Registrador de datos de 6 señales

Pablo Vivar Colina

18 de junio de 2023

1. Introducción

Un Registrador de datos (data logger) es un dispositivo electrónico que registra datos de sensores o entradas externas en intervalos de tiempo determinados.

Un Registrador de datos de 6 señales analógicas es un tipo de data logger que puede medir y almacenar hasta 6 variables físicas diferentes, como temperatura, presión, humedad, nivel de luz, etc. Estas variables se convierten en señales eléctricas que varían en función de la magnitud medida y se envían al data logger mediante cables o conexiones inalámbricas.

2. Necesidades del diseño

El laboratorio de Máquinas eléctricas encomendó a IDEA 1.61 la implementación de un dispositivo con el uso de medición de equipo generador de energía tipo maremotriz.

Para este proyecto se solicitó la lectura de voltaje y corriente de 3 fases, es decir 6 señales en total, la conversión de dichas variables físicas ya está implementada en el Laboratorio de Máquinas Eléctricas, además de ello que los datos fueron guardados en una memoria micro SD y que la tasa de muestreo fuera de mínimo 480 datos por segundo por cada dato. Además de ello que pudiera almacenar datos por lo menos de 7 días continuos.

3. Implementación de Solución propuesta

Para implementar la solución utilizando un Arduino Nano para leer voltaje y corriente de 3 fases, junto con un módulo para escribir en una tarjeta microSD, puedes seguir los siguientes pasos:

3.1. Hardware necesario

- Arduino Nano: Esta placa microcontroladora tiene suficientes pines analógicos para leer las señales de voltaje y corriente. (A0 a A5)
- Módulo de lectura de voltajes y corrientes: Se puede utilizar sensores de voltaje y de corriente adecuados para medir cada una de las fases. Esta parte del diseño ya está implementada en el laboratorio de Máquinas Eléctricas con la tarjeta Taraz technologies

  USM-3IV1
- Módulo de tarjeta microSD: Para escribir los datos, se necesitará un módulo que admita tarjetas microSD y se pueda conectar al Arduino Nano, como el módulo "MicroSD Card Adapter". en este caso se adquirió uno que pudiera leer hasta 32 Gb que es el estimado en valor para poder leer las señales requeridas

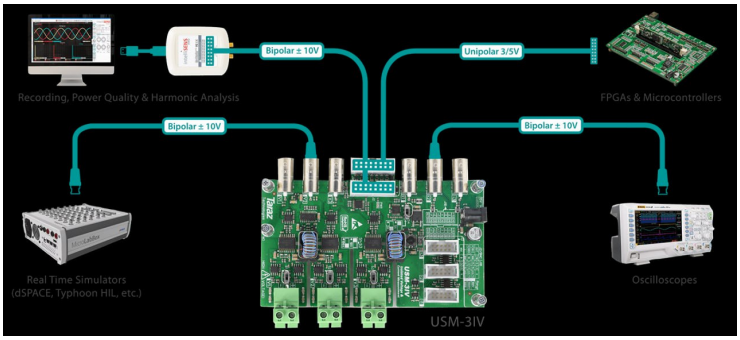

Figura 1: Tarjeta USM-3IV

3.2. Conexiones

- Se conectaron 6 potenciómetro simulando la tarjeta de adquisición de datosonecta los módulos de lectura de voltaje a los pines analógicos del Arduino Nano. Asegúrate de utilizar resistencias divisoras de voltaje si es necesario para adaptar los niveles de voltaje a los rangos de entrada del Arduino.
- Se conectó el módulo de tarjeta microSD al Arduino Nano a través de señales digitales.

En una primera etapa el prototipo se presentó en protoboard y después se implementó una tarjeta impresa con una carcasa de aluminio e impresión 3D.

3.3. Programación

3.3.1. Microcontrolador Arduino NANO

Se ha elaborado un código que toma los valores ingresados por el convertidor analogico digital de las 6 señales y los escribe en 6 columnas separados por tabuladores en la tarjeta de memoria de micro SD. (en un inicio se planteó hacer la conversión de datos a punto flotante pero se ha descartado ya que no se logra la transmisión de bits en el tiempo adecuado. Para su representación gráfica se ha utilizado código m ejectuado en MATLAB.

