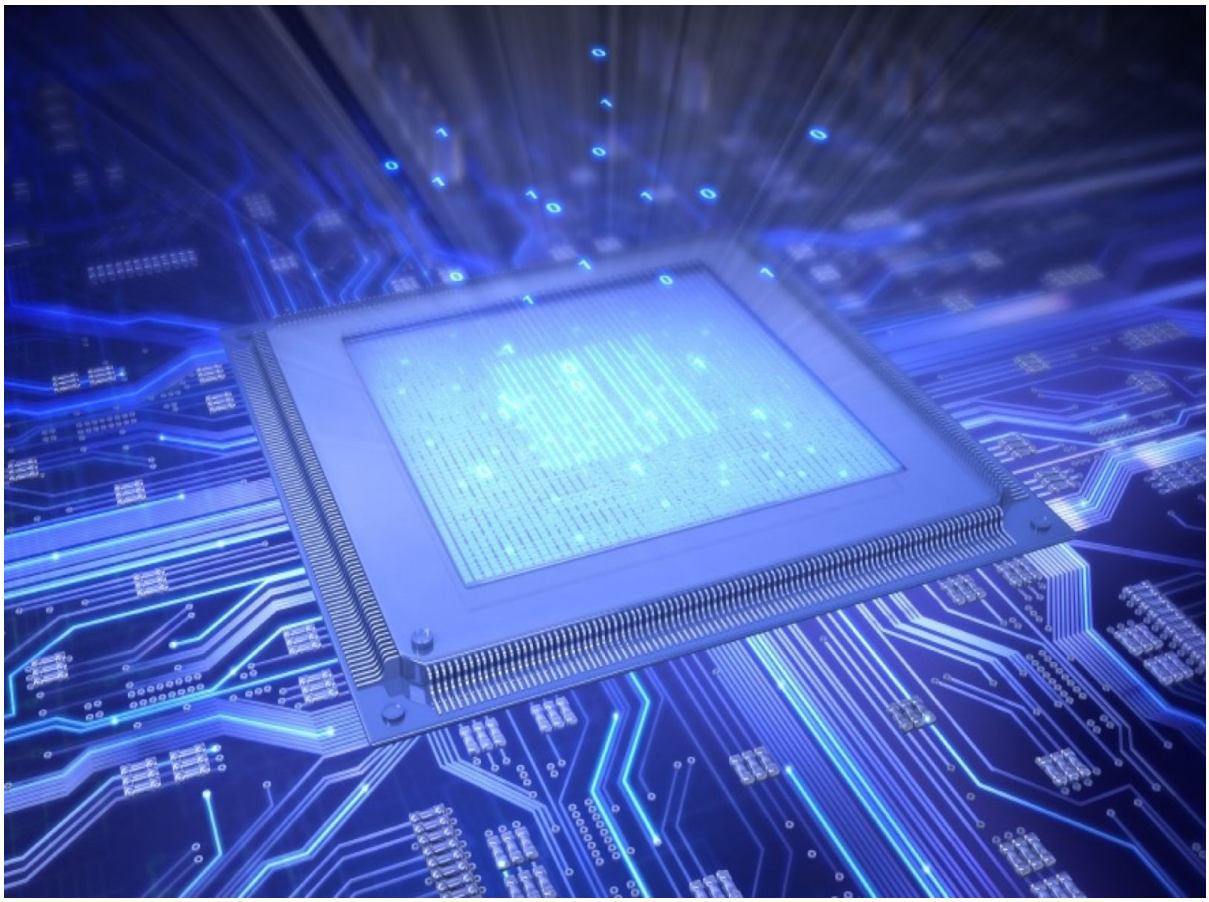
Modelado y Síntesis de Sistemas Electrónico Digitales

Apartado 4. Implementación del diseño completo



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

Índice

Introducción

Este apartado es la parte final de la práctica libre, por lo que el objetivo principal es conseguir la unión de todos los componentes y su implementación final en la placa Atlys de Digilent a través del CODEC de audio LM4550.

En la figura 1 podemos apreciar que la entidad *top_system* se compone de las entidades *codec_controller*, *decoder_epp*, y *epp_controller*, todas ellas modeladas en los apartados anteriores de la práctica libre.

