**PRÁCTICA 2.B:** **Filtros FIR e IIR**

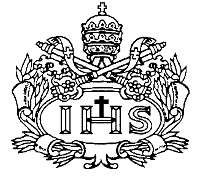
**AUTORES:**

**Nicolás Rodríguez Villate**

**Brandon González Benites**

**PRESENTADO A:**

**Francisco Carlos Calderón Bocanegra, Ing, Msc**



**PONTIFICIA UNIVERSIDAD JAVERIANA**

**FACULTAD DE INGENIERÍA**

**DEPARTAMENTO DE ELECTRÓNICA**

**BOGOTÁ D.C.**

**10/09/2021**

# INFORME

## Introducción

Se utiliza para la práctica la tarjeta de desarrollo para PIC18F46K40, suministrada por el laboratorio, también un módulo NI 6002 de Texas Instruments, un osciloscopio digital, un computador con los programas: Matlab, RealTerm y Signal Express.

## Objetivos

* Aprender las técnicas de implementación de filtros FIR e IIR, básicas y reconocer sus alcances y limitaciones
* Implementar un filtro FIR e IIR en un sistema microcontrolador.
* Reconocer y evaluar diferentes alternativas de implementación de filtros FIR e IIR

## Equipo utilizado

Para el desarrollo de esta práctica el equipo de trabajo necesito de los implementos que se observan en la **Tabla 1**. Estos dispositivos son suministrados por el laboratorio.

|  |  |  |
| --- | --- | --- |
| **Nombre del equipo** | **Marca** | **Modelo** |
| Fuente de voltaje | PeakTech | 6145 |
| Generador de funciones | Tektronix | AFG31000 SERIES |
| DVM | KEYSIGHT | 34450A |
| Osciloscopio | Tektronix | TBS2000 SERIES |

**Tabla 1.** Equipos utilizados en la práctica.

## Desarrollo

1. Desarrollar el código basado en los diagramas de flujo para FIR e IIR. Utilizando apuntadores e índices para comprobar el tiempo de ejecución en cada caso.
2. Para FIR en la implementación del cálculo usando la propiedad de simetría utilizar apuntadores.
3. Para IIR, implemente una librería (respectivos .c y .h) con una estructura única de entrada dónde guarde los coeficientes y estén las variables de cálculo intermedias, y existan como mínimo dos funciones, una para inicializar la estructura con los valores de los coeficientes y otra para operar el filtro cada nueva muestra. Se implementa en punto flotante.
4. Medir los tiempos de cálculo para cada una de las implementaciones de filtros.
5. Medir la respuesta en frecuencia de todos los filtros implementados en esta guía. Desarrolle un protocolo para realizar esta medida de forma automática para implementar en Labview o en el programa de preferencia. Compare la respuesta en frecuencia obtenida con la simulada y diseñada.
6. Medir el ENOB en la banda de paso para cada uno de los filtros implementados.
7. Recuerde que la entrada es un generador. Las mediciones sobre las señales, bode, ENOB o cualquier otra, debe realizarse con la salida del puerto serial y con la salida analógica del DAC.
8. El código final debe estar comentado adecuadamente y corresponder a los diagramas de tiempo, de flujo o de estado propuestos inicialmente.

## Procesamiento de datos en Matlab

Se presenta la transmisión por USART en Matlab:

Se extraen los coeficientes de Matlab y después se genera un encabezado en c, acoplando dichos coeficientes al código a partir de un archivo de Excel con el siguiente fragmento de código:

dat4 = readtable('Archivo.xlsx');

da =table2array(dat4);

x=1:800;

for N=1:800

plot(x(1:N),da(:,1:N))

hold on

**Filtros FIR**

Gráfico

Descripción generada automáticamente

Figura . FIR orden 7

Gráfico, Histograma

Descripción generada automáticamente

Figura . FIR orden 11

Gráfico, Histograma

Descripción generada automáticamente

Figura . FIR orden 17

Gráfico, Histograma

Descripción generada automáticamente

Figura . FIR orden 30

Estos resultados se pueden observar también desde el osciloscopio haciendo un barrido en frecuencia para cada caso.

