|  |  |
| --- | --- |
| **PROYECTO FINAL** | |
| **Título:** | **Ecualizador digital** |
| **Tipo de Documento:** | Reporte Técnico |
| **Fecha:** | 1-06-2022 |
| **Preparado por:** | Gasperín Castelán Hugo Joshua |
| **Aprobado por:** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Edición** | **Rev** | **Fecha** | **Razones del cambio** | **Sección afectada** |
| **1** | - | 22-04-2022 | Idea | - |
| **2** | Dr. Saul | 23-05-2022 | No abarcaba el espectro completo | Procesamiento |
| **3** | Dr. Saul | 1-06-2022 | Presentación final |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

1. Planteamiento del proyecto

* Objetivo:

Creación de un ecualizador digital capaz de filtrar señales analógicas de acuerdo con rangos prestablecidos en frecuencia apoyado en los periféricos disponibles en la tarjeta de desarrollo Tiva C Series TM4C1294NCPDT.

* Descripción del proyecto:

El proyecto consiste en realizar un ecualizador digital de 9 bandas capaz de separar los tonos o frecuencias de cualquier señal analógica que reciba el microcontrolador utilizando al convertidor ADC. Además, para esto realizaré el procesamiento digital de señales y se utilizaré un convertidor DAC externo comunicado por SPI, una pantalla comunicada por I2C para el despliegue de la información, y un módulo bluetooth conectado por UART para permitir el control remoto de mi aplicación, utilizando botones para la elección del canal de salida que se desea.

* Metodología

1. Para el diseño de una solución eficiente para el objetivo propuesto consideraré la siguiente metodología de solución:
2. Definir el problema general a resolver.
3. Definir las especificaciones mínimas que debe cumplir el proyecto.
4. Proponer un sistema en lo general que permita la resolución del problema planteado.
5. Proponer las partes fundamentales del sistema que conforma la solución.
6. Seleccionar los componentes necesarios para cada una de las partes del diagrama general.
7. Diseñar cada una de las partes del sistema contemplando los componentes seleccionados, es decir, realizar la programación, esquemas, estructuras, etc, contemplando las características de los componentes previstos.
8. Probar el funcionamiento individual de cada una de las partes
9. Integrar el sistema general y probar su funcionamiento.
10. Marco teórico

Como panorama general debemos contemplar que la creación y posterior evolución de los ecualizadores se debe principalmente a las líneas de redes telefónicas y a la inclusión del audio en el cine de Hollywood. Un ejemplo de esto sucede cuando a finales de 1920 La National Bell Telephone Company (la actual AT&T) usó los ecualizadores en las redes pasivas de las líneas telefónicas para corregir la perdida de las respuestas de frecuencias que la misma ruta de audio provocaba sobre la señal original. Esto contemplo un avance significativo en el procesamiento de las señales, pero aun poseía la desventaja de que al ser un procesamiento analógico la única manera que existía de variar el funcionamiento del circuito era alterándolo o cambiando sus componentes.

Como mejora de lo anterior surgen los filtros digitales, que realizan la misma función que un filtro analógico con la gran ventaja de ser programables, es decir, que su funcionamiento está determinado por un programa almacenado en la memoria contigua al procesador. Donde esto significa que puede ser variado fácilmente sin afectar al hardware.

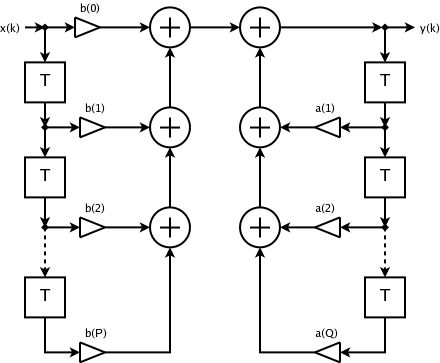
A su vez, existen diferentes técnicas de filtrado digital, por lo que para el desarrollo del proyecto contemplaré el uso de filtros de tipo IIR, por sus siglas en inglés, que significa filtros de respuesta al Impulso infinita.

En general este es un tipo de filtro digital que si su entrada es un impulso (una delta de Kronecker) la salida será un número ilimitado de términos no nulos, es decir, que nunca volverá a un estado de reposo. Para obtener la salida se emplean valores de la entrada actual y anteriores y, además, valores de salida anteriores que son almacenados en memoria y realimentados a la entrada. También se llaman filtros digitales recursivos.

Además, se define su expresión en el dominio discreto como:

Donde P y Q corresponden al orden del filtro, y son los coeficientes de este.

Esto puede ser descrito también por el siguiente diagrama:



1. Filtro IIR

Donde se puede ver cómo la salida y(k) es introducida de nuevo en el sistema. La transformada Z de este es:

