Universidad de Antioquia.

Facultad de Ingeniería

**DESAFIO 1**

Autores:

Esteban Rodríguez Monsalve

Melany Sierra Torres

Asignatura:

Informática 2

Docente:

Aníbal Guerra

Fecha:

18 de septiembre del 2024

Informe del desarrollo del proyecto

# Análisis del problema y consideraciones para la alternativa de solución propuesta

El proyecto consiste en medir y analizar la señal de entrada en términos de amplitud, frecuencia y tipo de onda (senoidal, cuadrada o triangular). Utilizamos un Arduino para adquirir los datos analógicos de la señal a través de un pin analógico, procesarlos, y mostrar los resultados en una pantalla LCD. Los pulsadores permiten iniciar y pausar la adquisición de datos, lo que permite al usuario controlar el proceso.

Se decidió usar memoria dinámica para almacenar los valores de la señal. Además, se utilizaron las funciones de la librería **Adafruit\_LiquidCrystal** para interactuar con la pantalla LCD.

# Consideraciones para la solución:

* + Manejo eficiente de la memoria para los datos de la señal.
  + Implementación de un control preciso de botones.
  + Procesamiento correcto de las características de la señal.
  + Detección robusta del tipo de onda.

# Esquema de tareas en el desarrollo de los algoritmos

1. **Inicialización de hardware:**
   * Configuración de los pines de entrada (sensor y pulsadores) y la pantalla LCD.
   * Asignación de memoria dinámica para almacenar los valores analógicos.

# Adquisición de datos:

* + Leer valores analógicos del pin de entrada y almacenarlos en un arreglo.
  + Controlar la adquisición con los pulsadores de inicio y pausa.

# Procesamiento de la señal:

* + Calcular la amplitud y frecuencia de la señal.
  + Determinar el tipo de onda basado en la variación de los valores analógicos.

# Mostrar resultados:

* + Visualizar la amplitud, frecuencia y tipo de onda en la pantalla LCD.

# Algoritmos implementados

1. **Manejo de botones (manejarBotones)**: Verifica el estado de los botones para iniciar o pausar la adquisición de datos.
   * **Problema detectado:** No funciona como se espera. El botón de inicio no activa la adquisición correctamente.
2. **Adquisición de muestras (adquiriendoMuestras)**: Lee los valores analógicos y los almacena en un arreglo hasta que se completa un ciclo de muestras.
3. **Procesamiento de la señal (procesarSenal)**: Calcula la amplitud (máximo valor leído) y la frecuencia (basada en la detección de cruces por 0).
   * **Problema detectado:** Aunque las ecuaciones están implementadas correctamente, los resultados impresos no son los esperados.
4. **Determinación del tipo de onda (determinarTipoOnda)**: Analiza los cambios entre los valores para identificar si la onda es senoidal, cuadrada o triangular.
   * **Problema detectado:** El algoritmo confunde las ondas senoidales con las triangulares.
5. **Mostrar resultados (mostrarResultados)**: Muestra en la pantalla LCD la amplitud, frecuencia y tipo de onda.

# Problemas de desarrollo que afrontó

1. **Control de botones:** Se esperaba que un botón iniciara la adquisición y el otro la detuviera. Sin embargo, ambos botones no funcionaron como se anticipó, lo que causó confusión en la secuencia de adquisición de datos.
2. **Procesamiento de la señal:** Aunque los cálculos de amplitud y frecuencia siguen las ecuaciones correctas, los valores resultantes no se muestran de manera coherente con la señal real. Esto podría estar relacionado con la velocidad de adquisición o el filtrado de datos.
3. **Determinación del tipo de onda:** El algoritmo de clasificación confunde ondas senoidales y triangulares. Además, no hay una salida para cuando el tipo de onda no se puede determinar, lo que debería mostrar "Desconocido".

# Evolución de la solución y consideraciones para la implementación

1. **Mejoras en el control de los botones:**
   * Se deberá revisar el debounce (rebote) de los pulsadores.

# Corrección en el procesamiento de la señal:

* + Verificar que las lecturas analógicas se hagan a intervalos regulares para evitar ruido en los datos.
  + Ajustar el cálculo de frecuencia para garantizar que detecta los cruces por cero correctamente.

