:
                        cycles_count_clock_enable <= '1';
                        all counters count up <= '0';
                        if cycles_count is 0:
                              load into cycles_count from 120000;
                              do load cycles_count;
                              play_cseg_count_clock_enable <= '1';
                              if play_cseg_count is 0;
                                    load into play_cseg_count from 99;
                                    do load play_cseg_count;
                                    play_seg_count_clock_enable <= '1';
```

La FSMD entonces debe habilitar dos salidas que son los segundos y las centésimas de segundos que debe mostrar la pantalla. Para ello, dependiendo del estado actual, se multiplexan así:

```
with st_reg select mo_seg <= qp_seg_reg when play,
                             qr_seg_reg when others;

with st_reg select mo_cseg <= qp_cseg_reg when play,
                              qr_cseg_reg when others;
```

Estas señales son las que toma el controlador del display de 7 segmentos para mostrar dichas cuentas.

Muestra del estado actual del sistema

Implementar esta funcionalidad es sencilla, simplemente el módulo FSMD Main habilita una salida con la codificación binaria del estado actual, para ser utilizada por el controlador del display de 7 segmentos:

```
with st_reg select mo_st <= "000" when idle,
                            "001" when rec,
                            "010" when play,
                            "011" when clr,
                            "111" when others;
```

El funcionamiento del display de 7 segmentos con esta mejora se detalla en el último apartado.

Control de volumen en la salida de audio

Esta mejora se ha implementado de forma independiente al funcionamiento del procesamiento de audio grabado o del estado actual de la FSMD Main. La mejora ataca a distintos niveles:

- Modificación de la salida en el módulo PWM a razón de una nueva entrada *factor*
- Actuación sobre el control de volumen y decodificación del nivel y factor de volumen en un nuevo módulo *vol_ctl*
- Habilitación de salida de información sobre el nivel de volumen actual a través de *audio_interface*
- Muestra en pantalla del nivel volumen actual

El funcionamiento de muestra en pantalla quedará cubierta en el último apartado.

En primer lugar, el módulo PWM deberá aceptar una nueva entrada *factor* mediante la cual se modificará el ancho de los pulsos modulados **independientemente** al ancho ya controlado por el propio modulador.

Para ello se acepta un valor codificado en coma fija sin signo <4.5>, esta representación ha sido escogida por el rango de valores de *factor* a tratar (0-8) y por la precisión deseada.

El proceso de modulación PWM con la mejora es el siguiente:

- Se realiza la cuenta hasta 299 con el circuito adaptado del Dr. P. Chu como se haría originalmente para emitir el request de sample
- Al emitir la señal request y obtener la muestra entrante, se obtiene paralelamente una conversión de la muestra a *unsigned* (<8.0>) y este valor se multiplica por el factor (<4.5>), que resulta en una representación unsigned <12.5> en coma fija
- Para evitar un valor excesivo de volumen por overflow, se limita el volumen máximo a 255 en <7.5>, si el valor de volumen está dentro del margen aceptable se obtiene de su valor convertido de la operación anterior en los bits superiores (<8.0>)
- Finalmente para la modulación, se genera un ‘1’ PWM si el contador cuenta por debajo del valor del volumen obtenido en el paso anterior o no hay sample de entrada (o hay reset), de otra manera (cuenta por encima del volumen requerido y sample de entrada presente) se genera el ‘0’ PWM

El factor que se le proporciona al modulador PWM proviene del módulo de control del volumen, que responde a la acción en los botones BTNU (vol +) BTND (vol -), comenzando por un factor de volumen de 1 (correspondiente al nivel 10).

