Laboratorio de DSP y FPGA

Trabajo Práctico N° 2

Grupo 2

KAMMANN, Lucas Agustín

FARALL, Facundo David

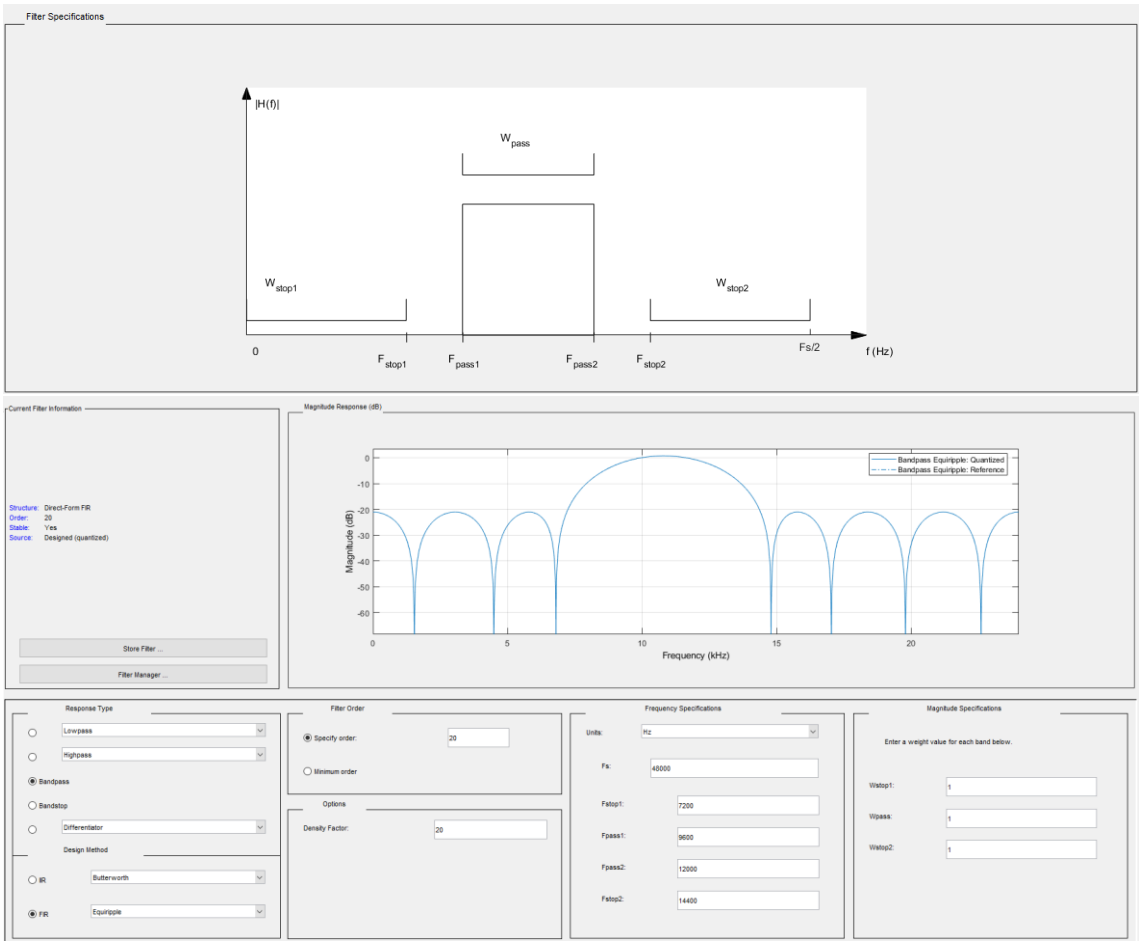
DAVIDOV, Gonzalo Joaquín

TROZZO, Nicolás Rafael

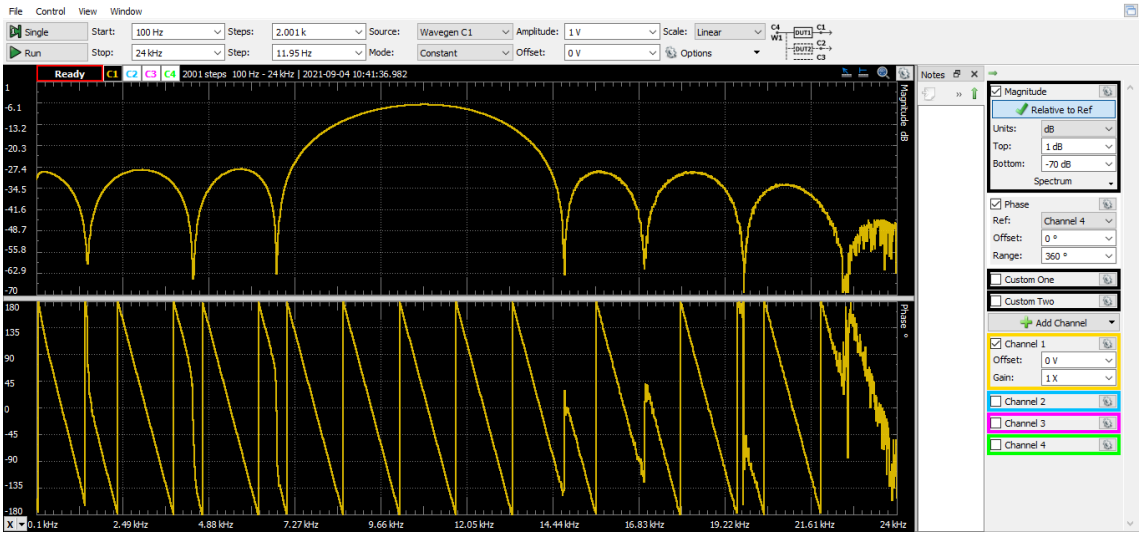
Ejercicio 1

Item b