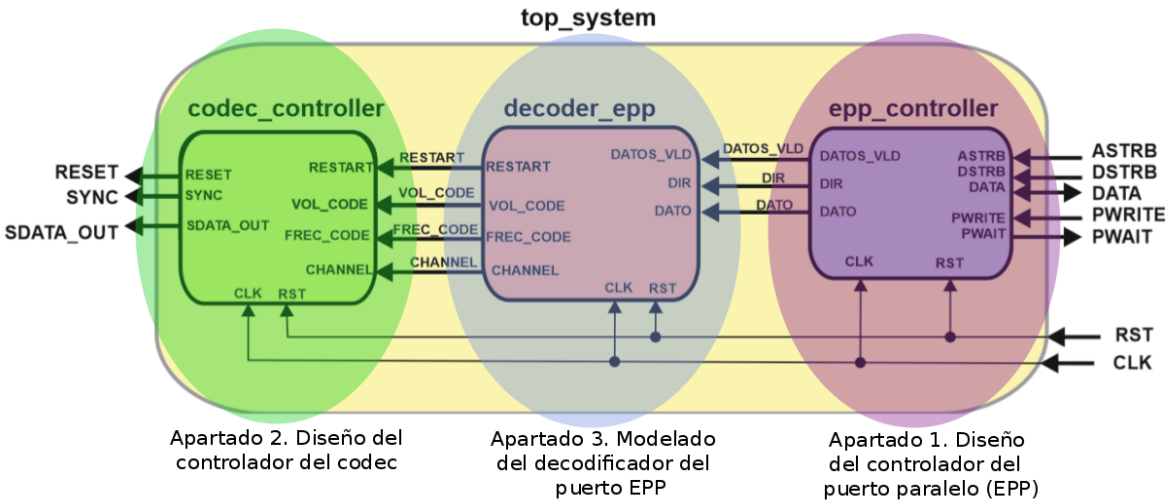


Figura 1. top_system

Para conseguir el completo funcionamiento del sistema se debe poder generar una señal de audio con el codec LM4550. Esta señal tendrá unos valores de frecuencia y amplitud que se transmiten a la FPGA de la placa Atlys a través del puerto USB del ordenador.

Como vemos en la figura 2, las datos enviados por el PC son convertidos por el driver CY7C68013A, que está en la placa Atlys y convierte los datos recibidos a un formato paralelo (EPP- Enhanced Parallel Port), enviandolos a la FPGA.

