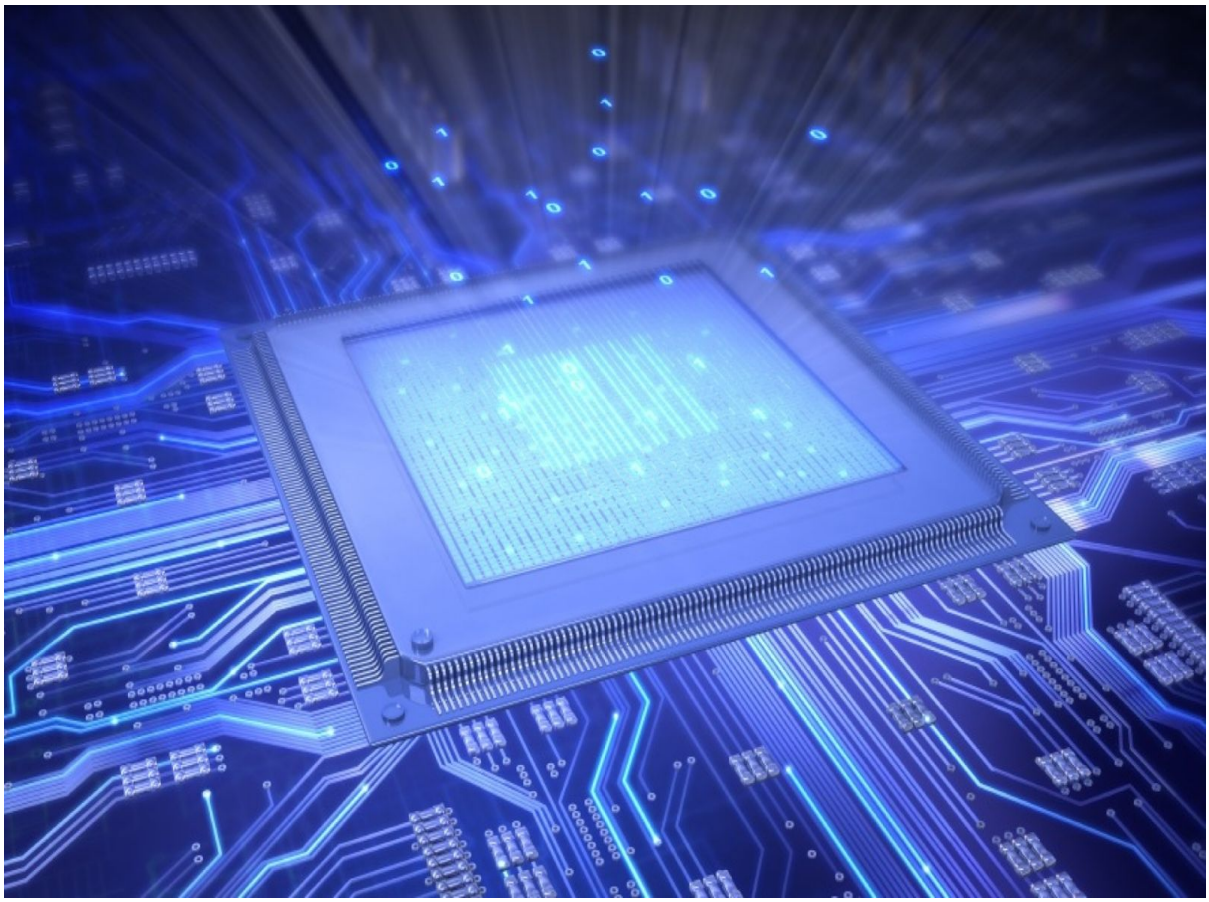
Modelado y Síntesis de Sistemas Electrónico Digitales

Apartado 2. Diseño del controlador del códec



Realizada por:
Guillermo Alba Sánchez
Adrián García-Vera de Lope

Índice

Apartado 2. Diseño del controlador del códec

Introducción 2

Comprobando la funcionalidad del controlador del puerto paralelo 2

Modelar la entidad codec_controller 2

1. Generación de la señal de reset del CODEC 3
2. Generación de la señal de SYNC del CODEC 5
3. Memoria (frecuencia a la que se lee de la misma) 7
4. Trama de salida (sdata_out) 8
5. Slot 0 (Fase de control) 8
6. Slots 1 y 2 (Dirección Registro y Datos de Control) 9
7. Slots 3 y 4 (Datos PCM canales Izquierdo / Derecho) 11
8. Visualización canales en MatLab 12

Simulación funcional de la entidad codec_controller 15

Simulación temporal de la entidad codec_controller 16

Informe sobre los recursos utilizados 17

Descarga en placa del controlador del puerto paralelo 17

Introducción

Para este apartado se nos proporciona en clase una completa explicación del diseño y el funcionamiento del codec a modelar. El objetivo es conseguir controlar el codec de audio LM4550, el cual se va a utilizar para obtener una señal de audio, cuyos valores de frecuencia y amplitud (volumen) se pueden modificar, al igual que su transmisión por el canal (izquierdo o derecho).

En el anterior apartado "*Apartado 1. Diseño del controlador del puerto paralelo (EPP)*" modelamos la entidad **epp_controller**. En este apartado vamos a modelar la entidad **codec_controller** de nuestro "top_system" (Figura 1).

