Desafío 1 – Analizador de señales analógicas

Andrés David Durán Quiñones
Ingeniería Electrónica
Universidad de Antioquia
Medellín, Colombia
andres.duranq@udea.edu.co

Daniela Escobar Velandia
Ingeniería Electrónica
Universidad de Antioquia
Medellín, Colombia
daniela.escobarv@udea.edu.co

***Resumen*—Este informe describe el desafío 1, enfocado en implementar un sistema de adquisición y visualización de datos para una señal analógica generada en la plataforma Tinkercad, utilizando un Arduino Uno. El sistema trabajará con tres tipos de señales: senoidal, triangular y cuadrada, permitidas por la aplicación. Para cada una, se determinarán su tipo de onda, frecuencia y amplitud, con el fin de explorar conceptos clave como punteros, arreglos y memoria dinámica en C++. Asimismo, se estudiarán las limitaciones del Arduino Uno en comparación con sistemas de mayor capacidad.**

***Index Terms*—Señal, Frecuencia, Amplitud, Arduino, Puntero, Memoria dinámica.**

I. Análisis del problema y consideraciones para la alternativa de solución propuesta

Después de analizar el problema, se realizó un debate entre los integrantes para lograr una mejor comprensión del mismo. Tras el debate, se concluyó que el problema se divide en tres áreas principales. En primer lugar, los conceptos teóricos, los cuales se enfocan en el estudio de señales, amplitud, frecuencia, muestreo, microcontroladores (en este caso, Arduino Uno), entre otros. En segundo lugar, está el problema de las conexiones, donde se investigó cómo conectar un generador de señales, cómo conectar dos botones y la importancia de usar resistencias en configuración pull-down o pull-up, así como la conexión y funcionamiento de un display LCD de 16x2. Por último, pero no menos importante, está el código, que permitirá cumplir con las funciones principales del sistema, tales como la adquisición de datos por un pin analógico, la lógica de los botones, el despliegue en el display, la lógica para encontrar la amplitud y frecuencia de la señal, y la lógica para determinar el tipo de señal.

Tras el análisis, logramos idear una alternativa que ofreciera una solución al problema. Esta alternativa se presenta en el diagrama de la figura 1.

Este diagrama expresa la solución implementada, donde se observa el flujo y las funciones que ejecuta cada parte del sistema. La parte más importante es el procesamiento de los datos, en la cual se obtienen la frecuencia, amplitud y tipo de onda. Para ello, se sigue la siguiente estrategia:

Frecuencia: Para determinar la frecuencia de la señal, se identifica un cruce ascendente (cuando la señal pasa de un valor bajo a uno alto) comparando el valor actual con el valor medio o valor DC de la señal. Al detectar el primer cruce, se registra el tiempo inicial, y al detectar un segundo cruce, se mide el tiempo final. La diferencia entre el tiempo de inicio y el tiempo final nos da el período de la señal, es decir, el tiempo que tarda en completarse un ciclo. La frecuencia se obtiene invirtiendo el período. Después de cada cruce, se reinicia el tiempo para medir el siguiente ciclo, actualizando así la frecuencia constantemente.

Amplitud: Se capturan el valor máximo y el valor mínimo de la señal. La diferencia entre estos valores se divide por 1023, que es la resolución del Arduino Uno (10 bits), y luego se multiplica por 5, que es el valor de referencia.

Tipo de señal: Para identificar el tipo de onda, se comenzó con la onda cuadrada, la cual fue la más fácil de reconocer, ya que sus valores se repiten con distinto signo. Si los valores se repiten más de 100 veces, se clasifica como una onda cuadrada. Para las señales senoidal y triangular, se calculó la pendiente entre el punto anterior y el actual, y entre el actual y el siguiente. Restando estas dos pendientes, se obtiene el cambio entre ellas. Si el cambio es suave, se incrementa un contador, y si este llega a 100, la señal se clasifica como senoidal. Si el cambio es más brusco, se cuenta en otro contador, y si este llega a 100, la señal se clasifica como triangular.