Con un valor de comunicación serial máximo de la placa de 2000000 se lograron tomar diversos ejercicios de lectura

- Se logró leer hasta 600 datos por segundo (18 segundos a 10700 líneas de 1 columna)
- Se logró leer hasta 116 datos por segundo (22 segundos a 2563 líneas de 6 columnas) (con conversión a flotante)
- Se logró leer hasta 483 datos por segundo (20 segundos a 9644 líneas de 6 columnas) (sin conversión a flotante)

```
   ////////////////////////////////////////////////////////////////
//              Data Logger de 6 señales     v1.00             //
////////////////////////////////////////////////////////////////
#include <SD.h>
#include <SPI.h>

int CS_PIN = 10;

File file;

// El pin digital 2 tiene un pulsador conectado. Dale un nombre:
int pushButton = 2;

void setup()
{

  // Serial.begin(9600);
  // Serial.begin(115200);

  Serial.begin(2000000); // pude leer hasta 600 datos por segundo (18 segundos a 10700 línea
                         // pude leer hasta 116 datos por segundo (22 segundos a 2563 líneas
                         // pude leer hasta 483 datos por segundo (20 segundos a 9644 líneas
  pinMode(pushButton, INPUT);
```

```
  initializeSD();
  createFile("test2.txt");
  writeToFile("Comienza Lectura");

  int buttonState;
  int sensorValue1, sensorValue2, sensorValue3, sensorValue4, sensorValue5, sensorValue6;
  float conv1, conv2, conv3, conv4, conv5, conv6;

  do
  {
    // lee el estado del pulsador:
    buttonState = digitalRead(pushButton);
    Serial.println(buttonState);
    sensorValue1 = analogRead(A0);
    sensorValue2 = analogRead(A1);
    sensorValue3 = analogRead(A2);
    sensorValue4 = analogRead(A3);
    sensorValue5 = analogRead(A4);
    sensorValue6 = analogRead(A5);


    writeToFileValue(sensorValue1);
    file.print("\t");
    writeToFileValue(sensorValue2);
    file.print("\t");
    writeToFileValue(sensorValue3);
    file.print("\t");
    writeToFileValue(sensorValue4);
    file.print("\t");
    writeToFileValue(sensorValue5);
    file.print("\t");
    writeToFileValue(sensorValue6);

    writeToFileString(".");
    // closeFile();
  } while (buttonState == 1);

  closeFile(); // necesario para el guardado final
  Serial.println("FIN DEL PROGRAMA");

}

void loop()
{
}
```

```
void initializeSD()
{
  Serial.println("Inicializando la tarjeta SD...");
  pinMode(CS_PIN, OUTPUT);

  if (SD.begin())
  {
    Serial.println("La tarjeta SD está lista para usar.");
  }
  else
  {
    Serial.println("Error al inicializar la tarjeta SD");
    return;
  }
}

int createFile(char filename[])
{
  file = SD.open(filename, FILE_WRITE);

  if (file)
  {
    Serial.println("Archivo creado exitosamente.");
    return 1;
  }
  else
  {
    Serial.println("Error al crear el archivo.");
    return 0;
  }
}

// funciOn de textos
int writeToFile(char text[])
{
  if (file)
  {
    file.println(text);
    Serial.println("Escribiendo en el archivo: ");
    Serial.println(text);
    return 1;
  }
  else
  {
```

```
    Serial.println("No se pudo escribir en el archivo");
    return 0;
  }
}


int writeToFileValue(int value)
{
  if (file)
  {

    file.print(value);

    return 1;
  }
  else
  {
    // Serial.println("No se pudo escribir en el archivo");
    return 0;
  }
}

int writeToFileString(char text[])
{
  if (file)
  {
    file.println(text);
    Serial.println("Escribiendo en el archivo: ");
    Serial.println(text);
    return 1;
  }
  else
  {
    Serial.println("No se pudo escribir en el archivo");
    return 0;
  }
}

void closeFile()
{
  if (file)
  {
    file.close();
    Serial.println("Archivo cerrado");
  }
}
```

