Trabajo Final: diseño y desarrollo de un medidor de conductividad térmica

Alesandria, Alejo Samuel (Autor)
Departamento de Ingeniería Electrónica, UTN Facultad Regional San Francisco
Av. de la Universidad 501, San Francisco, Córdoba – Argentina
alealesandria@gmail.com

Cignetti, Mateo Antonio (Autor)
Departamento de Ingeniería Electrónica, UTN Facultad Regional San Francisco
Av. de la Universidad 501, San Francisco, Córdoba – Argentina
mateo@cignetti.ar

Delfino, Ramiro Jesús (Autor)
Departamento de Ingeniería Electrónica, UTN Facultad Regional San Francisco
Av. de la Universidad 501, San Francisco, Córdoba – Argentina
pokramiro12@gmail.com

Galliano, Ignacio (Autor)
Departamento de Ingeniería Electrónica, UTN Facultad Regional San Francisco
Av. de la Universidad 501, San Francisco, Córdoba – Argentina
igalliano@facultad.sanfrancisco.utn.edu.ar

Rinaudo, Facundo Nicolás (Autor)
Departamento de Ingeniería Electrónica, UTN Facultad Regional San Francisco
Av. de la Universidad 501, San Francisco, Córdoba – Argentina
facurinaudo99@gmail.com

Rodriguez, Manuel (Autor)
Departamento de Ingeniería Electrónica, UTN Facultad Regional San Francisco
Av. de la Universidad 501, San Francisco, Córdoba – Argentina
manuel.rodr.98@gmail.com

RESUMEN

La conductividad térmica es una propiedad física de los materiales que mide la capacidad de conducción de calor. En este contexto, se emplea un medidor especializado para determinar esta propiedad en diversos materiales.

Por medio de diferentes controles, como la regulación de temperatura, la activación controlada de un dimmer, entre otros, se puede lograr medir la conductividad de los materiales sometidos a la prueba. El presente informe tiene como objetivo detallar este procedimiento, abordando tanto su programación específica como su desarrollo práctico.

Palabras clave: materiales, conductividad térmica, control, programación.

ABSTRACT

Thermal conductivity is a physical property of materials that measures their heat conduction capacity. In this context, a specialized meter is used to determine this property in various materials.

Through different controls, such as temperature regulation, controlled activation of a dimmer, among others, it is possible to measure the conductivity of the materials being tested. The objective of this project is to detail this procedure, addressing both its specific programming and its practical development.

Keywords: materials, thermal conductivity, control, programming.

INTRODUCCIÓN

Mediante el medidor de placa caliente con guarda se procura establecer dentro de las probetas, cuya forma es la de una placa uniforme de caras plano-paralelas, una densidad de flujo de calor uniforme y unidireccional en condiciones de régimen estacionario, tal como la que existiera en una placa infinita rodeada por dos superficies planas isométricas.

La unidad calefactora está constituida por una sección de medición independiente, donde puede establecerse una densidad de flujo de calor unidireccional, constante y uniforme, rodeada por una sección de guarda separada por una ranura estrecha. La unidad de enfriamiento consiste en una placa plana de aluminio del tamaño de la unidad calefactora (Dominguez Vega & Culzoni, 2008). La temperatura de cada una de las unidades se censa con sensores NTC.

Los sensores NTC (Negative Temperature Coefficient) se basan en la propiedad de ciertos materiales de cambiar su resistencia eléctrica en función de la temperatura. En este caso, su resistencia disminuye a medida que aumenta la temperatura. La estructura de un sensor NTC consiste en un material semiconductor, como óxido de metal, que presenta esta propiedad de variación de resistencia con la temperatura.

La curva de la resistencia del termistor NTC no es lineal, pero se pueden utilizar unos circuitos sencillos que ayudan a linealizar su característica de forma apreciable. En este caso se utiliza el circuito que se muestra en la figura 1.

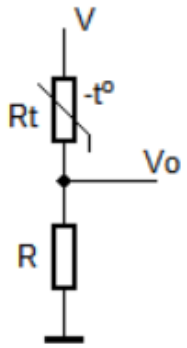


Figura 1: Circuito de utilización de NTC.

Este circuito proporciona una variación positiva con la temperatura. Los márgenes de temperatura entre los que pueden trabajar estos termistores están normalmente entre -55 °C y 125 °C, con una tolerancia del valor nominal comprendida entre ±10% y ±5% (BetaTHERM, 2020). Para este caso se linealiza para un rango de temperatura de entre 10 y 80 °C.