**Filtros IIR**

Gráfico

Descripción generada automáticamente

Figura . IIR pasabajos orden 4

Gráfico

Descripción generada automáticamente

Figura . IIR rechaza banda orden 4

## Procedimiento y resultados

Tiempo de filtrado FIR

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

Figura . Tiempo FIR

Tiempo de filtrado, filtro IIR

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura . Tiempo IIR

**ENOB**

Para la medición del ENOB, el número eficaz de bits, para cuantificar la calidad de una conversión de analógico-digital, a partir de la transmisión SPI por el conversor análogo-digital se empleó Signal Express para adquirir la señal del DAC y por medio del bloque “distortion” obtener el SINAD para posteriormente en el bloque “formula” calcular el ENOB con la siguiente expresión:

|  |  |
| --- | --- |
| FILTRO | ENOB |
| FIIR ORDEN 7 | 7.72 |
| FIIR ORDEN 11 | 7.65 |
| FIIR ORDEN 17 | 6.92 |
| FIIR ORDEN 30 | 7.14 |
| IIR RECHAZA BANDA | 6.82 |
| IIR PASA BAJAS | 5.8 |
| IIR PASA ALTAS | 5.22 |

Tabla 2. ENOB para filtros

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . SINAD para calcular ENOB

Patrón de fondo

Descripción generada automáticamente

Figura . Fórmula en Signal Express para calcular el ENOB a partir del SINAD

**Diagramas de BODE:**

**-Filtros FIR:**

Imagen que contiene ventana, edificio, grande, jaula

Descripción generada automáticamente

Figura . FIR pasa bajas orden 7, fc=16Hz

Imagen que contiene edificio, persiana

Descripción generada automáticamente

Figura . FIR pasa bajas orden 11 fc=16Hz

Imagen que contiene monitor, computadora, microondas

Descripción generada automáticamente

Figura . FIR pasa bajas orden 17, fc=16Hz

Imagen de la pantalla de una ventana

Descripción generada automáticamente con confianza baja

Figura . FIR pasa bajas orden 30, fc=16Hz

**-Filtros IIR:**

Imagen que contiene edificio

Descripción generada automáticamente

Figura . IIR rechaza banda, orden 4

Imagen de la pantalla de una ventana

Descripción generada automáticamente con confianza baja

Figura . IIR Pasa bajas, orden 4

Pantalla de una ventana

Descripción generada automáticamente con confianza media

Figura . IIR pasa altas, orden 4

## Análisis de resultados

* Filtro FIR: Teniendo en cuenta las consideraciones iniciales de los diseños de los filtros FIR por medio de la herramienta de Matlab (FilterDesigner) se puede evidenciar que se cumplió con la frecuencia de corte especificada en cada uno de los filtros FIR de los diferentes ordenes (7,11,17,30). Para ello se obtuvieron los diagramas de bode por medio del generador de signal Express de National Instruments, generando un barrido a la entrada de 1Hz – 100Hz y evidenciando a curva característica de cada pasa bajos se puede afirmar que mientras más alto sea el orden del filtro, más pronunciada será la pendiente, por lo tanto, el desempeño del filtro se asemejará más al comportamiento de un filtro ideal.
* Filtro IIR: Como es bien sabido, un filtro IIR se caracteriza por depender de las entradas actuales y pasadas, y además de las salidas en instantes anteriores. Esto se tuvo en cuenta en la implementación del filtro por medio de apuntadores y direcciones con el objetivo de garantizar una mayor velocidad y eficiencia al filtro (tanto en FIR como el IIR), esto se ve reflejado en los resultados de cada filtro IIR teniendo en cuenta que tanto el pasa bajos, pasa altos y rechaza banda, funciona de manera apropiada sin embargo no logra cumplir específicamente con los –60dB de ganancia, sin embargo las frecuencias de corte son apropiadas según el diseño inicial de los filtros.