# Identificación precisa del tipo de onda:

* + Implementar una lógica más robusta que diferencie entre ondas senoidales y triangulares, posiblemente introduciendo un filtro o análisis más profundo de los datos de las muestras.

Explicación del código

Definición de variables e importación de librerías necesarias.

* Se importó la biblioteca <Adafruit\_LiquidCrystal.h> que me permite trabajar Con la pantalla LCD.
* **Adafruit\_LiquidCrystal lcd(0)** Inicializa un objeto para controlar una pantalla LCD.
* ● **botonInicio** y **botonPausa** son botones conectados a pines digitales configurados como entradas.
* **pinEntrada** es el pin analógico donde se recibe la señal que será leída.
* **amplitud** y **frecuencia** son variables flotantes que almacenan los valores calculados de la amplitud y la frecuencia.
* **tipoOnda** es un arreglo de caracteres que almacenará el tipo de onda detectado (por ejemplo, “Senoidal”, “Cuadrada”, “Triangular” o “Desconocido”).
* **valoresAnalogicos** es un arreglo dinámico que almacena las lecturas analógicas de la señal.
* **numMuestras** es la cantidad de muestras a tomar para el análisis de la señal.
* **adquiriendo** **y** **enPausa** son variables de control para el estado de Adquisición de datos.
* **esSenoidal**, esCuadrada y esTriangular estas son banderas que indican el tipo de onda detectada todas iniciadas en false.
* **indice** se utiliza para iterar sobre las lecturas de las muestras en el arreglo “valoresAnalogicos”
* **val** variable que almacena la lectura analógica actual.
* **tiempoAnterior** Tiempo de referencia utilizado para medir la duración entre los cruces por cero (cruces de la señal por el valor 512)
* **ultimoValor** Almacena el valor de la última muestra para detectar cruces por cero.

1. **Descripción del setup()**

La función setup() se ejecuta una sola vez al encender el dispositivo o reiniciar el programa. Aquí se configuran los pines y otros componentes iniciales.

* pinMode(botonInicio, INPUT);: Configura el pin asociado al botón de inicio (botonInicio) como entrada.
* pinMode(botonPausa, INPUT);: Configura el pin del botón de pausa (botonPausa) como entrada.
* pinMode(pinEntrada, INPUT);: Configura un pin para recibir señales analógicas, que recibe la señal desde señales analógicas de tinkercad.
* Serial.begin(9600);: Inicializa la comunicación serie a 9600 baudios para enviar datos a través del puerto serie.
* lcd.begin(16, 2);: Inicializa un LCD con 16 columnas y 2 filas.
* lcd.print("Esperando...");: Imprime el mensaje "Esperando..." en la pantalla LCD.
* valoresAnalogicos = new int[numMuestras];: Reserva memoria dinámica para un array de enteros de tamaño numMuestras, utilizado para almacenar datos analógicos adquiridos.

2. Descripción del loop()

La función loop() se ejecuta repetidamente mientras el Arduino esté encendido y

gestiona las acciones continuas.

* manejarBotones();: Llama a una función que gestiona las interacciones con los botones de inicio y pausa.
* if (adquiriendo && !enPausa): Si la variable adquiriendo es verdadera y enPausa es falsa, se realiza:
* adquiriendoMuestras();: Llama a una función que adquiere las muestras analógicas.

3. Descripción de la función manejarBotones()

Esta función monitorea el estado de los botones de inicio y pausa para controlar el

proceso de adquisición de datos.

Primer if (Maneja el botón de inicio)

* digitalRead(botonInicio) == LOW: Verifica si el botón de inicio está presionado.
* !adquiriendo: Verifica si no se está adquiriendo datos actualmente.
* Si ambas condiciones son verdaderas:
* adquiriendo = true;: Inicia la adquisición de datos.
* lcd.clear();: Limpia la pantalla LCD.
* lcd.print("Adquiriendo...");: Muestra el mensaje "Adquiriendo..." en el LCD.
* delay(1000);: Pausa de 1 segundo para evitar la lectura múltiple debido al rebote del botón.