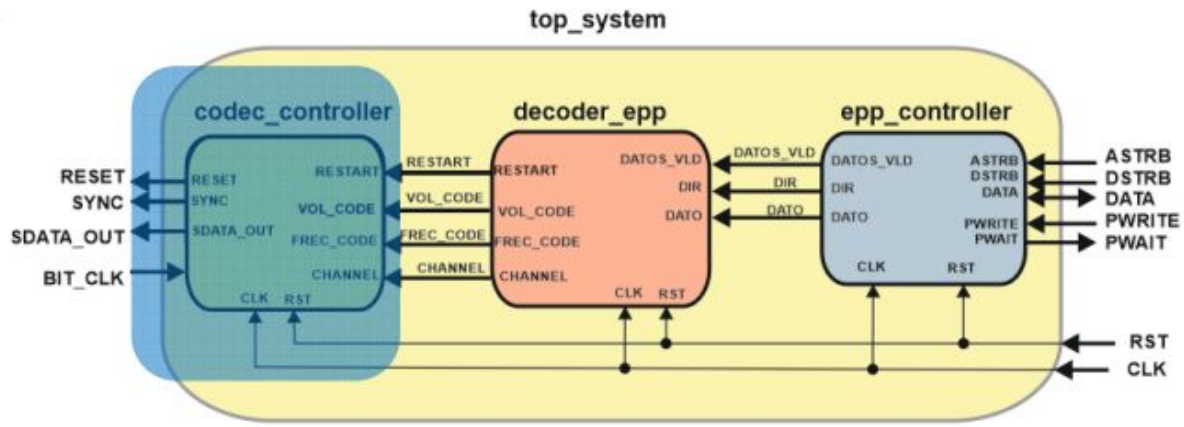


Figura 1. top_system

La frecuencia y el canal permanecerán fijos hasta que se envíe un nuevo valor. El sistema también dispone de la posibilidad de reiniciar el codec realizando un reset del mismo que comentaremos en profundidad más adelante.

Para el modelado de este sistema hemos usado el anexo del codec de audio LM4550 proporcionada en los archivos de la práctica libre.

Comprobando la funcionalidad del controlador del puerto paralelo

Modelar la entidad *codec_controller*

Para la codificación del modelo en VHDL de esta entidad se nos han proporcionado diversos archivos como el *codec_controller.vhd* para modelar la entidad **codec_controller**, y el archivo *seno_4kX20.vhd* que genera una señal sinusoidal usando una memoria ROM, la cual contiene los datos correspondientes a un periodo (hemos tenido que instanciarla en la entidad **codec_controller** como un componente).

Hemos creado diversos procesos para cada parte que hemos necesitado modelar. Estas contarán con una explicación lo más clara posible junto con alguna captura de nuestra simulación para comprobar que se cumple lo que deseamos hacer.

1. Generación de la señal de *reset* del CODEC

El codec LM4550 soporta varios tipos de *reset*, sin embargo, en esta práctica se utiliza el COLD RESET, que consiste en inicializar el codec cuando en el pin *reset* hay un nivel bajo y este tenga una duración superior a 1µs (figura 2).

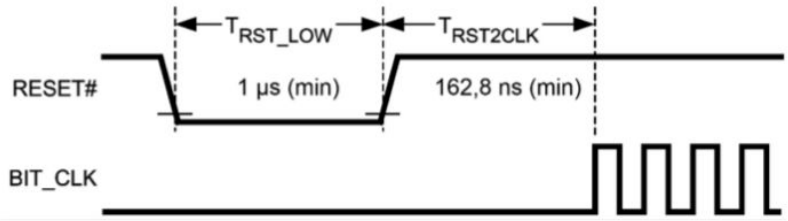


Figura 2. Temporización del reset del CODEC.

Lo primero que necesitamos es calcular cuántos pulsos de reloj equivalen a 1µs. Como sabemos que cada pulso de reloj son 10ns hacemos el siguiente cálculo:

$$1\mu s \;=\; 1000ns \;;\; \frac{1000ns}{10ns} = 100\; pulsos\; de\; clk$$

No se especifica que el valor tenga que ser exacto, es más, se recomienda que sea mayor por lo que le daremos 10 pulsos de margen, quedando un total de 110 pulsos de clk en los que el reset tiene que estar a nivel bajo.

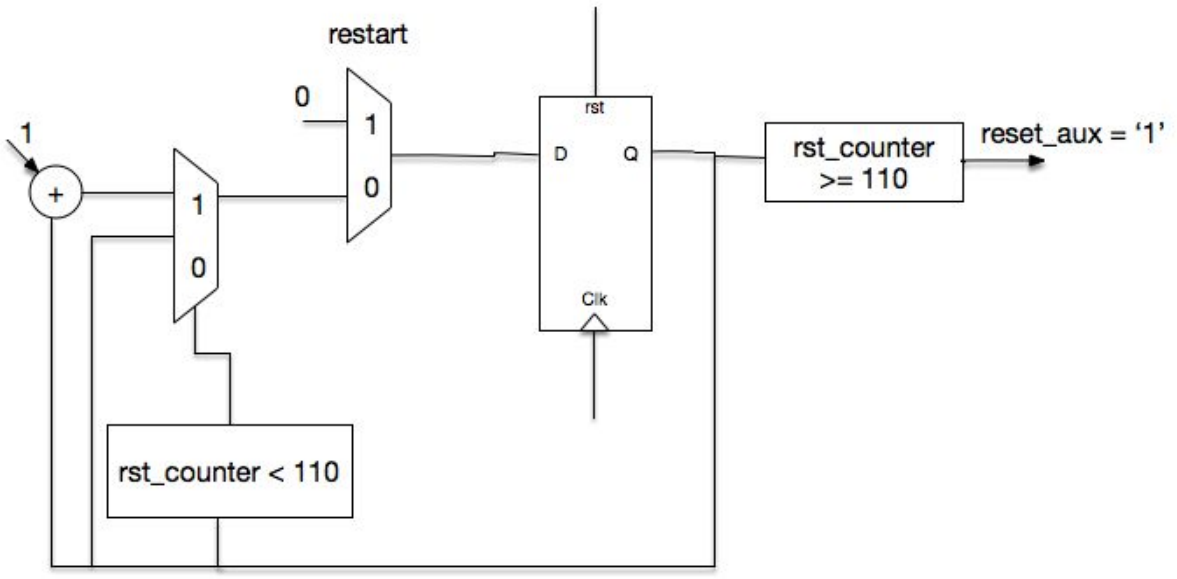


Figura 3. Diagrama reset

```
-- reset
constant CLK_DIV     : integer := 110;
signal   rst_counter : integer range 0 to CLK_DIV;
signal   reset_aux   : std_logic;

================================================

process(clk, rst)
begin
  if rst = '1' then
    rst_counter <= 0;
    reset_aux <= '1';
  elsif clk'event and clk = '1' then
    if restart = '1' then
      rst_counter <= 0;
      reset_aux <= '1';
    else
      if rst_counter < CLK_DIV then
        rst_counter <= rst_counter + 1;
        reset_aux <= '0';
      else
        reset_aux <= '1';
      end if;
    end if;
  end if;
end process ; -- reset_aux

reset <= '1' when reset_aux = '1' else '0';
```

Figura 4. Código para generar la señal reset

Como podemos apreciar en la figura 5, sólo activaríamos el reset en caso de que se cumpla la condición que comentamos anteriormente de 110 pulsos de **clk**, en los cuales el reset estará a nivel bajo, si no se llegaran a cumplir este número de pulsos, se resetearía el contador, empezando el proceso de nuevo. También vemos como la señal de **bit_clk** no se empieza a generar hasta pasados los 162,8 ns.