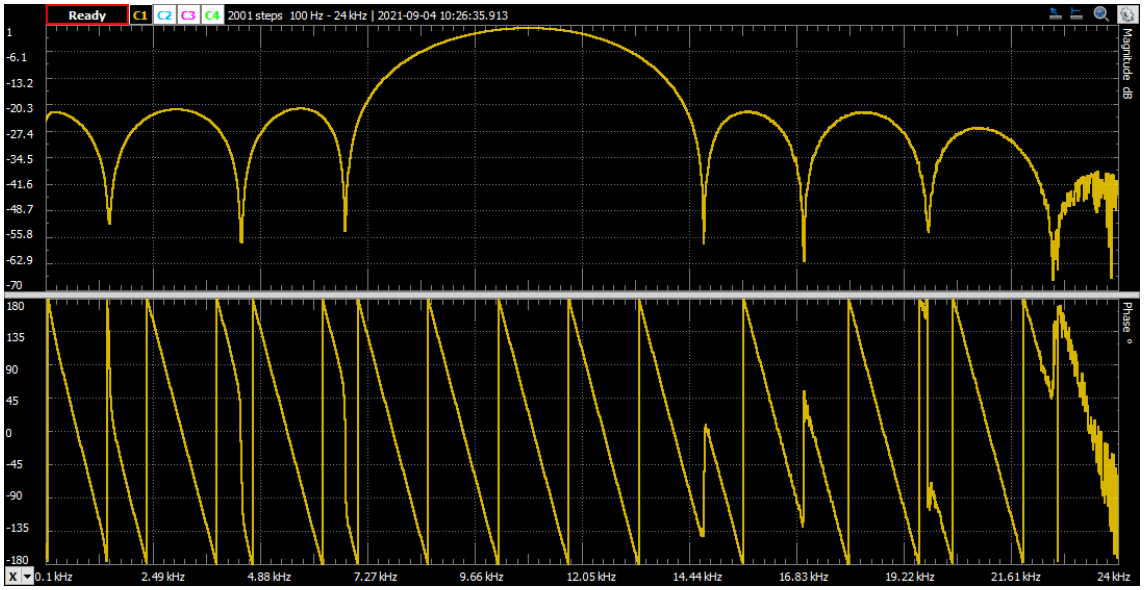
Utilizando la herramienta FilterDesigner de Matlab se diseña un filtro FIR pasa banda de orden 20 que cumpla la siguiente plantilla para una frecuencia de muestreo fs=48kHz y se obtienen los correspondientes coeficientes.



