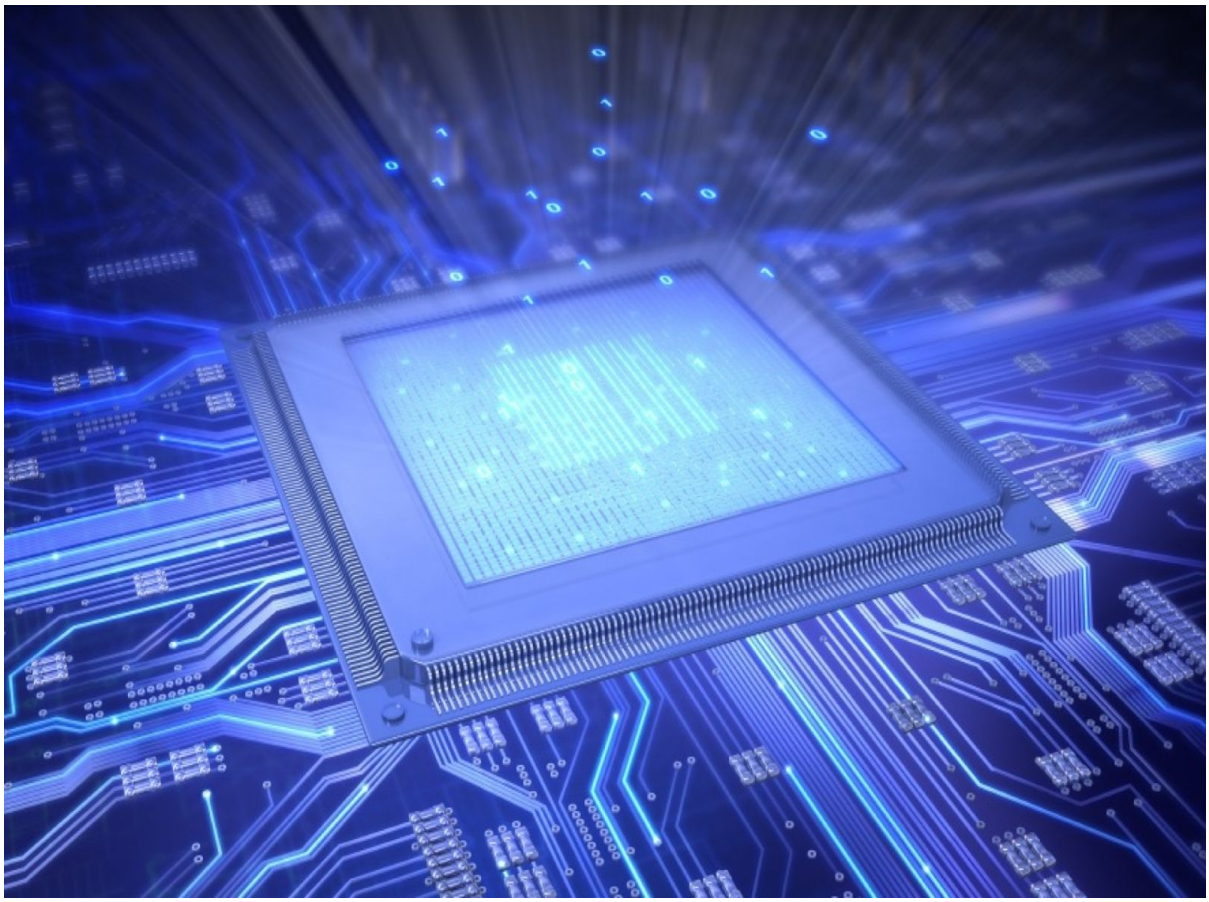
Modelado y Síntesis de Sistemas Electrónico Digitales

Apartado 3. Modelado del decodificador del puerto EPP



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

Índice

Introducción

Para este apartado se nos proporciona en clase una completa explicación del diseño y el funcionamiento de la entidad a modelar. El objetivo es decodificar las direcciones y datos proporcionados a través del puerto epp (entidad *epp_controler que* se modeló en el "*Apartado 1. Diseño del controlador del puerto paralelo (EPP)*") y enviarlos al controlador del codec de audio (entidad *codec_controller* que se modeló en el "Apartado 2. Diseño del controlador del códec").