## Conclusiones

* En comparación a los filtros FIR, los filtros IIR presentan mejor desempeño, ya que sus frecuencias de corte fueron más cercanas a las de los filtros que se diseñaron en Matlab.
* Se presenta una mayor resolución en la banda de paso de cada filtro respecto al ENOB, si se aumentara la frecuencia la resolución disminuiría.
* Hay una mayor resolución en la transmisión por USART respecto a SPI, ya que por USART se envían la cantidad total de datos.
* En los filtros IIR al tener dos sumatorias se requiere más tiempo de procesamiento respecto a los FIR que sólo tienen una.
* La simetría de los coeficientes de los filtros permite tiempos de ejecución menores.
* Se puede evidenciar que los filtros IIR son inestables. Esto se debe a que no tienen una fase lineal, mientras que los filtro FIR son más estables debido a la linealidad de su fase.
* Para comprobar la correcta implementación de los filtros FIR se muestra la respuesta impulso en el monitor serial y la salida de este debe ser los coeficientes del filtro.
* Los filtros digitales muestran una implementación precisa, ya que estos no se ven afectados por tolerancia de los componentes como los filtros analógicos.
* Se evidencia en el histograma que la distribución del ruido generado corresponde al comportamiento de una distribución Gaussiana.
* En los filtros FIR, si se quiere tener una atenuación similar a los filtros IIR, se debe aumentar el orden. Los filtros IIR por su parte tienen una atenuación mucho mayor que los filtros FIR como se pudo ver entre el filtro IIR pasa-bajas y cada uno de los diseños de los filtros FIR.
* El tiempo de cálculo depende en los filtros FIR del orden y en los filtros IIR de los coeficientes.

## Anexos

A continuación, se hace referencia al enlace en la plataforma de Github donde se encuentra el repositorio al correspondiente laboratorio:

* <https://github.com/Nicolas338/Proyecto_II_Nico/tree/main/Lab%202>

## Código

Main FIR:

#include "mcc\_generated\_files/mcc.h"

/\*

                         Main application

 \*/

volatile int estados=0;

volatile bool funciona=0;

volatile bool error\_num = 0;

volatile uint8\_t entrega[7];

volatile uint16\_t adc =0;

volatile bool bandera=0;

volatile bool TengoADC=0;

volatile int16\_t palabra1=0b0011000000000000;

volatile int16\_t palabra2=0b0011000000000000;

volatile uint8\_t bloque[2];

volatile uint16\_t DAC\_1=0;

const int N = 31;

volatile int k=0;

volatile int par=0;

#define BL 31

int entradas\_FIR[BL] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

const int coeficientes\_FIR[BL] = {

        -1, 0, 3, 6, 6, 4, -2, -10, -16, -17, -7, 13,

        43, 76, 101, 111, 101, 76, 43, 13, -7,

       -17, -16, -10, -2, 4, 6, 6, 3, 0, -1

    };

long filtrarFIR(){

    int i=0;

    entradas[k]=adc; // guarda el dato del ADC

    int indx=k;

    int indx2= (indx+1)%(N);

    int \*apuntadorB=&coeficientes\_FIR[0];

    int \*apuntadorX=&entradas\_FIR[indx];

    int \*apuntadorX2=&entradas\_FIR[indx2];

    long salida=0;

    par= ((N%2)==0) ? 0:1;

    for (i=0; i<(N-par)/2; ++i){

        salida+=((long)(\*apuntadorX)+(long)(\*apuntadorX2)) \* (long)(\*apuntadorB);

    \*apuntadorB++;

        if(indx2!=(N-1)){

            apuntadorX2++;

            indx2++;

        }

        else{

            apuntadorX2=&entradas\_FIR[0];

            indx2=0;

        }

        if(indx!=0){

            apuntadorX--;

            indx--;

        }

        else{

            apuntadorX=&entradas\_FIR[N-1];

            indx=N-1;

        }

    }

    if(par==1){

        salida+=((long)(\*apuntadorX)) \* (long)(\*apuntadorB);

    }

    k= (k>=N-1) ? 0:(k+1);

    return salida>>16;

}

void ADC\_Trabajando(void){

    adc=ADCC\_GetConversionResult();

    IO\_RD5\_Toggle();

    TengoADC=1;

}

void main(void)

{

    // Initialize the device

    SYSTEM\_Initialize();

    ADCC\_Initialize();

    // If using interrupts in PIC18 High/Low Priority Mode you need to enable the Global High and Low Interrupts

    // If using interrupts in PIC Mid-Range Compatibility Mode you need to enable the Global and Peripheral Interrupts

    // Use the following macros to:

    // Enable the Global Interrupts

    INTERRUPT\_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts

    INTERRUPT\_PeripheralInterruptEnable();

volatile char pack [5];

volatile int valor;

uint16\_t cont\_T6PR;

    ADCC\_SetADIInterruptHandler(ADC\_Trabajando);

    IO\_RE0\_SetHigh();

    IO\_RB4\_SetHigh();