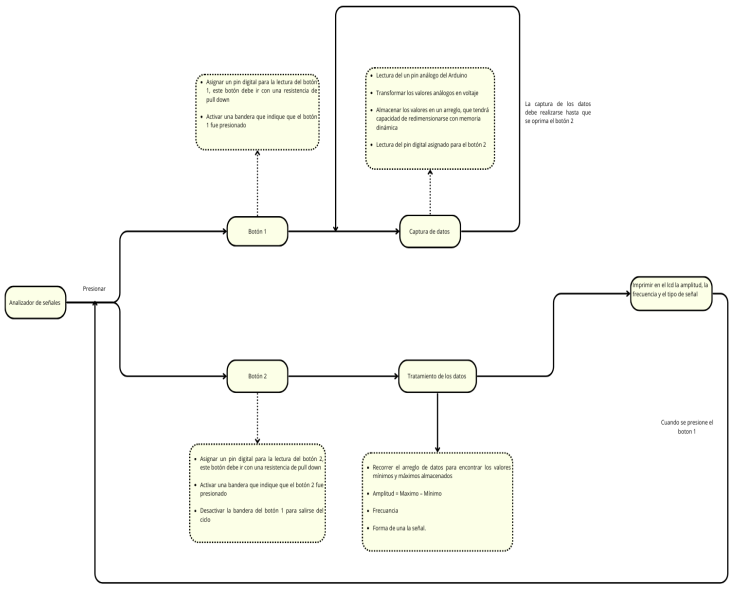

Figura 1. Diagrama de la solución

II. Esquema de tareas para el desarrollo

El siguiente esquema muestra las tareas que cumplimos e implementamos para lograr el éxito del desafío.

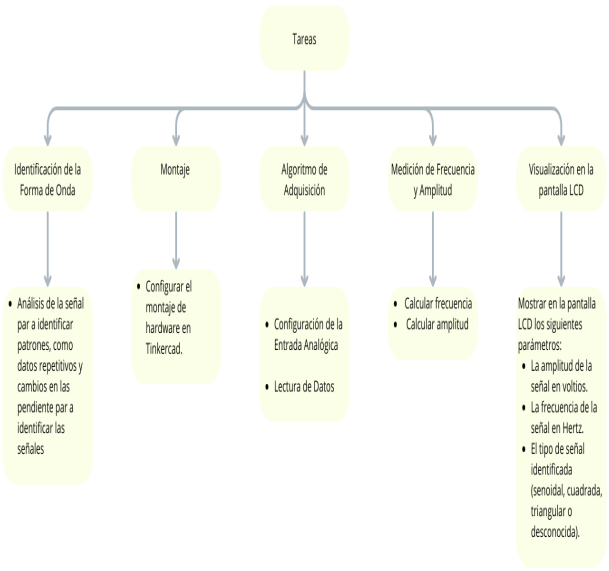

Figura 2. Esquema de tareas

III. Algoritmos implementados

- **Cálculo de la frecuencia:** Al detectar dos cruces consecutivos por cero de la señal analógica, se calcula el período y, a partir de este, la frecuencia en Hz.

```
void detectarFrecuencia() {
  valorActual = analogRead(generador);
  if (valorActual >= valorMedio + histeresis && !
       cruceDetectado) {
    if (tiempoInicio == 0) {
      tiempoInicio = micros();
    } else {
      tiempoFin = micros();
      periodo = tiempoFin - tiempoInicio;
      frecuencia = 1000000.0 / periodo;
      tiempoInicio = tiempoFin;
      detecto_fre = true;
    }
    cruceDetectado = true;
  }
  if (valorActual < valorMedio - histeresis) {
    cruceDetectado = false;
  }
}
```

- **Memoria dinámica:** Se implementa un arreglo dinámico para almacenar los valores de la señal, expandiendo su capacidad cuando sea necesario.

```
void almacenarValor() {
  valorActual = analogRead(generador);
  if (contador < maximo_elementos) {
    if (contador >= capacidad) {
      capacidad += incremento;
      int* temp = new int[capacidad];
      for (int i = 0; i < contador; i++) {
        temp[i] = arreglo[i];
      }
      delete[] arreglo;
      arreglo = temp;
    }
    arreglo[contador] = valorActual;
    contador++;
  }
}
```

- **Clasificación de señales:** Se clasifica la señal comparando las pendientes entre valores consecutivos del arreglo, contando los cambios para identificar señales cuadradas, senoidales o triangulares.

```
for (int i = 1; i < contador - 1; i++) {
  int pendiente_actual = abs(arreglo[i+1]) - abs(
       arreglo[i]);
  int pendiente_anterior = abs(arreglo[i]) - abs(
       arreglo[i-1]);
  if (arreglo[i] == abs(arreglo[i + 1])) {
    contador_C++;
  }
  if (arreglo[i+1] > arreglo[i]) {
    if (abs(pendiente_actual - pendiente_anterior) >
         3) {
      contador_T++;
    }
  } else {
    if (abs(pendiente_anterior - pendiente_actual) >
         3) {
      contador_T++;
    }
  }
}
```

IV. Problemas de desarrollo que afrontó

Durante el desarrollo se presentaron varios tipos de problemas, de los cuales los más destacados fueron:

Limitaciones de un microcontrolador frente a un PC: Estamos acostumbrados a tener muchos recursos de cómputo, tanto en potencia de procesamiento como en espacio de memoria. Este desafío nos puso a investigar las limitaciones que se tienen al trabajar con recursos limitados. El mayor afectado fue el uso de la memoria, ya que en un PC convencional contamos con una memoria RAM de 8 gigabytes, pero en el Arduino solo disponemos de 2 kB de SRAM, lo que nos limita considerablemente. Esto significa que, para almacenar un entero de 2 bytes, solo es posible almacenar aproximadamente 1000 valores de estos enteros, lo que nos demuestra que cada bit es importante y que el recurso de la memoria es muy limitado.