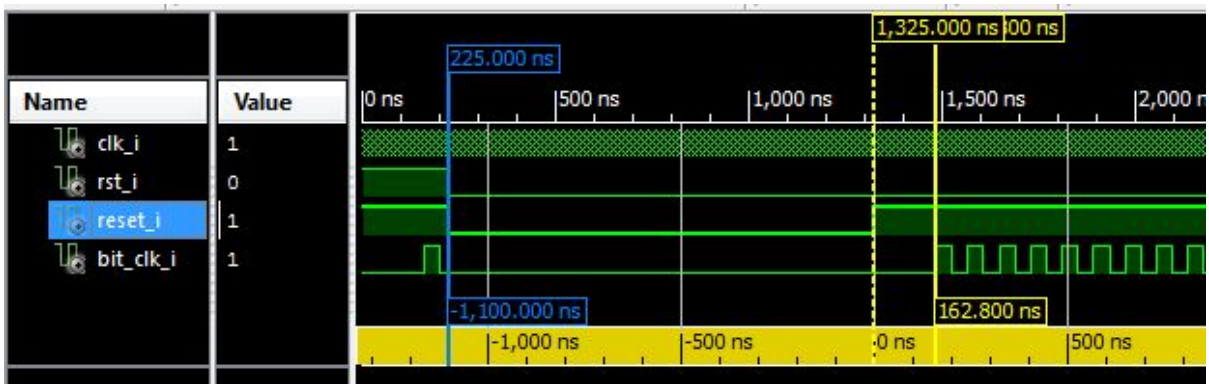


Figura 5. Captura de simulación en el que vemos la temporización de la señal reset

2. Generación de la señal de *SYNC* del CODEC

La señal **SYNC** indica los límites de una trama. Cada trama tiene una duración de 256 periodos de la señal **BIT_CLK** (señal de reloj que sincroniza la entrada y salida de datos del CODEC, aunque en nuestro modelo actúa sólo como señal de entrada).

Cabe destacar que durante los primeros 16 periodos de **BIT_CLK** (desde el 255 al 14) la señal **SYNC** está a nivel alto y no se envía ningún dato de la trama. Durante los siguientes 240 periodos, la señal **SYNC** si está a nivel bajo y es cuando se envía la trama de salida (Figura 6). Esta última fase es llamada “Data Phase”.

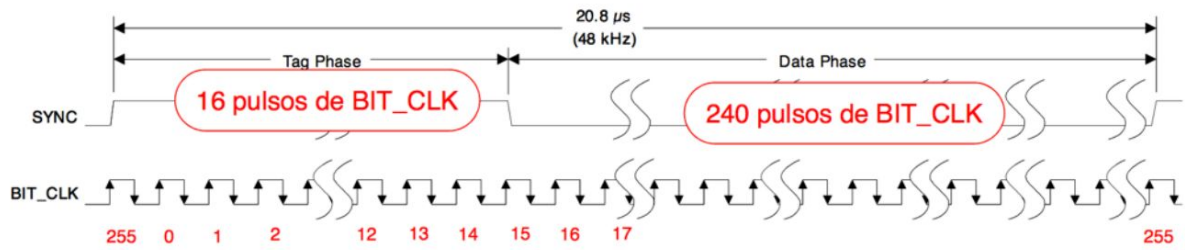


Figura 6. Temporización de las señales bit_clk y sync

Para modelar este proceso hemos seguido el diseño dado por el profesor (figura 7). Una vez que se inicia la trama y se empiezan a contar periodos de **BIT_CLK**, la señal SYNC será ignorada hasta que no se completen los 256 periodos.

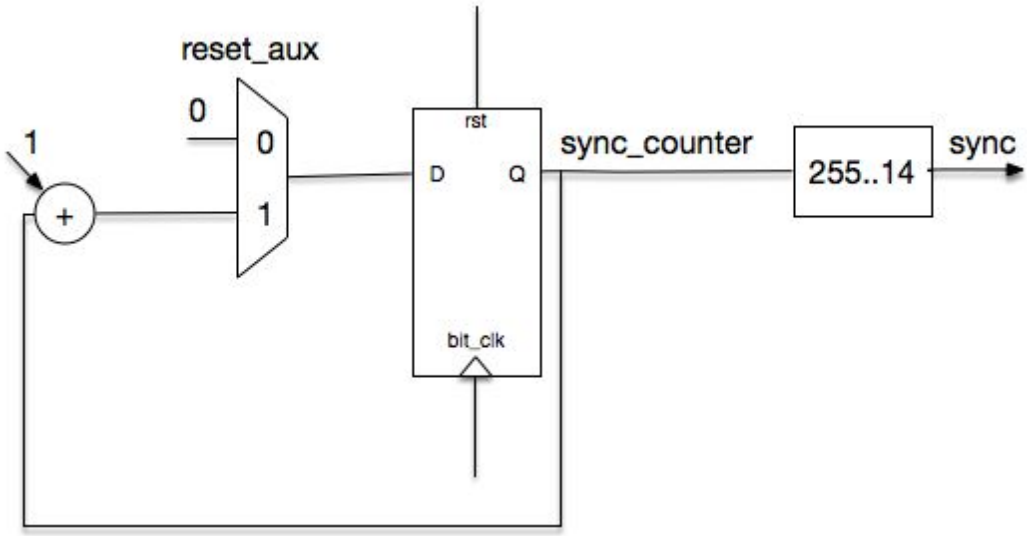


Figura 7. Diagrama sync

```
-- sync
signal sync_counter : unsigned(7 downto 0);

=====================================================

process(bit_clk, rst)
begin
  if rst = '1' then
    sync_counter <= x"FE";
    sync <= '0';
  elsif bit_clk'event and bit_clk = '1' then
    sync_counter <= sync_counter + 1;
    if sync_counter = 254 then
      sync <= '1';
    elsif sync_counter = 14 then
      sync <= '0';
    end if ;
  end if;
end process ; -- sync_counter
```

Figura 8. Código sync

Como podemos apreciar en la Figura 9, la simulación funcional del proceso anterior concuerda con la temporización esperada de la señal **sync** (figura 6).