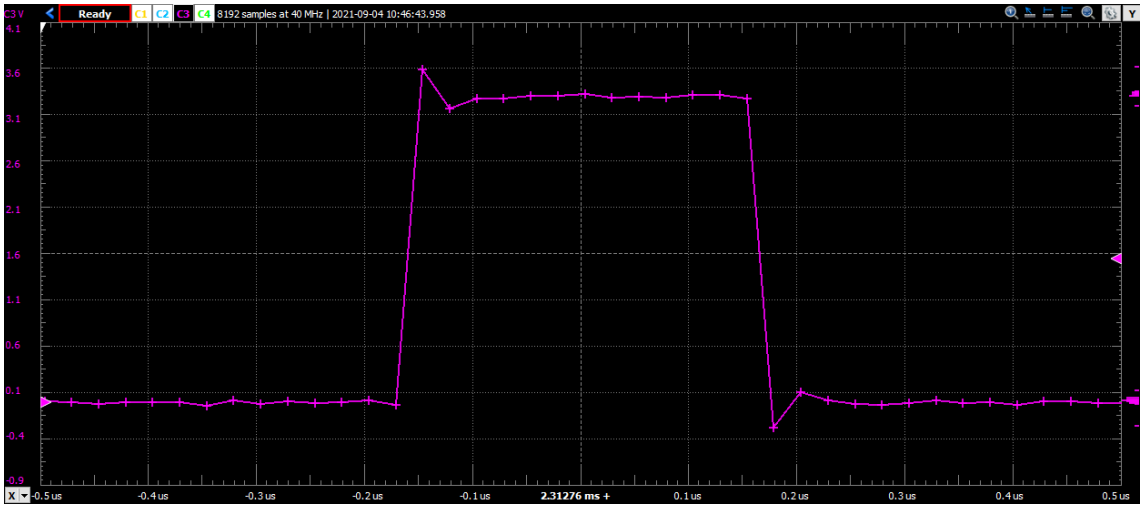
Al implementar el filtro en el DSP se obtuvo la siguiente respuesta en frecuencia:



En donde, como se puede ver, la amplitud de la banda de paso está por debajo de los 0dB debido a atenuaciones internas presentes en el circuito. Para solucionar esto y poder ver los resultados con mayor claridad se multiplica por un factor de compensación de 2,2 para que la banda de paso esté por encima de los 0dB. Las deformaciones con respecto al filtro deseñado en MATLAB se deben a la respuesta propia del DSP en ausencia de filtro (es decir, no es una respuesta plana para todo este rango de frecuencias, ya que, de base, se cuenta con el filtro antialias).



Finalmente, con el filtro ya diseñado y compensado, se procede a medir el tiempo de procesamiento de la interrupción resultando en t_{int}=324.39ns.



Item c

Se busca diseñar un filtro FIR del tipo sinc inverso pasa altos del mayor orden posible teniendo en cuenta la frecuencia de muestreo y la cantidad de memoria a utilizar.

Se usa una f_s = 48kHz. Sabiendo que el clock fue configurado a 86MHz aproximadamente, y que las instrucciones que corren en la rutina de interrupción son:

```
  movep   #$0001,X:M_HDR  ;1->PB0, sube el pin
  fir     ntaps           ;do fir
  movep   #$0000,X:M_HDR  ;0->PB0, baja el pin

fir     macro   ntaps
fir     ident   1,0
;
;       FIR - direct form FIR filter
;
;       FIR filter macro
;



        clr     a    x0,x:(r0)+  y:(r4)+,y0     ;save first state
        rep     #ntaps-1
        mac     x0,y0,a  x:(r0)+,x0  y:(r4)+,y0
        macr    x0,y0,a  (r0)-



        endm
```

