Desafío 1 – Informática II

Yohan Mauricio Valencia Uribe

Augusto Enrique Salazar Jiménez

Materia: Informática II (2570201)

Grupo: 03

Universidad de Antioquia

Medellín, 2024

Tabla de contenido

Justificación

La empresa Informa2 tiene la necesidad de que, dada una señal analógica, se deban identificar sus características principales y visualizar los resultados de acuerdo con lo siguiente: • El comienzo de la adquisición de datos se hará mediante la activación de un pulsador.

- Una vez iniciada la adquisición, en cualquier momento se puede solicitar la información de la señal capturada. Esta petición se hará mediante la activación de un pulsador. Mientras se procesa la información, la adquisición de datos puede ser suspendida (alternativa: detenerse). Una vez se termine de procesar la información (y se entreguen las salidas,) la adquisición se debe reanudar.
- El valor de las características de la señal capturada se debe visualizar a través de una pantalla LCD.

Análisis del problema

Requerimientos iniciales

- Se necesita un sistema que mida y visualice en tiempo real la **frecuencia, amplitud y tipo de señal** de una señal analógica.
- Debe haber dos botones: uno para iniciar/detener la adquisición de datos y otro para alternar capturar y mostrar o no los resultados.
- La amplitud debe reflejar correctamente el voltaje pico-pico de la señal analógica en estudio y en función del tiempo y de los cambios en la señal.

Identificación del problema inicial

Se requiere el diseño de una solución que permita el análisis por medio de un Arduino de las diferentes señales generadas tales como Sinusoidal, Cuadrada y Triangular, teniendo en cuenta sus variables principales como la amplitud (Voltaje Pico-Pico) y su frecuencia de operación.

Teniendo en cuenta que los parámetros ingresan de forma digital y no en voltios, se requiere de análisis matemáticos y de funciones periódicas con el fin de formular la solución al problema planteado.

Diseño de la solución

Diagrama de bloques:

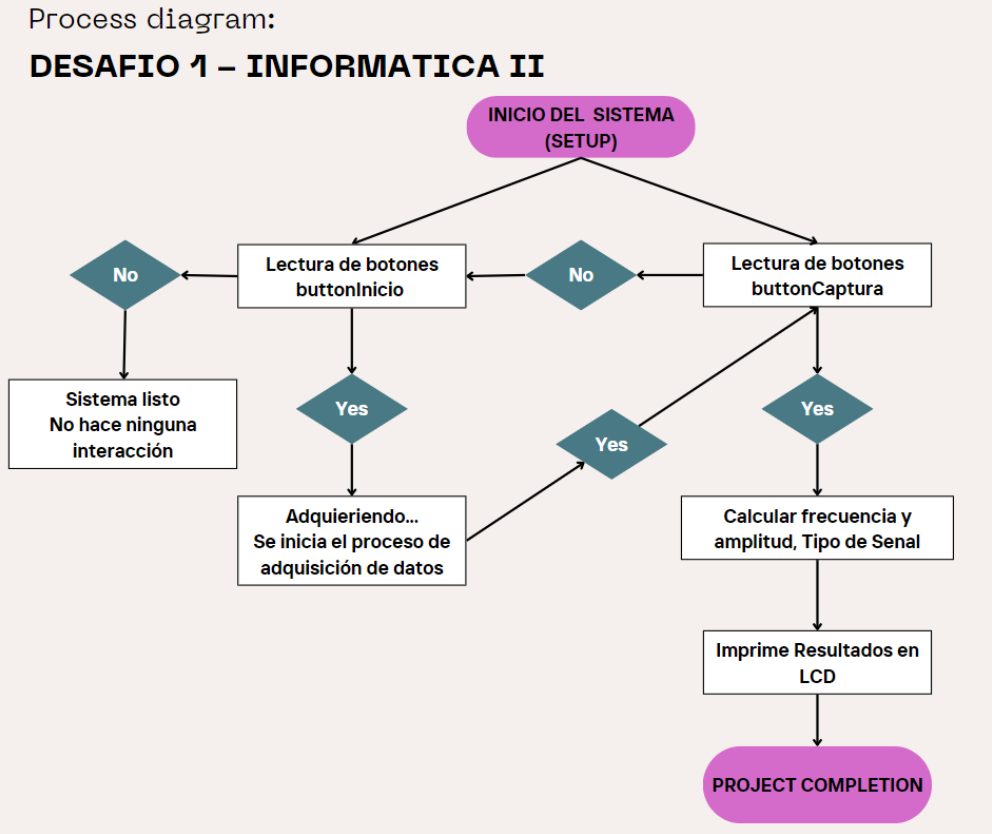

Ilustración 1 Diagrama de Bloques

- Inicio del sistema:El programa inicia con la configuración de pines, la pantalla LCD y la comunicación serie en la función setup().
- Lectura de botones:El sistema está continuamente revisando el estado de los botones (buttonInicio y buttonCaptura). Se usa lógica de debounce para evitar lecturas múltiples incorrectas.

- Verificar estado de adquisición de datos: Si el botón buttonInicio está activado, el sistema inicia la adquisición de datos. Si no está activado, el sistema muestra el mensaje "Detenido" en el LCD.
- Iniciar adquisición de datos: Se lee la señal analógica del pin A0 y se comienza a calcular el voltaje máximo y mínimo para determinar la amplitud de la señal.
- Calcular frecuencia, amplitud y tipo de señal: La frecuencia se calcula detectando cruces por cero en la señal. La amplitud se calcula como la diferencia entre el voltaje máximo y mínimo.
- Verificar estado de visualización: Si el botón buttonCaptura está presionado, el sistema alterna entre mostrar o no los resultados en la pantalla LCD.
- Mostrar resultados en el LCD:Si el estado de visualización está activo, se muestran los valores de amplitud y frecuencia en la pantalla LCD.
- Mantener adquisición sin visualización: Si el botón de visualización no está activo, el sistema sigue adquiriendo datos, pero sin mostrar los resultados en la pantalla.
- Fin del ciclo: El ciclo se repite, con nuevas lecturas y cálculos mientras el sistema esté en adquisición.

Este diagrama de bloques cubre las principales etapas del sistema de adquisición de datos y visualización, representando el flujo de control del programa.

Estructuración del sistema

Cantidad	Nombre	Componente
1	U1	Arduino Uno R3
1	FUNC1	1 Hz, 2 V, 0 V, Seno Generador de función
1	U2	LCD 16 x 2
2	S inicio	Pulsador
	S captura	
1	Rpot2	10 kΩ Potenciómetro
1	R1	220 Ω Resistencia
2	R2	10 kΩ Resistencia
	R3	

