Uso del DAC y ADC con microcontrolador ARDUINO

Daniel Roberto Garcia Miranda
Carrera de Física, Universidad Mayor de San Andrés

Mayo del 2025

Resumen

Se estudió el funcionamiento de los conversores analógico digital (ADC) y digital analógico (DAC), estudiando su uso en 3 aplicaciones, ayudándonos del microcontrolador Arduino. La primera práctica consistió en el estudio de la respuesta en frecuencia de una onda senoidal reconstruida en dos DACs de 8 bits, uno de baja impedancia y el otro de alta impedancia. Los anchos de banda calculados son de 249303.18 Hz para el de baja impedancia; 6847.77 Hz para el de alta impedancia. La segunda aplicación fue el desarrollo de un generador de ondas a partir del mismo DAC, utilizando el DAC de baja impedancia, las reconstrucciones de las ondas cuadrada, diente de sierra y senoidal fueron fielmente reconstruidas. Como última aplicación, se desarrolló un detector de nota musical, que calcula la FFT en el mismo Arduino, se pretendió analizar cómo se deben determinar los diferentes parámetros para el uso de la FFT en este tipo de aplicaciones, se tomaron 128 muestras para el cálculo de la FFT, y una frecuencia de muestreo de 2000 Hz.

Abstract

The operation of Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) was studied, exploring their use in 3 applications with the aid of the Arduino microcontroller. The first experiment involved studying the frequency response of a sinusoidal wave reconstructed by two 8-bit DACs, one with low impedance and the other with high impedance. The calculated bandwidths were 249303.18 Hz for the low impedance DAC and 6847.77 Hz for the high impedance DAC. The second application was the development of a waveform generator using the same DAC, specifically the low impedance one, and the reconstructions of square, sawtooth, and sinusoidal waves were faithfully achieved. The final application involved developing a musical note detector that calculates the FFT on the Arduino itself. The aim was to analyze how to determine the different parameters for using the FFT in this type of application. 128 samples were taken for the FFT calculation, with a sampling frequency of 2000 Hz.

I. Introducción

En las últimas décadas, la tecnología ha cambiado la forma en la que controlamos y vemos el mundo que nos rodea. Junto al desarrollo de la arquitectura de computadoras y la fabricación de circuitos integrados, nace el microprocesador o 'Computadora en Chip' en los años 70; en esta década se fue comercializando y expandiendo mucho más en el mercado.

Un subproducto de la fabricación de los microprocesadores son los microcontroladores, los cuales eran muy similares en fabricación como en programación a los microcontroladores, pero no tan conocidos, incluso para los especialistas.

A pesar de ello, se dejó muy en claro que cualquier electrodoméstico, videojuego, teléfono,

auto y otros artículos, eran inteligentes y programables [2].

I. Microcontroladores y microprocesadores

I. Microcoprocesadores

Un microprocesador es una Unidad Central de Procesamiento (CPU) de una computadora general de propósito general. En la figura 1, se muestra el diagrama de bloques del microprocesador; contiene una unidad de aritmética y lógica, contador de programa, puntero de pila, registros de programa, circuito de reloj y circuitos de interrupción.

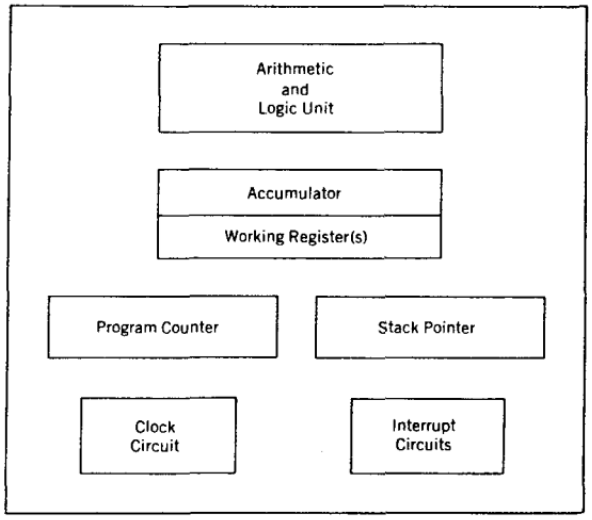

Figura 1: Diagrama de bloques de un microcontrolador, [2].

El término clave para el diseño del microprocesador, es el de Propósito General, el hardware del microprocesador puede ser muy pequeño o muy grande, el sistema puede ser configurado alrededor del CPU según la demanda. Su uso principal es el de tomar datos, hacer cálculos complejos con ellos y almacenarlos en algún dispositivo o mostrarlos. Los programas que el microprocesador usa se guardan y se cargan en la RAM para su uso, algunos lo guardan en la ROM.

Diseñar un microprocesador, tiene como fin ser lo más expandible posible [2].

II. Microcontroladores

Un microcontrolador es una verdadera computadora en chip, su diseño tiene todas las características del microprocesador; para ser una computadora completa se le añade ROM, RAM, Input/Output paralelo y serial, contadores y reloj, figura 2.