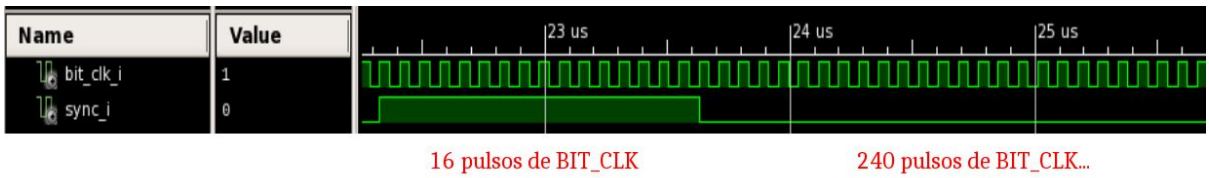


Figura 9. Señales bit_clk y sync

3. Memoria (frecuencia a la que se lee de la misma)

Disponemos de una memoria con 4096 posiciones que alberga una señal sinusoidal, para acceder a la misma y mediante el codec_controller, en base a una frecuencia que se nos da como entrada en este. El sistema es sencillo, necesitaremos un prescaler que generará un **prescaler_ce** cuando el contador, que va sincronizado con clk sea igual que la frecuencia que nos dan como referencia (**frec_code**).

Una vez activado, tendremos un contador cuyo propósito es iterar sobre las 4096 posiciones de la memoria, sincronizado con clk y a la frecuencia que le marca el prescaler.

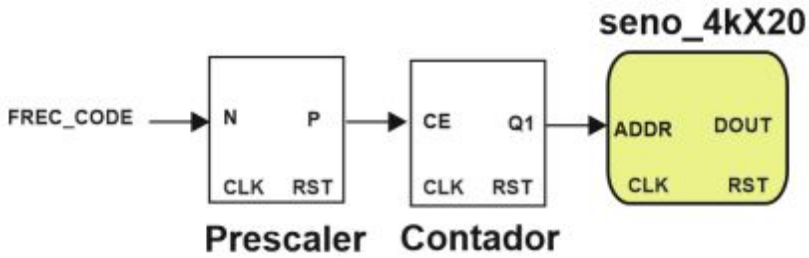


Figura 10. Diagrama lectura memoria

```
-- prescaler
signal prescaler_counter : unsigned(7 downto 0);
signal prescaler_ce      : std_logic;
signal prescaler_aux     : std_logic;

-- counter
signal contador_aux : unsigned(11 downto 0);

-- memoria
signal address : unsigned(11 downto 0);

=================================================

prescaler : process(clk, rst)
begin
  if rst = '1' then
    prescaler_counter <= (others => '0');
  elsif clk'event and clk = '1' then
    prescaler_counter <= prescaler_counter + 1;
    if (prescaler_counter = unsigned(FREC_CODE)) then
      prescaler_aux <= '1';
      prescaler_counter <= (others => '0');
    else
      prescaler_aux <= '0';
    end if ;
  end if;
end process ; -- prescaler
```

```
prescaler_ce <= '1' when prescaler_aux = '1' else '0';


contador : process(clk, rst, prescaler_ce)
begin
  if rst = '1' then
    contador_aux <= (others => '0');
  elsif clk'event and clk = '1' then
    if prescaler_ce = '1' then
      contador_aux <= contador_aux + 1;
    end if;
  end if;
end process ; -- contador

address <= contador_aux;
```

Figura 11. Código lectura memoria

4. Trama de salida (sdata_out)

La trama de datos que se envía al códec está compuesta por un total de 256 bits (explicado en el punto 2 de este apartado), los cuales se empaquetan en 13 slots (Figura 12). Los slots tienen un tamaño de 20 bits cada uno, de los que sólo hemos usado los slots 0 al 4, y que explicaremos en los siguientes apartados.

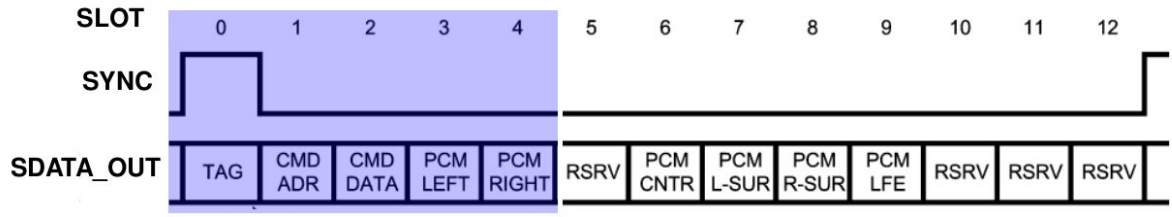


Figura 12. Formato de las tramas de entrada/salida.

5. Slot 0 (Fase de control)

Indica cuáles de los siguientes contienen información, además el primero indica si la trama es válida.

Tabla 4. Función de los bits del slot 0.

Bit	Descripción	Comentario
15	Trama Válida	1: Datos válidos en al menos un slot
14	Dirección de registro de Control	1: Dirección de Control válida en Slot 1
13	Dato de registro de Control	1 : Dato de Control válido en Slot 2
12	Dato DAC izquierdo en Slot 3	1 : Dato de Control válido en Slot 3
11	Dato DAC derecho en Slot 4	1 : Dato de Control válido en Slot 4
10:0	No utilizado	El controlador debe rellenar estos bits con 0's

Figura 13. Bits Slot 0

En base a los datos de la tabla anterior (Figura 13) podemos inicializar este slot de la siguiente manera:

```
constant Tag : std_logic_vector(15 downto 0) := X"F800";
```

6. Slots 1 y 2 (Dirección Registro y Datos de Control)

Contamos con una máquina de estados [DE MILY O MOORE] que nos permite acceder a distintos registros según el estado en que se encuentre, en cada uno de los cuales escribiremos la dirección del registro y los datos de control, estas señales, de 20 bits cada una se llaman **cmd_addr (Slot 1)** y **cmd_data (Slot 2)** respectivamente.

