Laboratorio de Sistemas Electrónicos Digitales

Universidad Politécnica de Madrid
Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación

Memoria del proyecto desarrollado en la asignatura "Diseño de circuitos y sistemas electrónicos" (DCSE)

Curso 2012/2013

"Procesamiento digital de señales en VHDL"

Autor : DAVID PORTILLA ABELLÁN

ÍNDICE GENERAL

1 Introducción

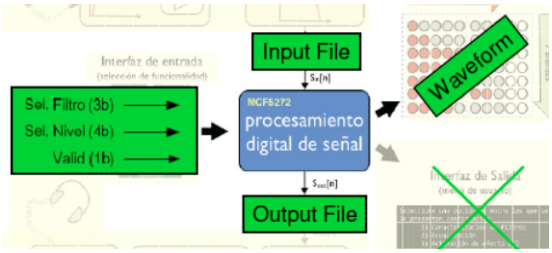
El proyecto consistía en programar un subsistema de ecualización de audio en VHDL. Aparte las mejoras, que consistían en implementar las estructuras de sumadores y multiplicadores estudiadas en la asignatura.

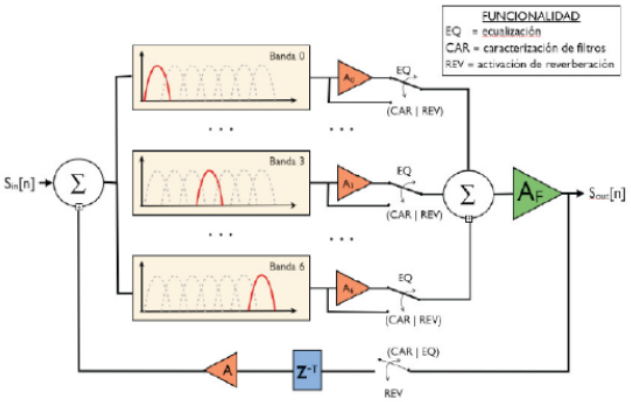
Para programar se ha utilizado el programa Modelsim. Para probar que todo funciona se han empleado las simulaciones que podemos hacer en Modelsim (visualización de formas de onda, ficheros de entrada y salida) y Matlab.

Para representar los números, se ha utilizado un formato de coma fija con 6 bits de signo y 10 bits de parte decimal. Los números negativos se representan en complemento a dos.

2 Diagrama de subsistemas

El sistema a desarrollar es el siguiente:





A través de un fichero de entrada con valores de una señal, el sistema procesa la señal y da un fichero de salida con los valores apropiados. Además se podrá visualizar la forma de onda de la energía de la señal de salida al simular en Modelsim.

Las entradas al sistema son:

- Señal de entrada
- Ecualización, caracterización y reverberación.
- Selector de nivel, selector de filtro y bit de validez.

Las ganancias de los amplificadores y el retardo son fijos.

3 Descripción del subsistema

3.1 Sumadores y multiplicadores

Esta parte es la que más problemas ha dado durante la realización del proyecto. Como operamos en complemento a dos y formato de coma fija, se ha optado por utilizar la librería **ieee.fixed_pkg.all;**

Partiendo de los sumandos a y b de tipo std_logic_vector(15 downto 0), el código para sumarlos es el siguiente:

```
variable a1, b1: sfixed(5 downto -10); -- 6 bits de parte entera, 10 de parte decimal
variable c1: sfixed(6 downto -10); --7 bits de parte entera, 10 de parte decimal
a1 := sfixed(a);
b1 := sfixed(b);
c1 := a1 + b1;
```

Después asignamos a la salida c los bits de c1 correspondientes a c1(5 downto -10) despreciando el primer bit.

Para las realizar las multiplicaciones se ha utilizado el mismo paquete y se ha seguido un desarrollo similar.

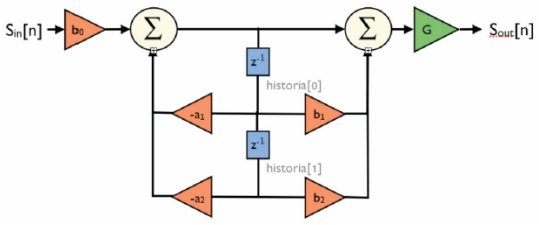
```
variable c1: sfixed(11 downto -20);
c1 := a1 * b1;
```

La salida serán los bits de c1 correspondientes a c1(5 downto -10).