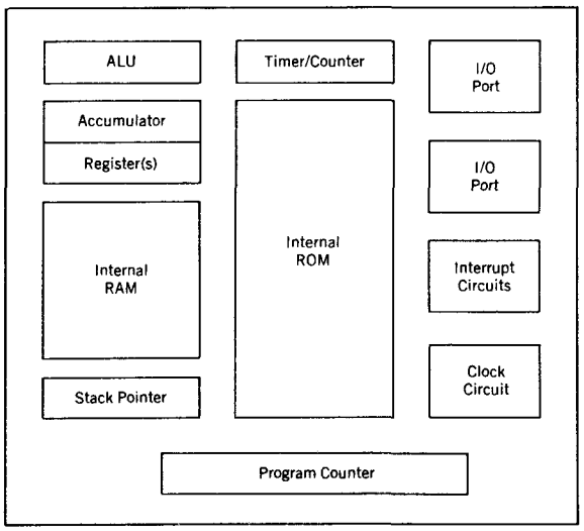

Figura 2: Diagrama de bloques de un microcontrolador, [2].

Al igual que el microprocesador, el microcontrolador es de propósito general, pero está diseñado para tomar datos, hacer cálculos limitados y controlar su entorno en base a estos cálculos. El propósito general del microcontrolador es controlar una máquina, mediante un programa fijo, que se almacena en la ROM y que no cambia durante la vida útil del sistema. La actividad del microprocesador se basa en transferir datos y código de la memoria externa a la CPU, esto desde y hacia sus propios pines, toda la arquitectura y el conjunto de instrucciones están optimizados para manejar datos en bits y bytes [2].

Los microcontroladores también se los denomina sistemas embebidos, sistemas informáticos con función específica y dedicada. Algunos ejemplos de microcontroladores son Arduino o Teensy 3.

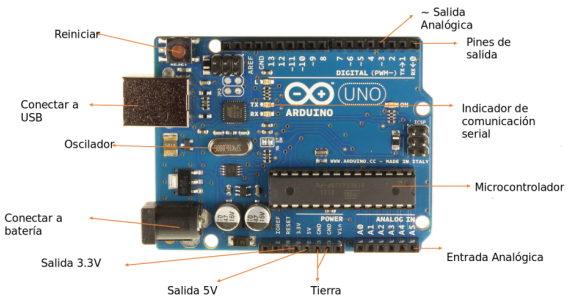

Figura 3: Microprocesador Arduino UNO.

Estos microcontroladores pueden tener conectados sensores y leer sus datos, hacer cálculos e implementar las leyes del control; sin embargo, estos dispositivos son digitales, lo que significa que su interpretación de datos es discretizada, diferente al mundo real, que es analógico; todo lo que vemos es continuo.

Para lograr esto, los microcontroladores utilizan tanto la conversión Digital Analógica (DAC), que convierte valores binarios a voltajes de salida reales; como la conversión Analógica Digital (ADC), para convertir una señal de entrada a datos digitales, que el microcontrolador puede usar [6]. Tanto el DAC como el ADC son circuitos electrónicos que están integrados en el microcontrolador.

II. Conversor digital analógico (DAC)

Un DAC representa un conjunto de códigos de entrada digital discreta, que corresponde a un número de valores de salida analógica discreta. Un DAC puede considerarse como un potenciómetro controlado digitalmente, cuya salida es un valor de voltaje analógico de la escala completa determinado por la entrada digital.

La salida analógica del DAC depende de una entrada analógica de referencia, la precisión de esta referencia es el factor limitante de la precisión absoluta de un DAC, algunos DACs tienen referencias externas o internas, los más simples no cuentan con ninguna [1].

I. DAC R-2R

Una de las estructuras más comunes de DACs es la red de escalera resistiva R-2R, figura 4. Utiliza resistencias de solo dos valores diferentes, con relación de 2:1, un DAC de N bits requiere 2N resistencias para ajustar. La red R-2R tiene dos formas de ajustar, conocidas como modo de voltaje y modo de corriente.

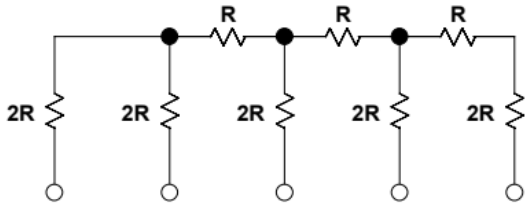

Figura 4: Arquitectura de un DAC R-2R

En el modo voltaje, los brazos de la escalera conmutan entre la referencia y tierra, y la salida se toma en un extremo como voltaje o corriente. Tiene como ventajas una salida de voltaje e impedancia de salida constante; sin embargo, la impedancia de referencia varía con el código. Los interruptores de voltaje operan en rangos de voltaje amplios, además de que la ganancia no se puede ajustar fácilmente [5].

III. Conversor analógico digital (ADC)

Un ADC ideal representa de forma única todas las entradas analógicas dentro de un rango determinado de códigos de salida digital. Como la escala analógica es continua, y la salida digital es discreta, existe un proceso de cuantificación que introduce un error. Este error se puede reducir, aumentado el número de salidas discretas (el número de bits). El ancho de paso, se define como 1 LSB (Last significant bit), es también una medida de la resolución del convertidor, ya que define el número de divisiones del rango analógico completo, es decir, 1/2 LSB representa una cantidad analógica igual a la mitad de la resolución analógica [1].

Al igual que los DACs, la relación entre la salida digital y la entrada analógica de un ADC depende de un valor de referencia, siendo el factor limitante en la precisión absoluta de un ADC.

Al igual que los DAC, gran parte de los ADC utilizan referencias externas (insertar figura) y tienen un terminal de entrada de referencia, mientras que otros disponen de una salida

de una referencia interna. Los ADC más simples, por supuesto, no cuentan con ninguna: la referencia está integrada en el chip del ADC y no tiene conexiones externas, figura 5, [5].