Gracias al **sync_counter** que vimos anteriormente, somos capaces de generar un ce en base a dicha señal cuando su valor sea 222, además cabe destacar que esta máquina de estados está sincronizada con **bit_clk**.

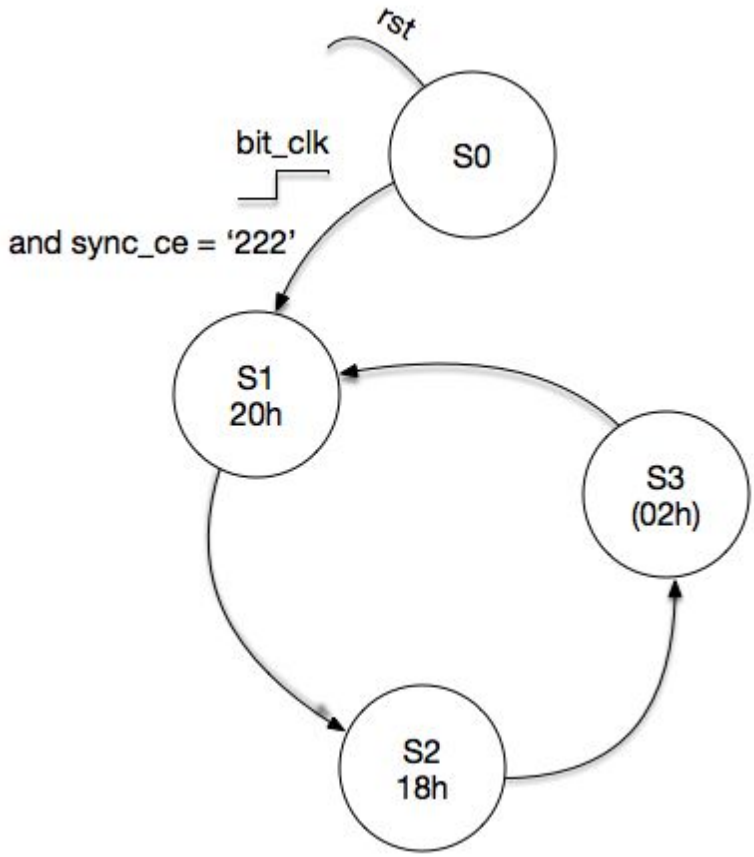


Figura 14. Máquina de estados

En caso de estar en el estado inicial ambas señales tomarán valor 0 (evidentemente son 20 bits, pero ponemos '0' para acortar).

En el estado uno, que corresponde con el registro **20h (Registro de propósito general)**, esto significa que **cmd_addr = x"20000"**, y **cmd_data = x"80000"**, esto es así porque en el anexo se nos indica que el valor que se nos indica es de 8000h, pero hay que añadir 4bits más por la derecha para completar los 20, ya que tal y como se indica en el Slot 2, los bits 3..0 están reservados.

En el estado dos estaremos hablando del registro **18h (PCM Out Volume)** cuya dirección será **cmd_addr = x"18000"**, nos especifican la ganancia de ambos canales, en este caso

debe ser de 0db, por tanto determinamos que los datos de control serán **cmd_data = x”08080”.**

Por último tenemos el registro **02h (Registro de Volumen Maestro),** cuya dirección será **cmd_addr = x”02000”,** y para sacar los datos de control usaremos la señal **vol_code**, señal de entrada en el **codec_controller**, por tanto nos queda que los datos de control serán: **cmd_data <= b"000" & vol_code & b"000" & vol_code & x"0";**

```
-- fsm
type FSM is (S0, S1, S2, S3);
signal std_act, prox_std : FSM;
signal sync_ce : std_logic;

=================================================

states : process(std_act)
begin
  case(std_act) is
    when S0 =>
      prox_std <= S1;
    when S1 =>
      prox_std <= S2;
    when S2 =>
      prox_std <= S3;
    when S3 =>
      prox_std <= S1;
  end case ;
end process ; -- states

change_state : process(bit_clk, rst)
begin
  if rst = '1' then
    std_act <= S0;
  elsif bit_clk'event and bit_clk = '1' then
    if sync_ce = '1' then
      std_act <= prox_std;
    end if ;
  end if ;
end process ; -- change_state
```

Figura 15.1. Código fsm

```
assigns : process(std_act, vol_code)
begin
  case(std_act) is
    when S0 =>
      cmd_addr <= x"00000";
      cmd_data <= x"00000";
    when S1 => -- 20h
      cmd_addr <= x"20000";
      cmd_data <= x"80000";
    when S2 => -- 18h
      cmd_addr <= x"18000";
      cmd_data <= x"08080";
    when S3 => -- 02h
      cmd_addr   <= x"02000";
      cmd_data   <= b"000" & vol_code & b"000" & vol_code & x"0";
  end case ;
end process ; -- assigns
```

Figura 15.2. Código fsm

7. Slots 3 y 4 (Datos PCM canales Izquierdo / Derecho)

En estos slots se almacena la información de cada uno de los canales, en código corresponden con las señales **left_data** y **right_data**, usando el ce que vimos anteriormente proveniente del **sync_counter,** enviaremos **0's** o **data** (Figura 16) dependiendo del valor del canales en el momento de actualizar las señales.