3.2 Filtros

Se ha programado una entidad con una entrada y una salida de tipo **std_logic_vector**. Además la entidad tiene seis genéricos para los coeficientes y la ganancia G.

Como el filtro parte del reposo inicial, los registros (que retrasan la señal un ciclo de reloj) tienen una entrada de reset. Aunque no es necesario, los sumadores y multiplicadores también tienen un reset.



Filtro IIR. Forma directa II.

3.2.1 Pruebas de los filtros

Para probar el funcionamiento de los filtros se han empleado ficheros de texto de entrada y salida. Se ha comparado con la respuesta al impulso del mismo filtro en Matlab.

3.3 Amplificadores

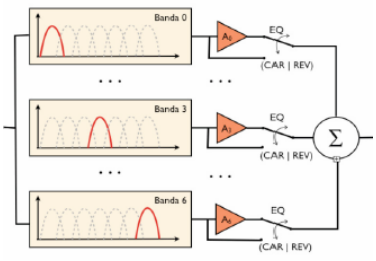
Tiene como entradas las salidas de las filtros y su salida es la salida final del sistema. Además tiene seis entradas más:

- Tres para programar los amplificadores: selector de filtro, selector de nivel y bit de validez.
- Tres para seleccionar el modo de funcionamiento: eq, car y rev

Para programar los amplificadores: cuando el bit de validez sea uno, se le asignará el nivel deseado al filtro seleccionado. Al hacer un *reset* del sistema, se ponen a cero todas las ganancias de los filtros.

De cara a conectar el vúmetro, hay siete salidas auxiliares, correspondientes a las siete ramas que entran al sumador.

Para el sumador de siete sumandos se ha utilizado una estructura en árbol de Wallace. Se han necesitado siete sumadores.



Esquema con los filtros y los amplificadores.

3.4 Reverberación

Para el sistema de reverberación se han necesitado varios componentes:

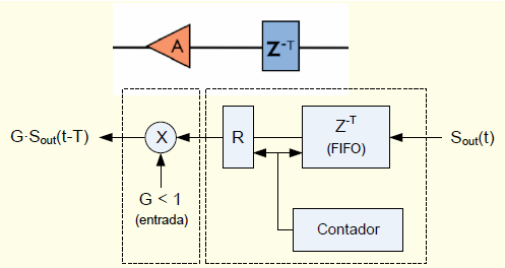
- Una cola FIFO. Introduce un retardo de T ciclos de reloj.
- Un contador. Tiene una entrada enable para poder parar la cuenta.
- Un registro para que antes de alcanzar el retardo no se realimente la salida de la FIFO, que sería un valor indefinido.

Con estos tres componentes, el proceso que controla todo es el siguiente:

```
controlador: PROCESS (clk) IS
BEGIN
   if rising_edge(clk) then
      if(unsigned(count) >= unsigned(retardo)) then
         retardo_alcanzado <= '1';
         enable <= '0'; -- Deja de incrementar el contador
      else
         retardo_alcanzado <= '0';
         enable <= '1';
      end if;
   end if;
END PROCESS controlador;
```

Cuando retardo alcanzado es ‘1’, entonces se extrae un valor de la FIFO cada flanco de subida del reloj. Además a partir de ese momento el registro deja de sacar ceros y envía a su salida lo que le llega de la FIFO.

Esta parte del subsistema de ecualización solo funciona en modo reverberación. En otro caso, se queda en circuito abierto (la salida del registro siempre vale cero).



Sistema de reverberación.

3.5 Vúmetro

Consta de siete entradas: cada una es una banda de frecuencias distinta. Tienen asociada una salida de 8 bits, y cada uno de ellos alumbraría un LED.

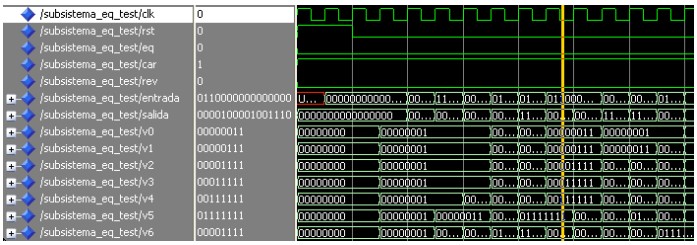
Se han definido siete umbrales: si la señal los supera todos, se encenderían los 8 LED’s.