Como vemos en la Tabla 1, es necesario controlar todas las situaciones según la dirección y el dato que se reciba a través de los puertos de entrada **dir** y **dato**.

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

Como vemos en la figura 1, esta entidad hace de intermediario entre las otras entidades para completar nuestro *top_system* en el siguiente apartado de la práctica libre.

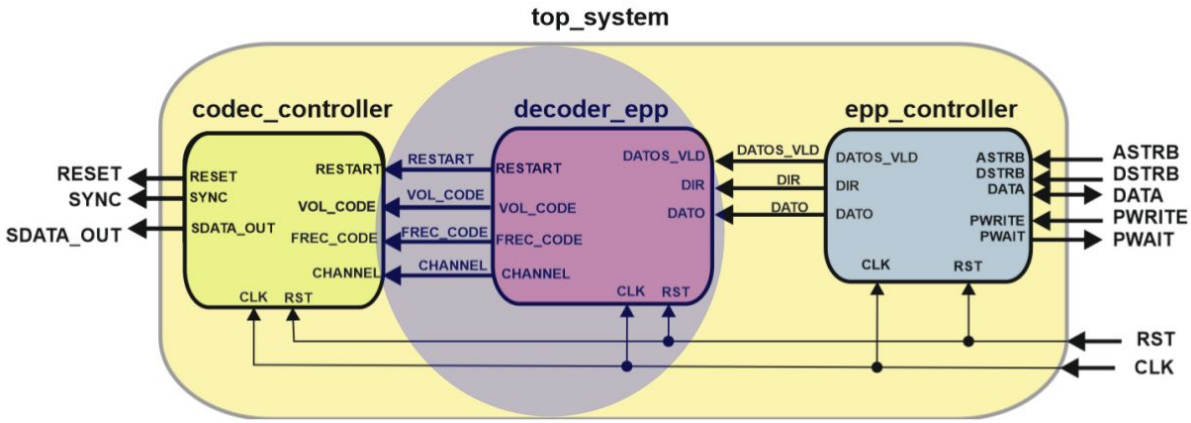


Figura 1. top_system

Modelar la entidad *decoder_epp*

1. *RESTART*

La señal **RESTART** se debe activar cuando se recibe la dirección 11x y el dato 11x (comando restart) y sólo puede estar a nivel alto durante un periodo de la señal **CLK**, que dura 10 ns.

Para conseguir tal fin necesitamos una variable o señal auxiliar ya que la señal **RESTART** es de salida y no podemos hacer comprobaciones con respecto a ella. La lógica es sencilla, tenemos una variable **flag** que inicializamos a 0, cuando se cumplan las condiciones de **DIR** y **DATO**, si la variable está en su estado inicial, activamos el **RESTART** y cambiamos el estado de la variable **flag**, la siguiente vez que se llame al proceso y puesto que las variables conservan su valor cuando se vuelve a llamar al proceso, pondríamos **RESTART** en 0 para que se cumpla un solo tick de **clk**.

```
process(clk, rst)
  variable flag : std_logic := '0';
begin
  if rst = '1' then
    RESTART <= '0';
    flag := '0';
  elsif clk'event and clk = '1' then
    if DIR = x"11" and DATO = x"11" then
      if flag = '0' and datos_vld = '1' then
        RESTART <= '1';
        flag := '1';
      else
        RESTART <= '0';
      end if;
    end if;
  end if;
end process; -- restart
```

Figura 2. Código del proceso RESTART

Para comprobar si hemos modelado correctamente el proceso, cogemos una captura de la simulación funcional (figura 3) en la que vemos como la señal **RESTART** se activa cuando ha cumplido todas las condiciones necesarias, y sólo está a nivel alto durante 1 tick completo de reloj o señal **CLK,** que dura 10 ns.

