#include <LiquidCrystal.h> // Librería para el control de la pantalla LCD

// Configuración de los pines para la pantalla LCD (12, 11, 5, 4, 3, 2)

LiquidCrystal pantalla(12, 11, 5, 4, 3, 2);

// Definimos los pines y variables globales

const int entradaAnalogica = A0; // Entrada analógica para la señal

const int botonInicio = 7; // Botón para iniciar la adquisición de datos

const int botonVer = 8; // Botón para mostrar información

// Variables para la adquisición de señal

float\* datosAmplitud; // Puntero para el arreglo dinámico de amplitudes

float\* segDerivadas; // Puntero para almacenar las segundas derivadas

int numMuestras = 60; // Número de muestras para la señal

float maxAmplitud = 0;

float minAmplitud = 1023;

float picoAmplitud = 0;

float valorFrecuencia = 0;

unsigned long tiempoPrevio = 0;

String tipoOnda = "Indefinida"; // Variable para guardar la forma de la onda

// Variables de control de pantalla

bool alternarCaracteristicas = true; // Control para alternar entre características

// Función para inicializar el sistema

void setup() {

// Inicialización de la pantalla LCD

pantalla.begin(16, 2); // Configura una pantalla de 16 columnas y 2 filas

pantalla.print("En espera..."); // Muestra un mensaje de espera

// Configuración de los pines de los botones con resistencias pull-up internas

pinMode(botonInicio, INPUT\_PULLUP);

pinMode(botonVer, INPUT\_PULLUP);

// Inicializar memoria dinámica para las amplitudes y segundas derivadas

datosAmplitud = new float[numMuestras]; // Reserva memoria para el arreglo dinámico

segDerivadas = new float[numMuestras - 2]; // Reserva para las segundas derivadas

// Inicializar comunicación serial

Serial.begin(9600); // Configura la velocidad de comunicación serial a 9600 baudios

}

// Función principal que corre en bucle

void loop() {

// Verificar si el botón de inicio fue presionado (se leerá LOW cuando esté presionado)

if (digitalRead(botonInicio) == LOW) {

pantalla.clear();

pantalla.print("Capturando...");

capturarDatos(); // Ahora adquirimos las 60 muestras cuando se presiona el botón

delay(200); // Pequeño retardo para evitar lecturas múltiples al presionar el botón

}

// Verificar si el botón de mostrar fue presionado

if (digitalRead(botonVer) == LOW) {

desplegarResultados(); // Mostrar los resultados en pantalla

delay(200); // Pequeño retardo para evitar lecturas múltiples

}

}

// Función para la adquisición de la señal analógica

void capturarDatos() {

// Leer 60 muestras consecutivamente

for (int i = 0; i < numMuestras; i++) {

// Lee el valor de la señal analógica y lo almacena en el arreglo

datosAmplitud[i] = analogRead(entradaAnalogica);

Serial.println(datosAmplitud[i]); // Imprimir las amplitudes en el monitor serial

// Actualizar el valor máximo y mínimo de la señal

if (datosAmplitud[i] > maxAmplitud) {

maxAmplitud = datosAmplitud[i]; // Actualizar valor máximo

}

if (datosAmplitud[i] < minAmplitud) {

minAmplitud = datosAmplitud[i]; // Actualizar valor mínimo

}

// Calcular el tiempo entre picos (para la frecuencia)

unsigned long tiempoActual = millis();

if (datosAmplitud[i] > 512 && tiempoActual - tiempoPrevio > 20) { // Detectar un cruce por cero

valorFrecuencia = 1000.0 / (tiempoActual - tiempoPrevio); // Calcular frecuencia en Hz

tiempoPrevio = tiempoActual;

}

// Introducir un pequeño retardo entre lecturas para evitar que se tomen todas al instante

delay(10);

}

}

// Función para calcular la segunda derivada (en valor absoluto) y contar cuántos elementos son menores que 5

void calcularFormaOnda() {

int contadorBajos = 0; // Contador para elementos menores que 5

// Calcular la segunda derivada en valor absoluto

for (int i = 2; i < numMuestras; i++) {

// Calcular la segunda derivada como la magnitud absoluta de las diferencias

float derivada2 = abs(datosAmplitud[i]) - 2 \* abs(datosAmplitud[i - 1]) + abs(datosAmplitud[i - 2]);

segDerivadas[i - 2] = abs(derivada2); // Guardar la segunda derivada en valor absoluto

// Si la segunda derivada es menor que 5, incrementar el contador

if (abs(derivada2) < 5) {

contadorBajos++;

}

}

// Imprimir las segundas derivadas en el monitor serial

Serial.println("Derivadas 2:");

for (int i = 0; i < numMuestras - 2; i++) {

Serial.print(segDerivadas[i]);

if (i < numMuestras - 3) Serial.print(", "); // Imprimir coma entre elementos

}

Serial.println(); // Nueva línea al final del arreglo

// Identificar la forma de la onda basada en el número de elementos menores que 5

if (contadorBajos > 10) {

tipoOnda = "Triangular";

} else {

tipoOnda = "Senoidal";

}

}

// Función para mostrar las características de la señal cíclicamente en la pantalla LCD

void alternarPantalla() {

if (alternarCaracteristicas) {

// Mostrar la forma de la onda

pantalla.clear();

pantalla.setCursor(0, 0);

pantalla.print("Tipo de Onda: ");

pantalla.setCursor(0, 1);

pantalla.print(tipoOnda);

} else {

// Mostrar la amplitud y la frecuencia

pantalla.clear();

pantalla.setCursor(0, 0);

pantalla.print("Amp: ");

pantalla.print(picoAmplitud);

pantalla.print("V");

pantalla.setCursor(0, 1);

pantalla.print("Freq: ");

pantalla.print(valorFrecuencia);

pantalla.print(" Hz");

}

alternarCaracteristicas = !alternarCaracteristicas; // Alternar entre características

delay(2000); // Pausa de 2 segundos antes de cambiar la pantalla

}

// Función para mostrar los resultados en la pantalla LCD

void desplegarResultados() {

// Identificar la forma de la onda usando la segunda derivada

calcularFormaOnda();

// Calcular amplitud pico a pico

picoAmplitud = (maxAmplitud - minAmplitud) \* (5.0 / 1023.0); // Convertir a voltios

// Mostrar características cíclicamente hasta que el usuario pulse el botón de inicio

int ciclos = 0;

while (digitalRead(botonInicio) == HIGH && ciclos != 6) {

ciclos++;

alternarPantalla();

}

pantalla.clear();

pantalla.setCursor(0, 0);

pantalla.print("Presiona:");

pantalla.setCursor(0, 1);

pantalla.print("INICIO");

// Reiniciar las variables de adquisición

maxAmplitud = 0;

minAmplitud = 1023;

tiempoPrevio = 0;

}

// Función para liberar la memoria asignada dinámicamente al finalizar el programa

void limpiarMemoria() {

delete[] datosAmplitud;

delete[] segDerivadas;

}