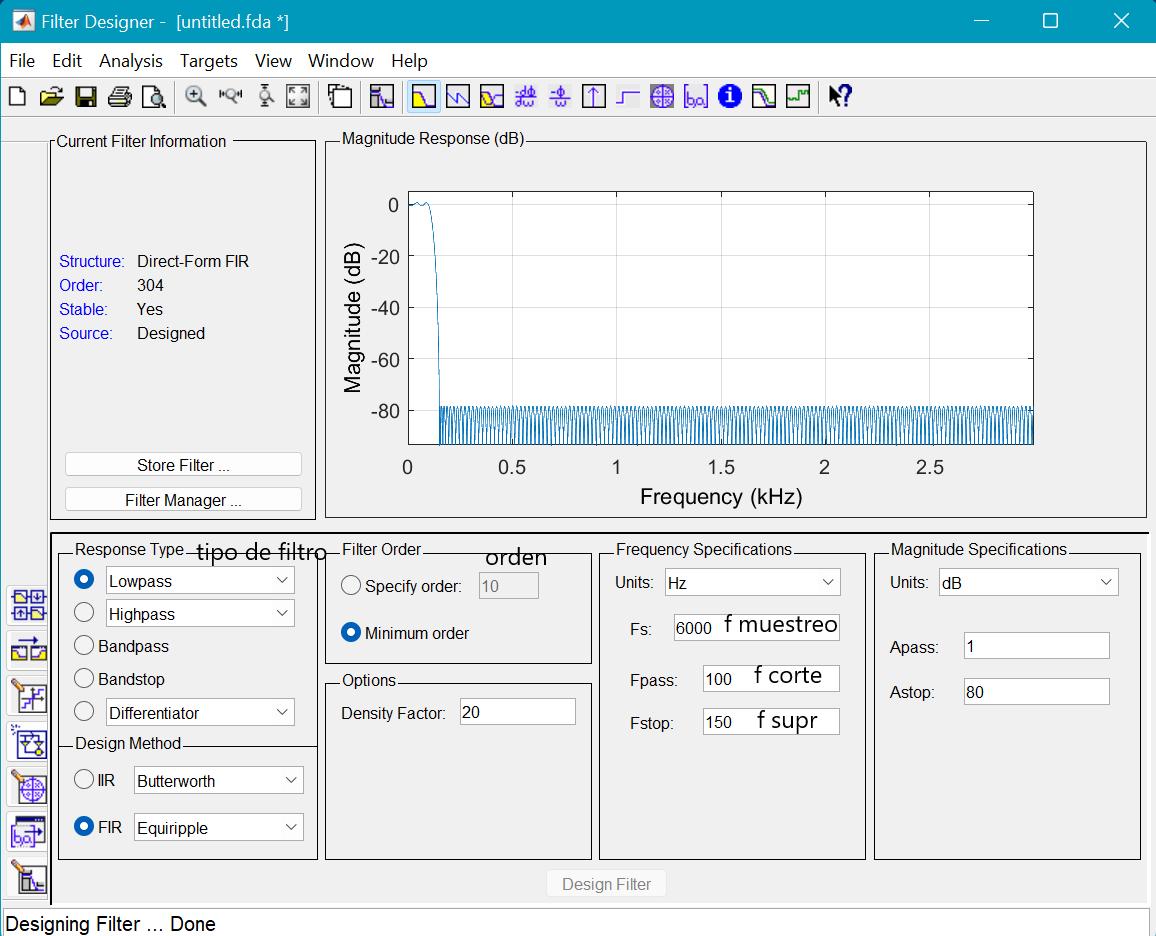
Imagen que contiene objeto, reloj

Descripción generada automáticamente

Vemos que ahora tenemos un denominador, es decir, ceros además de polos, que son los causantes de las posibles inestabilidades que pueden comprometer la estabilidad y causalidad del sistema.

Las ventajas de los filtros IIR respecto a los FIR es que pueden conseguir una misma respuesta empleando un número de coeficientes en el filtro mucho menor, requiriendo un menor tiempo de cálculo. El inconveniente es, además de la inestabilidad ya comentada, la introducción de desfases en la señal, que pueden ser compensados, pero a costa de añadir más coeficientes al filtro.

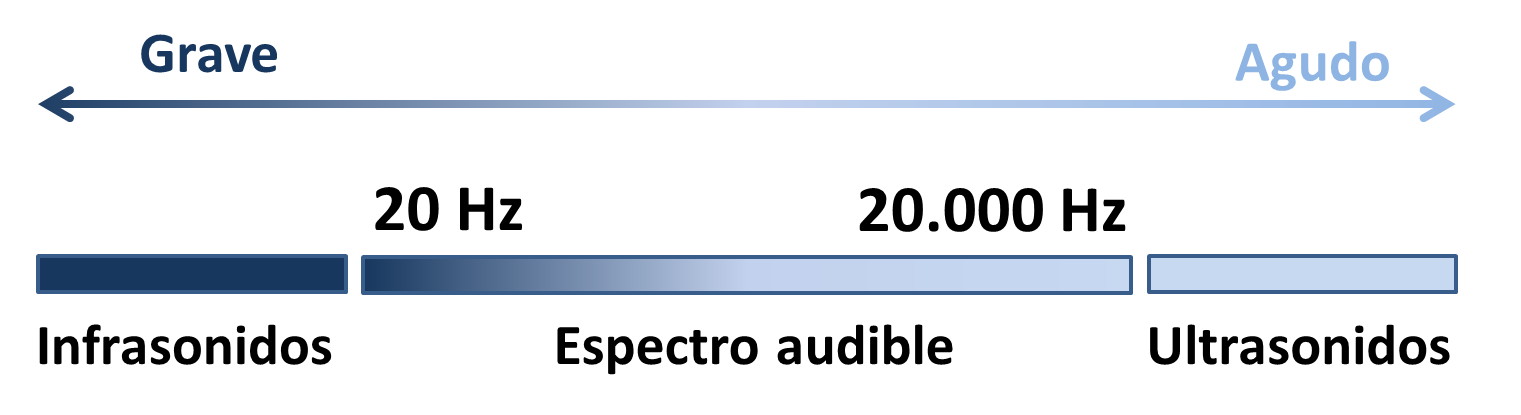
Esto también pudo ser confirmado en el momento del diseño de los coeficientes de los filtros utilizando el software auxiliar de MATLAB, filterDesigner, el cual presenta la siguiente interfaz:



2. Filter Designer

La cual nos permite diseñar seleccionar diferentes parámetros como se observa en la imagen.

Además, conociendo que el espectro auditivo humano abarca desde aproximadamente 20 Hz hasta los 20kHz como lo enseña la siguiente imagen:



3. Espectro audible

Por lo que se contemplará el uso de 9 filtros digitales con tal de demostrar el funcionamiento en diferentes condiciones.

También es importante para esto considerar el teorema de muestreo de Nyquist el cual nos dice que, para poder reconstruir una señal muestreada, la frecuencia de muestreo debe ser superior al doble del ancho de banda. Por lo que necesitamos idealmente una frecuencia de muestreo de alrededor de 40kHz para abarcar el espectro audible del ser humano.

1. Diseño

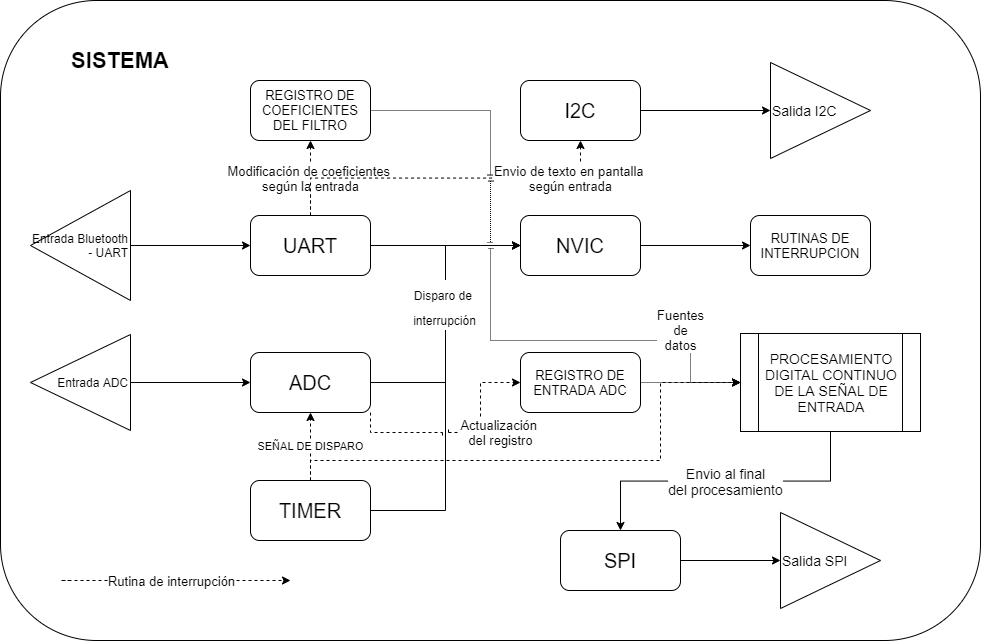
-Concepto

Crear un ecualizador digital utilizando al menos un elemento de todos los periféricos incluidos en la tarjeta de desarrollo TIVA tm4c1294xl los cuales son:

* Comunicación UART
* Comunicación SPI
* Comunicación I2C
* Puertos
* Temporizadores
* Controlador de interrupciones

-Diseño funcional

Partimos del siguiente esquema general:



4. Diagrama de bloques del ecualizador digital

De donde diseñaré cada una de sus partes para posteriormente ser integradas.

