PROYECTO FINAL:
GENERADOR DE EFECTOS DE AUDIO USANDO EL HDL CODER DE SIMULINK

Sherneyko Plata Rangel
e-mail: shpr2005@gmail.com

RESUMEN:*Este proyecto final muestra las capacidades y defectos del hdl coder de simulink, y como puede implementarse esta herramienta de prototipado rápido para un procesamiento de señales relativamente complejo en el dominio de frecuencia discreta (z).*

PALABRAS CLAVE: Simulink, matlab, HDL coder,fpga..

1 INTRODUCCIÓN

.

El procesamiento digital de señales, y en este caso, el de audio, es una tarea exigente en cuanto a recursos y velocidad,desde sus inicios se han usado integrados especializados para esta tarea, sin embargo con los avances de las ultimas décadas dispositivos programables han adquirido las capacidades para realizar estos procesos, entre ellos se encuentra la FPGA, que por sus capacidades y estructura pueden ser configuradas para realizar tareas de forma similar a los asic,debido a esto y a su paralelismo, se han hecho muy populares en este campo.

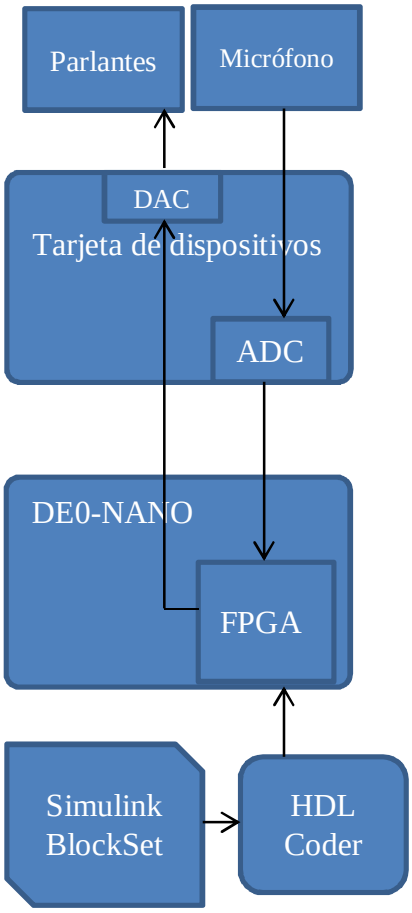
Describir la lógica para este procesamiento puede llegar a ser tedioso sobre todo si el numero de bloques, funciones y el orden de las mismas es alto, sin embargo existen herramientas con la capacidad de generar un código hdl en base a un diagrama de bloques que ilustra de forma simplificada las tareas a realizar sobre los datos,facilitando asi su implementación, y permitiendo que el diseñador se concentre mas en el diagrama teorico y no en su traducción a HDL.En este proyecto se implementa un generador de efecto basado en simulink y generador de código HDL.

2 PROPUESTA

Mediante simulink, realizar un procesamiento digital de señales que tome una señal de audio de entrada mediante un adc, genere los efectos sobre esta y la convierta de nuevo al mundo analógico con un dac, todo esto usando una tarjeta de dispositivos diseñada para la tarjeta de desarrollo DE0-NANO.

Hay que tener en cuenta las imperfecciones del código generado por simulink, cuya lógica es correcta pero no es confiable, generando dispositivos indeseados como latches en el compilador de la fpga.