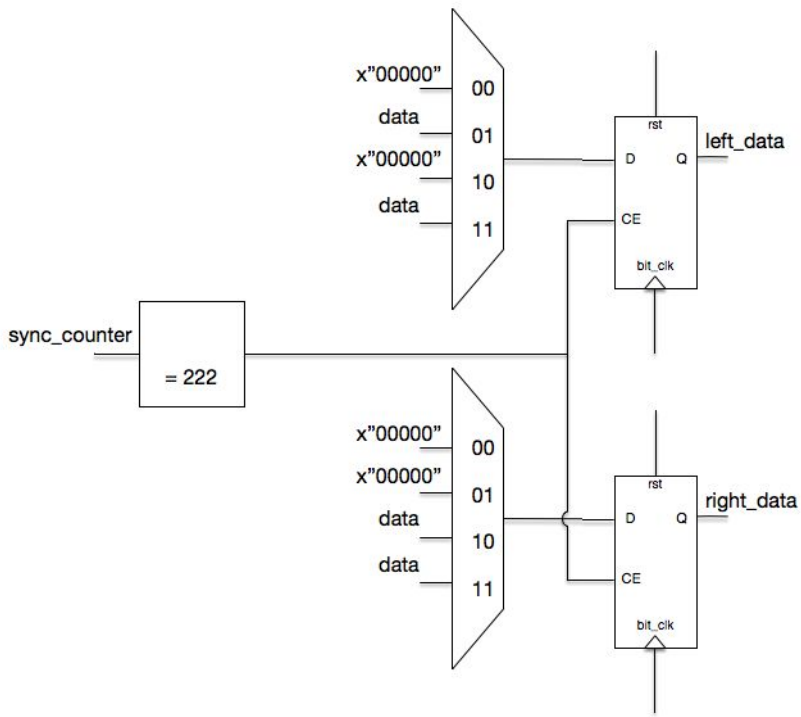


Figura 16. Diagrama left & rigth data

```
channels : process(bit_clk, rst, sync_ce)
begin
  if rst = '1' then
    left_data  <= (others => '0');
    right_data <= (others => '0');
  elsif bit_clk'event and bit_clk = '1' then
    if sync_ce = '1' then
      if channel = "00" then
        left_data  <= x"00000";
        right_data <= x"00000";
      elsif channel = "01" then
        left_data  <= data;
        right_data <= x"00000";
      elsif channel = "10" then
        left_data  <= x"00000";
        right_data <= data;
      else
        left_data  <= data;
        right_data <= data;
      end if ;
    end if ;
  end if ;
end process ; -- channels
```

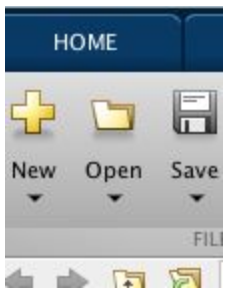
Figura 17. Código left & right data

8. Visualización canales en MatLab

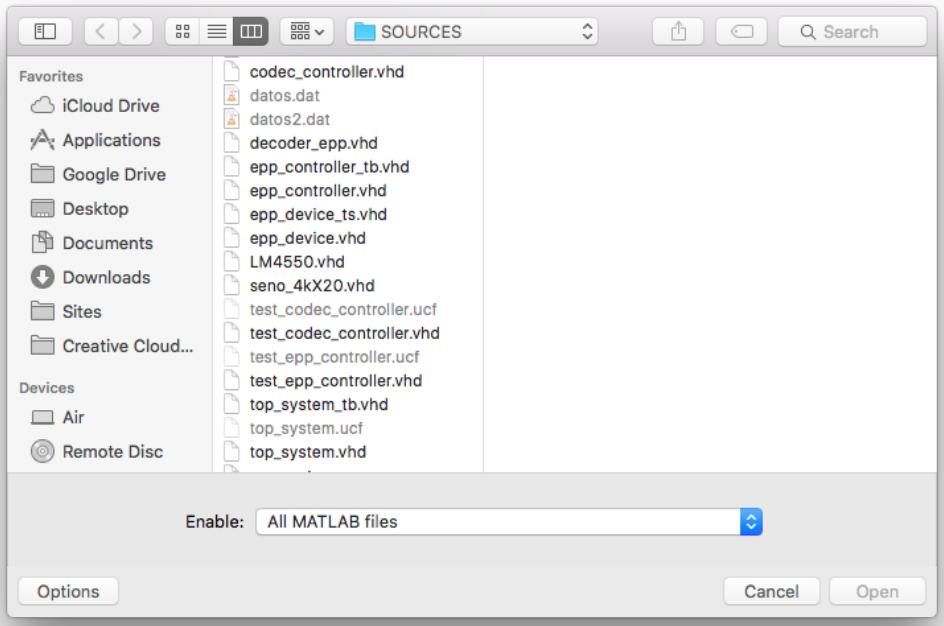
Como se comentó en clase se pueden visualizar las señales de ambos canales, procedentes de los slots 3 y 4, para lo cual necesitaremos el software MatLab.

Los pasos a seguir para obtener dichas señales son:

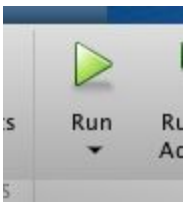
1. Abrir MatLab
2. Pulsamos el botón **Open**



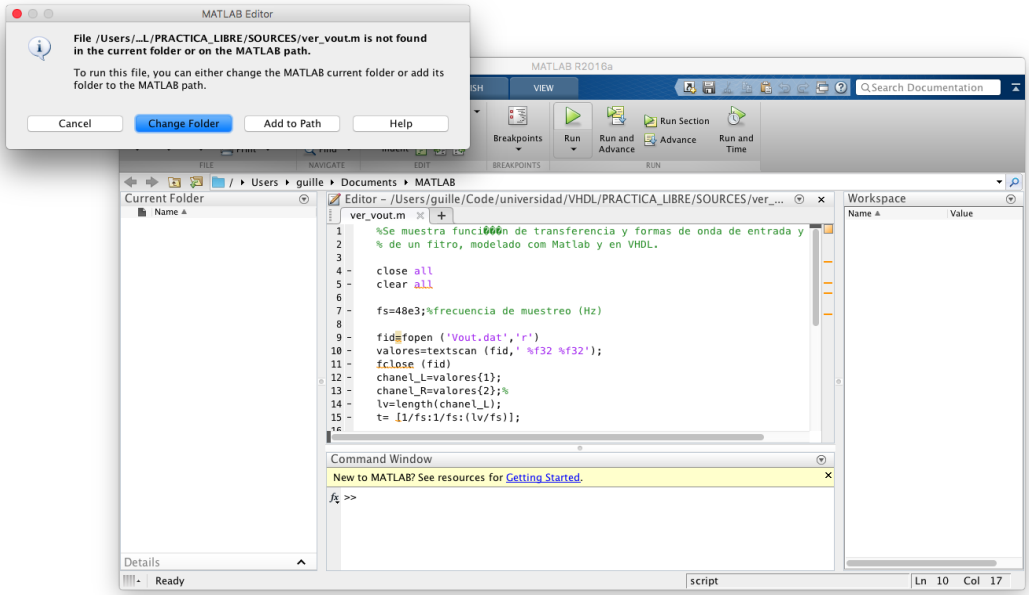
3. Navegamos hasta la ruta donde tenemos el fichero **ver_vout.m**, que es un script de MatLab.