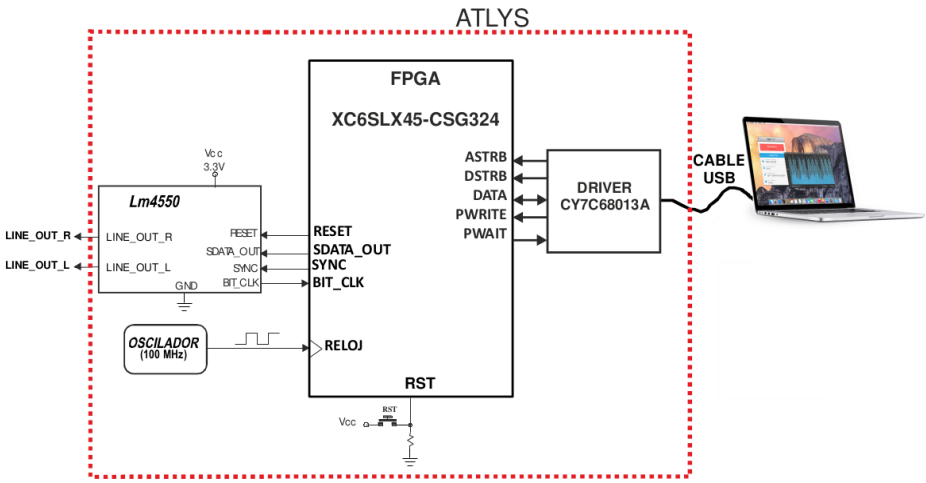


Figura 2. Conexión del hardware a utilizar

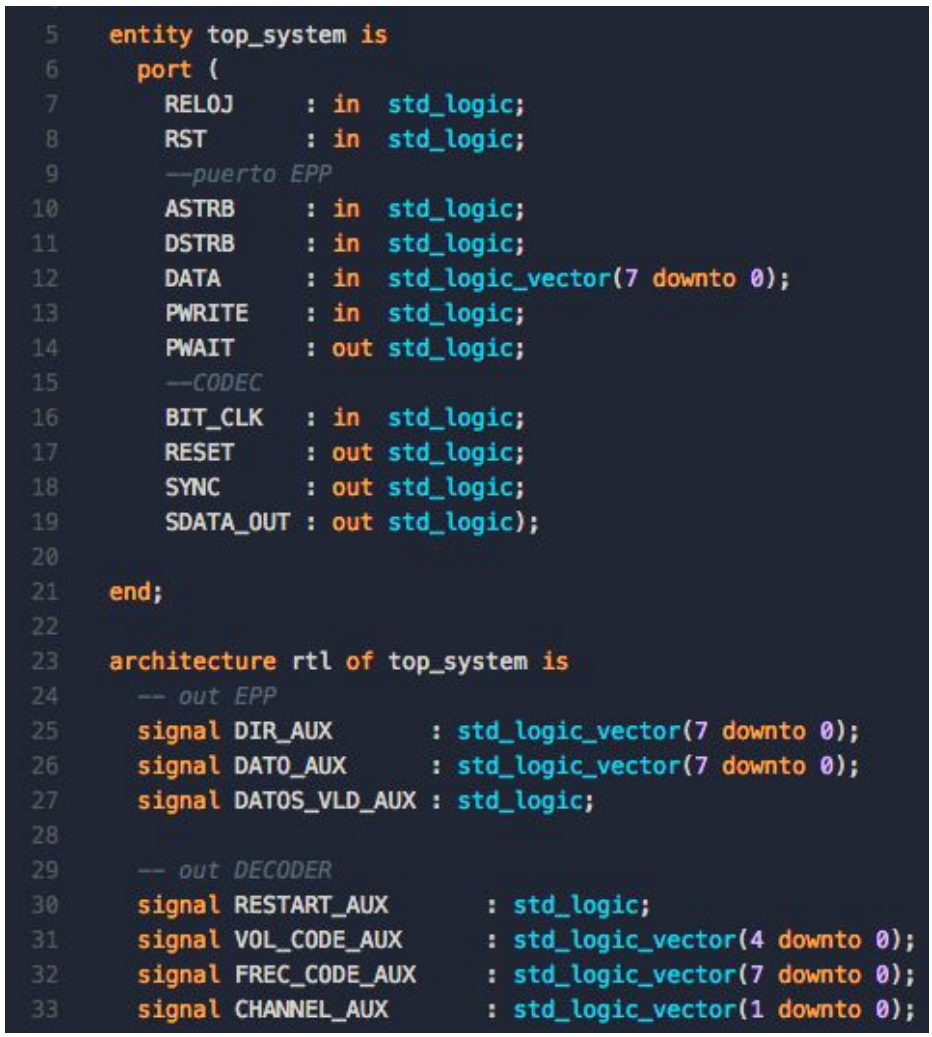
Modelar la entidad *top_system*

Antes de hablar sobre las interconexiones de los componentes principales del sistema, haremos una introducción de aquellos componentes que se nos han dado y que usamos por debajo de estos principales:

top_system

El modelado de este sistema ha sido muy sencillo ya que sólo quedaba unir los distintos componentes, para ello tenemos que conectar las entradas del *top_system* al *epp_controller* (**ASTRB**, **DSTRB**, **DATA**, **PWRITE**, **PWAIT**), las salidas al codec_controller (**RESET**, **SYNC**, **SDATA_OUT**) además de las señales de sincronización (**RELOJ** y **RST**).

Visto esto tan solo nos faltaría “interconectar” los componentes (señales de las líneas 24-33), para ello usaremos señales auxiliares que nos permiten conectar los componentes que las tienen como salida y en los que la tienen como entrada, parece difícil pero dejamos muestras del código utilizado para que se vea con mayor claridad en las Figuras 3 y 4.



```
5   entity top_system is
6     port (
7       RELOJ     : in  std_logic;
8       RST       : in  std_logic;
9       --puerto EPP
10      ASTRB     : in  std_logic;
11      DSTRB     : in  std_logic;
12      DATA      : in  std_logic_vector(7 downto 0);
13      PWRITE    : in  std_logic;
14      PWAIT     : out std_logic;
15      --CODEC
16      BIT_CLK   : in  std_logic;
17      RESET     : out std_logic;
18      SYNC      : out std_logic;
19      SDATA_OUT : out std_logic);
20
21  end;
22
23  architecture rtl of top_system is
24    -- out EPP
25    signal DIR_AUX       : std_logic_vector(7 downto 0);
26    signal DATO_AUX      : std_logic_vector(7 downto 0);
27    signal DATOS_VLD_AUX : std_logic;
28
29    -- out DECODER
30    signal RESTART_AUX       : std_logic;
31    signal VOL_CODE_AUX      : std_logic_vector(4 downto 0);
32    signal FREC_CODE_AUX     : std_logic_vector(7 downto 0);
33    signal CHANNEL_AUX       : std_logic_vector(1 downto 0);
```

Figura 3. Señales del top_system

```
36  begin
37
38    epp : entity work.epp_controller
39      port map (
40        CLK       => RELOJ,
41        RST       => RST,
42        ASTRB     => ASTRB,
43        DSTRB     => DSTRB,
44        DATA      => DATA,
45        PWRITE    => PWRITE,
46        PWAIT     => PWAIT,
47        DIR       => DIR_AUX,
48        DATO      => DATO_AUX,
49        DATOS_VLD => DATOS_VLD_AUX
50      );
51
52    decoder : entity work.decoder_epp
53      port map (
54        CLK       => RELOJ,
55        RST       => RST,
56        DIR       => DIR_AUX,
57        DATO      => DATO_AUX,
58        DATOS_VLD => DATOS_VLD_AUX,
59        RESTART   => RESTART_AUX,
60        VOL_CODE  => VOL_CODE_AUX,
61        FREC_CODE => FREC_CODE_AUX,
62        CHANNEL   => CHANNEL_AUX
63      );
64
65    codec : entity work.codec_controller
66      port map (
67        RST       => RST,
68        BIT_CLK   => BIT_CLK,
69        CLK       => RELOJ,
70        VOL_CODE  => VOL_CODE_AUX,
71        RESTART   => RESTART_AUX,
72        CHANNEL   => CHANNEL_AUX,
73        RESET     => RESET,
74        SYNC      => SYNC,
75        FREC_CODE => FREC_CODE_AUX,
76        SDATA_OUT => SDATA_OUT
77      );
78
79  end rtl;
```

Figura 4. Conexión de los componentes con las señales

1. epp_controller

Componente que modelamos en el Apartado 1, este es un controlador de puerto paralelo que recibe datos a través del cable usb que conectamos a nuestro ordenador y mediante el software Atlantis podremos interactuar con la placa.

Lo principal de este componente es que nos proporciona las señales **dir** y **dato**, que usaremos posteriormente en el componente *decoder_epp*, para más detalles sobre su funcionamiento podríamos usar la memoria de dicho apartado.

2. decoder_epp

Componente que modelamos en el Apartado 3, su función consiste en decodificar las señales **dir** y **dato** que se nos proporcionan a través de la entidad *epp_controller* y proporcionar a la entidad *codec_controller* las señales necesarias (**restart**, **vol_code**, **frec_code** y **channel**). Para recordar mejor su funcionamiento adjuntamos la Tabla 1, en la que podemos observar el comportamiento del sistema según los valores de **dir** y **dato**, además debemos recordar que cuando estos cambiaban, se mandaba también la señal **datos_vld** = '1', lo cual nos asegura la integridad de los datos enviados.