3.3.2. Código m MATLAB

Se necesitan 2 códigos para el flujo de trabajo, uno que convierte los datos que van de un intervalo d 0 a 1023 (por el conversor analógico digital) a valores propuestos de 0 a 20 con punto flotante:

```
    % Nombre del archivo de entrada
archivo_entrada = 'datos.txt';

% Nombre del archivo de salida
archivo_salida = 'datos_convertidos.txt';

% Leer el archivo de entrada
datos = dlmread(archivo_entrada, '\t');

% Convertir los valores enteros a flotantes de 0 a 20 con 2 decimales
datos_convertidos = datos * (20.0 / 1023.0);
datos_convertidos = round(datos_convertidos, 2);

% Guardar los datos convertidos en un archivo de texto
dlmwrite(archivo_salida, datos_convertidos, 'delimiter', '\t', 'precision', '%.2f');

disp('Datos convertidos guardados en el archivo datos_convertidos.txt.');
```

El segundo código es uno que toma los datos convertidos y los despliega en una gráfica.

```
    % Nombre del archivo de entrada
archivo = 'datos_convertidos.txt';

% Leer el archivo de entrada
datos = dlmread(archivo, '\t');

% Obtener el número de filas y columnas
[num_filas, ~] = size(datos);

% Calcular el vector de tiempo en segundos
tiempo = (0 : num_filas-1) / 480; % Se asume una frecuencia de muestreo de 480 muestras por

% Crear una figura para la gráfica
figure;
hold on;

% Generar la gráfica para cada columna
for col = 1 : 6
    amplitud = datos(:, col);
```

```
    plot(tiempo, amplitud);
end

% Configurar el título y etiquetas de los ejes
title('Gráfica de Amplitud vs Tiempo');
xlabel('Tiempo (s)');
ylabel('Amplitud');

% Mostrar la leyenda para cada columna
legend('Columna 1', 'Columna 2', 'Columna 3', 'Columna 4', 'Columna 5', 'Columna 6');

% Mostrar la cuadrícula
grid on;

% Ajustar los ejes para que se vea toda la gráfica
axis tight;

% Mostrar la gráfica
hold off;
```

4. Diseño PCB

Se han dibujado las conexiones en el programa de generación de PCB Eagle de Autodesk, después se ha utilizado métodos tradicionales para su fabricación, aunque también está listo el archivo para manufactura CAM.