Tabla 1 Lista de Materiales

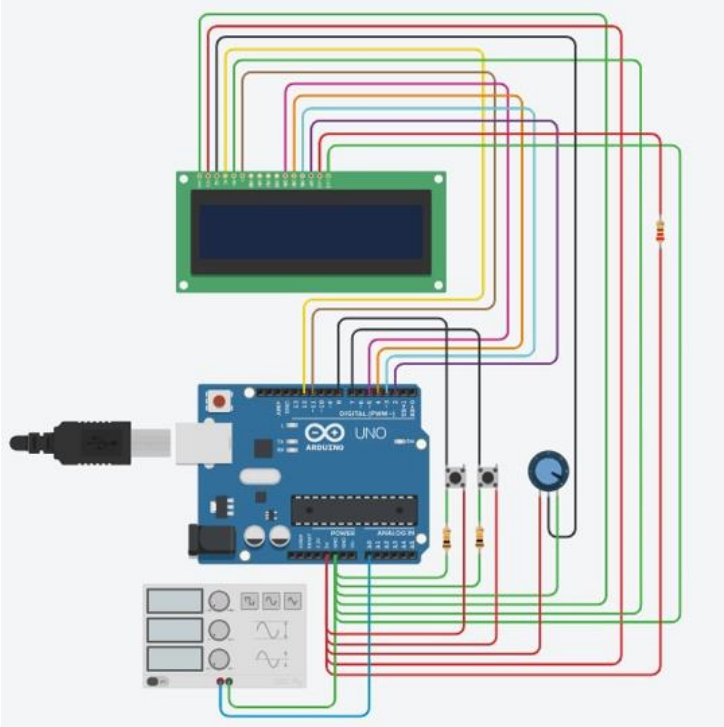

Ilustración 2 Diagrama de Conexiones

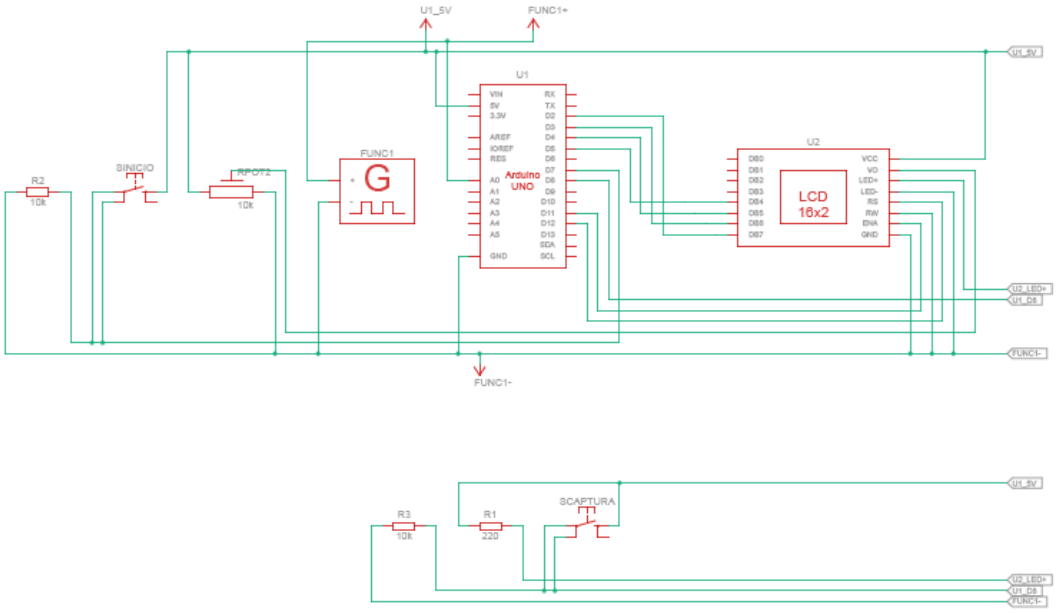

Ilustración 3 Diagrama Esquemático

Evidencia en el código

```
#include <LiquidCrystal.h>  // Incluir la biblioteca para el LCD

LiquidCrystal lcd(12, 11,5, 4, 3, 2);                    // Inicializar la biblioteca con los números de los pines de la interfaz

// Pines de los botones
const int buttonInicio = 7;
const int buttonCaptura = 8;

int buttonInicioState = LOW;
int lastButtonInicioState = LOW;
int buttonCapturaState = LOW;  // Estado inicial del botón de captura
int lastButtonCapturaState = LOW;  // Estado inicial previo del botón de captura
int analogPin = A0;
int val = 0;

// Variables globales
bool adquirirDatos = false;
bool capturandoDatos = false;     // Variable para controlar si estamos capturando datos

const int capacidadMaxima = 100; // Capacidad Máxima para almacear datos
float bufferVoltajes[capacidadMaxima]; // Buffer para almacenar los voltajes capturados
int indiceBuffer = 0;

float frecuencia = 0.0;
float amplitud = 0.0;
float voltaje = 0.0;            // Variable para almacenar el voltaje
float voltajeMaximo = 0.0;      // Variable para almacenar el voltaje máximo
float voltajeMinimo = 5.0;      // Variable para almacenar el voltaje mínimo (inicialmente 5V para garantizar que cualquier lectura sea menor)
float voltajeAnterior = 0.0;
float tasaDeCambio = 0.0;

const int numLecturas = 10;   // Número de lecturas para analizar el patrón
float lecturasVoltaje[numLecturas]; // Buffer para almacenar lecturas
int indice = 0;
String tipoDeSenal = "";

// Punteros para las variables
float* pvoltaje = &voltaje;
float* pvoltajeMaximo = &voltajeMaximo;
float* pvoltajeMinimo = &voltajeMinimo;

unsigned long tiempoAnterior = 0;  // Inicializamos tiempoAnterior
unsigned long lastDebounceTime = 0;  // Tiempo para debounce
unsigned long debounceDelay = 50;    // Retraso para el debounce
unsigned long lastDebounceTimeCaptura = 0;
unsigned long debounceDelayCaptura = 50;
unsigned long lastDisplayUpdate = 0;
unsigned long displayInterval = 1000;  // Intervalo para actualizar la pantalla (1 segundo)


// Configurar pines
void setup() {

  // Configurar pines de botones
  pinMode(buttonInicio, INPUT_PULLUP);
  pinMode(buttonCaptura, INPUT_PULLUP);

  // Iniciar LCD
  lcd.begin(16, 2);
  lcd.print("Sistema listo");
  delay(1000);

  // Iniciar comunicación serie para depuración
  Serial.begin(9600);
}
```

```
void loop() {

	val = analogRead(analogPin); // Función para mostrar la señal
	Serial.println(val);

	handleButtonInicio();
	handleButtonCaptura();

	if (adquirirDatos) {		// Si se está adquiriendo la señal, calcular sus características
		leerVoltaje(pvoltaje, pvoltajeMaximo, pvoltajeMinimo);
		amplitud = calcularAmplitud();
		frecuencia = calcularFrecuencia();
		tipoDeSenal = identificarSenal();
		if (capturandoDatos){
			leerVoltajeYAlmacenar();
		}
	}
}

 // Verificar si el botón de inicio/detención fue presionado
void handleButtonInicio() {
	int reading = digitalRead(buttonInicio);

	if (reading != lastButtonInicioState) {
		lastDebounceTime = millis();  // Actualiza el tiempo si hay cambio
	}

	if ((millis() - lastDebounceTime) > debounceDelay) {
		if (reading != buttonInicioState) {
			buttonInicioState = reading;

			if (buttonInicioState == LOW) {  // Cambiar el estado de adquisición de datos
				adquirirDatos = !adquirirDatos;

				if (adquirirDatos) {
					lcd.clear();
					lcd.setCursor(0, 0);
					lcd.print("Adquiriendo");
					lcd.setCursor(0, 1);
					lcd.print("Datos ...");

					// Reiniciar valores máximo y mínimo para cada adquisición
					voltajeMaximo = 0.0;
					voltajeMinimo = 5.0;  // Reiniciamos el mínimo a 5V

					lastDisplayUpdate = millis();  // Actualizar el temporizador para la pantalla
				}
				else {
					lcd.clear();
					lcd.setCursor(0, 0);
					lcd.print("Detenido");
					lastDisplayUpdate = millis();  // Actualizar el temporizador para la pantalla
				}
			}
		}
	}
	lastButtonInicioState = reading;
}

 // Verificar si el botón de captura fue presionado para alternar la visualización
void handleButtonCaptura() {
	int readingCaptura  = digitalRead(buttonCaptura);

	if (readingCaptura  != lastButtonCapturaState) {
		lastDebounceTimeCaptura = millis();  // Actualiza el tiempo si hay cambio en el botón de captura
		}
```

```
                if ((millis() - lastDebounceTimeCaptura) > debounceDelayCaptura) {
                        if (readingCaptura != buttonCapturaState) {