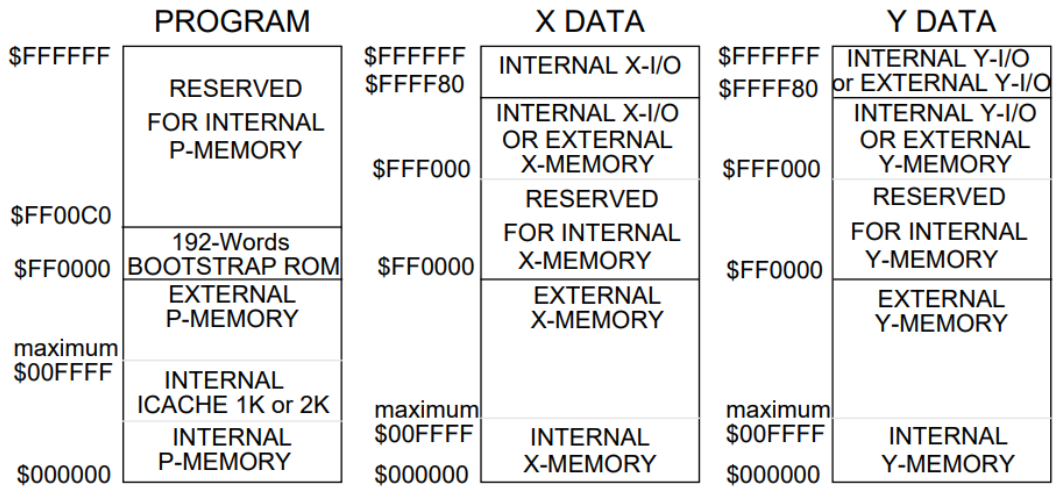
Se obtuvo que el orden máximo implementable en la DSP es 1786 utilizando el 100% del duty cycle, teniendo en cuenta cuántos ciclos de máquina toma cada instrucción. Se decidió dejar un margen y realizar un filtro de orden 1720 que utiliza el 96%.

```
fs = 48e3
cycle_time = 1/86e6
max_taps = ((1 / fs) / cycle_time - 2 - 1 - 1 - 1) / 1
print(max_taps)
```

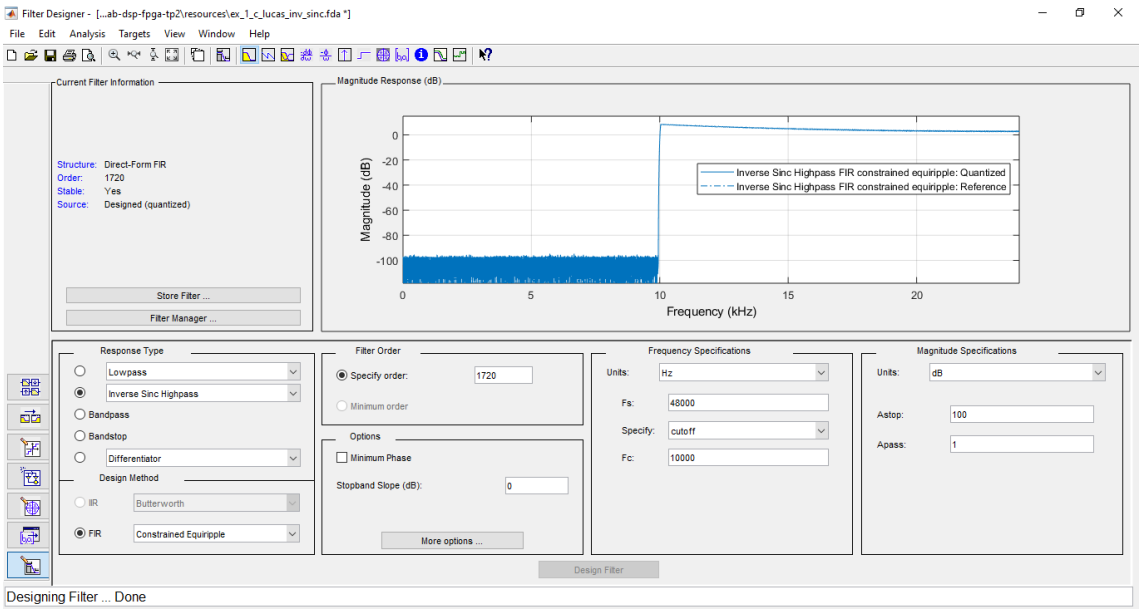
```
1786.6666666666665
```

Orden máximo debido a fs.