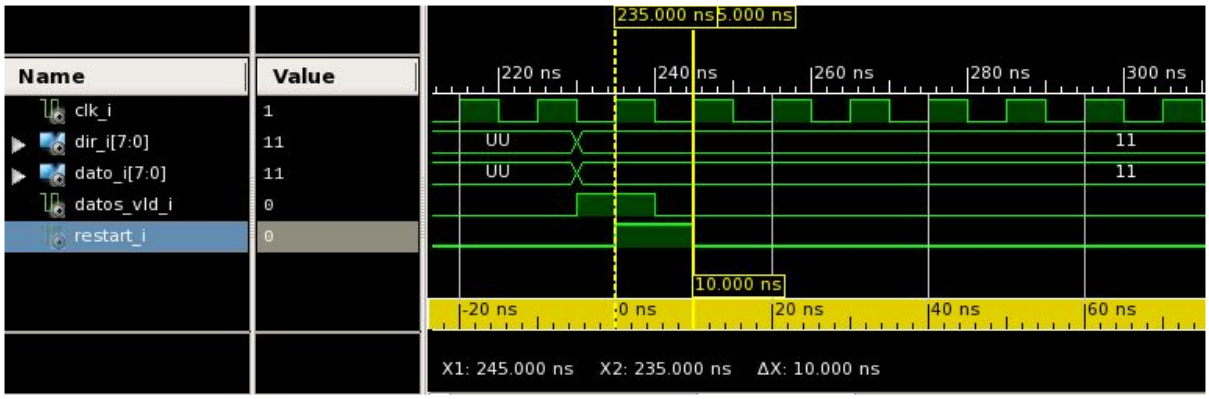


Figura 3. Captura de la simulación funcional

2. FREC_CODE

En este proceso tenemos que enviar el valor de frecuencia del tono, también llamado nota. Este valor se corresponde con el dato de un ciclo de escritura en la dirección **F0x**.

Para modelar esto, comprobamos que se produce un tick de reloj, y si la dirección corresponde con el valor **F0x** y la señal **DATOS_VLD** está a nivel alto (los datos se han transmitido de forma correcta), pasamos a la señal de salida **FREC_CODE** el valor de la señal de entrada **DATO.**

```
process(clk, rst)
begin
  if rst = '1' then
    frec_code <= (others => '0');
  elsif clk'event and clk = '1' then
    if datos_vld = '1' and dir = x"F0" then
      frec_code <= dato;
    end if;
  end if;
end process; -- frec_code
```

Figura 4. Código del proceso FREC_CODE

Para comprobar si el proceso está bien modelado vemos 2 situaciones en la simulación funcional:

1. En la figura 5 vemos que cuando se cumplen todos los requisitos descritos anteriormente, la señal de salida **FREC_CODE** es igual que la señal de entrada **DATO**. Ambas señales están resaltadas en color blanco.

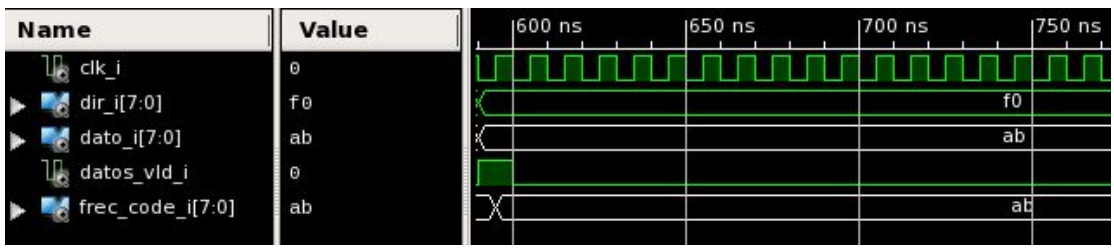


Figura 5. Captura de simulación.

2. En la figura 6 no estamos cumpliendo que la señal de dirección (resaltada en blanco) sea F0x, por lo que comprobamos que la señal de salida **FREC_CODE** se mantiene a 00x.