Se ha programado una función que tiene como parámetro un vector de 15 bits. La función calcula su valor absoluto (si la entrada es negativa hace el complemento a dos), y mira entre qué umbrales está. Devuelve un vector de 8 bits, que sería la barra de LED’s.

En cada flanco de subida se aplica esta función a cada una de las siete entradas para actualizar el vúmetro. Al ser solo una simulación, no se ha visto necesario hacer un promedio de la energía y actualizar solo cada x muestras.

3.6 Sistema final

Se ha fijado una frecuencia de reloj de 1MHz, que sería suficiente para funcionar introduciendo retardos de nanosegundos en los componentes. Se muestra a continuación una simulación:



Simulación del sistema completo.

4 Descripción de las mejoras

Las mejoras consisten en programar unas estructuras de sumadores y multiplicadores reales.

Se han implementado los sumadores *ripple-carry*, *carry-bypass* y *carry-select*; y el multiplicador en array. Para cada uno de ellos se han empleado puertas lógicas, multiplexores, y sumadores de un bit (*full adders* y *half adders*).

Los *full adders* y los *half adders* están formados por puertas lógicas, que serán los elementos más básicos de estas estructuras, junto con multiplexores de un bit.

Para simular los retardos, cada puerta lógica y cada multiplexor tarda 1 ns desde el último cambio en las entradas hasta provocar cambios en la salida.

4.1 Elementos básicos: puertas lógicas y multiplexor de un bit

Solo ha sido necesario programar las puertas *and*, *xor* y *or*. Se muestra el código del proceso que hay en la arquitectura de la puerta *and*. Las otras dos son similares. Se utilizan la funciones lógicas de la librería ieee.std_logic_1164.

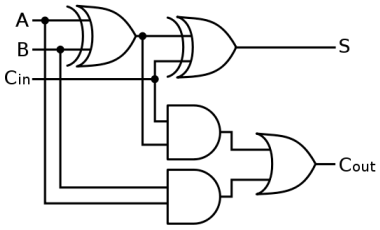
```
process (a, b) is
  begin
    c <= a and b after 1 ns;
end process;
```

El multiplexor de un bit simplemente manda la entrada cero a la salida si el selector vale cero, o la entrada uno si el selector vale uno tras 1 ns.

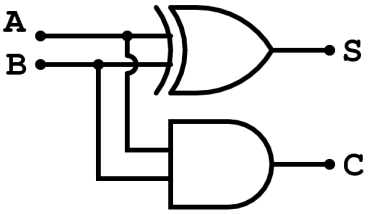
```
process (in0, in1, sel) is
 begin
   if sel = '0' then
     output <= in0 after 1 ns;
   elsif sel = '1' then
     output <= in1 after 1 ns;
   else null;
   end if;
end process;
```

4.2 Full adder, half adder

Sumadores de un bit con acarreo de entrada en un caso, sin acarreo de entrada en el otro. Están formados por puertas lógicas.



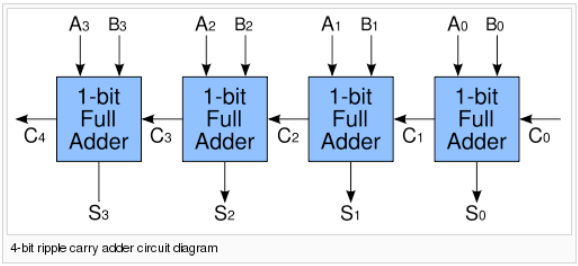
Full adder



Half adder

4.3 Sumador *ripple-carry*

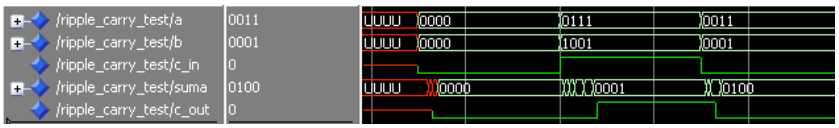
Teniendo programados los *full adders*, simplemente hay que montar este esquema:



4-bit ripple carry adder circuit diagram

Tiene dos entradas de tipo std_logic_vector (sumandos) y una de tipo std_logic (acarreo de entrada). Las salidas serán la suma y el acarreo de salida.

En la siguiente captura de pantalla, correspondiente a una simulación, se pueden apreciar los retardos.