-Diseño a detalle

Para el funcionamiento correcto del sistema se debe acondicionar la señal de entrada al ADC basándonos en la fuente que origina la señal, por lo que tomando como base mi celular tenemos los siguientes datos:

Por lo que considerando que el voltaje máximo posible a medir basándome en las características mencionados en la hoja de datos del ADC es de , entonces necesitamos una ganancia de .

Para esto propongo el siguiente circuito apoyándome en la estructura de un amplificador operacional en su configuración no inversora, donde se sabe que:

Y seleccionando .

Además, se plantea el uso del mismo circuito en la salida, junto con un capacitor en la entrada del amplificador de salida para eliminar el offset de CD generado por el DAC de manera que el sistema trabaje de manera adecuada.

Otro punto importante es considerar el uso de una fuente de 12 V simétrica con el fin de evitar que la señal de salida de los amplificadores se sature.

Por lo que el sistema queda de la siguiente forma:

Señal de entrada

Amplificador de salida

Amplificador de entrada

Señal de salida

Sistema

O como lo muestra el siguiente esquema eléctrico:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

5. Esquema eléctrico del ecualizador

Por otro lado, para la selección del DAC busqué un modelo que tuviera la misma resolución que el ADC, con el fin de evitar tener que realizar conversiones entre los datos obtenidos. Además de que también busque que fuera compatible con SPI para la comunicación.

A partir de esto tengo al circuito integrado MCP4921, el cual presenta, además de las características anteriormente mencionadas, un pin para la selección de voltaje de referencia y un pin de habilitación.

La configuración necesaria para la trama de datos consiste en 16 bits que son modificados acorde a lo señalado en la siguiente tabla:

|  |  |
| --- | --- |
| Bit | Función |
| 15 | Selección de pin de salida |
| 14 | Control de buffer de voltaje de referencia |
| 13 | Selección de ganancia |
| 12 | Habilitación de salida |
| 11-10 | Bits de datos del DAC |

Por lo que se realiza una función que permita el acomodo de los datos en los 12 bits menos significativos a enviar, además de la habilitación en los bits correspondientes al bit de selección de ganancia y de habilitación de la salida.

Así mismo, el ADC integrado en la tarjeta se configurará contemplando los parámetros básicos, con la particularidad de seleccionar que sea disparado por software, de manera que este sea accionado solamente cuando la rutina de interrupción del temporizador haya terminado de procesar la salida anterior.

Otro punto importante con el fin de abarcar una aplicación más notoria del ecualizador es realizar 9 filtros digitales repartidos en el espectro auditivo del ser humano, limitándome prácticamente al rango de 0 a 20 kHz.

De manera que considerando lo anterior propongo las siguientes bandas de paso:

1. Tonos graves – Frecuencias menores a 100 Hz
2. Tonos fundamentales de la voz – Frecuencias entre 100 y 350 Hz
3. Tonos altos dentro de la voz – Frecuencias entre 350 y 600 Hz
4. Tonos medios bajos – Frecuencias entre 600 y 1 kHz
5. Tonos medios – Frecuencias entre 1k y 1.8 kHz
6. Tonos medios altos – Frecuencias entre 1.8 y 2.5 kHz
7. Tonos altos audibles – Frecuencias entre 2.5 y 5 kHz
8. Tonos altos – Frecuencias entre 5 y 10 kHz
9. Tonos superiores a 10kHz – Frecuencias mayores a 10 kHz

A su vez, para el diseño de estos, me apoye en el uso del programa de diseño de filtros digitales integrado en Matlab llamado “Filter Designer”, donde colocando una configuración de ventana correspondiente a filtros pasa bajas, pasa bandas, pasa altas, según el caso, y seleccionando para todos un filtro IIR de orden 2 de tipo Butterworth con lo cual tenemos los siguientes coeficientes junto con su respuesta en frecuencia para cada uno de los filtros anteriormente señalados.

1. Tonos graves – Frecuencias menores a 100 Hz

Gráfico, Gráfico de líneas

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Tonos fundamentales de la voz – Frecuencias entre 100 y 350 Hz

Gráfico, Gráfico de líneas

Descripción generada automáticamenteTexto, Carta

Descripción generada automáticamente

1. Tonos altos dentro de la voz – Frecuencias entre 350 y 600 Hz

Gráfico, Gráfico de líneas

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Tonos medios bajos – Frecuencias entre 600 y 1 kHz

Gráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Tonos medios – Frecuencias entre 1k y 1.8 kHz

Gráfico, Gráfico de dispersión

Descripción generada automáticamenteTexto, Carta

Descripción generada automáticamente

1. Tonos medios altos – Frecuencias entre 1.8 y 2.5 kHz

Gráfico, Gráfico de líneas

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Tonos altos audibles – Frecuencias entre 2.5 y 5 kHz

Gráfico, Gráfico de líneas, Gráfico de dispersión

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Tonos altos – Frecuencias entre 5 y 10 kHz

Gráfico, Gráfico de dispersión

Descripción generada automáticamenteTexto

Descripción generada automáticamente

1. Tonos superiores a 10kHz – Frecuencias mayores a 10 kHz

Gráfico, Gráfico de dispersión

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Por otro lado, el módulo Bluetooth realiza por sí solo la recepción de los datos del celular para posteriormente transmitirlo por el bus de transmisión a la tarjeta, por lo que al solo necesitar la recepción de datos solo utilizaré el puerto de recepción UART en la tarjeta, y el correspondiente transmisor en el módulo bluetooth.

Así mismo, aparte de las habilitaciones correspondientes al uso del periférico solo se necesita configurar la recepción para datos de 8 bits pues este es el formato con el que trabaja el módulo inalámbrico.

Y finalmente, para poder comunicarme con el módulo, también desarrollo una aplicación móvil utilizando la plataforma App Inventor del MIT debido a su gran facilidad. Donde las configuraciones más relevantes se encuentran en los botones de conexión y desconexión, que enlazan y desenlazan al celular con el módulo, así como también en los botones de selección de filtro, que envían un numero de 8 bits en específico según cada uno de los filtros.

Pantalla de celular con aplicaciones

Descripción generada automáticamente

6.Pantalla principal de la aplicación

Datos que contemplo serán utilizados para la configuración de los coeficientes a emplear durante el filtrado de la señal, así como también para seleccionar la configuración de la pantalla adecuada a cada situación.

Para la pantalla se contempla el uso de una biblioteca que nos permite tanto realizar la inicialización del periférico I2C, así como también la inicialización básica de la pantalla y el posterior envío de los datos correspondientes en cada situación.