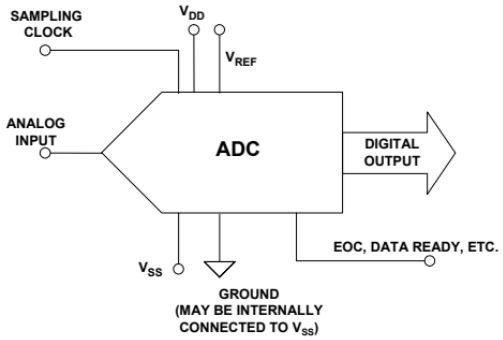

Figura 5: ADC básico con referencia externa.

IV. ARDUINO UNO

Arduino es una plataforma de código libre diseñada para facilitar proyectos de electrónica, posee un entorno gráfico de desarrollo que usa un lenguaje de programación processing/wiring (C/C++) y un gestor de arranque; en lo que respecta hardware está compuesta por un microcontrolador (ATMEGA328P) y puertos de entrada y salida [4].

Entre sus componentes tenemos:

1. Pines digitales: Son las conexiones digitales de los dispositivos conectados en la placa, Una placa de arduino cuenta con 14 pines digitales. Sus estados pueden ser LOW, 0V enviados desde la placa y HIGH, 5V enviados desde la placa.

2. Pines analógicos: Miden rangos de valores de voltaje, entre 0V y 5V, estos valores se representan con números enteros entre 0 y 1023 (números de 10 bits), la información que se escriben desde los pines son números entre 0 y 255 (8 bits).

3. Pines alimentación sensores: Son pines que permiten alimentar componentes, con voltajes de 5V y 3.3V, o tierra.

4. Microcontrolador de comunicaciones: Se encarga las comunicaciones con todo lo que se conecta a la placa.

5. Microcontrolador de programación: El cerebro del Arduino, en ésta, la placa almacena el programa que tiene que ejecutar y lo ejecuta. Se programa utilizando el IDE (Entorno de desarrollo Integrado).

6. Botón reset: Permite reinici al el programa cargado.

7. Puero USB: Nos permite comunicarnos con el arduino, alimenta la placa, carga los programas y envía información.

8. Conector de alimentación: Solamente sirve para alimentar la placa, debe alimentar de 7V a 12V.

Arduino es una buena opción para empezar a trabajar y/o estudiar temas relacionados con robótica, ya que es simple, económico, multiplataforma y cuenta con hardware abierto y extensible, [3].

En esta práctica de laboratorio se estudió el funcionamiento de DAC y del ADC, utilizando una placa de Arduino UNO como microcontrolador.

II. Metodología

Como base de la práctica, se utilizó una placa de Arduino UNO con su compilador Arduino IDE instalado en el ordenador.

En la figura 6 se muestra un DAC de 8 bits basado en una arquitectura R-2R. Sus 8 entradas digitales se conectan al PORTD del Arduino (pines digitales del 0 al 7). Gracias a que se pueden enviar directamente valores en formato byte a este puerto, no es necesario configurar cada pin individualmente como salida, lo que simplifica notablemente el código.

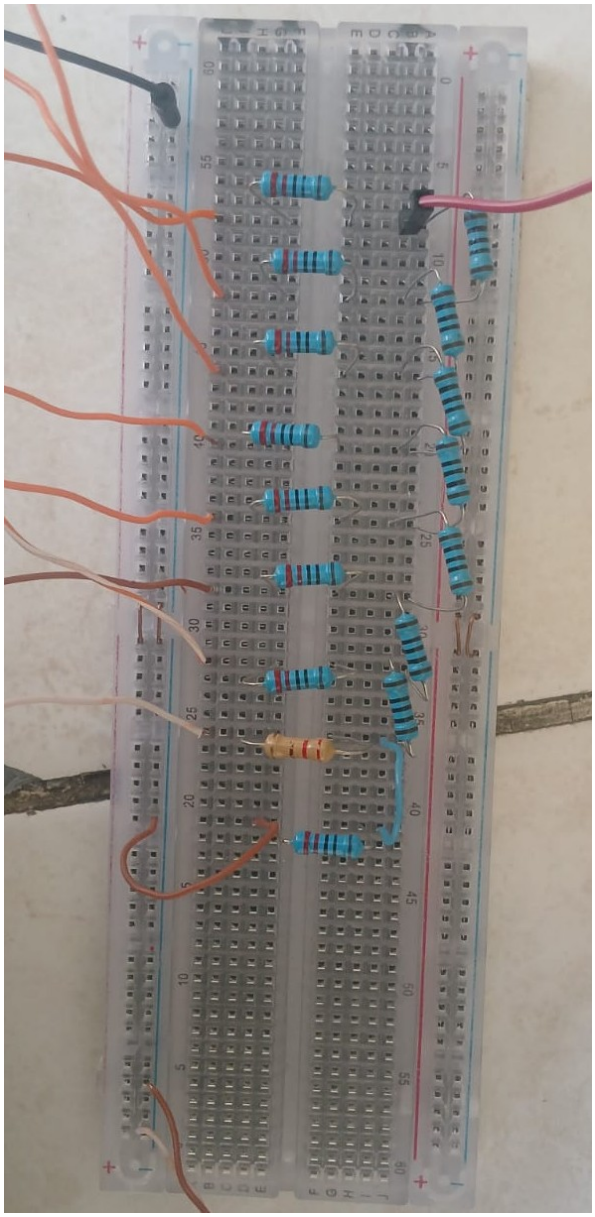

Figura 6: DAC construido con una arquitectura R-2R, se teien conectado al arduino, la entrada analógica al generador de funciones y la salida al osciloscopio

Con el DAC, se realizaron 2 prácticas:

I. Reconstrucción de señal

Como primera aplicación, se reconstruyó una señal analógica que recibe el Arduino. Como se explicó en la parte teórica, las entradas analógicas que puede recibir el ADC de Arduino son de 10 bits (un entero de 0 a 1023).