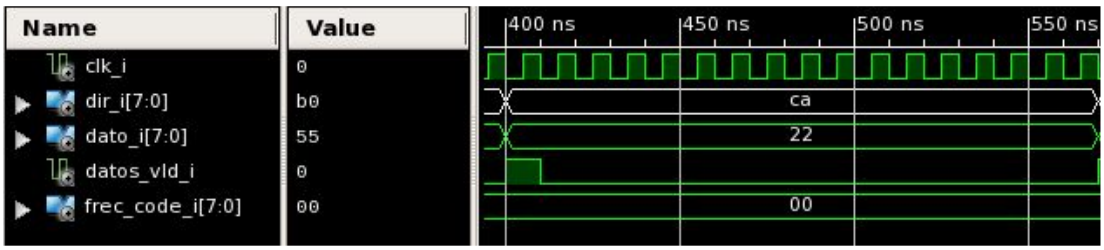


Figura 6. Captura de simulación.

3. VOL_CODE

En este proceso tenemos que cargar en el registro *Master Volume* (02x) el valor del volumen, ya que forma parte de su trama como vimos en el apartado anterior:

```
cmd_data <= b"000" & vol_code & b"000" & vol_code & x"0";
```

Este valor se corresponde con el dato de un ciclo de escritura en la dirección **B0x**.

Para modelar esto, comprobamos que se produce un tick de reloj, y si la dirección corresponde con el valor **B0x** y la señal **DATOS_VLD** está a nivel alto (los datos se han transmitido de forma correcta), pasamos a la señal de salida **VOL_CODE** el valor de la señal de entrada **DATO.** Como la señal **VOL_CODE** es de 5 bits, sólo le pasamos los 5 bits de mayor peso del dato.

```
process(clk, rst)
begin
  if rst = '1' then
    vol_code <= (others => '0');
  elsif clk'event and clk = '1' then
    if datos_vld = '1' and dir = x"B0" then
      vol_code <= dato(4 downto 0);
    end if;
  end if;
end process; -- vol_code
```

Figura 7. Código para el proceso de VOL_CODE

Siguiendo el mismo procedimiento que en los procesos anteriores, comprobamos si funciona correctamente en la simulación funcional de la entidad. En la figura 8 se están cumpliendo las condiciones necesarias y vemos que la señal **vol_code** es igual a los 5 bits de mayor peso de la señal **dato** (señales resaltadas en color blanco).

dato_i = 01010101
vol_code_i = 10101

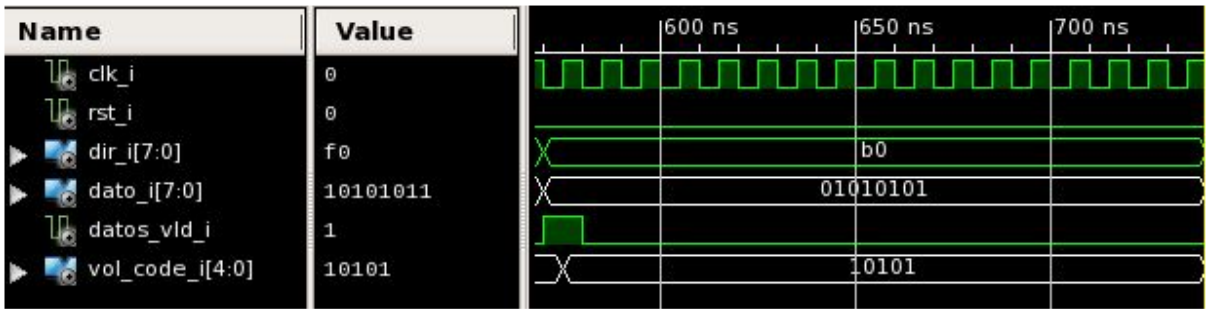


Figura 8. Captura de la simulación funcional

4. CHANNEL

La señal de salida **CHANNEL** permite seleccionar el canal por el que se proporciona el tono. Como sólo tenemos 4 posibles valores (tabla 1) necesitamos un vector de 2 bits.

Para modelar esto, comprobamos que se produce un tick de reloj, y si la dirección corresponde con el valor **CAx** y la señal **DATOS_VLD** está a nivel alto (los datos se han transmitido de forma correcta), pasamos a la señal de salida **CHANNEL** un valor siguiendo la tabla 2.