Para el temporizador se realiza la configuración básica de los registros considerando que será utilizado para generar una frecuencia de muestreo de 40 kHz, que cumple con el valor mínimo según Nyquist para el espectro de la voz humana. Por lo que teniendo en cuenta lo anterior necesitamos que el valor de recarga del registro del temporizador sea:

-Pruebas y refinamiento

* Rendimiento

Se comprobó mediante una prueba utilizando un contador y un cronometro que el número de muestras procesadas fuera igual al de la frecuencia de muestreo, lo que resultó en un total de 40,000 muestras procesadas por segundo, por lo que puedo decir que el sistema trabaja de manera eficiente.

* Prueba de la aplicación

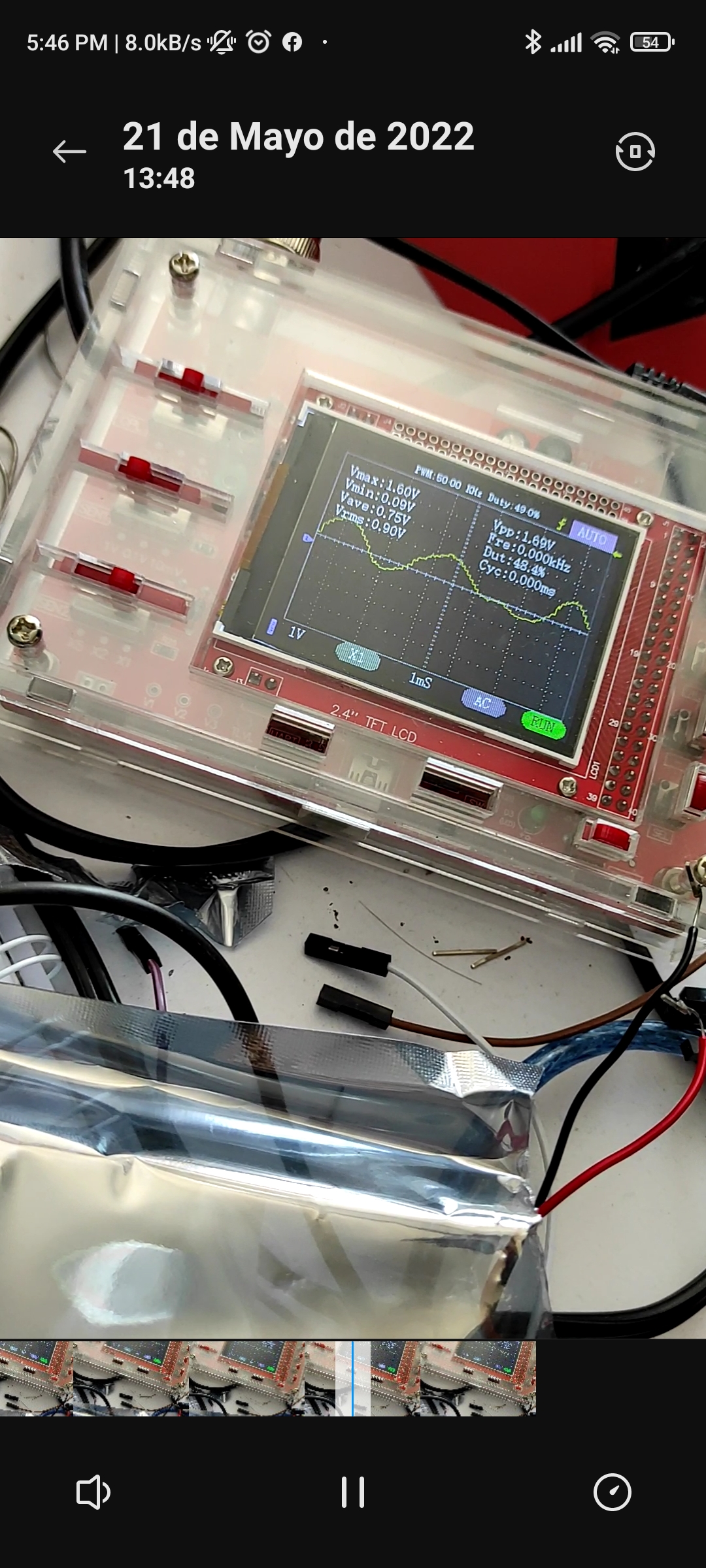
Como se observa en la imagen siguiente se puede notar una conexión exitosa entre el módulo y el celular.



7. Aplicación en funcionamiento

* Prueba de salida del DAC tras procesado

Se realizo la prueba de salida del DAC bajo las condiciones de no tener ninguna señal de entrada y de tener una melodía aleatoria obteniendo que el sistema responde solamente al tener una melodía, como era de esperarse.