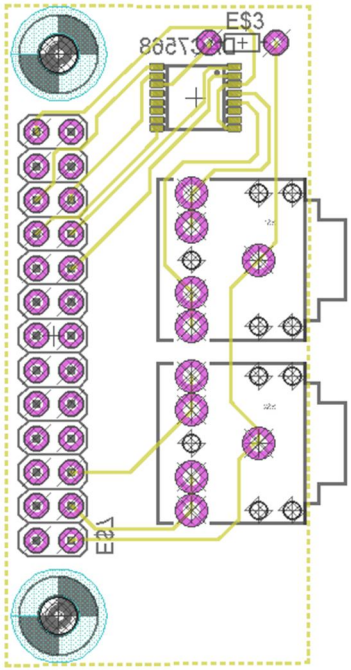
3 DIAGRAMA



4 ETAPAS

- Diseño del PCB de la tarjeta de dispositivos en Eagle pcb

Se realizo el diseño del siguiente pcb:



El cual contiene únicamente un dac7568 de Texas instruments, y usa el adc de la de0-nano para obtener los datos de audio, sin embargo tiene un inconveniente, y es el hecho de que no posee referencia negativa, por tanto para su correcto uso debe ajustarse primero la señal a un rango positivo (adc), sin embargo y para evitar desviarnos del objetivo principal de este proyecto se realizo un diseño alterno en protoboard con un dac0808, para asi tener un respaldo en caso de falla.

- Descripcion de los modulos ADC y DAC

Se utilizo el modulo adc suministrado con la board , autoria de terasic, y se instancio de la siguiente manera:

```
SPIPLL      U0 (
                .inclk0(CLOCK_50),
                .c0(wSPI_CLK),
                .c1(wSPI_CLK_n)
            );
ADC_CTRL    U1 (
                .iRST(KEY[0]),
                .iCLK(wSPI_CLK),
                .iCLK_n(wSPI_CLK_n),
                .iGO(KEY[1]),//
                .iCH(1'b0),
                .oLED(LED),
                .oDIN(ADC_SADDR),
                .oCS_n(ADC_CS_N),
                .oSCLK(ADC_SCLK),
                .iDOUT(ADC_SDAT),
                .salida(adc_out)
            );
```

El modulo spipll se encarga de generar el reloj principal del modulo adc_ctrl.

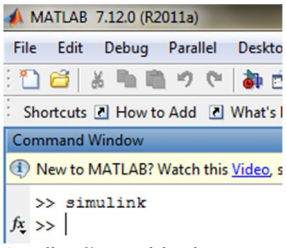
El modulo adc_ctrl se encarga de realizar la lectura de valores desde el adc, y de organizarlos de forma paralela para su uso en el diseño.
iRST➔Reset asincrono
iCLK➔Reloj principal
iCLK_n➔Reloj principal negado (se usa como entrada aparte para evitar retrasos de propagación diferentes a iCLK.
iGO➔Inicio de conversión.Se asigno al switch para realizar pruebas
iCH➔Canal de lectura.Se usara permanentemente el canal 0
oLED➔MSB's del dato,se imprimen en leds para confirmar su llegada a la fpga.
oDIN,oCS_n,oSCLK,iDOUT➔Pines del protocolo SPI, que es el que maneja el dac.
Salida➔puerto creado para tener la capacidad de obtener todo el dato y no solo los MSB de oLED.

Nota: la convención iVARIABLE y oVARIABLE es útil en programas extensos para recordar que función cumple el pin(entrada o salida).

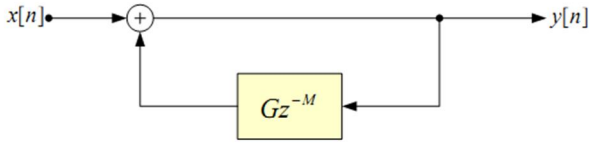
Para el dac7568 se realizo el siguiente modulo:

Para el dac0808 no es necesario modulo ya que es asíncrono , su manejo es similar a una red R-2R.

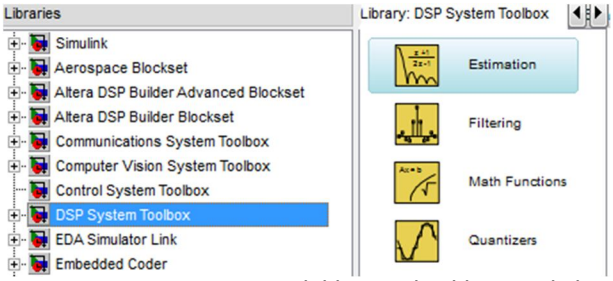
- Diseño del blockset



Se diseño un blockset en simulink basándose en el efecto teorico de un delay o reverb.Cabe aclarar que su creación es muy similar y solo varia en el tiempo de retraso entre muestras siendo el reverb mas corto que el delay.