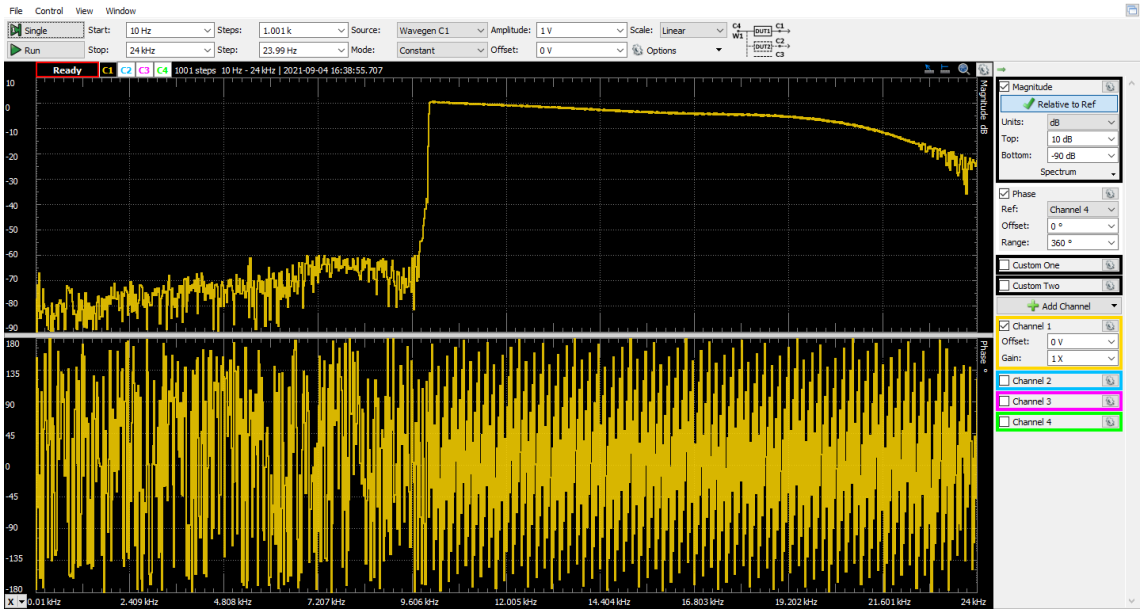
También fueron verificadas restricciones por tamaño de memoria utilizada.
Teniendo en cuenta el mapa de memoria a continuación, y sabiendo que los coeficientes en el programa actual comienzan a cargarse desde la posición Y:0, se tiene espacio de memoria inter na (con la cual es posible el parallel move) hasta Y:$00FFFF, es decir aproximadamente 65.500 palabras de 24 bits de los cuales solo utilizaremos 1720 espacios en la memoria Y, y luego la misma cantidad de memoria utilizada para los valores n-i (i=0,…,1719) que utiliza el filtro en el espacio de memoria X.



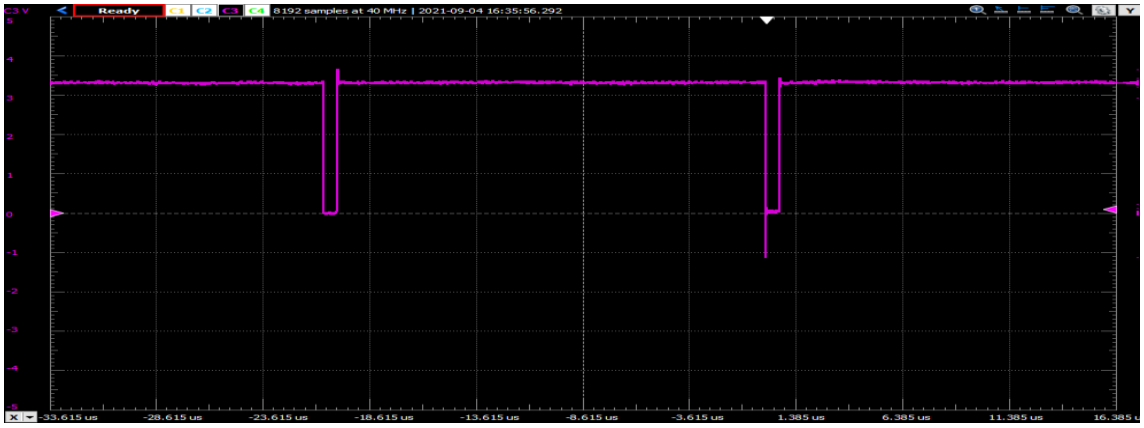
Teniendo en cuenta las restricciones previamente mencionadas, la más restrictiva es la dada por el tiempo de sample, y se procede a diseñar un filtro acorde a las conclusiones obtenidas.