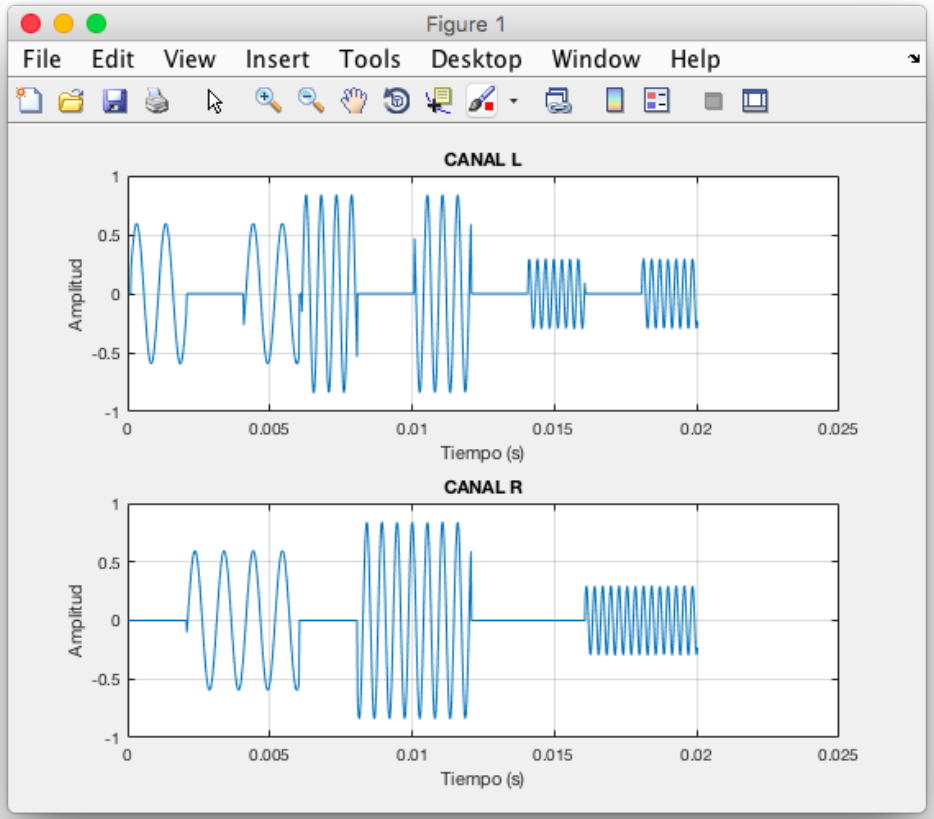
4. Pulsamos el botón **Run**



5. Nos saltará un aviso con varias opciones entre las que se encuentra cambiar de directorio (Change folder), lo que hará esto es cambiar la ruta actual dentro del explorador de MatLab.



6. Output con los datos escritos en **Vout.dat**



Simulación funcional de la entidad *codec_controller*

Para las simulaciones hemos usado los archivos *codec_controller_tb.vhd, seno_4kX20.vhd* y *LM4550.vhd*.

Al hacer la simulación funcional de nuestro modelo estamos comprobando que todo funciona según la funcionalidad deseada. En los apartados anteriores hemos ido comentando partes de la simulación funcional de forma específica para cada parte del modelo, por lo que sólo ponemos