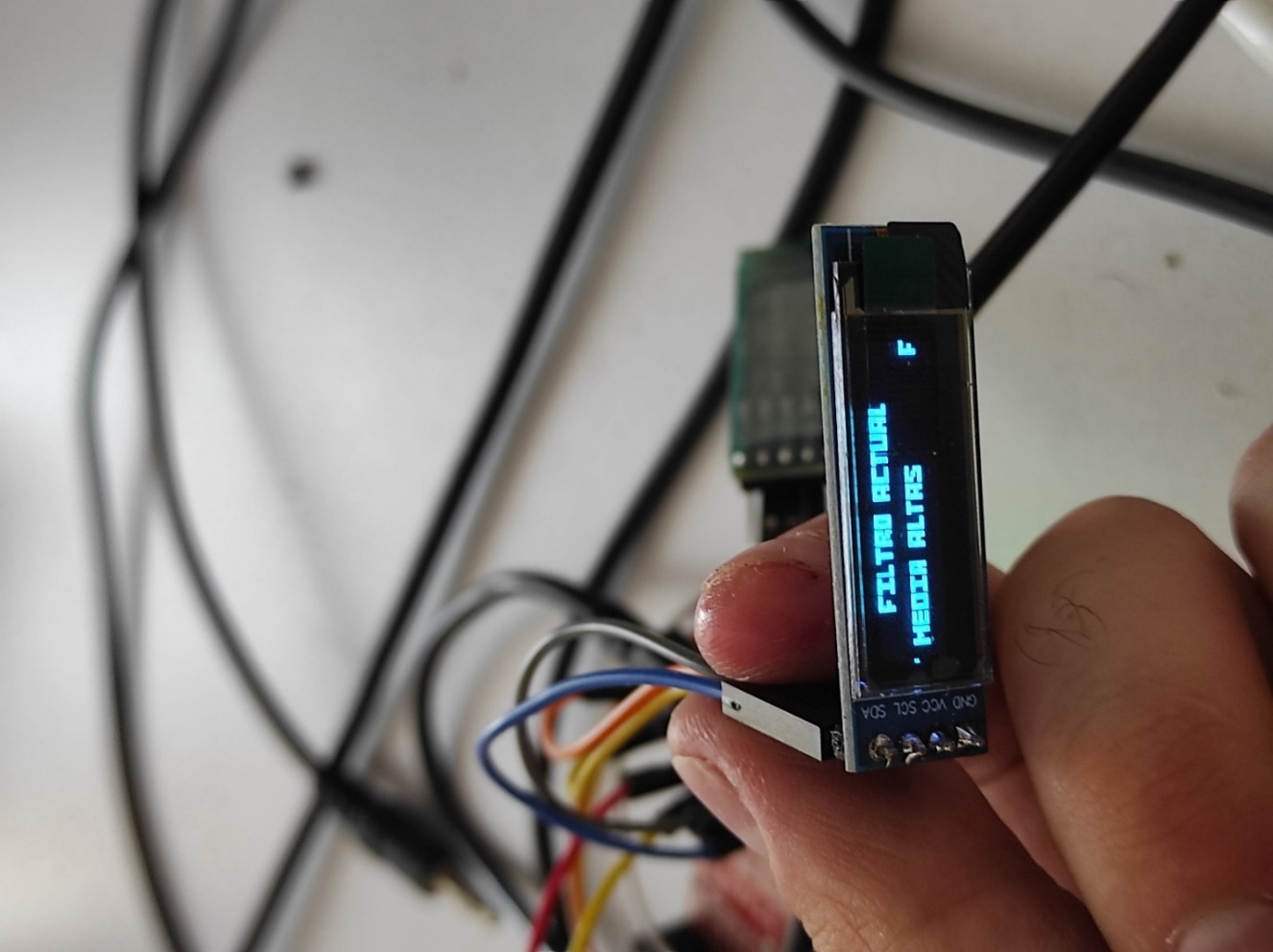


8. Salida del DAC postprocesado

* Prueba de comunicación bluetooth y funcionamiento de la pantalla

A través del accionamiento de uno de los botones en la aplicación se cambia el valor impreso en la pantalla LED, y a su vez, cambian los valores almacenados en el arreglo general de coeficientes, ambas cosas según el filtro seleccionado.

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

9. Salida de los estados y la pantalla

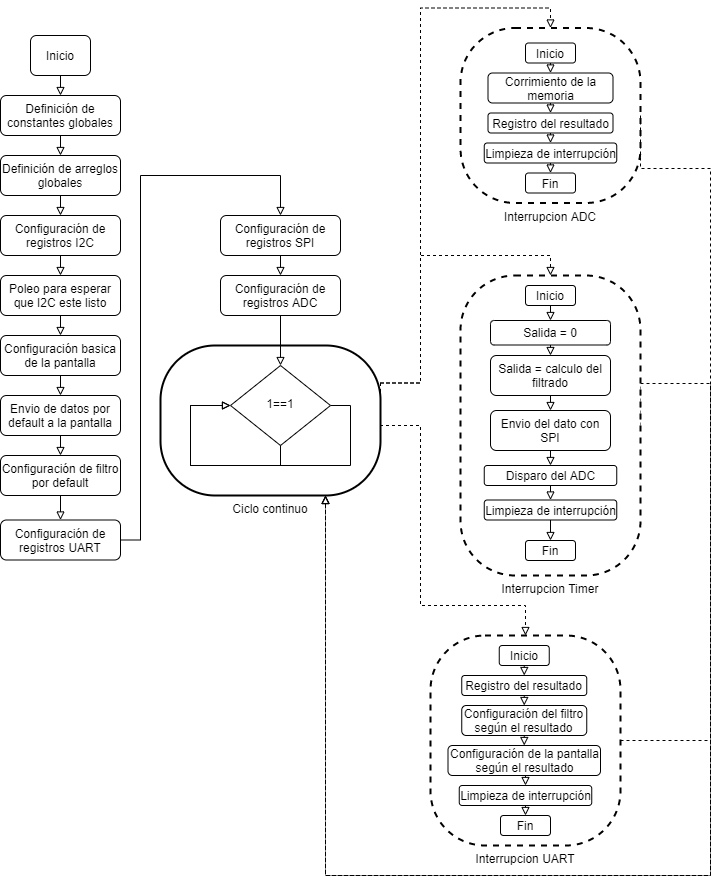
* Tiempo de procesamiento de la señal de salida

Contando el numero de instrucciones en lenguaje ensamblador necesarias para realizar el procesamiento de los datos encontré 38 comandos, además de 6 instrucciones extras utilizadas para el envió de los datos por SPI, por lo que tomando el caso ideal en el que solo se ocupe un ciclo de reloj por cada uno de los comandos tenemos que en total se requieren 44 ciclos tan solo para el procesamiento, lo que según el reloj de 16 MHz de la tarjeta corresponde a 2.75e-6 segundos.

Esto representa una ventaja respecto al anterior proyecto que utilizaba filtros FIR, puesto que en el anterior eran necesarios 900 ciclos de reloj en el caso ideal, lo que limitaba la tasa de salida a la capacidad de la tarjeta.

1. Diagrama de flujo y código comentado.

El código desarrollado puede ser visto en funcionamiento tal y como el siguiente diagrama de flujo:



. Diagrama de flujo del sistema

A continuación, se introduce el código comentado:

//puertos a utilizar

//PC4 Rx uart

//PC5 tx uart

//PB2 scl i2c

//PB3 sda i2c

//PE2 ADC IN-

//PE3 ADC IN+

//PA4 SI SPI SDI

//PA3 MOSI SPI CS

//PA2 MISO SPI SCK

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** <math.h>

**#include** "inc/tm4c1294ncpdt.h"

**#include** <stdlib.h>

**#include** "SSD1306.h"

//filtrado

**const** uint16\_t orden=3;//orden del filtro + 1

**float** salvoz;//salida final del sistema

**float** memvoz[orden]={0};

**float** ag1,ag2,ag3,bg2,bg3;

//filtro 100 a 350

**float** g1=0.019259274202337011566710245347167074215 ;

**float** a11,a12,a13;

**float** b12=-1.960634445434191075463559172931127250195;

**float** b13= 0.961481451595325942172109989769523963332;

//filtro 350 a 600

**float** g2=0.019259274202334010495096805470893741585;

**float** a21,a22,a23;

**float** b22=-1.956401243104590514931828693079296499491;

**float** b23= 0.961481451595331937376442965614842250943;

//filtro 600 a 1k

**float** g3=0.030468747091253921971398810342179785948 ;

**float** a31,a32,a33;

**float** b32=-1.924722152984461143176986297476105391979;

**float** b33= 0.939062505817492065851581628521671518683;

//filtro 1k a 1.8k

**float** g4=0.05919070381840586120292257987784978468 ;

**float** a41,a42,a43;

**float** b42=-1.839933812949782421242161944974213838577;

**float** b43= 0.881618592363188291471942648058757185936;

//filtro 1.8k a 2.5k

**float** g5=0.052162645814763396179269960839519626461 ;

**float** a51,a52,a53;

**float** b52=-1.79129862404605888492881149431923404336 ;

**float** b53= 0.895674708370473249274823501764331012964;

//filtro 2.5k a 5k

**float** g6=0.165910681040350477255529426656721625477 ;

**float** a61,a62,a63;

**float** b62=-1.414213562373095145474621858738828450441;

**float** b63= 0.668178637919299101000092377944383770227;

//filtro 5k a 10k

**float** g7=0.292893218813452482773840301888412795961 ;

**float** a71,a72,a73;

**float** b72=-0.585786437626905076569983066292479634285;

**float** b73= 0.41421356237309508996347062748100142926 ;

//filtro 10k a 20k

**float** g8=0.292893218813452593796142764404066838324 ;

**float** a81,a82,a83;

**float** b82=-0.000000000000000260141436226615165650146;

**float** b83= 0.171572875253809986606512438811478205025;

//filtro graves

**float** g0=0.000061006178758064257461237595014935664 ;

**float** a01,a02,a03;

**float** b02=-1.977786483776763803987819301255512982607;

**float** b03= 0.978030508491796068959445165091892704368;

uint32\_t data3[2];//datos UART

uint32\_t data1[2];//datos ADC

// En este programa se utiliza el ADC0 con el Secuenciador de muestras 2

// ya que el puede tomar hasta 4 muestras, y aquí se necesitan 2 muestras.