Este número de 10 bits recibido se puede transformar en un número de 8 bits con el comando $byte valor 8bits = valor 10bits \check{}2$;, este comando recorre los 2 bits menos significativos; esta señal es la que se manda al DAC.

Al conectar un generador de ondas a una entrada analógica del Arduino y cargar el código del anexo A, es posible observar la señal reconstruida mediante un osciloscopio.

El objetivo es visualizar en el osciloscopio la respuesta en frecuencia de la señal reconstruida, midiendo su amplitud y determinando su ancho de banda, así como observar las distintas formas que puede adoptar la señal de salida. También se pretende verificar que la frecuencia de muestreo definida en el código coincide con el umbral a partir del cual se manifiesta el fenómeno de aliasing, es decir, cuando el DAC deja de representar fielmente la forma original de la señal.

II. Generador de Ondas

Se mandó mediante código, señales digitales que reconstruyen la forma de una onda dada; se generaron 3 ondas: senoidal, cuadrada y diente de sierra. Un potenciómetro conectado a una entrada analógica, cambiará la frecuencia de la onda, y un pulsador cambiará el tipo de onda generada, figura 7, creando así, un generador de ondas.

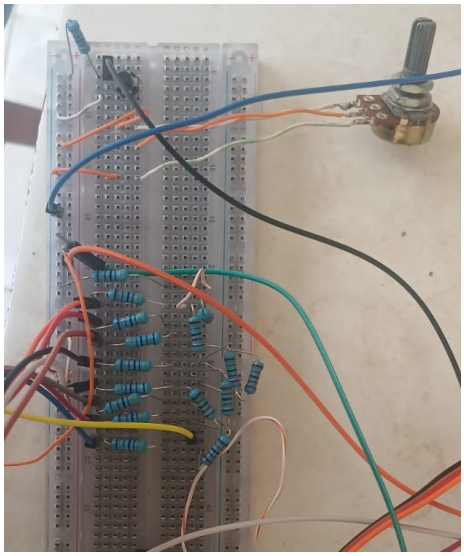

Figura 7: Generador de ondas, sobre el mismo DAC R-2R, se le añadió un pulsador y la entrada analógica es ahora un potenciómetro.

El código utilizado se encuentra en el anexo B).

Se pretende observar la reconstrucción de cada onda y el funcionamiento del generador de ondas.

III. FFT en Arduino

Como última aplicación, se construyó un circuito compuesto por 8 LEDs conectados al PORTD del Arduino. La entrada analógica estará a cargo de un módulo de micrófono comercial (modelo PIC 4/6V), cuya conexión se observa en la figura 8.

Con este circuito se desarrolló un código que implementa la FFT para detectar notas musicales, cada nota corresponde a un led.

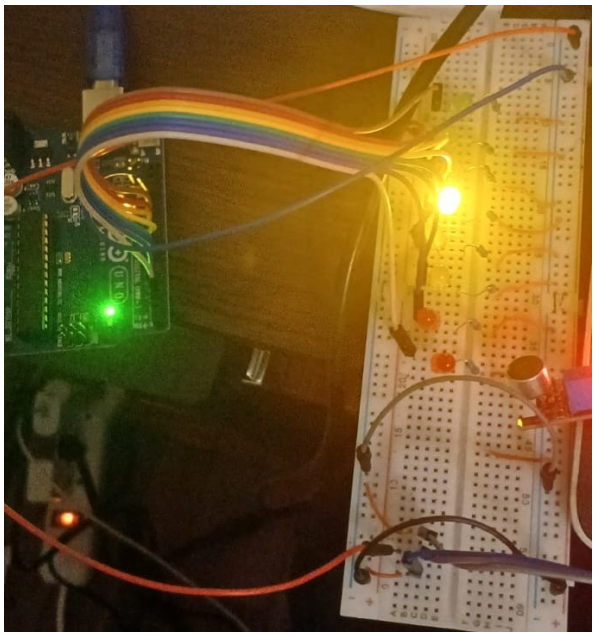

Figura 8: Circuito que detecta la nota reproducida al micrófono con un led y en el puerto serial de Arduino.

En el código de Arduino se programó la FFT. Para el análisis en tiempo real, los valores de la entrada analógica se agrupan en bloques de 128 muestras. A partir de estos datos Arduino calcula la FFT en tiempo real.

El objetivo es analizar como se deben ajustar los parámetros para la FFT en este tipo de aplicaciones, tales como la frecuencia de muestreo, el número de muestras, la tolerancia, entre otros.

El código utilizado se encuentra en el anexo C.

III. Análisis de resultados

I. DAC

Se construyeron 2 DACs, ambos con arquitectura R-2R, uno de 100 Ω y 200 Ω, el otro de 2000 Ω y 4000 Ω. Se inició el análisis utilizando el primer DAC. A continuación se muestran una serie de reconstrucciones de una onda senoidal.

