Muestreo de Señal PWM

Oscar Ivan Moreno Gutierrez

May 10, 2024

1 Descripción

Este proyecto se centra en realizar medidas sobre dos señales que se obtienen al filtrar una señal PWM generada con el dsPIC. Las medidas que se realizan incluyen el valor máximo, el valor medio y el valor eficaz de las señales. La información obtenida de ambas señales se muestra en una pantalla LCD. Además, se crea un filtro digital utilizando la herramienta de MATLAB para procesar las señales.

2 Archivos y Directorios

- `main.c`: Este es el archivo principal de mi proyecto.
- `pintar_lcd.c`: Este archivo se utiliza para funciones relacionadas con la visualización LCD.
- `funciones.c`: Este archivo contiene funciones basicas y necesarias para el proyecto, como la aplicacion de filtros.

3 Procedimiento

Empezamos con la creacion de las funciones basicas del dsPIC utilizando un SSCP para una salida PWM, Cumpliendo el con el esquema proporcionado (Figura 1).

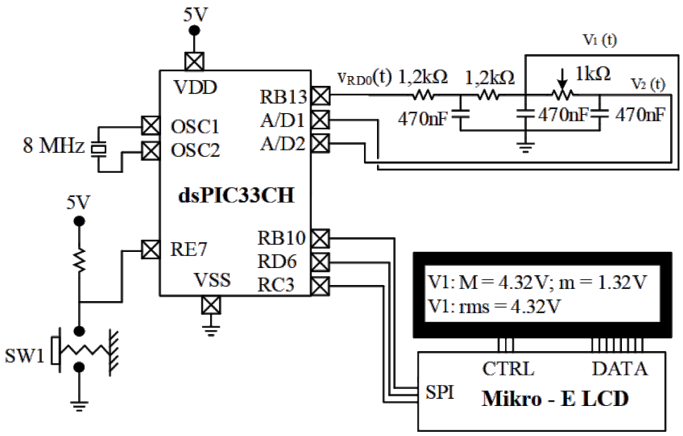


Figure 1: Esquema del proyecto

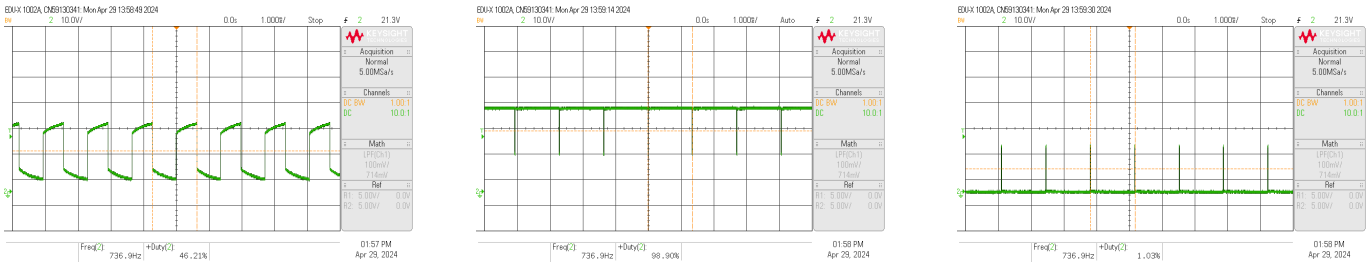
La salida PWM sale por el RB13 y se conecta con un filtro proporcionado por el profesor.Donde se obtienen las señales senoidal y triangular. Ademas de otros cambios:

- El pin RE9 (Switch 3) sale de la pantalla de inicio.
- El pin RE8 (Switch 2) cambia el tipo de filtro aplicado.
- El pin RE7 (Switch 1) cambia el tipo de señal visualizada.
- El pin RA0 (potenciometro) se utiliza para manipular el ciclo de trabajo de la señal PWM.
- El pin RC1 fue asignado para ADC1, que recibe la señal senoidal.
- El pin RC3 fue asignado para ADC2, que recibe la señal triangular.

Una mejora seria la implementacion de manipular el ciclo de trabajo con un modulo ADC. Utilizando un potenciometro para manipular el ciclo de trabajo de la señal PWM.

3.1 Señales

Posteriormente, realizo las medidas de las señales PWM,visualizado en un osciloscopio. Como se muestra en la Figura 2.