//

//inicializa los coeficientes con la ganancia correspondiente

**void** **coefInit**(**void**){

a01=1\*g0;

a02=2\*g0;

a03=1\*g0;

a11=1\*g1;

a12=0\*g1;

a13=-1\*g1;

a21=1\*g2;

a22=0\*g2;

a23=-1\*g2;

a31=1\*g3;

a32=0\*g3;

a33=-1\*g3;

a41=1\*g4;

a42=0\*g4;

a43=-1\*g4;

a51=1\*g5;

a52=0\*g5;

a53=-1\*g5;

a61=1\*g6;

a62=0\*g6;

a63=-1\*g6;

a71=1\*g7;

a72=0\*g7;

a73=-1\*g7;

a81=1\*g8;

a82=-2\*g8;

a83=1\*g8;

}

/\*Carga la configuracion necesaria

\* para utilizar el SPI

\* \*/

**void** **SSI0\_init** (**void**) {

SYSCTL\_RCGCSSI\_R = SYSCTL\_RCGCSSI\_R0; // Activa reloj al SSI0

**while** ((SYSCTL\_PRSSI\_R & SYSCTL\_PRSSI\_R0) == 0); // Espera a que este listo

SYSCTL\_RCGCGPIO\_R |= SYSCTL\_RCGCGPIO\_R0; // Activa reloj del GPIO A

**while** ((SYSCTL\_PRGPIO\_R & SYSCTL\_PRGPIO\_R0) == 0); // Espera a que este listo

GPIO\_PORTA\_AHB\_AFSEL\_R |= 0x3C; // Seleciona la función alterna de PA[2:5].

GPIO\_PORTA\_AHB\_PCTL\_R = (GPIO\_PORTA\_AHB\_PCTL\_R & 0XFF0000FF) | 0x00FFFF00; // Configura las terminales de PA a su función de SSI0.

GPIO\_PORTA\_AHB\_AMSEL\_R = 0x00; // Deshabilita la función analogica

GPIO\_PORTA\_AHB\_DIR\_R = (GPIO\_PORTA\_AHB\_DIR\_R & ~0x3C) | 0x1C; // Configura al puerto como salida

GPIO\_PORTA\_AHB\_DEN\_R |= 0x3C; // Habilita la función digital del puerto

SSI0\_CR1\_R = 0x00; // Selecciona modo maestro/deshabilita SSI0. (p. 1247)

SSI0\_CC\_R=0x05;//usa reloj de 16 MHz

SSI0\_CPSR\_R = 0x02; // preescalador (CPSDVSR) del reloj SSI (p. 1252)

// configura para Freescale SPI; 16bit; 4 Mbps; SPO = 0; SPH = 0 (p. 1245)

SSI0\_CR0\_R = (0x0100 | SSI\_CR0\_FRF\_MOTO | SSI\_CR0\_DSS\_16) & ~(SSI\_CR0\_SPO | SSI\_CR0\_SPH);

//0 en estado de espera del bus

SSI0\_CR1\_R |= SSI\_CR1\_SSE; // Habilita SSI0.

}

/\*Funcion para organizar la forma de envio de los datos al DAC

\* considerando dos bits que configura para tener ganancia unitaria en la salida

\* y permitir la salida del mismo

\* \*/

**void** **SSI0\_sendData** (uint16\_t dat) {

// Envia dato de 16-bit

// while ((SSI0\_SR\_R & SSI\_SR\_BSY) != 0); // espero si el bus está ocupado

SSI0\_DR\_R = (dat&0xFFF)|0x3000; // envia dato.

}

/\*Carga la configuracion necesaria

\* para utilizar el Timer

\* \*/

**void** **initTimer**(**void**){

SYSCTL\_RCGCTIMER\_R |= 0X08; //HABILITA TIMER 3

**int** con;

//retardo para que el reloj alcance el PORTN Y TIMER 3

**for**(con=0;con<10;con++){}

TIMER3\_CTL\_R=0X00000000; //DESHABILITA TIMER EN LA CONFIGURACION (p.986)

TIMER3\_CC\_R=1;//reloj interno de 16MHz

TIMER3\_CFG\_R= 0X00000004; //CONFIGURAR PARA 16 BITS (p. 976)

TIMER3\_TAMR\_R= 0X00000002; //CONFIGURAR PARA MODO PERIODICO CUENTA HACIA ABAJO (p. 977)

TIMER3\_TAILR\_R= 400; // VALOR DE RECARGA (p.1004)

TIMER3\_ICR\_R= 0X00000001 ; //LIMPIA POSIBLE BANDERA PENDIENTE DE TIMER3 (p.1002)

TIMER3\_IMR\_R |= 0X00000001; //ACTIVA INTRRUPCION DE TIMEOUT (p.993)

NVIC\_EN1\_R|= 1<<(35-32); //HABILITA LA INTERRUPCION 35 DE TIMER3 A

NVIC\_PRI8\_R=0x40000000;

TIMER3\_CTL\_R |= 0X00000001; //HABILITA TIMERA EN LA CONFIGURACION (p.986)

}

/\*Carga la configuracion necesaria

\* para utilizar el ADC

\* \*/

**void** **initADC**(**void**){

SYSCTL\_RCGCGPIO\_R |= 0x10; // 1) Habilita reloj para Puerto E

**while**((SYSCTL\_PRGPIO\_R & 0x10) == 0); // Se espera a que el reloj se estabilice (p.499)

GPIO\_PORTE\_AHB\_DIR\_R = 0x00; // 2) PE5-4 entradas (analógica)

GPIO\_PORTE\_AHB\_AFSEL\_R |= 0x0C; // 3) Habilita Función Alterna en PE2/3 (p. 770)

GPIO\_PORTE\_AHB\_DEN\_R &= ~0x0C; // 4) Deshabilita Función Digital en PE2-3 (p 781)

GPIO\_PORTE\_AHB\_AMSEL\_R |= 0x0C; // 5) Habilita Función Analógica de PE2-3 (p. 786)

SYSCTL\_RCGCADC\_R |= 0x01; // 6) Habilita reloj para ADC0(p. 396)

**while**((SYSCTL\_PRADC\_R & 0x01) == 0); // Se espera a que el reloj se estabilice

ADC0\_PC\_R = 0x07; // 7)Configura para 125Ksamp/s (p.1159)