Dirección	Dato	CHANNEL	Función
CAx	DDx	**10**	Tono por el canal derecho
CAx	11x	**01**	Tono por el canal izquierdo
CAx	22x	**11**	Tono por ambos canales
CAx	Resto combinaciones	**00**	Tono por ningún canal

Tabla 2. Asignación de valores en la señal CHANNEL

El código para este proceso quedaría así:

```
process(clk, rst)
begin
  if rst = '1' then
    chanel <= (others => '0');
  elsif clk'event and clk = '1' then
    if datos_vld = '1' and dir = x"CA" then
      case(dato) is
        when x"DD" =>
          chanel <= "10";
        when x"11" =>
          chanel <= "01";
        when x"22" =>
          chanel <= "11";
        when others =>
```

```
            chanel <= "00";
        end case;
      end if;
    end if;
end process; -- channel
```

Figura 9. Código para el proceso del CHANNEL

Comprobamos si funciona tal como se espera viendo la simulación funcional en cada caso. En todos los casos se están cumpliendo las condiciones necesarias para transmitir un valor en el canal (señal **CHANNEL**). Como en el resto de simulaciones anteriores, hemos resaltado las señales comentadas en color blanco.

1. Tenemos el valor DDx en la señal **dato**, por lo que la señal **channel** tiene el valor '10' binario y estamos transmitiendo los tonos por el canal derecho.

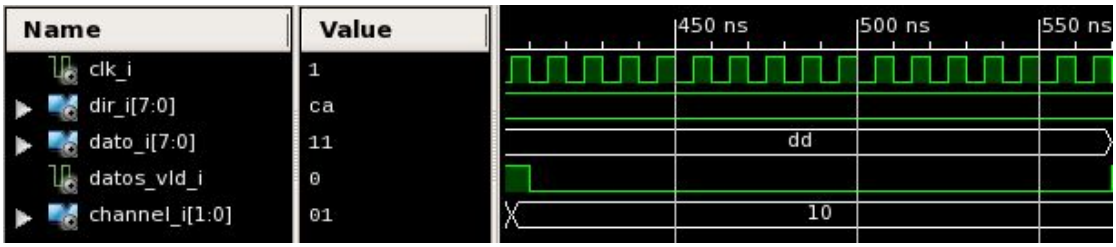


Figura 10. Captura de la simulación funcional

2. Tenemos el valor 11x en la señal **dato**, por lo que la señal **channel** tiene el valor '01' binario y estamos transmitiendo los tonos por el canal izquierdo.

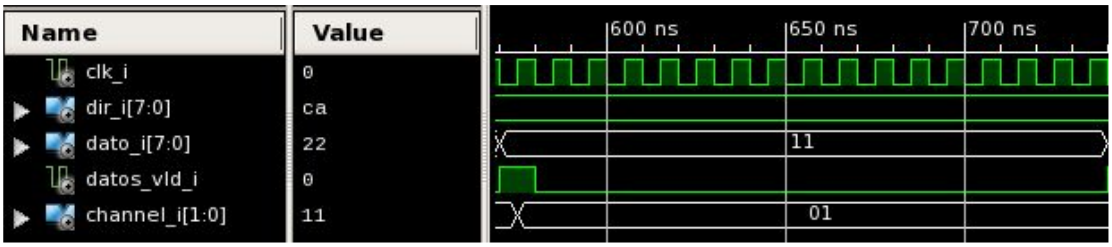


Figura 11. Captura de la simulación funcional

3. Tenemos el valor 22x en la señal **dato**, por lo que la señal **channel** tiene el valor '11' binario y estamos transmitiendo los tonos por ambos canales.

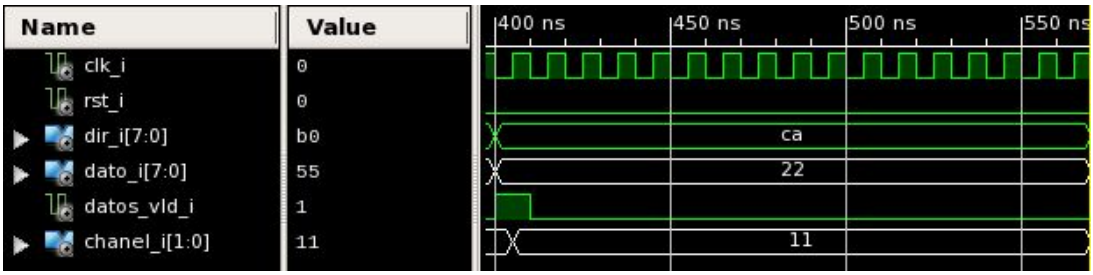


Figura 12. Captura de la simulación funcional

4. Tenemos otra combinación diferente de las comentadas antes en la señal **dato**, por lo que la señal **channel** tiene el valor '00' binario y hemos dejado de transmitir los tonos por ambos canales.