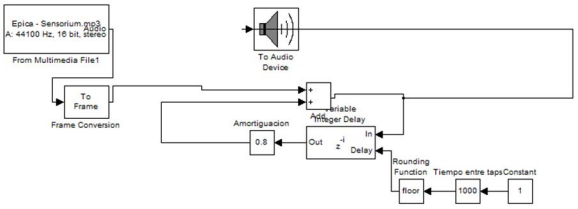


En este caso M es el numero de muestras y dependiendo de su magnitud escucharemos un delay o un reverb.G es la ganancia de eco y debe ser menor que uno ya que de lo contrario podemos saturar el sonido. El bloque utilizado tiene como nombre delay, y existen varios tipos del mismo en simulink.
Es importante depurar al máximo antes de convertir el blockset a hdl ya que esto ahorra complicaciones. El entorno grafico de simulink facilita el aprendizaje ya que se compone de bloques ya funcionales.



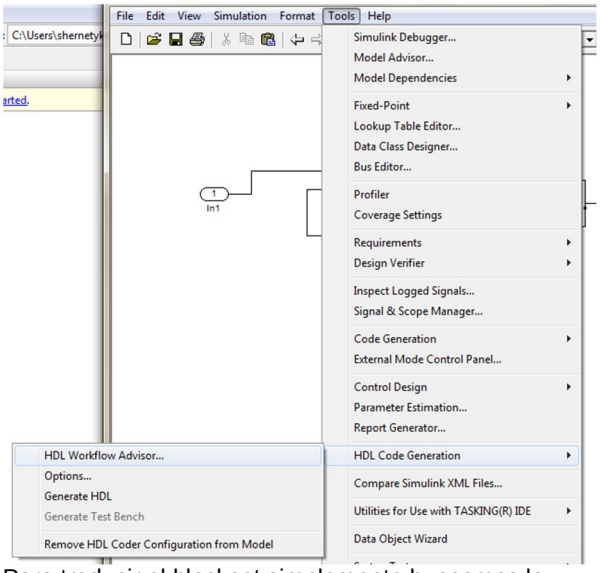
Para este caso es recomendable usar los bloques de la toolbox dsp system, ya que sus elementos (como filtros de n tipo) están especializados en procesar datos discretos

Con ese conocimiento previo tenemos entonces se diseño el siguiente blockset



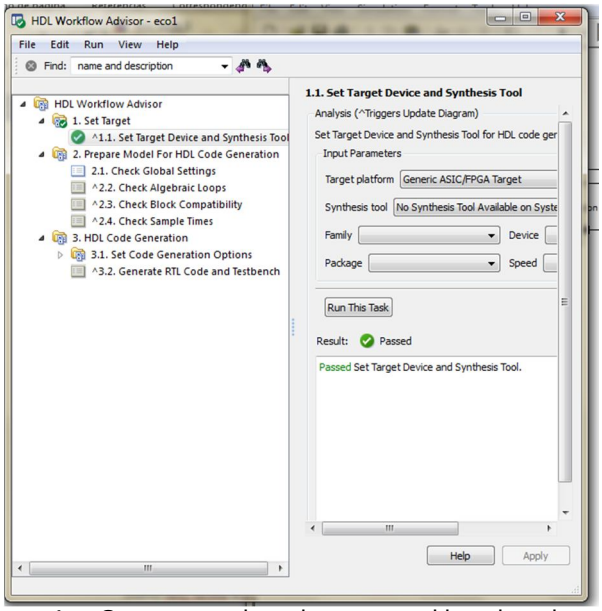
Que virtualmente puede crear un eco de cualquier duración y atenuación.Sin embargo existen problemas en su traducción como veremos a continuación.