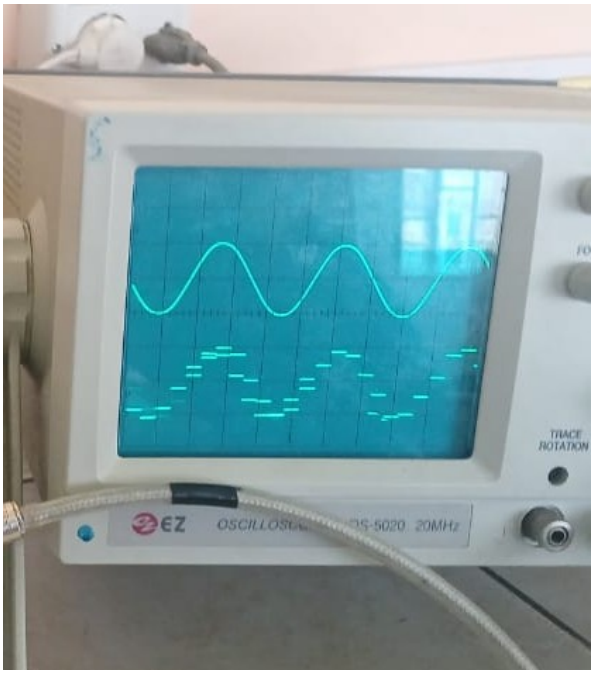

Figura 9: Arriba se encuentra la onda senoidal y abajo la reconstrucción del DAC, la frecuencia del generador no es muy alta

En la figura 9, se observa la reconstrucción a baja frecuencia del DAC, se observa que se reconstruye la forma de la onda senoidal mediante escalones, esto debido a que la señal digital que se manda del Arduino al DAC es discreta. Cuando los escalones empiezan a dejar de reconstruir la forma de la onda de entrada, empieza el aliasing. El aliasing debería a comenzar a presentarse, a partir de la frecuencia de Nyquist (la mitad de la frecuencia de muestreo). La frecuencia de muestreo está dada en el código, anexo A. El

periodo de muestreo del DAC es de 100 microsegundos, es decir, la frecuencia de muestro es de 10000 Hz.

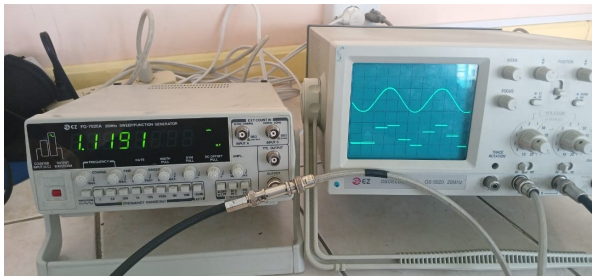

Figura 10: Se observa la reconstrucción de la onda y la frecuencia a la que está la onda senoidal,

En la figura 10, se empieza a observar que el DAC está empezando a reconstruir la señal con aliasing, sin embargo, el aliasing comienza a presentarse a una frecuencia mucho más baja que la de Nyquist (5000 Hz), esto se puede deber a diferentes defectos propios del DAC, como ser la existencias capacitancias parásitas.

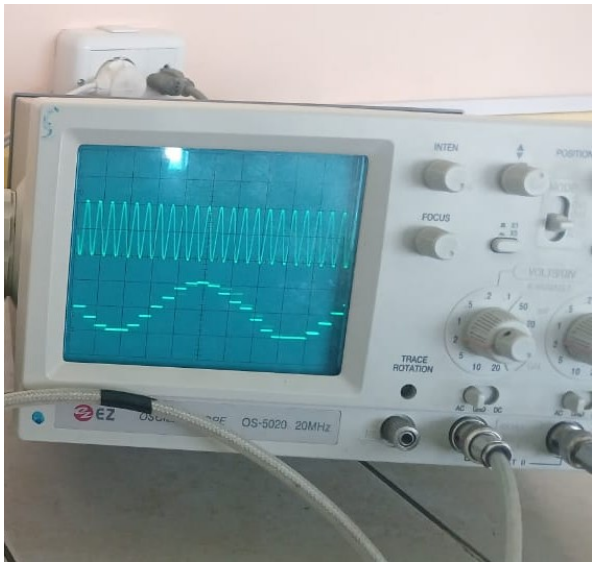

Figura 11: Reconstrucción de la señal con Aliasing.

En la figura 11, se observa el aliasing a una frecuencia muy alta. Cabe resaltar que a pesar de superar la frecuencia de muestreo, la amplitud aun no disminuye.

Luego de observar y anotar la respuesta en frecuencia del primer DAC, se procedió a estudiar el segundo.

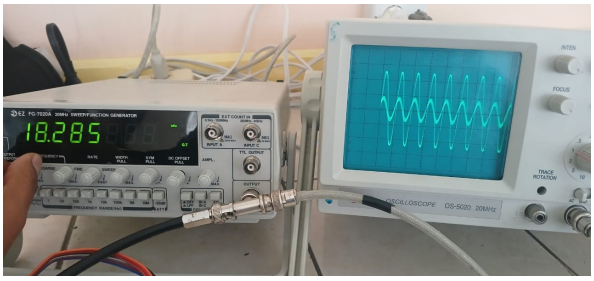

Figura 12: Señal reconstruida por el segundo DAC en aliasing.

En la figura 12 se observa la reconstrucción de la señal en aliasing. Aunque la señal reconstruida siga teniendo la forma de la señal original, el efecto del aliasing se presenta en la amplitud, a medida que se seguía aumentando la frecuencia, la amplitud bajó a casi cero y por poco se veía que la reconstrucción ya no se parecía a la onda senoidal.