Dirección	Dato	Función
11x	11x	Inicialización del códec LM4550 (RESTART)
F0x	00x-FFx	Frecuencias de la nota
B0x	00x-1Fx	Volumen de la señal de audio
CAx	DDx	Tono por el canal derecho
CAx	11x	Tono por el canal izquierdo
CAx	22x	Tono por ambos canales
CAx	Resto combinaciones	Tono por ningún canal

Tabla 1. Direcciones y datos a utilizar por la entidad decoder_epp

3. codec_controller

Componente que modelamos en el Apartado 2, su función es comunicarse con el códec LM4550, básicamente podríamos afirmar que es una interfaz por encima de este, toma los datos que se pasan por el *decoder_epp* (**restart**, **vol_code**, **frec_code** y **channel**) para producir una salida de audio acorde a esas señales.

Merece la pena recordar también que este componente hace uso de una memoria con 4096 posiciones (señal sinusoidal) a la que accedemos según la señal **frec_code**.

Banco de pruebas de la entidad *top_system*

top_system_tb

El funcionamiento es sencillo, se instancian los componentes que veremos a continuación. Se generan las señales de **rst** y **reloj** y se escribe la salida en "*vout.dat*" (que veremos posteriormente, con mayor detalle) como mostramos en la Figura 5.

```
process(sync_i)
  file arch_out      : text open write_mode is "../sources/vout.dat";
  variable buffer_wr : line;
  variable vout      : real := 0.0;
begin  -- process
  if sync_i = '1' and sync_i'event then
    vout := LINE_OUT_L_i;
    write (buffer_wr, vout);
    write (buffer_wr, " ");
    vout := LINE_OUT_R_i;
    write (buffer_wr, vout);
    writeline (arch_out, buffer_wr);
  end if;
end process;
```

Figura 5. Escritura de salida de los canales en el fichero vout.dat

1. epp_device_ts

Este componente se nos proporciona, pero merece la pena analizar su funcionamiento para poder entender el conjunto del sistema.

¡**Atención!** Las siguientes capturas están desordenadas como se puede observar en las líneas, pero se ponen en este orden para facilitar la comprensión del código.

El código es bastante sencillo, lo principal es el archivo de datos que leemos ("*datos2.dat*") como se puede apreciar en la Figura 6 y la llamada recursiva al procedimiento epp_cycle, que veremos a continuación, el resto son inicializaciones de las señales, algo propio de los bancos de pruebas.

```
48        file arch_in : text open read_mode is "../SOURCES/datos2.dat";
49        variable bf   : line;
50        variable dato : std_logic_vector(7 downto 0);
51        variable dir  : std_logic_vector(7 downto 0);
52        variable tiempo  : time;
53      begin
54        --inicializacion
55        data   <= (others => '0');
56        PWRITE <= '1';
57        DSTRB  <= '1';
58        ASTRB  <= '1';
59
60        dir  := (others => '0');
61        wait for 330 ns;
62
63        while not endfile(arch_in) loop
64          readline (arch_in, bf);
65          hread (bf, dir);
66          hread (bf, dato);
67         read (bf, tiempo);
68
69          epp_cycle ( address => dir,
70                      data_out => dato);
71          wait for tiempo;
72        end loop;
73
74        report "FIN CONTROLADO DE LA SIMULACION" severity failure;
75      end process;
```

Figura 6. Main del TB del epp_device_ts

Al no disponer de una placa para trabajar en casa, con este banco de pruebas podremos hacer las simulaciones pertinentes y verificar que nuestro sistema funciona perfectamente antes de realizar la descarga en placa, en él se define un procedimiento que lee del fichero "*datos2.dat*" hasta que llega al final de este.

Vemos en la Figura 7 que desde las líneas 27 a la 34 tenemos escrituras en la dirección poniendo las señales **PWRITE** y **ASTRB** a '0' y transmitiendo el valor de la señal **data** a la dirección. Desde las líneas 37 a la 44 tenemos un proceso de escritura del dato poniendo las señales **PWRITE** y **DSTRB** a '0' y transmitiendo el valor de la señal **data** al dato.

```
23      procedure epp_cycle ( address : in  std_logic_vector(7 downto 0);
24                            data_out : in std_logic_vector(7 downto 0)) is
25      begin
26        -- escritura direccion.
27          PWRITE <= '0';
28          wait for 33 ns;
29          ASTRB  <= '0';
30          data   <= address;
31           wait for 133 ns;
32          ASTRB  <= '1';
33           wait for 33 ns;
34          data   <= (others => 'Z');
35          PWRITE <= '1';
36           wait for 133 ns;
37          PWRITE <= '0';
38          data   <= data_out;
39           wait for 33 ns;
40          DSTRB  <= '0';
41           wait for 133 ns;
42          DSTRB  <= '1';
43           wait for 33 ns;
44          data   <= (others => 'Z');
45          PWRITE <= '1';
46      end procedure;
```

Figura 7. Procedimiento epp_cycle dentro del epp_device_ts

2. LM4550

Quizá deberíamos haber hablado de primeras sobre el códec para poder entender con mayor claridad por qué hemos realizado así el sistema, este archivo se nos proporciona y es sin duda el que más complejidad tiene ya que es el core de nuestro sistema.