                                buttonCapturaState = readingCaptura;

                                if (buttonCapturaState == LOW) {  // Cambiar el estado de visualización en el LCD

                                        if (adquirirDatos) {
                                                capturandoDatos = !capturandoDatos;

                                                if (capturandoDatos) {          // Inicio de la captura
                                                        lcd.clear();
                                                        lcd.setCursor(0, 0);
                                                        lcd.print("Capturando");
                                                        lcd.setCursor(0, 1);
                                                        lcd.print("Datos...");

                                                        indiceBuffer = 0;       // Reiniciar el buffer y el índice de almacenamiento
                                                }

                                                else {          // Detención de la captura
                                                        lcd.clear();
                                                        lcd.setCursor(0, 0);
                                                        lcd.print("Captura detenida");
                                                        lcd.setCursor(0, 1);
                                                        lcd.print("Procesando...");
                                                        delay(1000);

                                                        procesarDatosCapturados();      // Procesar los datos capturados

                                                        int repeticiones = 0;  // Contador de repeticiones
                                                        int maxRepeticiones = 3;  // Número máximo de repeticiones

                                                        while (repeticiones < maxRepeticiones) {
                                                                if (millis() - lastDisplayUpdate >= displayInterval) {
                                                                        lastDisplayUpdate = millis();
                                                                        mostrarDatos();  // Muestra los datos en el LCD
                                                                        repeticiones++;  // Incrementar el contador de repeticiones
                                                                }
                                                        }
                                                        lcd.clear();
                                                        lcd.print("Sistema listo");
                                                }
                                        }
                                        else {          // Si no se ha iniciado la adquisición de datos, mostrar advertencia
                                                lcd.clear();
                                                lcd.setCursor(0, 0);
                                                lcd.print("Inicie primero");
                                                lcd.setCursor(0, 1);
                                                lcd.print("la captura");
                                                delay(2000);  // Mostrar advertencia por 2 segundos
                                                lcd.clear();
                                                lcd.print("Sistema listo");
                                        }
                                }
                        }
                }
        lastButtonCapturaState = readingCaptura;
}

// Función para leer voltaje y almacenar
void leerVoltajeYAlmacenar() {
        // Leer el voltaje desde el pin analógico
        float voltajeActual = analogRead(analogPin) * (5.0 / 1023.0);
        if (capturandoDatos && indiceBuffer < capacidadMaxima) {                // Almacenar el voltaje en el buffer si estamos capturando datos
                bufferVoltajes[indiceBuffer] = voltajeActual;
                indiceBuffer++;
        }
}
```

```
// Función para procesar datos capturados
void procesarDatosCapturados() {
	// Procesar los datos almacenados en el buffer
	if (indiceBuffer > 0) {
		float sumaVoltajes = 0.0;
		float maxVoltaje = 0.0;
		float minVoltaje = 5.0;
		for (int i = 0; i < indiceBuffer; i++) {
			sumaVoltajes += bufferVoltajes[i];
			if (bufferVoltajes[i] > maxVoltaje) {