Este módulo está compuesto por una serie de registros con lógica del estado siguiente y lógica de salida que convierte el nivel controlado por los botones (aumentando o disminuyendo el registro que contiene el nivel de volumen) y saca el factor correspondiente para que sea utilizado por el módulo PWM.

Para convertir el valor de nivel al factor en <4.5> se emplea la fórmula proporcionada por el enunciado de la práctica como mejora sugerida. Como tenemos un número restringido de niveles que tratar, se ha optado por convertir manualmente los valores reales obtenidos de la función a los valores en la representación final y almacenarlos en una pequeña memoria ROM en la lógica de salida del módulo. La tabla siguiente contiene las equivalencias calculadas:

Level	Real	Unsigned <4.5>
0	0,000000	000000000
1	0,035802	000000001
2	0,079296	000000010
3	0,132132	000000100
4	0,196318	000000110
5	0,274292	000001000
6	0,369016	000001011
7	0,484088	000001111
8	0,623879	000010011
9	0,793700	000011001
10	1,000000	000100000
11	1,250616	000101000
12	1,555069	000110001
13	1,924922	000111101
14	2,374224	001001011
15	2,920043	001011101
16	3,583112	001110010
17	4,388617	010001100
18	5,367156	010101011
19	6,555899	011010001
20	8,000000	100000000

La pequeña memoria ROM asigna un valor de nivel codificado en binario de 9 bits a un número sin signo del formato requerido.

Finalmente el módulo genera una salida del nivel actualmente usado en una salida de 5 bits que exporta a *audio_interface* y ésta al módulo *main* para poder ser conectado al módulo del display.

Las entradas de los botones también se hacen disponibles a través de *audio_interface* para poder ser enrutadas desde *main* y consiguientemente desde la placa.

En las pruebas de funcionamiento se ha visto como si se incrementa el volumen a partir del nivel 14 o 15, no se escucha nada. Creemos que esto se debe a que el pulso del PWM está siendo ensanchado tanto que en el periodo del PWM no cabe.

Display de 7 segmentos

Este apartado detalla como todas las mejoras muestran en la pantalla de 7 segmentos información relevante para el usuario. Se van a crear dos módulos para la interacción con el display: *matrix_driver* y *matrix_blinker*.

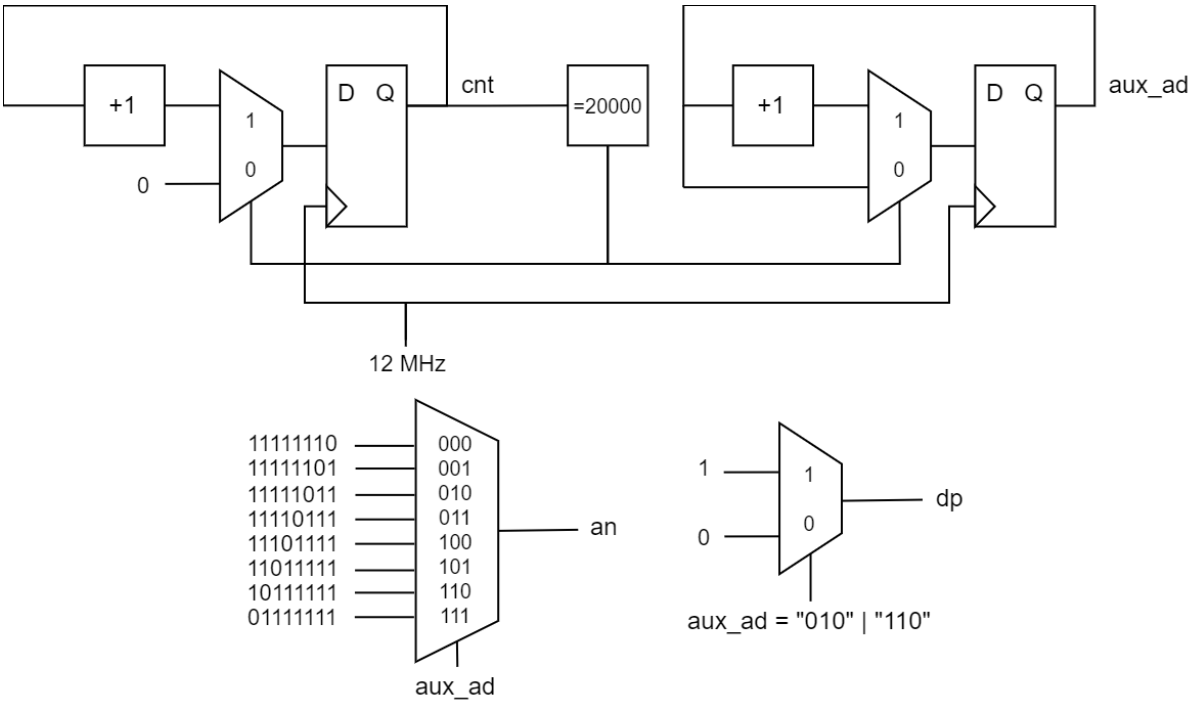
Reuniendo todas las salidas de los distintos módulos de las mejoras tenemos:

- cseg: centésimas de segundo (de rec o play) → 7 bits
- seg: segundos (de rec o play) → 7 bits
- st: estado actual: → codificación del estado en 3 bits
- level: nivel de volumen → 5 bits

Para operar con la matriz de 7 segmentos, cabe recordar que su funcionamiento es de modo ánodo común y que en un determinado instante de tiempo sólo se puede controlar uno de los 8 displays de 7 segmentos que forman la matriz. Para ello será necesario refrescar cada dígito y dependiendo del dígito que se controle enviar a los diodos de la matriz de 7 segmentos el conjunto de segmentos que queremos representar (número o letra), el módulo *matrix_blinker* se encarga de esto último.

matrix_blinker

Este módulo se encarga de refrescar cada dígito a una frecuencia de 75 Hz, (pasa a controlar el siguiente dígito a 600 Hz). Este módulo se compone de un contador y de un decodificador 3 a 8 como muestra el siguiente esquemático:



matrix_driver

Este módulo se encarga de recoger la señal de an para los ánodos de la matriz y decidir según el display que se está pintando cuál es el dígito que se debe mandar

Para ello se utilizan los siguientes elementos:

- Conversor de binario (7 bits) a dos dígitos BCD (8 bits)
- Conversor de un dígito BCD (4 bits) a su representación en 7 segmentos
- Decodificador del estado actual de la FSMD Main a una representación en dos dígitos de la matriz de 7 segmentos
- Un multiplexor 8x7

La implementación de cada uno de los bloques se puede revisar en el código fuente. Cabe mencionar que los conversores de formato Binario ↔ BCD ↔ 7seg y el decodificador de estado están implementados con pequeñas memorias ROM.

La interconexión entre los componentes se puede ver en la página final adjunta