Como comentamos anteriormente la interfaz que controla este códec es el *codec_controller*, iremos descendiendo en el diseño para entender el funcionamiento, para ello podemos ver que al codec_controller se le conectan las señales **RESTART**, **VOL_CODE**, **FREC_CODE** y **CHANNEL**. La señales **RESTART** y **FREC_CODE** las usaremos directamente en el *codec_controller* y "no afectan", por tanto nos quedamos con las otros dos.

VOL_CODE es el encargado de modificar el volumen de la salida, esto se realiza mediante el registro Master Volume (0x02) y el PCM Out Volumen (0x18), podemos ver cómo funciona esta parte con el siguiente código (Figura 8):

```
master_vol_r <= REG_02(4 downto 0);
master_vol_l <= REG_02(12 downto 8);
mute_master  <= REG_02(15);

process (master_vol_r, master_vol_l, mute_master) is
 begin
   if mute_master = '1' then
     master_gn_r <= 0.0;
     master_gn_l <= 0.0;
   else
     master_gn_r <= 10.0**((0.0-real(to_integer(unsigned(master_vol_r)))*1.5)/20.0);
     master_gn_l <= 10.0**((0.0-real(to_integer(unsigned(master_vol_l)))*1.5)/20.0);
   end if;
 end process;
```

Figura 8. Registro 0x20 Master Volume

En la figura 10, lo primero que hacemos es separar los bits del registro PCM Out Volume según la figura 9. Así que, usamos los bits GR[4:0] para controlar la ganancia o atenuación del canal derecho (**out_vol_r**), y los bits GL[4:0] controlando el canal izquierdo (**out_vol_l**). El último bit controla el valor mute, silenciando las salidas de los dos DACs.

El proceso comprueba si los canales están silenciados (valor de **mute_out** a '1'), y en caso contrario se ajusta la ganancia o atenuación de los canales izquierdo y derecho en el rango de 12 dB y en pasos de 1.5 dB

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
Mute	0	0	GL4	GL3	GL2	GL1	GL0	0	0	0	GR4	GR3	GR2	GR1	GR0

Figura 9. Funcionalidad de los bits del registro PCM Out Volume

Comentar que las últimas líneas, de pop en adelante son "innecesarias" en el sentido de que pop por nuestra configuración no cambia como comentaremos posteriormente, por tanto los valores de **output_r** y **stereo_mix_r** serán los mismos, lo mismo sucede con el canal izquierdo.

```
dac_out_r <= real(to_integer(signed(dac_in_r(19 downto 2))))*2.0/262144.0;
dac_out_l <= real(to_integer(signed(dac_in_l(19 downto 2))))*2.0/262144.0;

out_vol_r <= REG_18(4 downto 0);
out_vol_l <= REG_18(12 downto 8);
mute_out  <= REG_18(15);

process (out_vol_r, out_vol_l, mute_out) is
begin
  if mute_out = '1' then
    out_gn_r <= 0.0;
    out_gn_l <= 0.0;
  else
    out_gn_r <= 10.0**((12.0-real(to_integer(unsigned(out_vol_r)))*1.5)/20.0);
    out_gn_l <= 10.0**((12.0-real(to_integer(unsigned(out_vol_l)))*1.5)/20.0);
  end if;
end process;

output_r <= dac_out_r*out_gn_r;
output_l <= dac_out_l*out_gn_l;

pop <= REG_20(15);

stereo_mix_r <= output_r when pop = '1' else 0.0;

stereo_mix_l <= output_l when pop = '1' else 0.0;
```

Figura 10. Registro 0x18 PCM Out Volume

Por último tan solo queda multiplicar las ganancias / atenuaciones de los registros 0x02 y 0x18 para generar las señales de salida finales en ambos canales como podemos ver en la Figura 11.

```
LINE_OUT_R <= stereo_mix_r*master_gn_r;
LINE_OUT_L <= stereo_mix_l*master_gn_l;
```

Figura 11. Salida según canal

La señal **CHANNEL** se configura cuando se pasan datos y cuando no desde el *codec_controller* (recordemos la Tabla 1), dichos valores se pasan a la trama para que se pueda generar correctamente las señales de salida de ambos canales, dependiendo de los valores de la trama se pueden mutear ambos canales o pasar sólo datos por uno de ellos como podemos ver en las siguientes muestras de código de la FIgura 12:

```
master_vol_r <= REG_02(4 downto 0);
master_vol_l <= REG_02(12 downto 8);
mute_master  <= REG_02(15);

process (master_vol_r, master_vol_l, mute_master) is
 begin
   if mute_master = '1' then
     master_gn_r <= 0.0;
     master_gn_l <= 0.0;
   else
     master_gn_r <= 10.0**((0.0-real(to_integer(unsigned(master_vol_r)))*1.5)/20.0);
     master_gn_l <= 10.0**((0.0-real(to_integer(unsigned(master_vol_l)))*1.5)/20.0);
   end if;
 end process;
```

Figura 12. Se mutean ambos o se toma valor en cada canal

Por último comentar que el siguiente pedazo de código se utiliza para saltarse 3D Sound de National tal y como se especifica en el Anexo 1 que se proporciona para realizar esta práctica, Figura 13:

```
pop <= REG_20(15);

stereo_mix_r <= output_r when pop = '1' else 0.0;

stereo_mix_l <= output_l when pop = '1' else 0.0;
```

Figura 13. Deshabilitar 3D Sound de National

Simulación funcional de la entidad *top_system*

Para las simulaciones funcional y temporal cogemos el fichero *top_system_epp_tb.vhd* explicado en el apartado anteriormente.