Segundo if (Maneja el botón de pausa)

* digitalRead(botonPausa) == LOW: Verifica si el botón de pausa está presionado.
* adquiriendo: Verifica si actualmente se están adquiriendo datos.
* Si ambas condiciones son verdaderas:
* enPausa = !enPausa;: Cambia el estado de la variable enPausa.
  + - lcd.clear();: Limpia la pantalla LCD.
    - lcd.print(enPausa ? "En Pausa..." : "Reanudando...");: Muestra el mensaje adecuado según el estado de enPausa.
    - delay(1000);: Pausa de 1 segundo para evitar el rebote del botón.

4. Descripción de adquiriendoMuestras()

Esta función se encarga de adquirir muestras analógicas y procesarlas.

* Leer el valor analógico:
* analogRead(pinEntrada);: Lee un valor analógico desde el pin de entrada.
* valoresAnalogicos[indice] = val;: Almacena el valor en el array valoresAnalogicos.
* indice++;: Incrementa el índice para la próxima muestra.
* Serial.println(val);: Envía el valor leído al monitor serie.
* Verificar el número de muestras:
* ○ if (indice >= numMuestras): Comprueba si se ha alcanzado el número máximo de muestras.
* Si se han adquirido todas las muestras:
* procesarSenal();: Llama a una función para procesar las muestras.
* mostrarResultados();: Llama a una función para mostrar los resultados.
* indice = 0;: Reinicia el índice para una nueva adquisición de muestras.
* Pausa entre adquisiciones:
* delay(100);: Pausa de 100 ms entre lecturas para controlar la frecuencia de adquisición.

5. Descripción de la función procesarSenal()

Esta función procesa las muestras adquiridas para calcular la amplitud y frecuencia de

la señal.

1. Inicialización de variables:

* amplitudMax y amplitudMin: Encuentra los valores máximo y mínimo de la señal.
* tiempoAnterior: Guarda el tiempo del último cruce por cero.
* cruces: Cuenta cuántas veces la señal cruza el umbral.
* frecuenciaProm: Acumula la frecuencia calculada.
* ciclosContados: Cuenta ciclos completos detectados.

2. Bucle para procesar las muestras:

* float voltaje = valoresAnalogicos[i] \* (5.0 / 1023.0);: Convierte el valor analógico en voltaje.
* if (voltaje > amplitudMax) y if (voltaje < amplitudMin): Actualiza amplitudMax y amplitudMin.

3. Detección de cruces por cero:

* if (ultimoValor < 512 && valoresAnalogicos[i] >= 512): Detecta cruces por cero.
* unsigned long tiempoActual = millis();: Guarda el tiempo actual.
* unsigned long intervalo = tiempoActual - tiempoAnterior;: Calcula el intervalo entre cruces.
* if (intervalo > 10 && intervalo < 1000): Filtra cruces erróneos.
* frecuenciaProm += 1000.0 / intervalo;: Acumula la frecuencia calculada.
* ciclosContados++;: Incrementa el contador de ciclos.
* tiempoAnterior = tiempoActual;: Actualiza el tiempo del último cruce.

4. Calcular amplitud y frecuencia:

* Calcula la amplitud y la frecuencia promedio basada en los ciclos contados.

5. Determinar el tipo de onda:

* La función clasifica el tipo de onda basado en la amplitud y frecuencia calculadas (no se muestra en el código).

6. Descripción de la función determinarTipoOnda()

Esta función clasifica el tipo de onda en función de los cambios entre muestras

consecutivas.

1. Inicialización de variables:

* maxDelta: almacena el mayor cambio entre dos muestras consecutivas de la señal.
* cuentaConstante: cuenta cuántas veces el valor de la señal no cambia (la diferencia entre dos muestras es cero).
* transiciones: almacena el número de veces que la señal cruza el valor de 512 (nivel medio).
* tieneCambiosLineales: Verifica cambios constantes entre muestras.
* esSenoidal, esCuadrada, esTriangular: Indicadores de tipo de onda. (se inicializan en false para clasificar la forma de la onda posteriormente).