Para ambos DACs, se tomaron diferentes valores de las amplitudes y de frecuencias de las reconstrucciones. Se graficaron para calcular el ancho de banda (en amplitud) de cada DAC, figura 13:

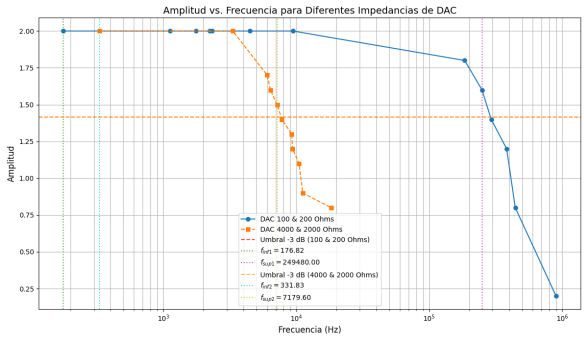

Figura 13: Ancho de banda para el DAC de 200 y 100, 249303.18. Ancho de banda para el DAC de 4000 y 2000, 6847.77

En la figura 13 se observa la gráfica de amplitud contra frecuencia de las reconstrucciones de cada DAC. Se observa que en el primer DAC (línea azul), la amplitud de la señal empezó a bajar a frecuencias muy altas, llegando casi al límite de frecuencia del generador de ondas. Cabe aclarar que, para las frecuencias mayores a los 5000 Hz, la señal ya se encontraba en aliasing (figura 10). Por otro lado, la amplitud de la señal del

segundo DAC (línea naranja), a partir de casi los 5000 Hz, empezó a disminuir y comenzaba a presentar aliasing.

II. Generador de ondas

Una vez preparado el generador de ondas, se comenzó observando cada onda en el osciloscopio, las ondas cuadrada, diente de sierra y senoidal generadas se observan en las figuras 14, 15 y 16.

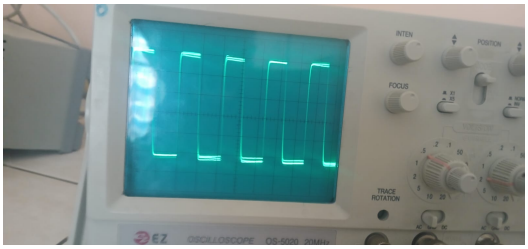

Figura 14: Onda cuadrada generada por el DAC.

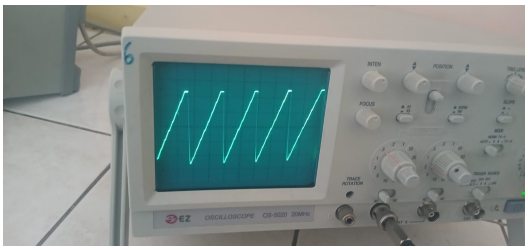

Figura 15: Onda Diente de Sierra generada por el DAC.

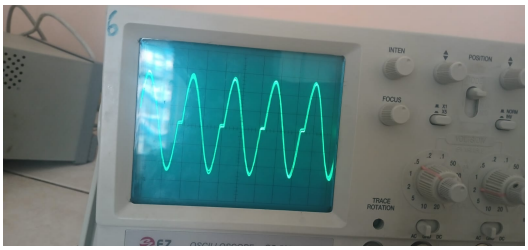

Figura 16: Onda senoidal generada por el DAC.

Para la reconstrucción de estas ondas mediante el DAC, se tomó en cuenta la frecuencia a la que el Arduino lee la señal del potenciómetro y cuánto tarda en procesarla y enviarla. En el código el delayTime es el que maneja esta frecuencia de actualización. Una fórmula para la frecuencia es $F_{DAC} = \frac{1}{delaytTime} = 256 * Frec$, al imponer esta fórmula el DAC se actualizará mucho más rápido que la onda que genera, lo cual nos ahorra el preocuparnos de los problemas relacionados con estas dos frecuencias.

Por otro lado, como se observa en las figuras 14, 15 y 16, las reconstrucciones de cada tipo de onda presentan una forma bastante fiel. Sin embargo, ciertos detalles como los pequeños tiempos muertos en la onda senoidal o la suavidad en las esquinas de la onda cuadrada pueden atribuirse al settling time del DAC, el cual está relacionado con el delayTime establecido en el código.

Adicionalmente, estos efectos también pueden ser consecuencia de las capacitancias parasitarias internas del propio circuito DAC, que afectan la respuesta dinámica del sistema.

III. FFT

Una vez instalado el circuito, se procedió a reproducir diferentes notas musicales, observando la respuesta tanto en los LEDs como en la salida serial.

En la figura 17 se muestra la activación de los LEDs al generar la nota C4 (Do), mientras que en la figura 18 se observa la salida del monitor serial del entorno Arduino IDE.

El programa está diseñado para indicar la frecuencia que se está reproduciendo, así como la nota correspondiente, considerando una tolerancia de 15 Hz.

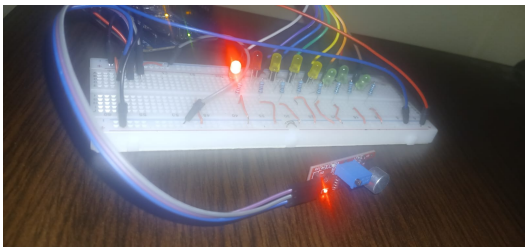

Figura 17: Respuesta de los leds cuando se reproduce la noda C4 (Do).