Comprobamos que nuestra entidad funciona tal como queremos, en la figura 14 estamos metiendo los valores 11x tanto en dirección como en dato, por lo que según la Tabla 1 estaríamos inicializando el códec LM4550 (**RESTART**). Después de que los datos se hayan validado con la señal **datos_vld** vemos que se activa la señal **restart** poniendo a 0 la salida de los 2 canales.

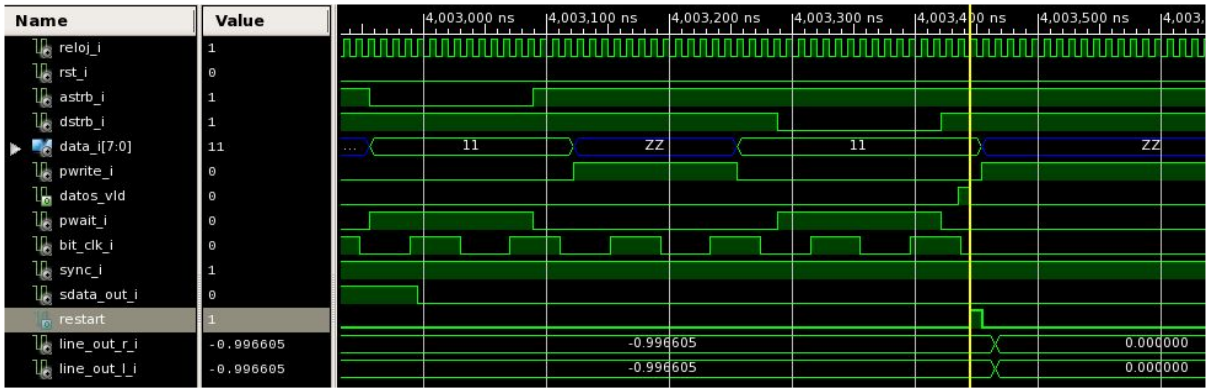


Figura 14. Captura de simulación funcional (dirección y dato a 11x)

Comprobamos otro ejemplo mirando las figuras 15 y 16. Al igual que antes, comprobando la Tabla 1, vemos que si tenemos el valor CAx en dirección y 11x en dato vamos a transmitir audio únicamente por el canal izquierdo. En la Figura 16 vemos como al inicio de la trama, se deja de transmitir información por el canal derecho poniéndolo a 0.

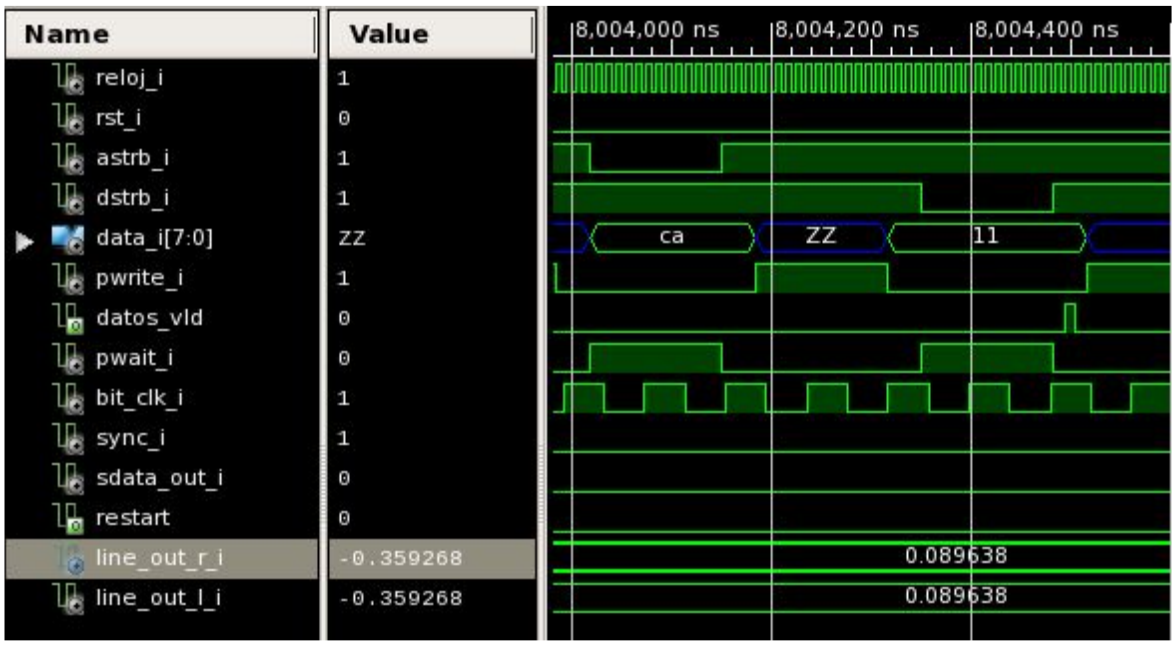


Figura 15. Valores CAx en dirección y 11x en dato.