- Traduccion del blockset

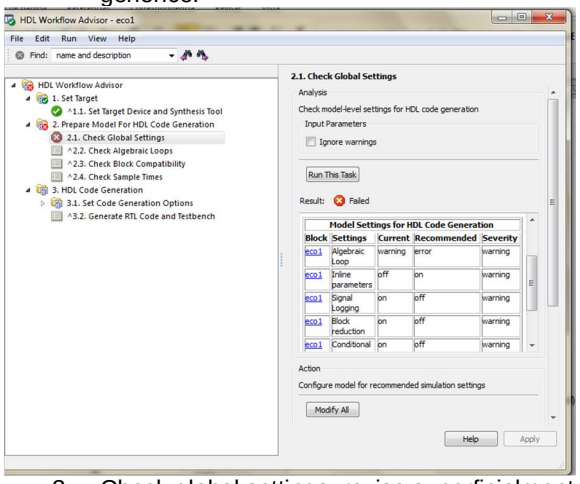


Para traducir el blockset simplemente buscamos la opción geneate HDL en el menú tolos de simulink, sin embargo, para depurar nuestro diagrama de bloques recomiendo primero la opción de workflow advisor que señala y corrige algunos de los errores mas comunes como:

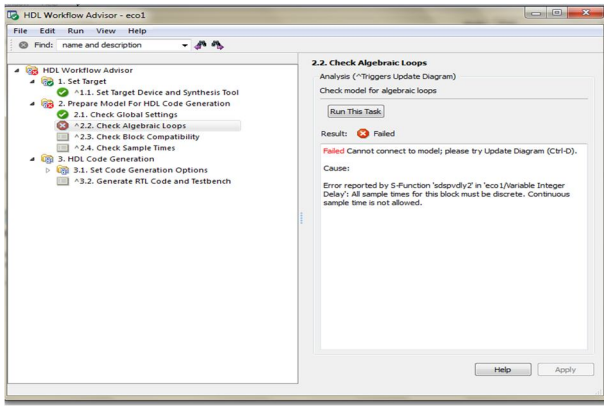
- Bloques no sintetizables
- Bucles algebraicos(algebraic loops)
- Tiempos de muestreo o herencia de los mismos incorrecta (todo el sistema debe funcionar a un mismo reloj,facilita la traduccion)



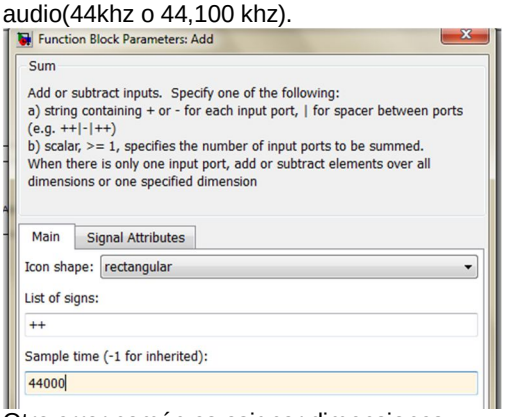
1. Set target: solo es importante si la tarjeta de desarrollo que usemos es reconocida por matlab, como no es el caso, lo dejaremos genérico.