				maxVoltaje = bufferVoltajes[i];
			}
			if (bufferVoltajes[i] < minVoltaje) {
				minVoltaje = bufferVoltajes[i];
			}
		}
	}
}

// Función para leer voltaje
void leerVoltaje(float* pvoltaje, float* pvoltajeMaximo, float* pvoltajeMinimo) {
	int valorAnalogico = analogRead(analogPin);
	*pvoltaje = ((valorAnalogico / 1023.0) * 5.0);
	// Actualizar buffer de lecturas
	lecturasVoltaje[indice] = *pvoltaje;
	indice = (indice + 1) % numLecturas;  // Incrementar índice y envolver
	if (*pvoltaje > *pvoltajeMaximo) {
		*pvoltajeMaximo = *pvoltaje;
	}
	if (*pvoltaje < *pvoltajeMinimo) {
		*pvoltajeMinimo = *pvoltaje;
	}
}

// Función para identificar tipo de señal
String identificarSenal() {
	float derivadas[numLecturas-1];
	// Calcular derivadas
	for (int i = 0; i < numLecturas - 1; i++) {
		derivadas[i] = lecturasVoltaje[(i + 1) % numLecturas] - lecturasVoltaje[i];
	}
	// Identificar señal cuadrada
	int cambiosBruscos = 0;
	for (int i = 0; i < numLecturas - 1; i++) {
		if (abs(derivadas[i]) > 1.0) {
			cambiosBruscos++;
		}
	}
	if (cambiosBruscos > (numLecturas / 2)) {
		return "Senal Cuadrada";
	}
	// Identificar señal senoidal
	bool cambioSuave = true;
	for (int i = 0; i < numLecturas - 1; i++) {
		if (abs(derivadas[i]) > 0.3) { // Ajusta el umbral según sea necesario
			cambioSuave = false;
			break;
		}
	}
	if (cambioSuave) {
		return "Senal Senoidal";
	}
	// Identificar señal triangular
	bool cambioConstante = true;
	for (int i = 1; i < numLecturas - 1; i++) {
		if (abs(derivadas[i] - derivadas[i - 1]) > 0.2) { // Ajusta el umbral según sea necesario
			cambioConstante = false;
```

```
                                        break;
                                }
                        }
                        if (cambioConstante) {
                                return "Señal Triangular";
                        }
                        return "No identificado";
}

// Función para calcular la amplitud
float calcularAmplitud() {
                        return (voltajeMaximo - voltajeMinimo);
}

// Función para calcular la frecuencia
float calcularFrecuencia() {
                        static int valorAnterior = 0;
                        static unsigned long tiempoAnterior = 0;  // Tiempo del último cruce
                        int valorActual = analogRead(analogPin);
                        unsigned long tiempoActual = micros();  // Obtener el tiempo actual en microsegundos

                        // Aplicar un filtro para evitar fluctuaciones por ruido
                        int umbral = 511;  // Umbral de cruce en el punto medio (511 o 512)

                        // Verificar si hay cruce por el umbral
                        if ((valorAnterior > umbral && valorActual <= umbral) || (valorAnterior < umbral && valorActual >= umbral)) {
                                float tiempoCiclo = (tiempoActual - tiempoAnterior) / 1000000.0;// Calcular el tiempo de un ciclo completo (en segundos)
                                tiempoAnterior = tiempoActual;                  // Actualizar el tiempo del último cruce
                                if (tiempoCiclo > 0) {                  // Si el ciclo es válido (mayor a cero), calcular la frecuencia
                                        return 1.0 / tiempoCiclo;  // Frecuencia en Hz
                                }
                        }
                        valorAnterior = valorActual;                        // Actualizar el valor anterior
                        return frecuencia;       // Retornar la última frecuencia calculada si no hay nuevo cruce
}

// Función para mostrar datos en LCD
void mostrarDatos() {
                        lcd.clear();
                        lcd.setCursor(0, 0);
                        lcd.print("Amplitud: ");
                        lcd.setCursor(0, 1);
                        lcd.print(amplitud);
                        lcd.print(" V");
                        delay(1000);
                        lcd.clear();
                        lcd.setCursor(0, 0);
                        lcd.print("Frecuencia: ");
                        lcd.setCursor(0, 1);
                        lcd.print(frecuencia);
                        lcd.print(" Hz");
                        delay(1000);
                        lcd.clear();
                        lcd.setCursor(0, 0);
                        lcd.print("Tipo: ");
                        lcd.setCursor(0, 1);
                        lcd.print(tipoDeSenal);
                        delay(1000);
}
```