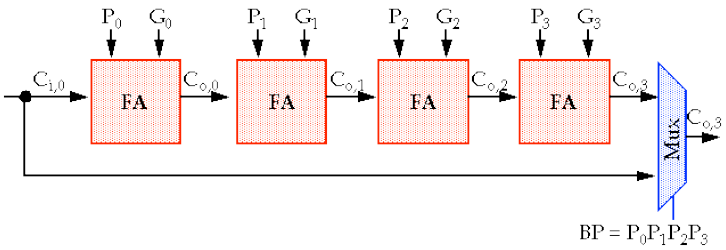


Forma de onda de la simulación.

4.4 Sumador *carry-bypass*

Está formado por un sumador *ripple-carry* y un multiplexor. El multiplexor está controlado por la señal BP.

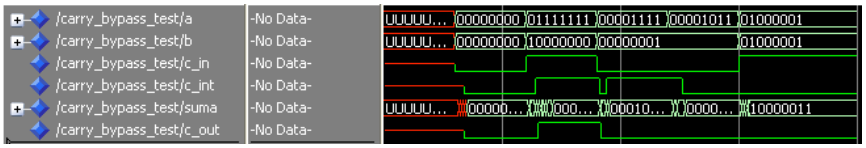
Se calcula la señal p (propagate) = a xor b, y a partir de ella BP = p_0 and p_1 and p_2 and p_3.



Si BP = 0 entonces la salida del multiplexor será igual a $C_{3,0}$, pero si BP = 1 entonces será igual a $C_{i,0}$, lo cual supone un ahorro importante de los retardos de propagación.

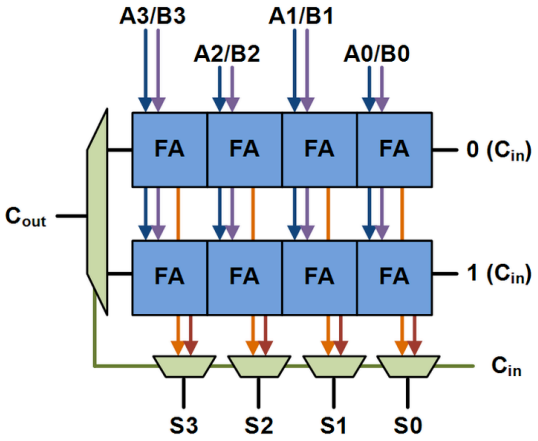
A diferencia del sumador *carry-bypass* que se muestra en la figura, las entradas a los *full adders* en el diseño programado son los sumandos A y B, no P y G.

Se ha simulado un sumador de 8 bits con dos etapas conectadas en serie.



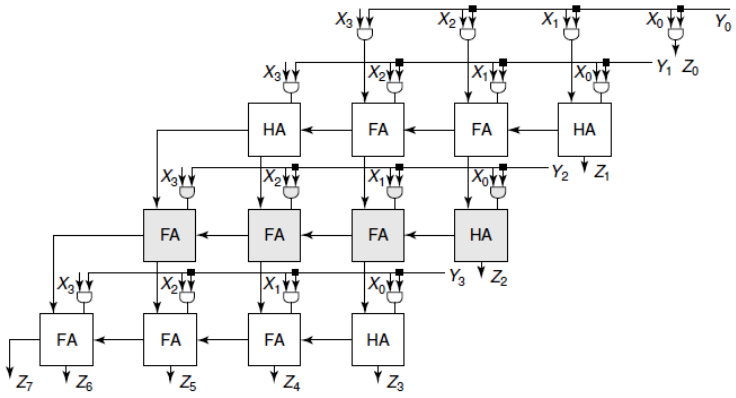
4.5 Sumador *carry-select*

Combinando multiplexores de un bit y *full adders* conseguimos este sumador. Se ha programado siguiendo el esquema de la figura.

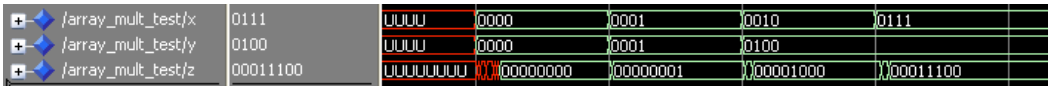


4.6 Multiplicador en array

Se ha implementado esta estructura:



Simplemente se ha creado una entidad con todos los full adders, half adders y puertas lógicas, y se han ido propagando las señales intermedias.



Simulación del multiplicador en array.