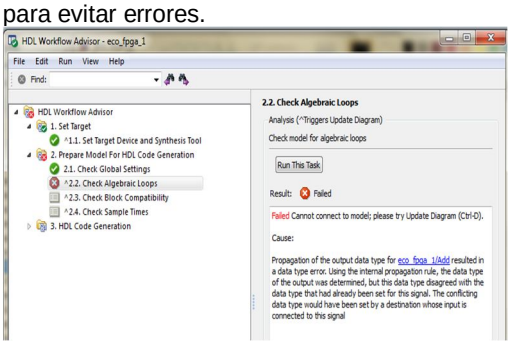
2. Check global settings: revisa superficialmente el blockset.En este caso encontró un error y varios warnings.Nos encargaremos del error ya que los warnings pueden corregirse en base al código generado.Presionamos modify all para autocorregir, si el error persiste hay que reestructurar el blockset



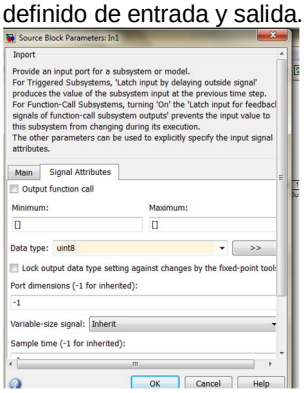
3. Check algebraic loops: revisión lógica del blockset. Un error muy común en este paso es tiempo de muestreo incorrecto.En realidad este error sucede por que simulink no conoce ningún tiempo de muestreo de ningún bloque ya que los hemos dejado todos en -1(heredado), para corregir esto basta con buscar un bloque al principio de todo el blockset y asignarle una fq de muestreo, para que asi los que están a continuación del mismo hereden dicha frecuencia.Simulink no crea divisores, esta frecuencia de muestreo solo la solicita para verificar que el sistema funcione bien.A continuación cambiamos la fq de muestreo del bloque add de -1 a 44000, que es una frecuencia común de muestreo de audio(44khz o 44,100 khz).

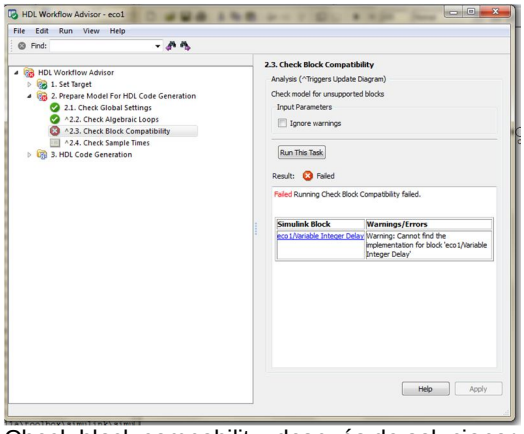


Otro error común es asignar dimensiones erróneas a las entradas y salidas de un bloque, o dejar que simulink las herede de bloques anteriores.Es recomendable asignarlas todas para evitar errores.

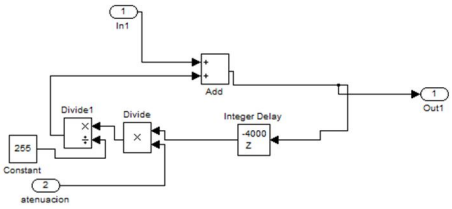


Por ejemplo un producto entre 2 uint8, da un uint16, y también es imposible heredar si no hay por lo menos un bloque con tamaño definido de entrada y salida.

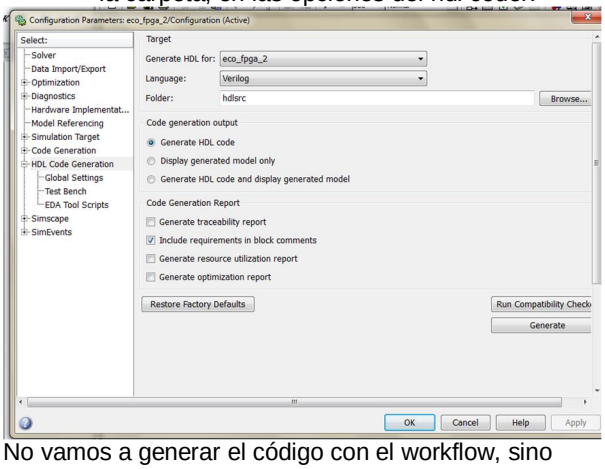




4. Check block compability: después de solucionar problemas varios del blockset, simulink revisara si los bloques usados son sintetizables, en nuestro caso, variable integer delay, el bloque principal de nuestro programa y el fundamento sencillo del eco , no es sintetizable, y tiene sentido, no existe un registro de tamaño variable en ejecución.Entonces simplificamos el modelo a un delay constante , el cual luego editaremos para hacerle variable.(el hdl coder es limitado).