A continuación, se puede observar el funcionamiento y la respuesta en frecuencia de este al implementarlo en la DSP.



Asimismo, se observa el duty cycle cercano al 96%, en concordancia con los valores calculados previamente



Duty cycle con filtro de orden 1720.

Ejercicio 2

El objetivo de este ejercicio es desarrollar el código fuente para ejecutar un sistema reverberador como se ilustra en la **Ilustración 1: Esquema reverberador**, es decir, dada una señal de audio de entrada, el sistema le agrega un efecto de reverberación, simulando la respuesta de una habitación por los reflexiones y ecos producidos en la misma.

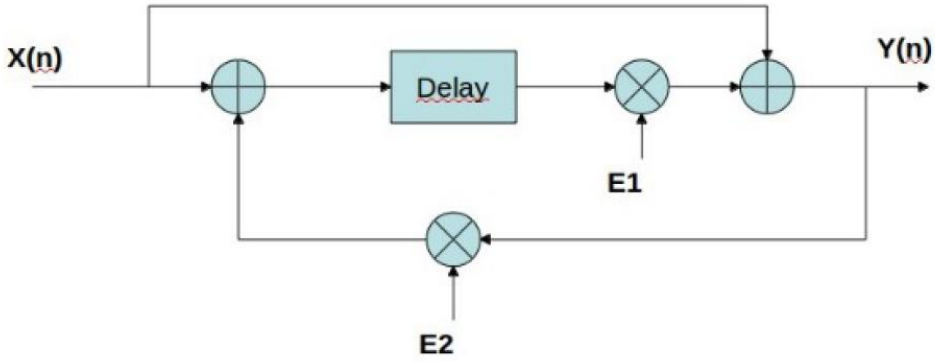


Ilustración 1: Esquema reverberador

El sistema propuesto por el esquema en bloques busca simular la reverberación. Para esto, la señal de salida siempre tiene una componente de la **señal directa**, y además las señales que son **reflexiones**. El bloque de retardo temporal busca modelar el tiempo que se tarda la señal en el espacio modelado para reflejar, y los coeficientes modelan la proporción de amplitud reflejada.

Para el diseño del algoritmo, se definió una señal auxiliar

$$W(n) = E_2 \cdot Y(n) + X(n)$$

*Ecuación 1: Señal auxiliar **W(n)***

De esta forma, la ecuación completa del esquema será

$$Y(n) = E_1 \cdot W(n - N) + X(n)$$

Ecuación 2: Cálculo de la salida del reverberador

El bloque de retardo temporal de la señal W(n) es implementado con un buffer en memoria, almacenando las muestras [W(n-1), …, W(n-N)]. El buffer es configurado como un buffer circular al cual se accede por un registro R que siempre apunta al final del mismo, es decir, a la muestra más antigua. En el siguiente esquema se ilustra la tabla de memoria que corresponde al buffer para implementar el bloque de retardo temporal, en el diagrama **izquierdo** se representa el estado inicial para el ciclo en un instante "n". El diagrama **derecho** representa el estado final para el ciclo en un instante "n". Para configurar este modo se cargó el registro modificador M.

R0 ➔	**W(n-N)**
	W(n-N+1)
	…
	W(n-2)
	W(n-1)

	W(n)
R0 ➔	**W(n-N+1)**
	…
	W(n-2)
	W(n-1)

Para un instante n, en primer lugar, se toma la muestra $W(n-N)$ del buffer y se calcula la salida $Y(n)$, con la **Ecuación 2: Cálculo de la salida del reverberador**. Luego, se computa el valor de $W(n)$ mediante la **Ecuación 1: Señal auxiliar W(n)** para ser guardado en el buffer, valor que será utilizado N muestras más adelante.