Frecuencia: Pudimos observar que las señales tienen comportamientos distintos a medida que la frecuencia aumenta. Si la frecuencia se incrementa, las velocidades de muestreo deben ser más rápidas, teniendo como limitante la capacidad del Arduino para muestrear estas señales.

Clasificación de las señales: Para poder muestrear una señal se debe cumplir con el teorema de Nyquist, que establece que debemos muestrear a más del doble de la frecuencia máxima. Como trabajábamos con un amplio rango de frecuencias, la tasa de muestreo variaba dependiendo de la frecuencia de la señal. Cuando intentábamos muestrear señales a frecuencias muy altas, las señales de frecuencias bajas llenaban rápidamente el búfer de la memoria. Por otro lado, al intentar muestrear a tasas bajas, las señales con frecuencias altas no podían reconstruirse correctamente, ya que no se

capturaban suficientes muestras en el período de esas señales de alta frecuencia.

Y como último, pero no menos importante, la lógica para programar los botones. Queríamos que el sistema fuera amigable con el usuario, por lo que era necesario evitar rebotes, permitir la acción al sostener el botón y asegurar que todo el proceso fuera fluido y natural.

V. Evolución de la solución y consideraciones

Se comenzó la solución con lo más básico pero muy importante: el flujo que queríamos darle al programa utilizando las interfaces de botones y display. Inicialmente, empezamos leyendo los botones e implementando la lógica para que al presionar un botón se ejecutara una tarea y no se saliera de esta hasta que se presionara el otro botón. Se imprimía en el LCD cuando se estaba ejecutando la tarea del botón 1 y luego cuando se pasaba al botón 2.

Luego, abordamos la tarea de encontrar la amplitud y la frecuencia. Esta tarea fue más rigurosa, especialmente en la parte de la frecuencia. La amplitud se podía encontrar de manera relativamente sencilla entendiendo que es la diferencia entre los picos máximos y mínimos. Dado que el Arduino lee estos valores con una resolución de 10 bits (de 0 a 1023), era fácil obtener el máximo y el mínimo, restarlos, dividir el resultado por la resolución y multiplicarlo por 5, que es el valor de voltaje de referencia.

Sin embargo, la frecuencia resultó ser más complicada. Inicialmente intentamos calcular la frecuencia a partir del arreglo de datos almacenados, pero descubrimos que esta metodología limitaba la capacidad a frecuencias muy bajas debido a la tasa de muestreo fija. Por lo tanto, decidimos medir la frecuencia a partir de todos los valores que nos daba la señal leída en el pin A0. Esto nos permitió medir un rango de frecuencias más amplio, ya que la señal era constante y el Arduino muestreaba toda la señal entrante a una frecuencia alta, cumpliendo así el teorema de Nyquist.

Luego pasamos a la parte más desafiante del proyecto: almacenar en una memoria de máximo 250 valores los datos necesarios para determinar el tipo de señal. Inicialmente, escogimos un rango de 1 a 50 Hz. Sin embargo, al establecer una tasa de muestreo que cumpliera con el rango de 50 Hz, las frecuencias bajas de 1 a 5 Hz se recortaban, ya que la muestreo era tan rápido para estas frecuencias bajas que solo capturaba un fragmento pequeño de la señal. Por otro lado, al usar tasas de muestreo muy pequeñas, la señal con frecuencias más altas no se reconstruía adecuadamente debido a la insuficiencia de muestras.

Se nos ocurrió la idea de implementar una tasa de muestreo dinámica. Esto significa que la tasa de muestreo se ajustaría dependiendo de la frecuencia de la señal ingresada, asegurando que la tasa fuera al menos 100 veces más rápida que la frecuencia de la señal. Con esta estrategia, pudimos reconstruir adecuadamente la señal en un intervalo de 250 muestras, lo que nos permitió obtener el tipo de señal en un rango de frecuencias de 1 a 120 Hz.

Las consideraciones a tener en cuenta son, primero, entender el sistema que será programado y sus limitaciones. Si se contara con un hardware mejor, es decir, un microcontrolador con prestaciones superiores, como una SRAM más grande, un reloj o cristal con frecuencias de trabajo mayores, y una resolución de 16 bits, por ejemplo, se podría lograr una mayor fiabilidad en la toma de datos, un espacio adicional para almacenarlos y un mejor rango de trabajo en frecuencias.

Como segunda recomendación, si en el proyecto con el que estamos trabajando estamos limitados en hardware, debemos ser capaces de conocer a fondo cada componente y función que este nos brinda. Esto es crucial para lograr resultados significativos, como los que obtuvimos en este primer desafío. Al tener en cuenta cada bit de recurso en la memoria y cada recurso limitado, podremos implementar los algoritmos de manera más eficiente, sin depender exclusivamente del hardware.