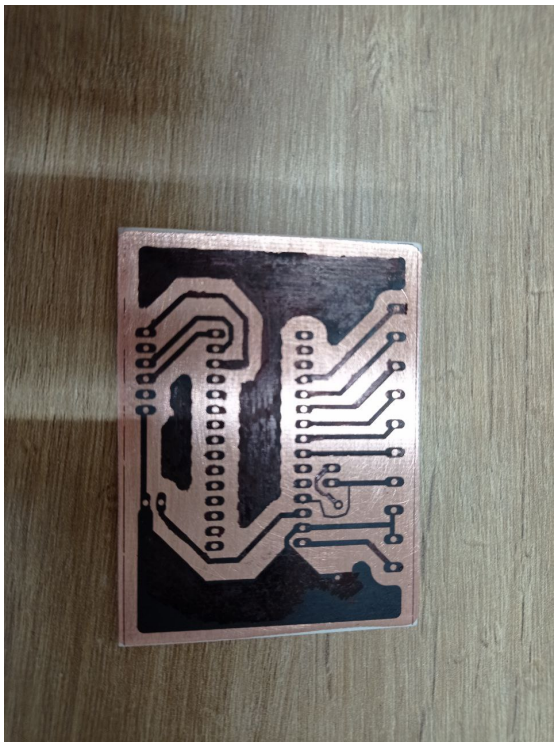

Figura 2: Primer paso de transferencia a placa

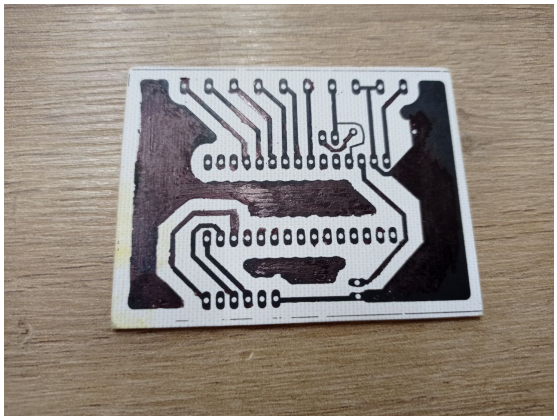

Figura 3: Desintegración de cobre con cloruro Férrico

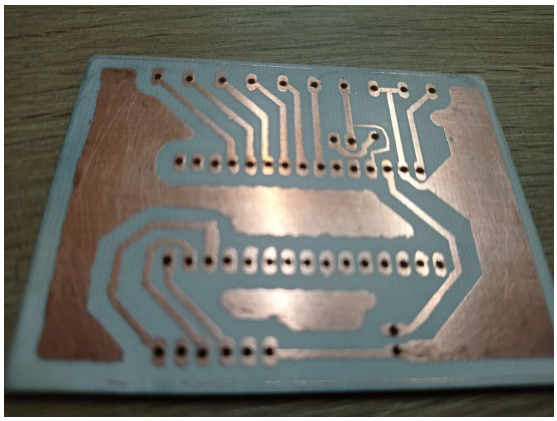

Figura 4: PCB antes de soldar componentes

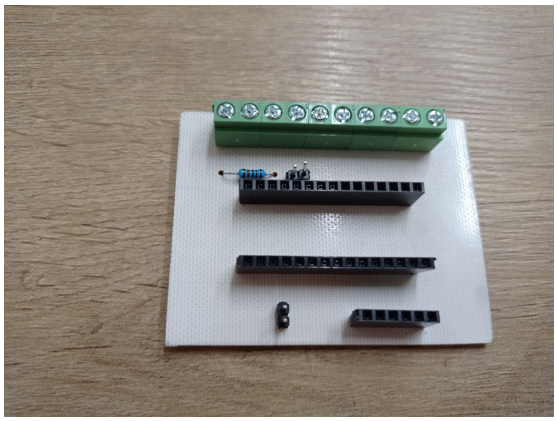

Figura 5: PCB con componentes soldados

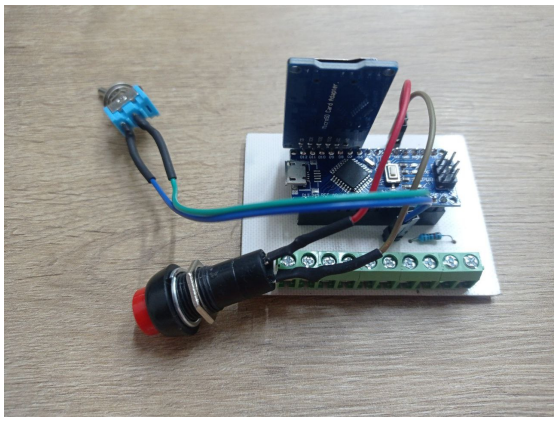

Figura 6: PCB con componentes montados

5. Fabricación

Para finalizar se ha integrado la tarjeta en un gabiete a la medida elaborado con Perfil de aluminio, acrílico y métodos de impresión 3D.

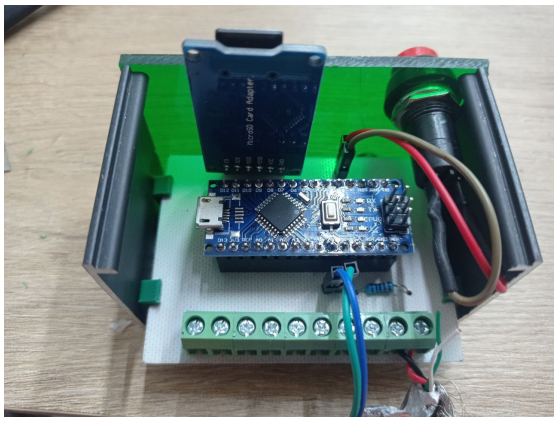

Figura 7: Interior del prototipo

6. Entregable y documentación

6.1. Ciclo de trabajo y operación

Para el ciclo de trabajo con el dispositivo, se necesita tener el interruptor superior en posición de encendido para que una vez conectado a la corriente (5V) en el borne de entrada el dispositivo tome lecturas, una vez abajo el interruptor