5. Pasando de nuevo por los pasos anteriores, verificamos que tenemos un blockset sintetizable , además con amplitud de eco variable (eso es bueno).Configuramos el hdl coder para que trabaje en verilog y escogemos la carpeta, en las opciones del hdl coder.



No vamos a generar el código con el workflow, sino directamente en tools>generate hdl (es mas sencillo).Esperamos a que simulink termine su proceso(en este caso es un poco demorado por el numero de registros utilizados).Al terminar vamos a la carpeta que seleccionamos y abrimos el código.

6. Revision del código hdl generado (confiabilidad)

```
2082    assign Integer_Delay_reg_next[3993] = Integer_Delay_reg[3992];
2083    assign Integer_Delay_reg_next[3994] = Integer_Delay_reg[3993];
2084    assign Integer_Delay_reg_next[3995] = Integer_Delay_reg[3994];
2085    assign Integer_Delay_reg_next[3996] = Integer_Delay_reg[3995];
2086    assign Integer_Delay_reg_next[3997] = Integer_Delay_reg[3996];
2087    assign Integer_Delay_reg_next[3998] = Integer_Delay_reg[3997];
2088    assign Integer_Delay_reg_next[3999] = Integer_Delay_reg[3998];
2089
2090
2091
2092    assign Divide_out1 = Integer_Delay_out1 * atenuacion;
2093
2094
2095
2096    assign Divide1_div_temp = (Constant_out1 == 0 ? 17'b11111111111111111 :
2097                Divide_out1 / Constant_out1);
2098    assign Divide1_out1 = (Divide1_div_temp[16:8] != 0 ? 8'b11111111 :
2099                Divide1_div_temp[7:0]);
2100
2101
2102
2103    assign Add_out1 = In1 + Divide1_out1;
2104
2105
2106
2107    assign Out1 = Add_out1;
2108
```

El hdl coder no es mas que un programa y si bien puede crear códigos sintetizables, puede que estos códigos sean exageradamente largos.Sin embargo esto no influye en el funcionamiento del bloque.Ahora si hay defectos que pueden llegar a influir como:

```
73
74    assign Constant_out1 = 255;
75
76
77
78    always @(posedge clk or posedge reset)
79      begin : Integer_Delay_process
80        if (reset == 1'b1) begin
4082      else begin
4083        if (enb) begin //aca falta el else de cerrar!!
8085      end
8086    end
8087
8088    assign Integer_Delay_out1 = Integer_Delay_reg[3999];
8089    assign Integer_Delay_reg_next[0] = Add_out1;
8090    assign Integer_Delay_reg_next[1] = Integer_Delay_reg[0];
8091    assign Integer_Delay_reg_next[2] = Integer_Delay_reg[1];
8092    assign Integer_Delay_reg_next[3] = Integer_Delay_reg[2];
8093    assign Integer_Delay_reg_next[4] = Integer_Delay_reg[3];
8094    assign Integer_Delay_reg_next[5] = Integer_Delay_reg[4];
8095    assign Integer_Delay_reg_next[6] = Integer_Delay_reg[5];
8096    assign Integer_Delay_reg_next[7] = Integer_Delay_reg[6];
8097    assign Integer_Delay_reg_next[8] = Integer_Delay_reg[7];
8098    assign Integer_Delay_reg_next[9] = Integer_Delay_reg[8];
8099    assign Integer_Delay_reg_next[10] = Integer_Delay_reg[9];
8100    assign Integer_Delay_reg_next[11] = Integer_Delay_reg[10];
8101    assign Integer_Delay_reg_next[12] = Integer_Delay_reg[11];
```

Sin embargo podemos compilar y revisar si el compilador corrige este error revisando cuidadosamente que no haya inferido latches en ningún lado.Afortunadamente para este código ese es el caso, no se tuvo la necesidad de modificarlo para su funcionamiento.