Para este circuito, la ecuación que rige para su linealización es la siguiente:

$$V_0 = \frac{V \times R}{R_T + R} \qquad (Ecuación\ 1)$$

El valor de la resistencia del termistor a una temperatura "T" se calcula de la siguiente manera:

$$R_T = R_0 \times e^{\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)} \qquad (Ecuación\ 2)$$

Para evaluar los parámetros R_0 y β se necesitan dos puntos de calibración (T1, R1) y (T2, R2). Se obtiene así de β dado por la expresión:

$$\beta = \frac{\ln\left(\frac{R_1}{R_2}\right)}{\frac{1}{T_1} - \frac{1}{T_2}} \qquad (Ecuación\ 3)$$

El calentamiento de la placa caliente se realiza a través de resistencias calentadoras, las mismas se controlan con un dimmer diseñado con un triac disparado por un optotriac. El cual es controlado por sofware.

Objetivos:

- Construir un prototipo con materiales localmente accesibles, considerando un razonable costo-beneficio en cuanto a efectividad de medición requerida y resultados confiables.
- Realizar la selección de los componentes para, el sistema de placa caliente y de igual manera para el de placa fría.
- Implementar un sistema de control de temperatura.
- Anexar un sistema de adquisición de datos.

Materiales:

- ESP32 devkit V1.
- Sensores NTC.
- Placas Peltier.
- Coolers.
- Disipadores de aluminio.
- Dimmer controlado.
- Medidor de conductividad térmica brindado por UTN Facultad Regional Rafaela.
- Transformador.
- Resistencia calentadora

DESARROLLO

Metodología de trabajo

La programación es realizada en FreeRTOS, en el marco ESP-IDF (Espressif Systems, 2023), y se utilizó Git para todo el código y la plataforma online de GitHub[1], una vez terminado el repositorio será público. Por medio de esta herramienta es posible organizar cada tarea a realizar, junto con los cambios pertinentes a arreglos, mejoras y agregados. Además, cabe destacar que la programación de cada etapa del trabajo se realizó por separado y posteriormente se procedió a implementación conjunta bajo un mismo proyecto.

Git es un software de control de versiones, pensado en la eficiencia, confiabilidad y compatibilidad del mantenimiento de versiones de aplicación, permitiendo en todo momento al acceso de las distintas versiones que fueron publicadas. Por otra parte, GitHub es un control de versiones de Git, permitiendo crear repositorios destinados para proyectos que son alojados en la nube. También brinda las opciones de *Issues* (propuestas), discussions (discusiones/debates), pull request (pedidos de admisión de código).

[1] https://github.com/MateoCignetti/Hornito2023

Driver de utilización del conversor analógico a digital (ADC)

Para la facilitación del uso del ADC del microcontrolador ESP32, se programó un controlador auxiliar utilizando las funciones del driver oficial de las librerías de esp-idf. El mismo se encuentra integrado en "adc.h" y "adc.c" y facilita la inicialización y configuración de las unidades del ADC y los canales a utilizar. Además, posee funciones para facilitar la obtención de los valores en mV en los pines del ADC, con o sin multisampling. Finalmente, provee una tarea opcional dedicada a la depuración, la cual imprime en pantalla los valores de todos los canales configurados del ADC.

Medición de temperatura

Los sensores utilizados son unos NTC de resistencia típica a 25 °C de 100 kΩ, este sensor es suficientemente lineal para rangos pequeños de temperatura, pero para rangos de unos 60 °C considerarlo lineal sería un error. Por ello, mediante la ecuación 4 de linealización se logra que el sensor cumpla con los requerimientos de acuerdo con el rango estipulado. Además, como se mencionó se realizó un circuito divisor de tensión con el mismo y una resistencia de 100 kΩ. La integración de lo mencionado se encuentra en "ntc.c" y "ntc.h", donde se encuentra una función que implementa la ecuación 4 y devuelve el valor de temperatura en °C.

$$T = \frac{\beta * T_{0_{NTC}}}{\beta + \left[T_{0_{NTC}} * \ln\left(\frac{R_{NTC}}{R_{0_{NTC}}}\right)\right]} - 273{,}15 \qquad (Ecuación\ 4)$$

Figura 2: Termistor utilizado.

Dimmer

El control de potencia de las resistencias del sistema de calefacción se logra mediante la implementación de un circuito de rectificación controlada o dimmer, basado en TRIACs, que se muestra en la Figura 3.

La etapa de control emplea un circuito de detección de cruce por cero que utiliza un optotransistor PC817 y un transformador 220 V – 12 V, para adecuar la tensión de entrada al pin PD0 del microcontrolador.