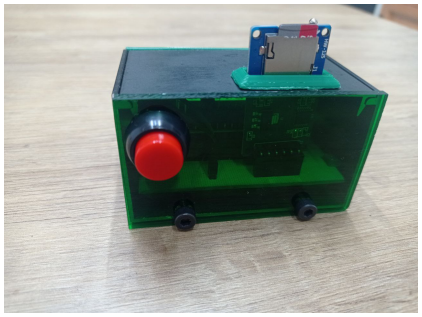

Figura 8: Vista 1 del prototipo

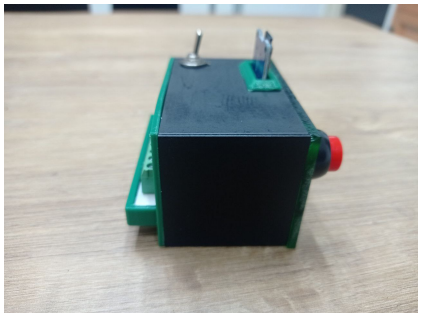

Figura 9: Vista 2 del prototipo

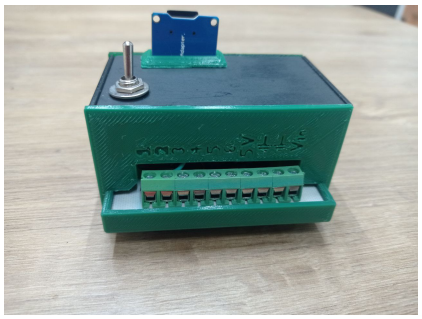

Figura 10: Vista 3 del prototipo

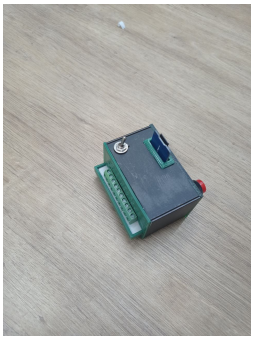

Figura 11: Vista 4 del prototipo

el dispositivo dejará de tomar lecturas, para reiniciar el proceso es necesario volver a colocar el interruptor en posición de encendido y oprimir el boton de reset.

Para usar los datos obtenidos es necesario borrar el texto **Continua lectura** ya que el programa en código m marcará un error de dato no reconocido, hay que hacer notar que cada vez que se reinicia el dispositivo volverá a escribir este mensaje, esto sirve para poder hacer una separación entre lectura y verificar el inicio del programa.

En la computadora es necesario primero convertir los datos obtenidos con el primer código mostrado o llamado **conversionArduino**. este toma los datos guardados en **datos.txt** y los convierte de enteros de 0 a 1023 a datos de variables físcas reales de 0 a 20 en punto flontante a el archivo de texto **datos convertidos.txt**. Después correr el programa o segundo código m mostrado llamado **graficaConversiónArduino** para poder ver los datos convertidos en un rango físico real.

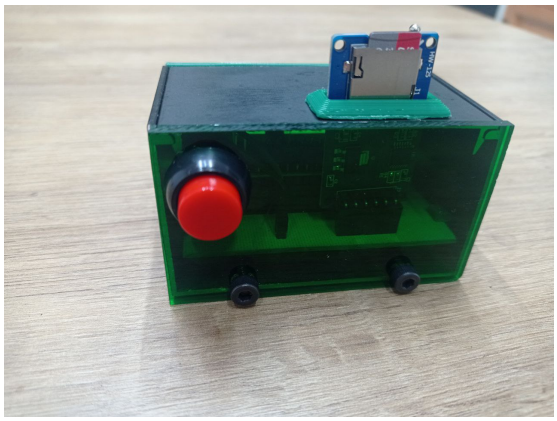

Figura 12: Vista Posterior

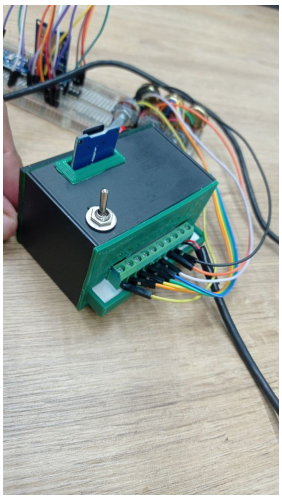

Figura 13: Vista frontal

6.2. Documentación

Para obtener información detallada sobre la implementación de este dispositivo, incluyendo el código fuente, los circuitos de conexión y los pasos de configuración, les invitamos a consultar la documentación completa en el siguiente enlace: Repositorio de GITHUB

Esperamos que esta herramienta sea de utilidad en sus proyectos y les ayude a realizar mediciones precisas y registros de datos confiables en aplicaciones de monitoreo y control. Si tienen alguna pregunta adicional, no duden en ponerse en contacto con nosotros.

Atentamente, Pablo Vivar Colina