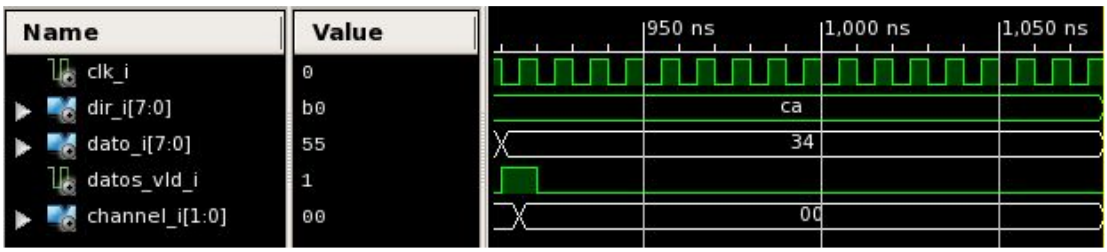


Figura 13. Captura de la simulación funcional

Banco de pruebas de la entidad *decoder_epp*

Hemos programado un banco de pruebas (testbench) para verificar el correcto funcionamiento de nuestra entidad.

Lo primero que hacemos es declarar las señales, importamos el *decoder_epp* y asociamos las señales anteriores a los puertos de la entidad, acto seguido configuramos las señales **RST_i** y **CLK_i** como podemos ver en la figura 14.

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_epp_tb is

end decoder_epp_tb;


architecture sim of decoder_epp_tb is

    signal CLK_i       : std_logic := '0';
    signal RST_i       : std_logic := '1';
    signal DIR_i       : std_logic_vector(7 downto 0);
    signal DATO_i      : std_logic_vector(7 downto 0);
    signal DATOS_VLD_i : std_logic := '0';
    signal RESTART_i   : std_logic;
    signal VOL_CODE_i  : std_logic_vector(4 downto 0);
    signal FREC_CODE_i : std_logic_vector(7 downto 0);
    signal CHANNEL_i   : std_logic_vector(1 downto 0);

begin  -- sim
```

```
begin  -- sim

  DUT : entity work.decoder_epp
    port map (
        CLK       => CLK_i,
        RST       => RST_i,
        DIR       => DIR_i,
        DATO      => DATO_i,
        DATOS_VLD => DATOS_VLD_i,
        RESTART   => RESTART_i,
        VOL_CODE  => VOL_CODE_i,
        FREC_CODE => FREC_CODE_i,
        CHANNEL   => CHANNEL_i);


  RST_i <= '1', '0'  after 223 ns;
  CLK_i <= not CLK_i after 5 ns;


  process
  begin  -- process
```

Figura 14.

Es necesario inicializar **DIR_I** y **DATO_I**, ya que de no hacerlo en la simulación temporal al primer cambio de datos, vendremos de valores *undefined*, lo que se traduce en un error en nuestra simulación con los fallos que llevaría a costa de esto.

Pasamos a hablar ahora de nuestro proceso, que nos proporcionará los resultados en simulación, haremos que este empiece tras el **RST_i** al que sometemos a nuestro sistema inicial, lo primero que hacemos es realizar la inicialización del codec, para lo cual como vimos anteriormente tenemos que comprobar que se está pasando una trama válida en **DATOS_VLD_i** y que **DIR_i** y **DATO_i** tienen un valor de **11x**, a destacar también que hay que esperar a que haya un flanco de bajada de la señal de reloj **CLK_i** pues es cuando se empieza a mandar a **DATOS_VLD_i**.