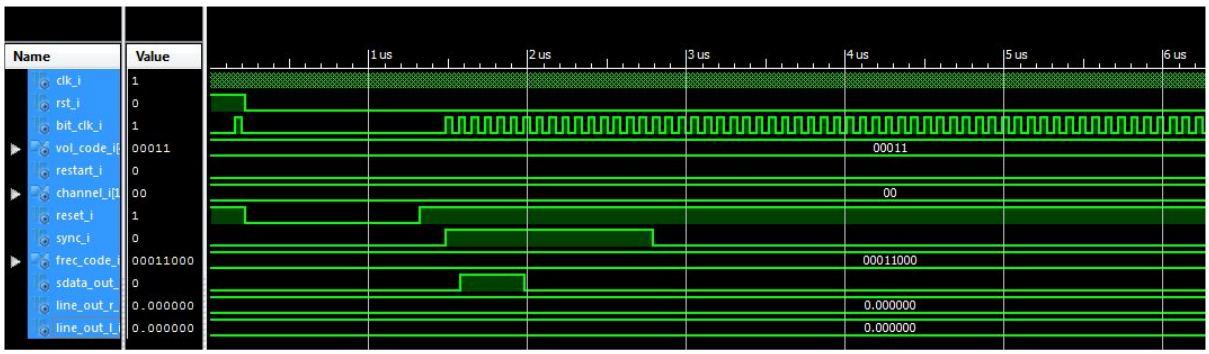


Figura 18. Inicio de la simulación

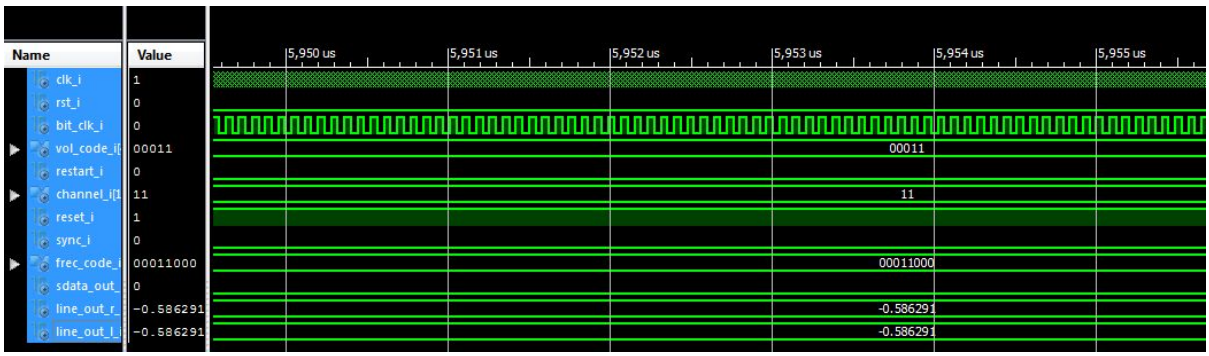


Figura 19. Simulación con channel = 11

Como podemos apreciar en la figura 19 al tener channel dicho valor, ambos canales toman el valor de **data**, que introduce el mismo valor para ambas señales como pudimos ver en el código de la figura 17.

Simulación temporal de la entidad *codec_controller*

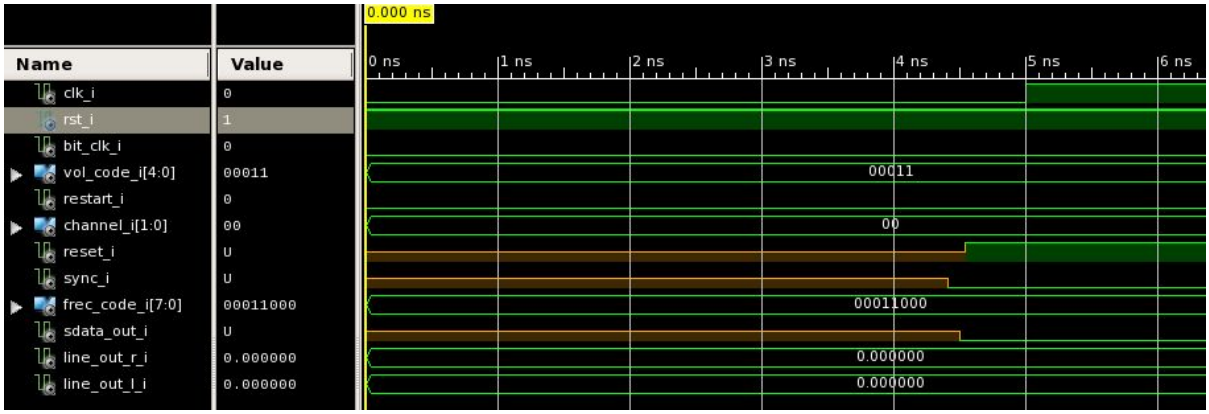


Figura 20. Captura de la simulación temporal con los valores antes de inicializarse.

En la figura 20 podemos ver como las señales reset, sync y sdata_out están sin inicializar hasta pasados los primeros instantes de tiempo esto es normal y proviene de los tiempos de inicialización de los componentes en la placa, al ser simulación temporal.

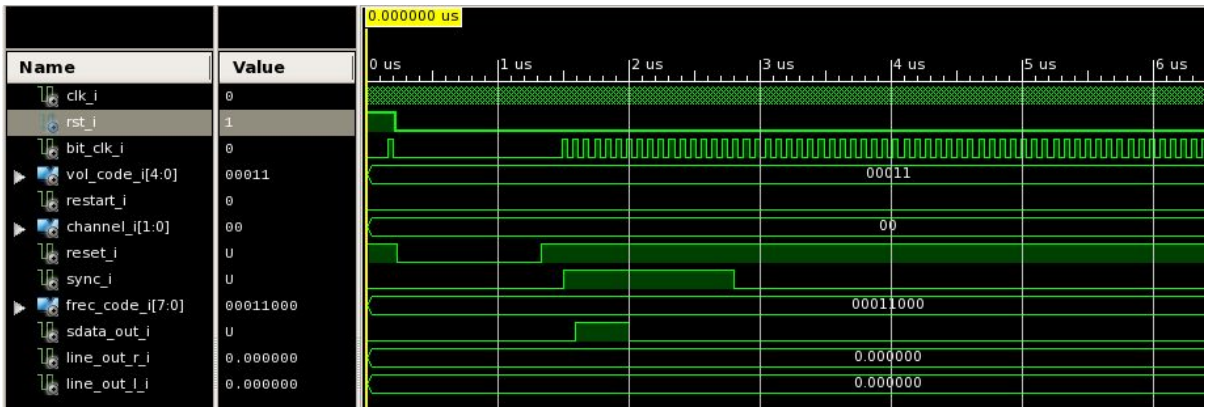


Figura 21. Captura del inicio de la simulación temporal

En la figura 21 podemos apreciar que al tener ser channel = 00, los valores que se meten a los canales izquierdo y derecho son 0’s también.

Informe sobre los recursos utilizados

Es condición imprescindible generar un informe sobre los recursos utilizados cuando se haga uso de la simulación temporal y posterior descarga en placa ya que de esta forma sabremos la cantidad de hardware en uso.

Una alternativa al código de la figura 17 es realizar un case en vez de un if anidado, ya que los if anidados necesitan más recursos hardware. Después de hacer el cambio en el código comprobamos esto con los siguientes informes:

Number of Slices Registers	95 out of 54,576 1%
Number of Slice LUTs	1,151 out of 27,288 4%
Number of occupied Slices	**364** out of 6,822 5%

Tabla de recursos con IF anidados

Number of Slices Registers	95 out of 54,576 1%
Number of Slice LUTs	1,151 out of 27,288 4%
Number of occupied Slices	**355** out of 6,822 5%

Tabla de recursos con CASE

Al ser un sistema relativamente pequeño la diferencia es insignificante, aunque si esto formara parte de un sistema mayor la diferencia sería muy significativa. Confirmamos entonces que la estructura de control “case” es mucho más recomendable que el uso de “if anidados”.

Descarga en placa del controlador del puerto paralelo

Después de haber realizado todas las pruebas pertinentes y comprobar el correcto funcionamiento de nuestro sistema hacemos la descarga en placa en el despacho del profesor Pedro Martín.