    SPI1\_Open(SPI1\_DEFAULT);

    IO\_RD0\_SetLow();

    IO\_RD1\_SetLow();

    IO\_RB3\_SetLow();

    IO\_RD5\_SetLow();

 while (1)

    {

        if(EUSART1\_is\_rx\_ready()){

            switch (estados){

                case 0:

                    IO\_RD0\_SetLow();

                    entrega[0]=EUSART1\_Read();

                    if (entrega[0]=='1'){

                    estados=1;

                    }

                break;

                case 1:

                    entrega[1]=EUSART1\_Read();

                    estados=2;

                break;

                case 2:

                    entrega[2]=EUSART1\_Read();

                    estados=3;

                break;

                case 3:

                    entrega[3]=EUSART1\_Read();

                    estados=4;

                break;

                case 4:

                    entrega[4]=EUSART1\_Read();

                    estados=5;

                break;

                case 5:

                    entrega[5]=EUSART1\_Read();

                    estados=6;

                break;

                case 6:

                    entrega[6]=EUSART1\_Read();

                    if (entrega[6]=='1'){

                        funciona=1;

                        IO\_RD0\_SetHigh();

                    }else{

                        error\_num=1;

                    }

                    estados=0;

                break;

            }

        }

        /\*if(EUSART1\_is\_tx\_ready()){

            EUSART1\_Write(entrega[0]);

            EUSART1\_Write(entrega[1] );

            EUSART1\_Write(entrega[2] );

            EUSART1\_Write(entrega[3] );

            EUSART1\_Write(entrega[4] );

            EUSART1\_Write(entrega[5] );

            EUSART1\_Write(entrega[6] );

            EUSART1\_Write(0x0D);

            EUSART1\_Write('\n');

            //bandera=0;

         }\*/

        if(funciona == 1){

            funciona=0;

            valor=(entrega[1]-'0')\*10000+(entrega[2]-'0')\*1000+(entrega[3]-'0')\*100+(entrega[4]-'0')\*10+(entrega[5]-'0')\*1;

            //EUSART1\_Write(valor);

        if(valor>=50 && valor<=4096){

            T6CON = 0xF0;

            cont\_T6PR = valor/16;

        }

        if(valor>=4097 && valor<=8193){

            T6CON = 0xF1;

            cont\_T6PR = valor/32;

        }

        if(valor>=4098 && valor<=12288){

            T6CON = 0xF2;

            cont\_T6PR = valor/48;

        }

        if(valor>=4099 && valor<=16384){

            T6CON = 0xF3;

            cont\_T6PR = valor/64;

        }

        if(valor>=4100 && valor<=20480){

            T6CON = 0xF4;

            cont\_T6PR = valor/80;

        }

        if(valor>=20481 && valor<=24576){

            T6CON = 0xF5;

            cont\_T6PR = valor/96;

        }

        if(valor>=24577 && valor<=28672){

            T6CON = 0xF6;

            cont\_T6PR = valor/112;

        }

        if(valor>=28673 && valor<=32768){

            T6CON = 0xF7;

            cont\_T6PR = valor/128;

            IO\_RD1\_SetHigh();

        }

        if(valor>=32769 && valor<=36864){

            T6CON = 0xF8;

            cont\_T6PR = valor/144;

        }

        if(valor>=36865 && valor<=40960){

            T6CON = 0xF9;

            cont\_T6PR = valor/160;

        }

        if(valor>=40961 && valor<=45056){

            T6CON = 0xFA;

            cont\_T6PR = valor/176;

        }

        if(valor>=45057 && valor<=49152){

            T6CON = 0xFB;

            cont\_T6PR = valor/192;

        }

        if(valor>=49153 && valor<=52000){

            T6CON = 0xFC;

            cont\_T6PR = valor/208;

        }

            T6PR=cont\_T6PR;

        }

        /\*if(TMR6\_HasOverflowOccured()){

            IO\_RD5\_Toggle();

        }\*/

        if(TengoADC==1){

            //resultado = ADCC\_GetConversionResult();

            //adc=valor;

            TengoADC=0;

            DAC\_1 = (uint16\_t)(filtrarFIR());

            palabra2 = palabra1 + DAC\_1;

            bloque[1]=palabra2;

            bloque[0]=palabra2>>8;

            IO\_RE0\_SetLow();