Resumen del código:

1. **Bibliotecas y pines:** Utiliza la biblioteca LiquidCrystal para controlar el LCD y define pines para los botones de inicio y captura, además de un pin analógico para medir voltajes.
2. **Variables globales:** Se crean variables para almacenar el estado de los botones, voltajes, amplitud, frecuencia, y otras relacionadas con el proceso de adquisición y captura.
3. **Configuración:** En la función setup(), se configuran los pines, se inicializa la pantalla LCD y se comienza la comunicación serie.
4. **Adquisición de datos:** El bucle principal (loop()) maneja la lectura de voltaje y el estado de los botones. Dependiendo de si la adquisición de datos está activa, se calculan la amplitud, frecuencia y tipo de señal.
5. **Botones:** Los botones son gestionados por las funciones handleButtonInicio() y handleButtonCaptura() que controlan cuándo se empieza o se detiene la adquisición y captura de datos.
6. **Procesamiento de datos:** Las funciones leerVoltajeYAlmacenar(), procesarDatosCapturados() y leerVoltaje() gestionan la captura de voltajes y el cálculo de sus características.
7. **Identificación de señal:** La función identificarSenal() analiza las derivadas de las lecturas de voltaje para clasificar la señal.
8. **Amplitud y frecuencia:** Se calculan en las funciones calcularAmplitud() y calcularFrecuencia(), respectivamente.
9. **Visualización:** Los resultados se muestran en la pantalla LCD mediante la función mostrarDatos().