ADC0\_SSPRI\_R = 0x1023; // 8)SS2 con la más alta prioridad

ADC0\_ACTSS\_R &= ~0x0004; // 9) Deshabilita SS2 antes de cambiar configuración de registros (p. 1076)

ADC0\_EMUX\_R = 0x0000; // 10) Se configura SS2 para disparar muestreo por soft (default) (p.1091)

ADC0\_SAC\_R = 0x0; // 11) Se configura para sobremuestreo de 2 por hardware(default)(p. 1105)

ADC0\_CTL\_R = 0x0; //12) Se configura con referencias internas (default VDDA and GNDA) (p. 1107)

ADC0\_SSOP2\_R = 0x0000; // 13) Se configure para salvar los resultados del ADC en FIFO (default)(p. 1134)

ADC0\_SSTSH2\_R = 0x000; // 14) Se configure el ADC para un periodo de 4 S&H (default) (p. 1139)

ADC0\_SSMUX2\_R = 0x0000; // 15) Se configura entradas 1°muestra=AIN9 2°muestra=AIN8 (p.1128)

ADC0\_SSEMUX2\_R &= ~0x0001; // 16) Canales del SS2 para 1°muestra y 2°muestra en AIN(15:0) (p.1137)

ADC0\_SSCTL2\_R = 0x0007; // 17) SI: AIN, Habilitación de INR2; No:muestra diferencial (p.1142)

ADC0\_ISC\_R = 0x0004; //Limpia la bandera RIS del ADC0

ADC0\_IM\_R ^= 0x0004;//desenmascara

ADC0\_ACTSS\_R |= 0x0004; // 19) Habilita SS2 (p. 1076)

SYSCTL\_PLLFREQ0\_R |= SYSCTL\_PLLFREQ0\_PLLPWR; // encender PLL

**while**((SYSCTL\_PLLSTAT\_R&0x01)==0); // espera a que el PLL fije su frecuencia

SYSCTL\_PLLFREQ0\_R &= ~SYSCTL\_PLLFREQ0\_PLLPWR; // apagar PLL

//bloque interrupcion

NVIC\_EN0\_R|=1<<16;//habilita int adc

NVIC\_PRI4\_R|=0x02;//prioridad adc

//se inicializa el timer pues sera usado para disparar el ADC

initTimer();

}

/\*Rutina de interrupcion de ADC

\* \*/

**int** k3=0;

**void** **rutinaiADC**(**void**){

k3++;

data1[0]= (ADC0\_SSFIFO2\_R&0xFFF);// se lee el resultado

uint16\_t k;

**for**(k=orden-1;k!=0;k--){

memvoz[k]=memvoz[k-1];//se recorre la memoria

}

memvoz[0]=data1[0];//se asigna el dato procesado a la primera posicion de memoria

ADC0\_ISC\_R = 0x0004; //Limpia la bandera RIS del ADC0

}

// RUTINA DE SERVICIO DE INTERRUPCIÓN

uint16\_t k2;

**float** y\_1=0;

**float** y\_2=0;

**void** **Timer03AIntHandler**(**void**)

{

salvoz=0;//se reinicia la variable utilizada como salida

salvoz=memvoz[0]\*ag1+ag2\*memvoz[1]+ag3\*memvoz[2]-bg2\*y\_1-bg3\*y\_2;

y\_2=y\_1;

y\_1=salvoz;

SSI0\_sendData((**int**)salvoz); //una vez terminado el procesado se envian el dato de salida al spi

TIMER3\_ICR\_R= 0X00000001 ; //LIMPIA BANDERA DE TIMER3

ADC0\_PSSI\_R = 0x0004; //Inicia conversión del SS2

}

/\*Carga la configuracion necesaria

\* para utilizar el UART

\* \*/

**void** **uartinit**(**void**){

SYSCTL\_RCGCUART\_R |= 0x00000080; // activa el reloj para el UART7 (p.388)

SYSCTL\_RCGCGPIO\_R |= 0x00000004; // activa el reloj para el Puerto C (p.382)

**while**((SYSCTL\_PRUART\_R&0x80) == 0){}; // Se espera a que el reloj se estabilice (p.505)

UART7\_CTL\_R &= ~0x00000001; // se deshabilita el UART (p.1188)

UART7\_IBRD\_R = 104; // IBRD = int(16,000,000 / (16 \* 115,200)) = int(8.681) (p.1184)

UART7\_FBRD\_R = 11; // FBRD = round(0.6806 \* 64) = 44 (p. 1185)

// Palabra de 8 bits (sin bits de paridad, un bit stop, FIFOs) (p. 1186)

UART7\_LCRH\_R = (UART\_LCRH\_WLEN\_8|UART\_LCRH\_FEN);

// UART toma su reloj del la fuente alterna como se define por SYSCTL\_ALTCLKCFG\_R (p. 1213)

UART7\_CC\_R = (UART7\_CC\_R&~UART\_CC\_CS\_M)+UART\_CC\_CS\_PIOSC;

// La fuente de reloj alterna es el PIOSC (default)(P. 280)

SYSCTL\_ALTCLKCFG\_R = (SYSCTL\_ALTCLKCFG\_R&~SYSCTL\_ALTCLKCFG\_ALTCLK\_M)+SYSCTL\_ALTCLKCFG\_ALTCLK\_PIOSC; //

//Bloque que configura la interrupcion y la desenmascara

UART7\_IFLS\_R &= ~0x0000003F; // Limpia las interrupciones de FIFO de TX y RX (p.1191)

UART7\_IFLS\_R += (UART\_IFLS\_TX1\_8|UART\_IFLS\_RX1\_8); // Habilita las interrupciones de la FIFO de TX y RX, y la interrupción por time-out de RX

UART7\_ICR\_R = 0x10; // Limpia la bandera interrupcion de FIFO RX (En este programa solo manejaremos esta interrupción)

UART7\_IM\_R |= 0x00000010; // Desenmascara la interrupción para UART7

//Fin de configuración de interrupcion

// Alta velocidad deshabilitada;divide el reloj por 16 en lugar de 8 (default)(1188)

UART7\_CTL\_R &= ~0x00000020;

UART7\_CTL\_R |= 0x00000001; // habilita el UART (p.1188)

// Se espera a que el reloj se estabilice

**while**((SYSCTL\_PRGPIO\_R&0x04) == 0){};

GPIO\_PORTC\_AHB\_AFSEL\_R |= 0x30; // habilita funcion alterna en PC5-4

GPIO\_PORTC\_AHB\_DEN\_R |= 0x30; // habilita digital I/O en PC5-4

// configura PC5-4 como UART

GPIO\_PORTC\_AHB\_PCTL\_R = (GPIO\_PORTC\_AHB\_PCTL\_R&0xFF00FFFF)+0x00110000;//asocia el puerto con el uart

GPIO\_PORTC\_AHB\_AMSEL\_R &= ~0x30; // deshabilita la funcionabilidad analogica de PC