2. Bucle para analizar las muestras

* + El bucle recorre todas las muestras de la señal almacenadas en valoresAnalogicos[], comenzando desde la segunda muestra (i = 1), porque se compara cada muestra con la anterior.
  + **delta**: almacena el valor absoluto de la diferencia entre dos muestras consecutivas (valoresAnalogicos[i] - valoresAnalogicos[i - 1]). Esto mide el cambio (variación) entre dos puntos de la señal.
* 3. Cálculo de maxDelta
* Actualiza **maxDelta** si el cambio entre las muestras actuales es mayor que el máximo cambio encontrado hasta el momento. maxDelta será el mayor cambio entre dos muestras consecutivas, útil para diferenciar entre formas de onda.

4. Detección de muestras constantes

* Si no hay diferencia entre dos muestras consecutivas (es decir, si delta == 0), aumenta el contador cuentaConstante. Esto se usa para detectar si la señal tiene largos periodos de valores constantes, típico de ondas cuadradas.

5. Detección de cambios lineales

* Aquí se verifica si los cambios entre las muestras son constantes o no. Si el cambio actual (delta) no es igual al cambio anterior (entre la muestra actual y la muestra anterior), tieneCambiosLineales se establece en false. Esto se utiliza para identificar una forma de onda triangular, que tiene cambios lineales.

#### 6. Detección de transiciones por el nivel medio (512)

* Este bloque detecta cuándo la señal cruza el valor de 512, que representa el nivel medio de la señal. Cada cruce cuenta como una transición, y **transiciones** se incrementa cada vez que la señal pasa por este umbral (tanto de abajo hacia arriba como de arriba hacia abajo).

7. Determinación del tipo de onda

* **Onda Cuadrada**: Si más de la mitad de las muestras son constantes (cuentaConstante > numMuestras / 2), entonces la onda es clasificada como cuadrada, ya que las ondas cuadradas tienen largos periodos donde la señal permanece constante.
* **Onda Triangular**: Si la señal tiene cambios lineales (tieneCambiosLineales == true) y al menos 2 transiciones por el nivel medio (transiciones >= 2), entonces la onda es clasificada como triangular, ya que las ondas triangulares cambian de forma constante y cruzan el nivel medio regularmente.
* **Onda Senoidal**: Si hay al menos 2 transiciones (transiciones >= 2), el cambio máximo entre muestras es suficientemente grande (maxDelta > 50), y los cambios no son lineales (!tieneCambiosLineales), la onda es clasificada como senoidal. Las ondas senoidales tienen transiciones suaves y un gran cambio entre los picos.
* **Desconocida**: Si ninguna de las condiciones anteriores se cumple, se clasifica la onda como "Desconocido”.

7. Descripción de la función mostrarResultados()

Esta función muestra los resultados de la adquisición en el LCD.

1. Mostrar la frecuencia:

* lcd.clear();: Limpia la pantalla del LCD.
* lcd.setCursor(0, 0);: Posiciona el cursor en la primera fila.
* lcd.print("Freq: ");: Imprime el texto "Freq: ".
* lcd.print(frecuencia);: Imprime la frecuencia calculada.
* lcd.print(" Hz");: Añade la unidad "Hz".

2. Mostrar la amplitud:

* lcd.setCursor(0, 1);: Posiciona el cursor en la segunda fila.
* lcd.print("Amp: ");: Imprime el texto "Amp: ".
* lcd.print(amplitud);: Imprime la amplitud calculada.
* lcd.print("V");: Añade la unidad "V".

3. Esperar 2 segundos:

* delay(2000);: Pausa la ejecución durante 2 segundos.

4. Mostrar el tipo de onda:

* lcd.clear();: Limpia la pantalla del LCD.
* lcd.setCursor(0, 0);: Posiciona el cursor en la primera fila.
* lcd.print("Tipo de onda:");: Imprime el texto "Tipo de onda:".
* lcd.setCursor(0, 1);: Posiciona el cursor en la segunda fila.
* lcd.print(tipoOnda);: Imprime el tipo de onda.
* Esperar 2 segundos más: delay(2000);: Pausa la ejecución durante 2 segundos.

8. Descripción de la función cleanup()

● delete[] valoresAnalogicos;: Libera la memoria dinámica asignada

previamente para valoresAnalogicos, evitando fugas de memoria