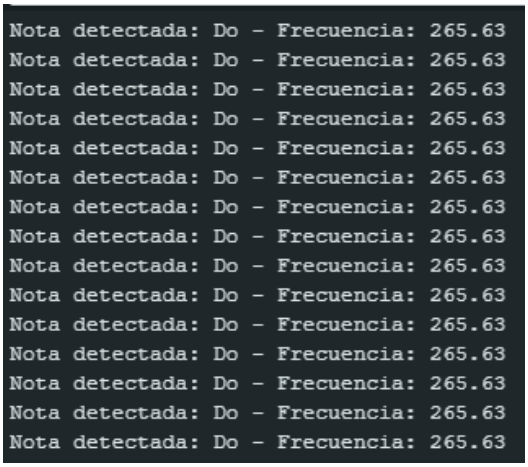

Figura 18: Monitor serial de Arduino IDE cuando se reproduce un sonido cuya frecuencia esta dentro del rango de tolerancia de la nota C4 (Do)

El programa realiza la FFT con 128 muestras, es decir, toma 128 valores de la entrada analógica del micrófono y, con ellos, calcula la transformada rápida de Fourier (FFT).

El valor de 128 está relacionado con las limitaciones de memoria RAM del Arduino, que dispone de aproximadamente 2 KB. En el código, tanto la parte real como la imaginaria de la señal se declaran como variables de tipo float, las cuales ocupan 4 bytes cada una. Estas dos componentes, en conjunto, utilizan alrededor de 1 KB de la RAM disponible.

En el código se observa también un ajuste a la señal analógica del micrófono, eliminando el offset y amplificando la entrada.

La frecuencia de muestreo de 2000 Hz, fue escogida tomando en cuenta la resolución con la que Arduino detecta la frecuencia dominante y la frecuencia de Nyquist.

La resolución del Arduino se calcula dividiendo la frecuencia de muestreo entre el número de muestras, para 2000 Hz aproximadamente la resolución es de 15 Hz, ésta discrepancia se compensa con la tolerancia de 15 Hz puesta en el código. Por otro lado, la frecuencia de Nyquist limita el máximo de frecuencia que la FFT nos puede calcular. Para los fines de la práctica, 1000 Hz es suficiente para poder identificar la escala musical.

IV. Conclusiones

Las 3 prácticas realizadas aportaron significativamente a la comprensión del comportamiento y uso de los microcontroladores y, específicamente, el del DAC. Para la construcción de los DACs, se decidió utilizar la arquitectura R-2R debido a la disponibilidad de los materiales, como resistencias, protoboards y cables. En la práctica, se observan diferencias significativas entre ambos DACs, figuras 11, 12 y 13. Entre las anomalías observadas al utilizar el DAC de menor impedancia, se detectó que la señal reconstruida comenzó a presentar aliasing a frecuencias inferiores a la frecuencia de Nyquist. Esto podría deberse a la sensibilidad del propio DAC al ser de baja impedancia. Específicamente, es probable que se manifiesten capacitancias parásitas que alteran la respuesta en frecuencia de la señal.

En contraste, el DAC de alta impedancia (2000Ω -4000 Ω), no mostró la misma sensibilidad a dichas capacitancias, lo que explica que su respuesta en frecuencia no presentara anomalías.

En el generador de funciones se utilizó el DAC de baja impedancia. Además de lograr una reconstrucción fiel de las señales, se observó que la suavidad en las transiciones de la onda cuadrada y el comportamiento de la onda senoidal se vieron influenciados por las capacitancias parásitas del circuito.

En la práctica del uso de la FFT en Arduino, se observó la gran importancia de manejar los parámetros de la frecuencia de muestreo y el número de muestras. Ya que, si se buscara detectar un mayor rango de frecuencias, por ejemplo, todo el espectro audible (20000 Hz), la frecuencia de muestreo tendría que ser de al menos 40000 Hz, dándonos una resolución de 312 Hz. Una limitación importante observada fue la del tamaño de los conjuntos de muestras; no es posible aumentar significativamente este número, pues se empezaría a tener problemas con la RAM del Arduino. Como se había explicado en el análisis, los parámetros escogidos en la práctica resultaron ideales para detectar la escala musical.

En general, la implementación de los micro-

controladores a la ciencia es un paso muy importante en cuanto a las ventajas científicas y tecnológicas posibles. En esta práctica se pretendió empezar a comprender y estudiar el cómo éstos funcionan; si bien la teoría está ampliamente desarrollada, se espera implementar los conocimientos teóricos y prácticos en las diferentes prácticas futuras.

Referencias

[1] *Understanding Data Converters.* Texas Instruments.

[2] Kennet J. Ayala. *The 8052 Microcontroller. Architecture, Programming and Application.* WEST Publishing Company, 1991.

[3] A. Moreno S. Córcoles. *Aprende Arduino en un fin de semana.* Time of Software.

[4] Grupo Halley. *Introducción a Arduino.* Universidad Industrial de Santander.

[5] J. Bryant W. Kester. *Analog - Digital Conversion.* Analog Devices, 2004.

[6] Tyler Ross Lambert. *An Introduction to Microcontrollers and Embedded Systems.* Tyler Ross Lambert, 2017.

Anexos

A. Código Arduino - DAC

```
void setup() {
  DDRD = 0xFF; // Configura todos los pines 0 a 7 como salida

  // Si no vas a usar Serial, esto no se necesita
  // Serial.begin(9600);
}

void loop() {
  int valor10bits = analogRead(A0);   // Lectura de 10 bits
  byte valor8bits = valor10bits >> 2; // Recorta 2 bits, ahora es de 8 bits

  PORTD = valor8bits; // Enviar directamente a los pines 0  7

  delayMicroseconds(100); // Para controlar la velocidad
}
```

Listing 1: Codigo para el DAC, el codigo convierte la señal analógica de 10 bits a 8 para mandar al DAC.