```
process
begin  -- process

  DIR_i <= (others => '0');
  DATO_i <= (others => '0');

  wait until RST_i'event and RST_i = '0';

  -- Init
  wait until CLK_i'event and CLK_i = '0';
  DATOS_VLD_i <= '1';
  DIR_i <= x"11";
  DATO_i <= x"11";
  wait until CLK_i'event and CLK_i = '0';
  DATOS_VLD_i <= '0';

  wait for 150 ns;
```

```
-- right channels
wait until CLK_i'event and CLK_i = '0';
DATOS_VLD_i <= '1';
DIR_i <= x"CA";
DATO_i <= x"DD";
wait until CLK_i'event and CLK_i = '0';
DATOS_VLD_i <= '0';

wait for 150 ns;

     -- left channels
wait until CLK_i'event and CLK_i = '0';
DATOS_VLD_i <= '1';
DIR_i <= x"CA";
DATO_i <= x"11";
wait until CLK_i'event and CLK_i = '0';
DATOS_VLD_i <= '0';

wait for 150 ns;
```

```
    -- any channels
wait until CLK_i'event and CLK_i = '0';
DATOS_VLD_i <= '1';
DIR_i <= x"CA";
DATO_i <= x"34";
wait until CLK_i'event and CLK_i = '0';
DATOS_VLD_i <= '0';

wait for 150 ns;

-- both channels
 wait until CLK_i'event and CLK_i = '0';
 DATOS_VLD_i <= '1';
 DIR_i <= x"CA";
 DATO_i <= x"22";
 wait until CLK_i'event and CLK_i = '0';
 DATOS_VLD_i <= '0';

 wait for 150 ns;
```

```
  -- vol
  wait until CLK_i'event and CLK_i = '0';
  DATOS_VLD_i <= '1';
  DIR_i <= x"B0";
  DATO_i <= x"55";
  wait until CLK_i'event and CLK_i = '0';
  DATOS_VLD_i <= '0';

  wait for 150 ns;
```

```
  -- frec
  wait until CLK_i'event and CLK_i = '0';
  DATOS_VLD_i <= '1';
  DIR_i <= x"F0";
  DATO_i <= x"AB";
  wait until CLK_i'event and CLK_i = '0';
  DATOS_VLD_i <= '0';