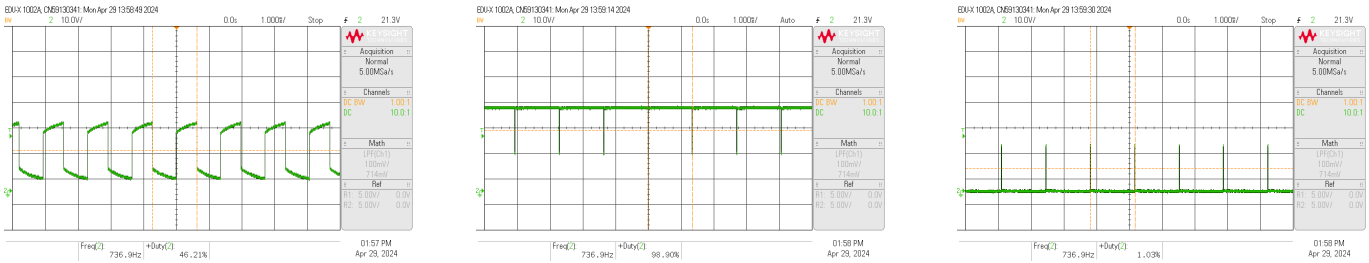


(a) Ciclo de trabajo 46% (b) Ciclo de trabajo 99% (c) Ciclo de trabajo 1%

Figure 2: señal PWM en osciloscopio

Se diseño que el limite de la señal PWM sea de 1% a 99% en su ciclo de trabajo. Se observa que la señal PWM se comporta de manera correcta en el osciloscopio.

La visualización de la señal senoidal se muestra en la Figura 3.

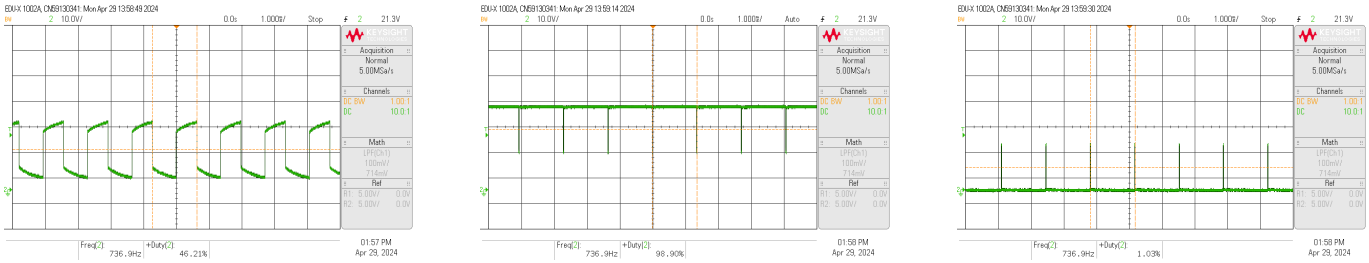


(a) Ciclo de trabajo 46% (b) Ciclo de trabajo 99% (c) Ciclo de trabajo 1%

Figure 3: señal senoidal en osciloscopio

La visualización de la señal triangular se muestra en la Figura 4.

change the im correct



(a) Ciclo de trabajo 46% (b) Ciclo de trabajo 99% (c) Ciclo de trabajo 1%

Figure 4: señal triangular en osciloscopio

change the im correct

3.2 Filtros

Se implemento un filtro digital a las señales senoidal y triangular. Se utilizo la herramienta de MATLAB Filter Desginer para diseñar el filtro. Primero conociendo la escogiendo una frecuencia adecuada de muestreo por la siguiente formula:

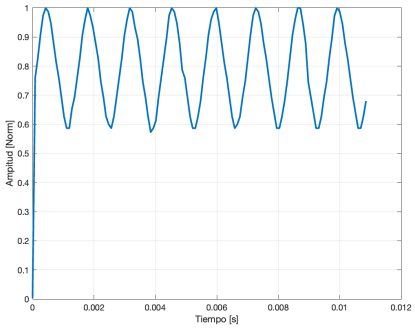
$$f_s = 16 * 737hz = 11792Hz \tag{1}$$

Donde 737 Hz es la frecuencia de nuestras señales. Se obtuvo el siguiente filtro FIR:

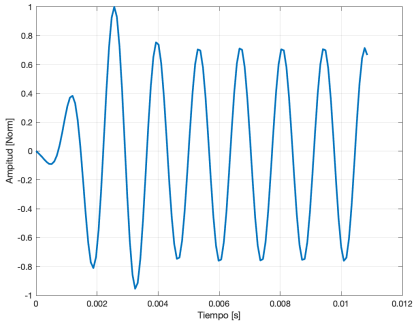
- **Filtro FIR**:Window
- **Frecuencia de muestreo**: 11792 Hz
- **Frecuencia de corte 1**: 710 Hz
- **Frecuencia de corte 2**: 750 Hz

- **Orden del filtro**: 50
- **Tipo de filtro**: Bandpass

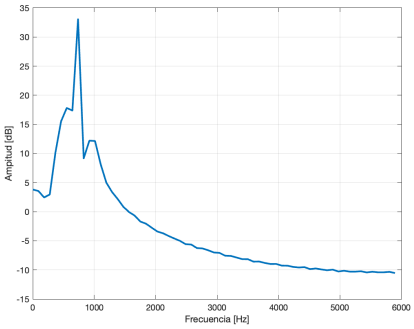
Aplicacion para el Señal senoidal:



(a) Señal de entrada con respecto al tiempo



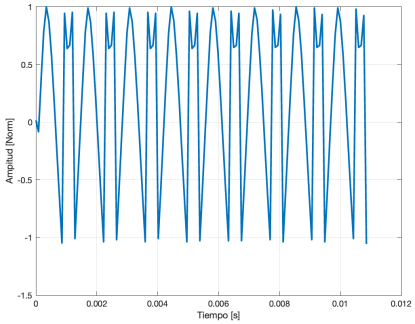
(b) Señal filtrada con respecto al tiempo



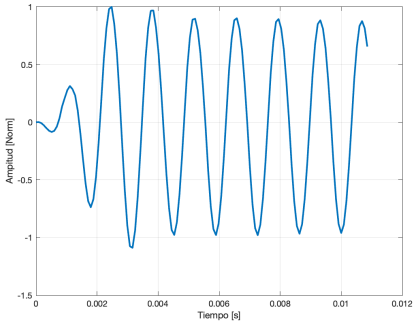
(c) Espectro de la señal filtrada

Figure 5: Filtro aplicado a la señal senoidal

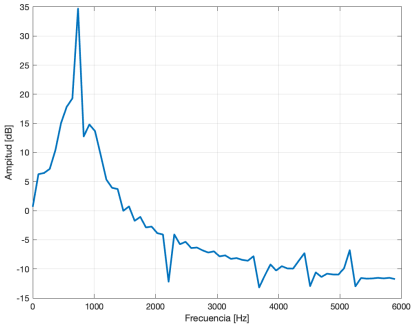
Aplication para la señal triangular:



(a) Señal de entrada con respecto al tiempo



(b) Señal filtrada con respecto al tiempo



(c) Espectro de la señal filtrada

Figure 6: Filtro aplicado a la señal triangular

4 Código

explica
unas p
del cod

Para calcular el RMS:

```
    -----------------funciones.c-----------------
    float calculate_rms(int16_t* samples, uint8_t num_samples) {
    float sum_squared = 0.0;
    float mean_squared=0.0;
    float rms=0.0;
    for (uint8_t i = 0; i < num_samples; i++)
        sum_squared += (float)samples[i] * (float)samples[i];
    mean_squared = sum_squared / num_samples;
    rms = sqrt(mean_squared);
    return rms;
}
```

Para aplicar el filtro FIR:

```
    --------------------main.c-------------------
    // multiplicacmos por 0x0100 para convertir a fractional
    buffer_fractional[buffer_move] = buffer[buffer_move] * 0x0100;
    aplicarFiltroFIR(buffer_fractional,buffer_filter,NUM_SAMPLES);
    // dividimos por 0x0100 para convertir a integer
    buffer[buffer_move] = buffer_filter[0] / 0x0100;
    -----------------funciones.c-----------------
    void aplicarFiltroFIR (fractional* buffer,fractional* output, uint8_t num_samples){
    extern FIRStruct FIR11792Filter;
    FIRDelayInit (&FIR11792Filter);
    FIR(num_samples, &output[0], &buffer[0], &FIR11792Filter);
    return;
}
```