B. Código Arduino - Generador de Ondas

```

#include <math.h>

const int potPin = A0;          // Potenci metro
const int buttonPin = 8;        // Pulsador
```

```
int waveType = 0;               // 0: seno, 1: cuadrada, 2: sierra
bool lastButtonState = HIGH;

byte sineTable[256];

void generateSineTable() {
  for (int i = 0; i < 256; i++) {
    sineTable[i] = (sin(i * 2 * PI / 256.0) * 127.5) + 127.5;
  }
}

void setup() {
  DDRD = 0xFF;                    // Pines  D 0 D7  como salida (PORTD)
  pinMode(buttonPin, INPUT_PULLUP);
  generateSineTable();          // Generar tabla de seno
}

void loop() {
  // Leer el bot n (cambio de forma de onda)
  bool currentState = digitalRead(buttonPin);
  if (lastButtonState == HIGH && currentState == LOW) {
    waveType = (waveType + 1) % 3;
    delay(200); // Antirrebote
  }
  lastButtonState = currentState;

  // Leer potenci metro y mapear a frecuencia
  int potValue = analogRead(potPin);
  float freq = map(potValue, 0, 1023, 1, 500); // de 1 Hz a 500 Hz
  float delayTime = 1000000.0 / (freq * 256);  // tiempo por muestra en  s

  // Generar la onda
  for (int i = 0; i < 256; i++) {
    byte value;

    switch (waveType) {
      case 0: value = sineTable[i]; break;      // Senoidal
      case 1: value = (i < 128) ? 255 : 0; break; // Cuadrada
      case 2: value = i; break;                  // Diente de sierra
    }

    PORTD = value;
    delayMicroseconds(delayTime);
  }
}
```

Listing 2: Codigo para el generador de funciones, se mandan señales de 8 bits al DAC.

C. Código Arduino - FFT

```
//Todo
#include <arduinoFFT.h>

#define SAMPLES 128
#define SAMPLING_FREQUENCY 2000
#define ANALOG_PIN A0
```

```
7
8  ArduinoFFT<float> FFT;
9
10 float vReal[SAMPLES];
11 float vImag[SAMPLES];
12
13 unsigned long sampling_period_us;
14
15 // Pines de LEDs para Do, Re, Mi, Fa, Sol, La, Si, Do'
16 const int ledPins[8] = {3, 4, 5, 6, 7, 8, 9, 10};
17 // Frecuencias objetivo para las notas (Hz)
18 double noteFrequencies[8] = {
19   261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25
20 };
21
22 const char* noteNames[8] = {
23   "Do", "Re", "Mi", "Fa", "Sol", "La", "Si", "Do'"
24 };
25
26 const double TOLERANCE = 15.0;  // Rango aceptable en Hz
27
28 void setup() {
29   Serial.begin(115200);
30   sampling_period_us = round(1000000.0 / SAMPLING_FREQUENCY);
31
32   for (int i = 0; i < 8; i++) {
33     pinMode(ledPins[i], OUTPUT);
34     digitalWrite(ledPins[i], LOW);
35   }
36 }
37
38 void loop() {
39   // Tomar muestras
40   for (int i = 0; i < SAMPLES; i++) {
41     unsigned long t0 = micros();
42     vReal[i] = analogRead(ANALOG_PIN);
43     vImag[i] = 0;
44     while (micros() - t0 < sampling_period_us);
45   }
46
47   // Eliminar offset y amplificar
48   double mean = 0;
49   for (int i = 0; i < SAMPLES; i++) mean += vReal[i];
50   mean /= SAMPLES;
51   for (int i = 0; i < SAMPLES; i++) vReal[i] = (vReal[i] - mean) * 10;
52
53   // FFT
54   FFT.windowing(vReal, SAMPLES, FFT_WIN_TYP_HAMMING, FFT_FORWARD);
55   FFT.compute(vReal, vImag, SAMPLES, FFT_FORWARD);
56   FFT.complexToMagnitude(vReal, vImag, SAMPLES);
57
58   // Buscar el pico de frecuencia
59   double maxMag = 0;
60   int maxIndex = 0;
61
62   for (int i = 1; i < SAMPLES / 2; i++) {
63     if (vReal[i] > maxMag) {
```

```
64        maxMag = vReal[i];
65        maxIndex = i;
66      }
67    }
68
69    double dominantFreq = maxIndex * ((1.0 * SAMPLING_FREQUENCY) / SAMPLES);
70
71    // Limpiar LEDs
72    for (int i = 0; i < 8; i++) digitalWrite(ledPins[i], LOW);
73
74    bool noteDetected = false;
75
76    // Detectar nota y encender LED correspondiente
77    for (int i = 0; i < 8; i++) {
78      if (abs(dominantFreq - noteFrequencies[i]) < TOLERANCE) {
79        digitalWrite(ledPins[i], HIGH);
80        Serial.print("Nota detectada: ");
81        Serial.print(noteNames[i]);
82        Serial.print(" - Frecuencia: ");
83        Serial.println(dominantFreq, 2);
84        noteDetected = true;
85        break;
86      }
87    }
88
89    if (!noteDetected) {
90      Serial.print("Frecuencia detectada: ");
91      Serial.print(dominantFreq, 2);
92      Serial.println(" Hz - Sin nota espec fica");
93    }
94
95    delay(100);
96  }
```

Listing 3: Código que calcula la FFT y depende la frecuencia maxima, de una muestra de 128 samples, con tolerancia de 15 Hz y frecuencia de muestreo de 2000 Hz.