En la **Ilustración 2: Subrutina reverb implementada** se muestra el código fuente desarrollado para implementar la subrutina **reverb**, que permite tomar una muestra de entrada y calcular la muestra de salida.

```
; ================================================================
; reverb
;
; Given the current sample in X0, the reverberated output is returned
; in the A accumulator. The reverberation has a delay of N samples.
;
; @requires Space of memory to store past samples being pointed by R0
;           with a length or module of M0+1.
; @param    R1 Pointer to the E1 coefficient in the X space
; @param    R2 Pointer to the E2 coefficient in the X space
; @param    X0 Input sample
; @return   A Output sample
; ================================================================
          ; Compute Y(n) and save it into the accumulator
reverb    MOVE    X:(R1),Y0       ; Y0 = E1
          MOVE    X:(R2),Y1     ; Y1 = E2
          MOVE    X:(R0),X1       ; X1 = W(n-N)
          MPY     X1,Y0,A         ; A = W(n-N).E1
          ADD     X0,A            ; A = W(n-N).E1 + X(n) = Y(n)
          NOP

          ; Compute W(n) and save it into the delay buffer
          MOVE    A,X1            ; X1 = Y(n)
          MPY     X1,Y1,B         ; B = Y(n).E2
          ADD     X0,B            ; B = Y(n).E2 + X(n)
          NOP
          MOVE    B,X:(R0)+       ; W(n) = Y(n).E2 + X(n)

          RTS
```

*Ilustración 2: Subrutina **reverb** implementada*

Es de interés mencionar que fue necesario agregar instrucciones NOP para evitar problemas de dependencias entre instrucciones dentro del pipeline a la hora de leer y cargar el contenido de los acumuladores. Además, en el código de inicialización (previo a la llamada de la subrutina), se configuraron los registros para utilizar **4096 taps**, **E1=0.7** y **E2=0.4**.

En la **Ilustración 3: Respuesta del reverberador** se observa la respuesta del reverberador medida a partir de una entrada periódica configurada como un tren de pulsos de muy baja frecuencia y un ciclo de trabajo muy bajo, simulando así un tren de deltas. Es importante notar que a medida que pasa el tiempo la amplitud va decayendo, logrando el conocido perfil de retardos de la reverberación

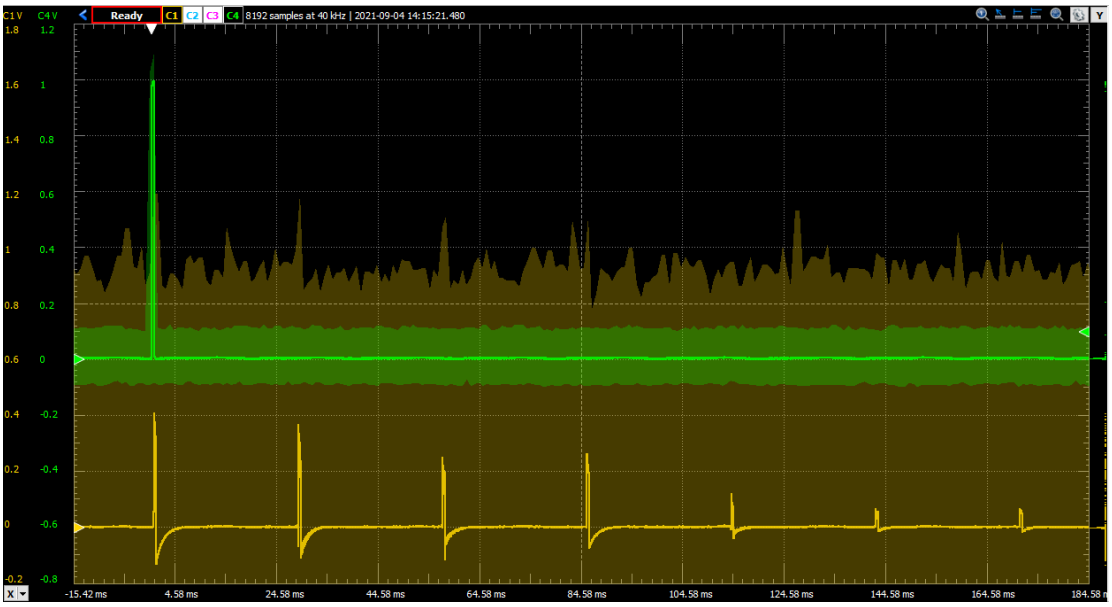


*Ilustración 3: Respuesta del **reverberador***