5. Pruebas

En la primera prueba el sonido salía con mucho ruido,y al ver en un osciloscopio se verifico que era un problema de signo, revisando el código:

```
2082    assign Integer_Delay_reg_next[3993] = Integer_Delay_reg[3992];
2083    assign Integer_Delay_reg_next[3994] = Integer_Delay_reg[3993];
2084    assign Integer_Delay_reg_next[3995] = Integer_Delay_reg[3994];
2085    assign Integer_Delay_reg_next[3996] = Integer_Delay_reg[3995];
2086    assign Integer_Delay_reg_next[3997] = Integer_Delay_reg[3996];
2087    assign Integer_Delay_reg_next[3998] = Integer_Delay_reg[3997];
2088    assign Integer_Delay_reg_next[3999] = Integer_Delay_reg[3998];
2089
2090
2091
2092    assign Divide_out1 = Integer_Delay_out1 * atenuacion;
2093
2094    assign Divide1_div_temp = (Constant_out1 == 0 ? 17'b11111111111111111 :
2095                Divide_out1 / Constant_out1);
2096    assign Divide1_out1 = (Divide1_div_temp[16:8] != 0 ? 8'b11111111  : //8'b11111111
2097                Divide1_div_temp[7:0]);
2098
2099
2100   assign Add_out1 = In1[7:1] + Divide1_out1[7:1];
2101
2102   assign Out1 = Add_out1;
2103
2104   assign ce_out = clk_enable;
2105
2106 endmodule  // eco_fpga_2
2107
2108
```

No estaba considerando el carry de la suma entonces, se tomaron los 7 mas significativos y se sumaron para que asi el carry cupiera en la salida.

Ya con el sonido base correcto, podemos empezar a variar parámetros.La instanciación del modulo de simulink fue la siguiente

```
eco_fpga_2 eco_fpga_2_inst
(
   .clk(clk_eco) ,   // input  clk_sig
   .reset(1'b0) , // input  reset_sig
   .clk_enable(1'b1) ,  // input  clk_enable_sig
   .In1(adc_out[11:4]) ,   // input [7:0] In1_sig
   .atenuacion({SW,4'd0}) ,   // input [15:0] atenuacion_sig 240
   .ce_out() , // output  ce_out_sig
   .Out1({GPIO_2[11],GPIO_2[9],GPIO_2[7],GPIO_2[5],GPIO_2[3],GPIO_2[1],GPIO_2[2],GPIO_2[0]})
);
```

Clk_eco va a originarse en un divisor y nos va a permitir variar el tiempo de delay, aunque introducirá un poco de distorsion (ya que este no es el método apropiado para realizar el delay, pero es un concepto muy cercano).

Experimentalmente se comprobó que :

Frecuencia_clk_eco<<f_muestreo ➔ eco
Frecuencia_clk_eco<f_muestreo ➔ reverb
Frecuencia_clk_eco=f_muestreo ➔ no efecto

El rango de valores del divisor va de 567 a 5000 (aproximadamente)

Tambien la atenuación juega un papel importante, ya que determina cuanto se disminuye el eco por rebote.Entre mas grande sea este valor menos se amortiguara el eco.No puede ser igual a 255 ya que el sistema seria inestable (no se atenuaría)

Ya con estos efectos básicos funcionando, podemos mejorar el sonido realizando un verdadero delay variable con suficientes muestras como para que no se sature la onda.Para ello se podría usar la sdram externa de la de0-nano, sin embargo simulink necesitaría de bloques dsp especiales para realizar el manejo de la memoria y por ello no fue implementado.