  wait for 150 ns;
```

Figura 16. Código para la simulación.

Las capturas anteriores corresponden al proceso del testbench, en el que vamos cambiando **DIR_i** y **DATO_i**, lo cual nos permite observar los cambios en las salidas cuando simulamos. Dichas simulaciones las podemos ver en el siguiente apartado.

Simulación funcional de la entidad *decoder_epp*

Para las simulaciones funcional y temporal cogemos el fichero *decoder_epp_tb.vhd* explicado en el apartado anterior.

Vemos que al principio de la simulación tenemos las señales **dir** y **dato** con valor 'U' debido a que las señales se están inicializando (señal **rst** con valor '1'), por lo que en estas entradas no hay ningún valor.

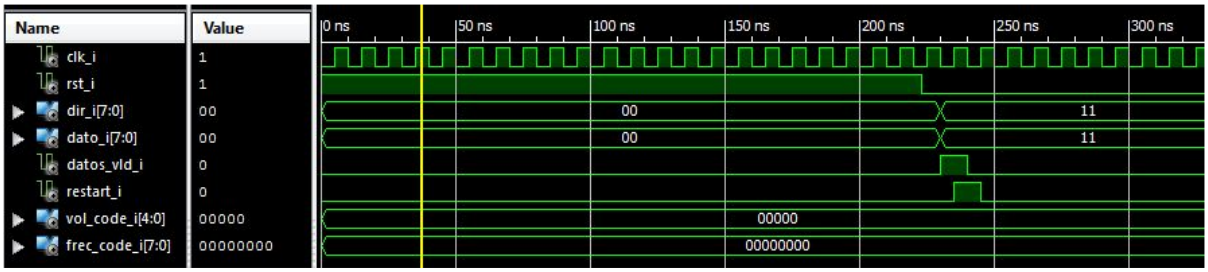


Figura 17. Captura al inicio de la simulación funcional

Al final de la simulación vemos los siguientes valores en los puertos de salida:

- Señal **vol_code** a '15'.
- Señal **frec_code** a 'ab'.
- Señal **channel** a '11'.

Según los valores de la tabla 1 tendríamos tono por ambos canales a una frecuencia con valor 'ab' y un volumen con valor '15'.

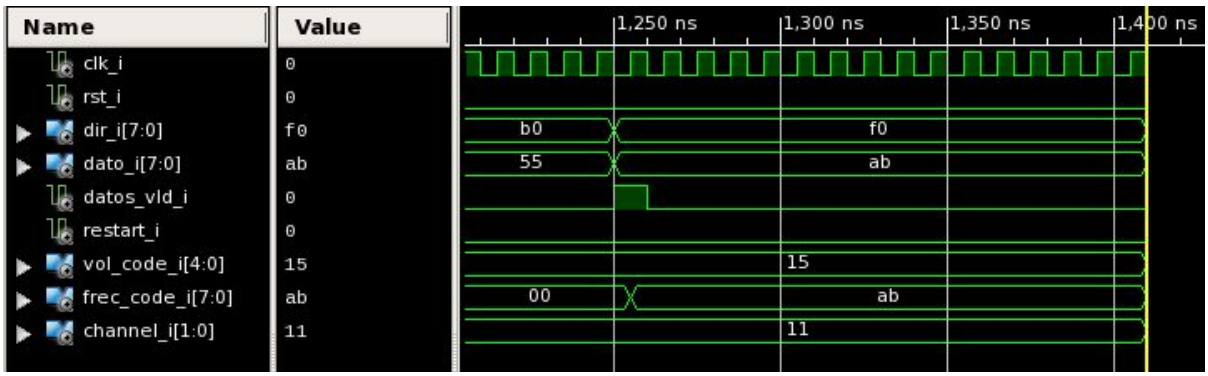


Figura 18. Captura al final de la simulación funcional

Una vez terminada la simulación se nos muestra por consola el fin de esta (figura 19), ya que no hemos hecho que se pudiera ejecutar indefinidamente y cambiando los datos "a mano" según fuera avanzando la simulación.

```
Console
ISim>
# run 1ms
Simulator is doing circuit initialization process.
Finished circuit initialization process.

** Failure:FIN CONTROLADO DE LA SIMULACION
User(VHDL) Code Called Simulation Stop
In process decoder_epp_tb.vhd:40

INFO: Simulator is stopped.
```

Figura 19. Captura de la consola

Simulación temporal de la entidad *decoder_epp*

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.

Comprobando la salida de las mismas señales que en la simulación funcional vemos que recoge perfectamente los valores en los puertos de salida según los valores de dirección y dato.

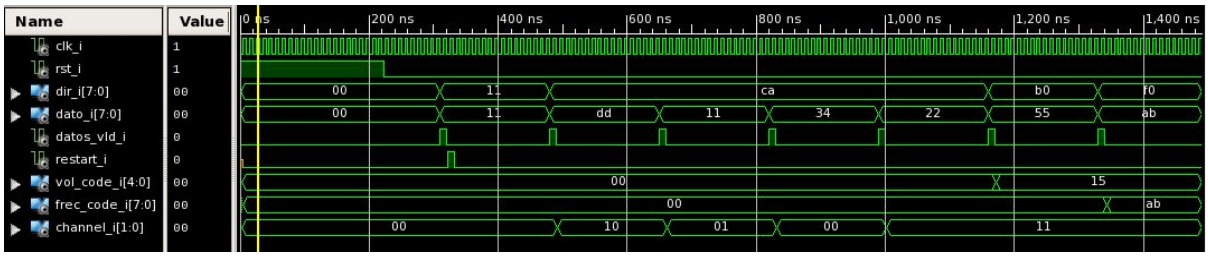


Figura 20. Captura de la simulación temporal

Como se puede observar en la figura 21, hay un retardo causado por la simulación temporal y muy similar al que puede producir la descarga en placa.

Figura 21. Captura de la simulación temporal al inicio

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.

Number of Slices Registers	2 out of 54,576 1%
Number of Slice LUTs	11 out of 27,288 1%
Number of occupied Slices	7 out of 6,822 1%

Figura 22. Tabla de recursos