Comprobación y ajuste del sistema

Mejoras Potenciales:

- Aunque el sistema funciona correctamente en general, algunos ajustes podrían mejorar su precisión y usabilidad:
    - Mejorar la precisión del cálculo de frecuencia para señales más complejas.
    - Implementar opciones para almacenar los datos capturados externamente (por ejemplo, en una tarjeta SD).
    - Ajustar los umbrales de las derivadas para una identificación más precisa de las señales bajo condiciones de ruido.

Resultados obtenidos

Resultados de las pruebas del sistema de adquisición y análisis de señales con Arduino:

1. Medición de Voltaje Exitosa:

- El sistema es capaz de medir correctamente el voltaje analógico en el rango de 0 a 5V, cumpliendo con las expectativas en cuanto a precisión de lectura. Las fórmulas utilizadas para la conversión de valores analógicos a voltajes reales funcionan de manera adecuada.

2. Debounce de Botones Funciona Correctamente:

- El debounce implementado en los botones de inicio y captura asegura que los cambios de estado ocurran solo cuando el botón es presionado intencionadamente, evitando múltiples

activaciones debido a ruido o rebotes mecánicos. Esto contribuye a la estabilidad del sistema.

3. Captura y Almacenamiento de Datos:

- El sistema almacena de manera eficiente los datos de voltaje en un buffer con una capacidad máxima de 100 lecturas, lo cual es suficiente para realizar análisis básicos. El índice del buffer no excede su capacidad y los datos capturados son procesados correctamente.

4. Cálculo de Amplitud y Frecuencia Aceptable:

- Las pruebas con señales de voltaje conocidas muestran que el sistema puede calcular la amplitud y la frecuencia de las señales con un nivel de precisión adecuado, aunque podría beneficiarse de mejoras en el algoritmo de cálculo de frecuencia para señales con mayor ruido.

5. Identificación de Tipo de Señal:

- El código es capaz de identificar correctamente diferentes tipos de señales (cuadrada, senoidal y triangular), utilizando el análisis de derivadas de las lecturas. Este proceso es útil para clasificar la naturaleza de la señal de entrada de manera automatizada.

6. Pantalla LCD Informativa:

- La pantalla LCD ofrece una forma clara y sencilla de visualizar los resultados del análisis (amplitud, frecuencia, tipo de señal), lo que facilita la interpretación de los datos en tiempo real sin necesidad de depender del monitor serie.

7. Robustez del Sistema:

- El sistema se muestra robusto en cuanto a la captura de datos y su procesamiento, siendo capaz de operar de manera continua sin fallas críticas ni sobrecarga en el buffer, siempre que se respeten los límites establecidos en cuanto al número de lecturas.

Pruebas y validación

Para garantizar que el código funciona correctamente, es importante realizar respectivas pruebas y validaciones en varias áreas:

1. Prueba de lectura de voltaje

- **Objetivo**: Validar que el Arduino está midiendo correctamente el voltaje a través del pin analógico.
- **Prueba**: Conectar una señal de voltaje conocida (por ejemplo, de 0V a 5V) al pin A0 del Arduino.
    - Observar los valores que se imprimen en el monitor serie (Serial.println(val);).
    - Asegúrar que los valores se ajusten a la fórmula usada: voltaje = (valorAnalogico / 1023.0) * 5.0.
- **Validación**: Los valores deben corresponder a los voltajes reales medidos. Se verifica si las lecturas en serie coinciden con el voltaje que estás aplicando.

2. Prueba de botones (Inicio y Captura)

- **Objetivo**: Verificar que los botones de inicio y captura cambien correctamente el estado de adquisición y captura de datos.

- **Prueba**:
    - Pulsa el botón de **inicio** y observar si el LCD muestra "Adquiriendo Datos...".
    - Pulsa el botón de **captura** cuando la adquisición esté activa y se verifica que el LCD muestre "Capturando Datos...".
    - Se repite el proceso para alternar entre los estados de inicio/detención y captura.
- **Validación**: Se confirma de que los cambios de estado ocurren solo después de presionar los botones, sin saltos ni falsos cambios de estado.