            SPI1\_WriteBlock(bloque,2);

            IO\_RE0\_SetHigh();

            IO\_RB4\_SetLow();

            \_\_delay\_us(4);

            IO\_RB4\_SetLow();

            IO\_RB4\_SetHigh();

            IO\_RD5\_Toggle();

            pack[4]= (adc%10);

            adc=(adc/10);

            pack[3]= (adc%10);

            adc=(adc/10);

            pack[2] = (adc%10);

            adc=(adc/10);

            pack[1]=(adc%10);

            adc=(adc/10);

            pack[0]=(adc%10);

            bandera=1;

        }

        if(bandera==1){

            EUSART1\_Write(pack[0] +'0');

            EUSART1\_Write(pack[1] +'0');

            EUSART1\_Write(pack[2] +'0');

            EUSART1\_Write(pack[3] +'0');

            EUSART1\_Write(pack[4] +'0');

            EUSART1\_Write(0x0D);

            EUSART1\_Write('\n');

            bandera=0;

        }

 }

 }

Main IIR:

SPI1(igual para los dos):

#include "spi1.h"

#include <xc.h>

typedef struct {

    uint8\_t con1;

    uint8\_t stat;

    uint8\_t add;

    uint8\_t operation;

} spi1\_configuration\_t;

//con1 == SSPxCON1, stat == SSPxSTAT, add == SSPxADD, operation == Master/Slave

static const spi1\_configuration\_t spi1\_configuration[] = {

    { 0b00100010, 0x40, 0x1, 0 }

};

void SPI1\_Initialize(void)

{

    //Setup PPS Pins

    SSP1CLKPPS = 19;

    SSP1DATPPS = 20;

    //RC3PPS    = 15;

    RC5PPS    = 16;

    //SPI setup

    SSP1STAT = 0x40;

    SSP1CON1 = 0b00100010;

    SSP1ADD = 0x01;

    TRISCbits.TRISC3 = 0;

    TRISCbits.TRISC5 = 0;

    SSP1CON1bits.SSPEN = 0;

}

bool SPI1\_Open(spi1\_modes\_t spi1UniqueConfiguration)

{

    if(!SSP1CON1bits.SSPEN)

    {

        SSP1STAT = spi1\_configuration[spi1UniqueConfiguration].stat;

        SSP1CON1 = spi1\_configuration[spi1UniqueConfiguration].con1;

        SSP1CON2 = 0x00;

        SSP1ADD  = spi1\_configuration[spi1UniqueConfiguration].add;

        TRISCbits.TRISC3 = spi1\_configuration[spi1UniqueConfiguration].operation;

        TRISCbits.TRISC5 = 0;

        SSP1CON1bits.SSPEN = 1;

        return true;

    }

    return false;

}

void SPI1\_Close(void)

{

    SSP1CON1bits.SSPEN = 0;

}

uint8\_t SPI1\_ExchangeByte(uint8\_t data)

{

    SSP1BUF = data;

    while(!PIR3bits.SSP1IF);

    PIR3bits.SSP1IF = 0;

    return SSP1BUF;

}

void SPI1\_ExchangeBlock(void \*block, size\_t blockSize)

{

    uint8\_t \*data = block;

    while(blockSize--)

    {

        SSP1BUF = \*data;

        while(!PIR3bits.SSP1IF);

        PIR3bits.SSP1IF = 0;

        \*data++ = SSP1BUF;

    }

}

// Half Duplex SPI Functions

void SPI1\_WriteBlock(void \*block, size\_t blockSize)

{

    uint8\_t \*data = block;

    while(blockSize--)

    {

        SPI1\_ExchangeByte(\*data++);

    }

}

void SPI1\_ReadBlock(void \*block, size\_t blockSize)

{

    uint8\_t \*data = block;

    while(blockSize--)

    {

        \*data++ = SPI1\_ExchangeByte(0);

    }

}

void SPI1\_WriteByte(uint8\_t byte)

{

    SSP1BUF = byte;

}

uint8\_t SPI1\_ReadByte(void)

{

    return SSP1BUF;

}

## MCC

**-PERIFÉRICOS:**

System module

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

ADCC

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

EUSART

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

MSSP1

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

TMR6

Interfaz de usuario gráfica

Descripción generada automáticamente

**-PIN MANAGER:**

Tabla

Descripción generada automáticamente