// Configuración de prioridad y hablitación de interrupción en NVIC

NVIC\_PRI15\_R = (NVIC\_PRI15\_R&0xFFFFFF00)|0x00000000; // Prioridad 0 (p.152)

NVIC\_EN1\_R |= 1<<(60-32); // Habilita la interrupción 60 en NVIC (p. 154)

}

**void** **UART7\_Handler**(**void**)

{

data3[0] = UART7\_DR\_R&0xFF;// se guarda el primer valor

data3[1] = UART7\_DR\_R&0xFF;// se guarda el segundo valor

//de acuerdo al valor recibido se configura el filtro seleccionado

//y se actualiza el texto en la pantalla

**switch**(data3[0]){

**case** 1:{

bg2=b12;

bg3=b13;

ag1=a11;

ag2=a12;

ag3=a13;

SSD1306\_Clear();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. VOZ");

}

**break**;

**case** 2:{

bg2=b22;

bg3=b23;

ag1=a21;

ag2=a22;

ag3=a23;

SSD1306\_Clear();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. ALTAS DE VOZ");

}

**break**;

**case** 3:

bg2=b32;

bg3=b33;

ag1=a31;

ag2=a32;

ag3=a33;

SSD1306\_Clear();

SSD1306\_SCROLLINGR();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. MEDIAS BAJAS");

**break**;

**case** 4:

bg2=b42;

bg3=b43;

ag1=a41;

ag2=a42;

ag3=a43;

SSD1306\_Clear();

SSD1306\_SCROLLINGR();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. MEDIAS");

**break**;

**case** 5:

{

bg2=b52;

bg3=b53;

ag1=a51;

ag2=a52;

ag3=a53;

SSD1306\_Init();

SSD1306\_Clear();

SSD1306\_SCROLLINGR();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. MEDIAS ALTAS");

}

**break**;

**case** 6:

{

bg2=b62;

bg3=b63;

ag1=a61;

ag2=a62;

ag3=a63;

SSD1306\_Init();

SSD1306\_Clear();

SSD1306\_SCROLLINGR();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F ALTAS AUDIBLES");

}

**break**;

**case** 7:

{

bg2=b72;

bg3=b73;

ag1=a71;

ag2=a72;

ag3=a73;

SSD1306\_Init();

SSD1306\_Clear();

SSD1306\_SCROLLINGR();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. ALTAS");

}

**break**;

**case** 8:

{

bg2=b82;

bg3=b83;

ag1=a81;

ag2=a82;

ag3=a83;

SSD1306\_Init();

SSD1306\_Clear();

SSD1306\_SCROLLINGR();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. SUPERIORES");

}

**break**;

**case** 9:

bg2=b02;

bg3=b03;

ag1=a01;

ag2=a02;

ag3=a03;

SSD1306\_Init();

SSD1306\_Clear();

SSD1306\_SCROLLINGR();

SSD1306\_Command(SSD1306\_SETPAGE0);

SSD1306\_WriteString("FILTRO ACTUAL: ");

SSD1306\_Command(SSD1306\_SETPAGE2);

SSD1306\_WriteString("F. GRAVES");

**break**;

**default**:

**break**;

}

UART7\_ICR\_R = 0x00000010; // Limpia bandera de interrupción

}

**void** **main**(**void**){

//inicializacion de coeficientes

coefInit();

//Configuracion de filtro grave por defecto

bg2=b02;

bg3=b03;

ag1=a01;

ag2=a02;

ag3=a03;

I2C\_Init(); //inicio de configuracion de i2c

**while**(I2C0\_MCS\_R&0x00000001){}; // espera que el I2C estÃ© listo

SSD1306\_Init(); // inicializacion de los comandos de la pantalla

SSD1306\_Clear(); //limpieza de la pantalla

SSD1306\_SCROLLINGR(); //scroll continuo

SSD1306\_Command(SSD1306\_SETPAGE0);//primer renglon de la pantalla

SSD1306\_WriteString("FILTRO ACTUAL: "); //texto por default

SSD1306\_Command(SSD1306\_SETPAGE2);//segundo renglon

SSD1306\_WriteString("VOZ");//texto por default

//asignar salida

uartinit();//habilita el uart bluetooth

SSI0\_init(); // Función que habilita el SPI

//inicio el ADC

initADC();

//procesamiento continuo

**while**(1==1){

}

}

1. Resultados y – Conclusiones

Con el desarrollo de este proyecto desarrolle mis habilidades de resolución de problemas a partir de una metodología que me permitió organizarme de una manera más eficiente, además de que observe que dividiendo al problema en partes más pequeñas es posible hacer una solución por etapas de manera que sea más ligero el proceso.

Por otro lado, también reafirme mis conocimientos de programación en lenguaje C puesto que no había realizado proyectos de este tipo en algo de tiempo.

Además, también entendí que el uso de los microcontroladores, como la TIVA, permiten realizar diferentes proyectos debido a su gran capacidad de instrumentación, lo que los vuelve opciones viables en prácticamente cualquier situación que empleé conocimientos del campo de la electrónica o el control automático.

Otro punto importante que mencionar es que comprendí cómo funcionan los diferentes protocolos de comunicación serial y como se configuran a partir de sus hojas de datos, lo que a mi parecer constituye una de las partes más interesantes del curso puesto que el entendimiento de esto me permite visualizar la forma en la que se comunican las maquinas, y, como parte de esto, lo que es una trama de datos.

Como consideración extra, el filtro pasa bajas para los tonos graves no presenta una salida acorde a lo descrito debido a que los coeficientes requeridos para el procesamiento requieren mas decimales de los soportados por mi tarjeta de desarrollo, lo que provoca que no se comporte de manera adecuada.

Y finalmente, como punto extra, aprendí a través del desarrollo de un proyecto precursor a este, que los filtros IIR son más eficientes que los filtros FIR, debido a que, con un orden menor, y, en consecuencia, menor procesamiento, es posible realizar resultados aceptables. Con la única desventaja que es un poco más difícil programarlos por tener una estructura diferente.

1. Referencias
2. De, A. (2019). Filtros FIR. Retrieved May 19, 2022, from Elipsis website: <https://www.elipsisaudio.com.mx/filtros-fir.html#:~:text=Los%20filtros%20FIR%20por%20sus,ingl%C3%A9s%20filtros%20de%20Respuesta%20Infinita>.‌
3. Filtro digital. (2019). Retrieved May 19, 2022, from Studio-22.com website: https://www.studio-22.com/blog/enciclopedia/filtro-digital#:~:text=Un%20filtro%20digital%20es%20programable,anal%C3%B3gico%20es%20alterando%20el%20circuito.
4. ‌ TivaTM C Series TM4C1294 Connected LaunchPad Evaluation Kit User’s Guide. (2014). Retrieved from <https://www.ti.com/lit/ug/spmu365c/spmu365c.pdf?ts=1644584905542&ref_url=https%253A%252F%252Fwww.google.com%252F>
5. Filtro digital IIR. (2019). Retrieved May 25, 2022, from Studio-22.com website: https://www.studio-22.com/blog/enciclopedia/filtro-digital-iir