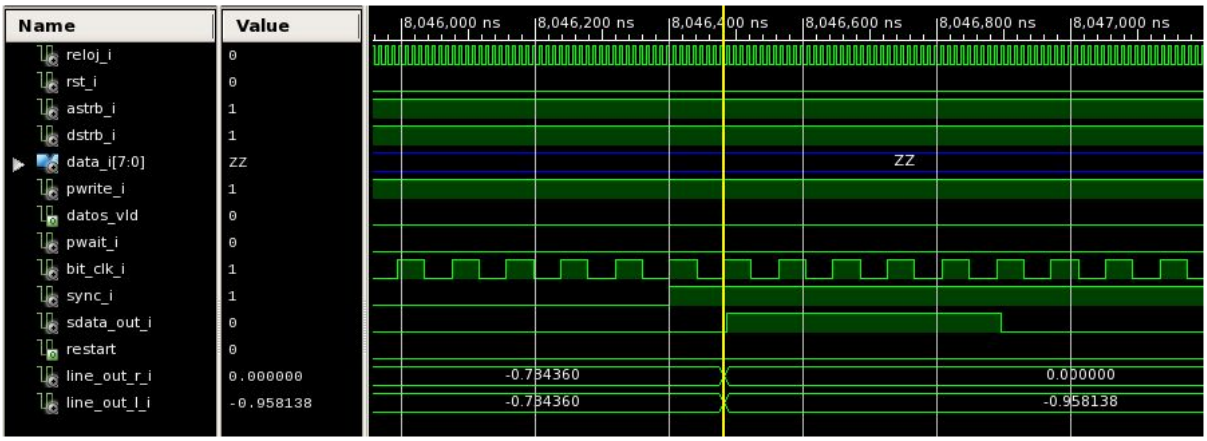


Figura 16. El canal derecho deja de transmitir información poniéndose a valor 0.

En la Figura 17 vemos un captura global de toda la simulación funcional. Primero transmitimos valores por el canal derecho, luego por los 2 canales y por último sólo transmitimos audio por el canal izquierdo.

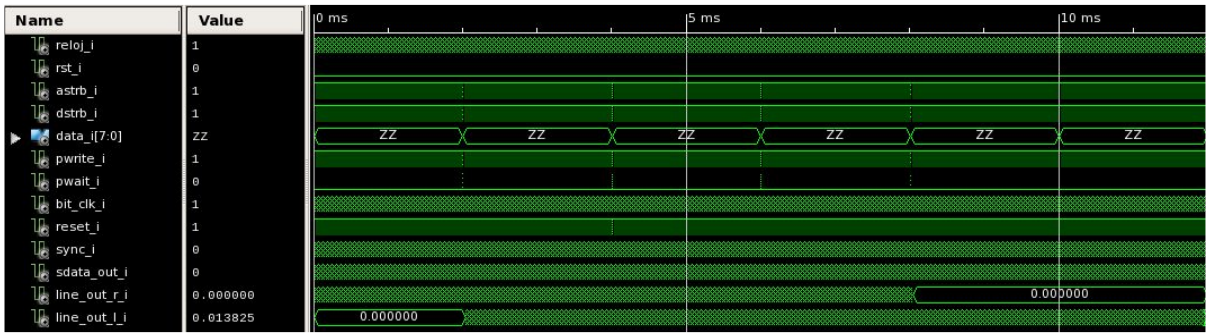


Figura 17. Captura de la simulación funcional

Simulación temporal de la entidad *top_system*

Con la simulación temporal comprobamos que la temporización del diseño cumple con las restricciones impuestas al mismo, fundamentalmente las relacionadas con la señal del reloj.

En la Figura 18 comprobamos que las señales resaltadas (**pwait**, **reset**, **sync** y **sdata_out**) salen en color naranja ya que no está inicializado su valor.

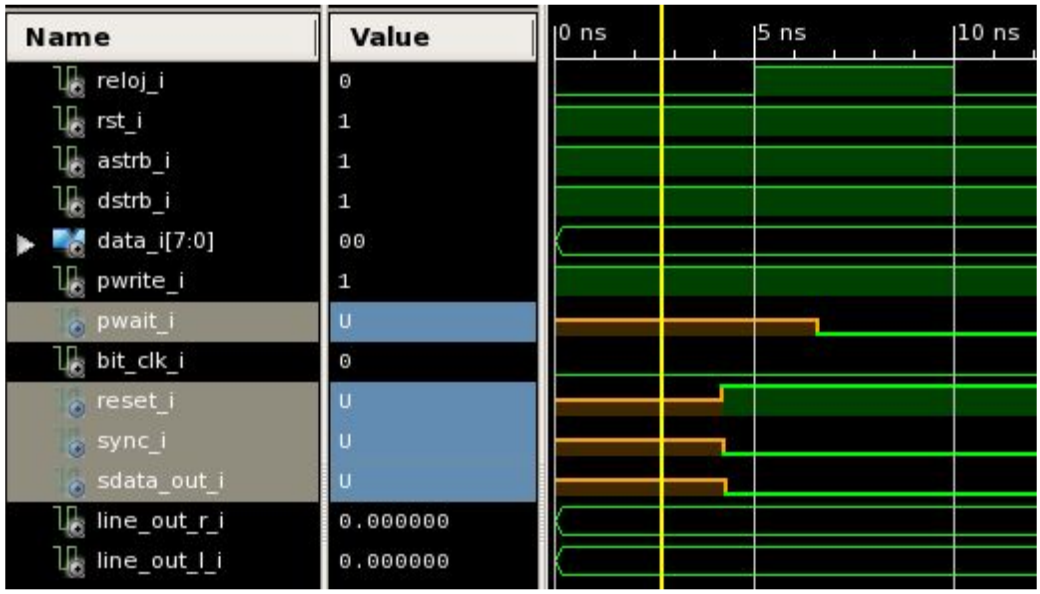


Figura 18. Captura al inicio de la simulación temporal

El resto de la simulación temporal (figura 19) cumple con lo que buscábamos en el diseño de la entidad *top_system*.