3. Prueba de procesamiento de datos capturados

- **Objetivo**: Validar que los datos se almacenan y procesan correctamente.
- **Prueba**: Capturar los datos y verificar que:
    - Los valores se almacenen en el buffer bufferVoltajes.
    - Se calculen el voltaje máximo, mínimo y el promedio de los datos capturados.
- **Validación**: Asegurar que los valores máximo y mínimo calculados correspondan a los voltajes reales capturados y que el almacenamiento no exceda el tamaño del buffer (100 valores).

4. Prueba de cálculo de amplitud

- **Objetivo**: Asegurarse de que la amplitud se calcule correctamente.
- **Prueba**: Aplicar una señal con una amplitud conocida, como una onda senoidal de 5V pico a pico.
    - Observa el valor de amplitud = calcularAmplitud().
- **Validación**: El valor de amplitud debe corresponder a la diferencia entre el voltaje máximo y mínimo de la señal aplicada.

5. Prueba de cálculo de frecuencia

- **Objetivo**: Verificar que la frecuencia de la señal medida es correcta.
- **Prueba**: Se aplica una señal de frecuencia conocida, como una onda senoidal de 50 Hz o 60 Hz, al pin A0.
    - Observa el valor de frecuencia = calcularFrecuencia().
- **Validación**: El valor de la frecuencia en Hz que se muestra en el LCD debe coincidir con la frecuencia real de la señal aplicada.

6. Prueba de identificación de señal

- **Objetivo**: Validar que el código identifica correctamente los diferentes tipos de señal (cuadrada, senoidal, triangular).
- **Prueba**: Se aplican tres tipos de señales diferentes al pin A0:
    - **Señal Cuadrada**: Con cambios bruscos entre niveles altos y bajos.
    - **Señal Senoidal**: Con variaciones suaves y periódicas.
    - **Señal Triangular**: Con un aumento y descenso lineal en cada ciclo.
- **Validación**: Se verifica que la clasificación de la señal en el LCD corresponda al tipo de señal real aplicada.

7. Prueba de manejo del buffer

- **Objetivo**: Verificar que el sistema maneja correctamente el almacenamiento en el buffer de voltajes sin exceder su capacidad.
- **Prueba**: Aplicar una señal continua por un período prolongado para llenar el buffer.

- **Validación**: Asegurar que el índice del buffer no supere el límite (100) y que los datos se almacenen adecuadamente dentro del rango establecido.

8. Prueba de debounce de los botones

- **Objetivo**: Asegurarse de que el debounce funcione para evitar múltiples activaciones por pulsación.
- **Prueba**: Pulsa rápidamente los botones de **inicio** y **captura** y verifica que solo una acción es reconocida por cada pulsación.
- **Validación**: Si el debounce funciona correctamente, no debe haber múltiples cambios de estado por una sola pulsación.

9. Prueba del LCD

- **Objetivo**: Verificar que los datos de amplitud, frecuencia y tipo de señal se muestren correctamente en el LCD.
- **Prueba**: Ejecutar el código con señales de prueba y observa las lecturas en el LCD.
- **Validación**: Asegurar que los datos mostrados en el LCD (voltaje, frecuencia, amplitud, tipo de señal) sean coherentes con las señales que estás midiendo.

10. Validación final:

- Se comparan los resultados mostrados en el monitor serie y en el LCD con los valores esperados.

Conclusión

- La evidencia del análisis y diseño de la solución está representada por el proceso iterativo que siguió al análisis de los problemas detectados, las decisiones sobre el diseño del flujo de datos, y las modificaciones del código para mejorar el comportamiento del sistema de adquisición de señales y visualización de datos.
- El sistema cumple su objetivo de medir voltajes, calcular amplitud, frecuencia e identificar el tipo de señal de entrada. Es un diseño funcional y estable para aplicaciones de adquisición de datos con Arduino. Sin embargo, tiene margen para mejorar en cuanto a precisión y almacenamiento de datos a largo plazo, especialmente en aplicaciones más exigentes.