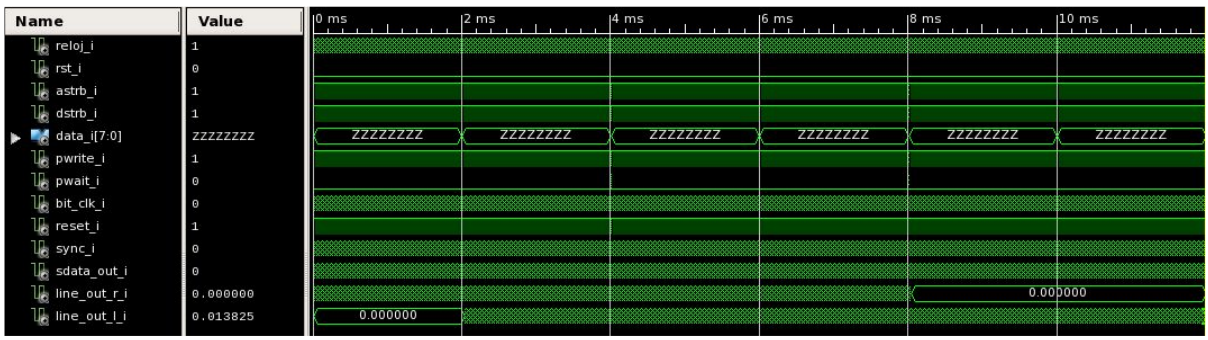


Figura 19. Captura de la simulación temporal

Salida MatLab

Como ya sucedía en el Apartado 2, el banco de pruebas además de la salida por pantalla de nuestra simulación nos generaba un fichero con las señales de ambos canales de audio que podíamos usar con el software matemático MatLab para verificar la salida por ambos canales de dicha simulación, en este caso nos genera la siguiente salida que vemos en la Figura 20:

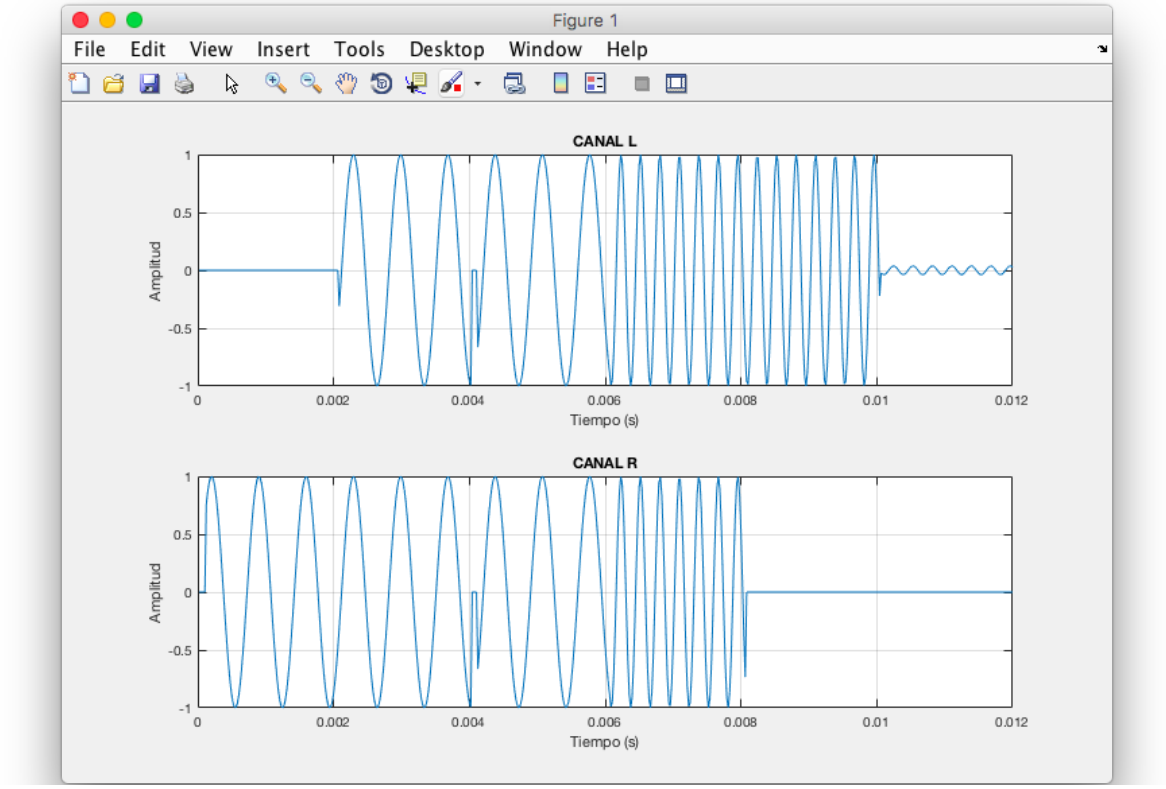


Figura 20. Captura del programa Matlab

Informe sobre los recursos utilizados

Es condición imprescindible generar un informe (Tabla 2) 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.

Number of Slices Registers	139 out of 54,576 1%
Number of Slice LUTs	1,171 out of 27,288 4%
Number of occupied Slices	379 out of 6,822 5%

Tabla 2. Recursos utilizados