Luego de un tiempo de espera (ajustado para coincidir con la mitad del periodo de la onda de 50 Hz de corriente alterna), se genera un pulso en el pin de salida (PT0 o PT1) que controla el optoacoplador MOC3021, que a su vez controla el ángulo de disparo del TRIAC. La variación de este ángulo de disparo modificará la potencia suministrada a la carga, permitiendo así el control preciso de la temperatura de la placa caliente.

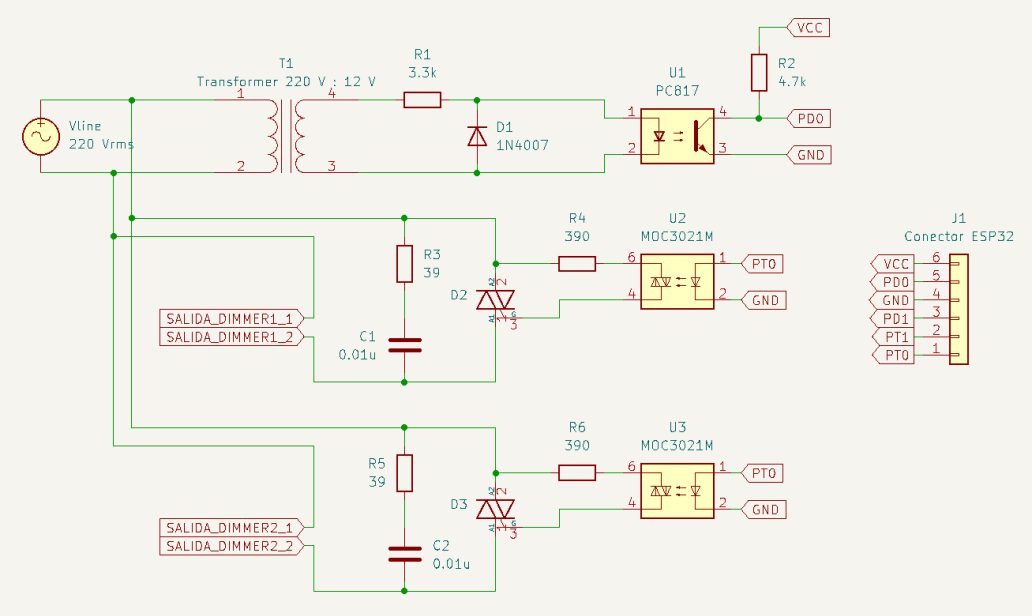


Figura 3: Circuito de ambos dimmer.

Lógica de programación

El código del dimmer está organizado en dos archivos que son “dimmer.c” y “dimmer.h”.

En este código, se utilizan técnicas como semáforos y rutinas de interrupción para sincronizar el código y garantizar un funcionamiento ordenado y eficiente del dimmer.

La lógica principal se encuentra en la tarea vTaskDimmer, que responde a los eventos de la interrupción para controlar la potencia de salida del dimmer.

Para sincronizar la tarea principal con la rutina de interrupción se utiliza un semáforo binario; esta rutina se activará cuando en el pin de entrada se detecte un cruce por cero de la señal de alterna y liberará el semáforo.

En la tarea vTaskDimmer, la función xSemaphoreTake se utiliza para esperar a que la interrupción libere el semáforo, y luego tomarlo, para ejecutar la sección critica del código. Si está disponible, la ejecución del código continua dentro del bloque if, y se controla el nivel del pin de salida para modular la potencia de salida del dimmer. Si no puede tomar el semáforo dentro del tiempo especificado por DIMMER_SEMAPHORE_TIMEOUT_MS, se registra un mensaje de error indicando un "timeout" al querer adquirir el semáforo. Este error sucederá en caso de que se encuentre mal conectado el dispositivo.

Control de temperatura placa caliente

Este sistema consta de dos resistencias, una interna y otra externa, las cuales se encuentran montadas sobre un ladrillo refractario, material que además de otorgar aislamiento térmico hace de aislante eléctrico. El material de la placa que se encuentra en contacto con la probeta es aluminio.

Figura 4: Ladrillo refractario con resistencias incrustadas.

El objetivo de este sistema es mantener una temperatura constante de 36,7° en toda la superficie del material; valor estipulado por el INTI para realizar este tipo de ensayo, para realizar pruebas en laboratorio se considera 60° el set-point de temperatura. Si la temperatura comienza a bajar indica que el material está disipando la potencia entregada, por ello se debe suministrar más para conseguir la misma temperatura, y viceversa. El sistema busca controlar ese suministro de potencia, a través de la realimentación del sensor NTC.

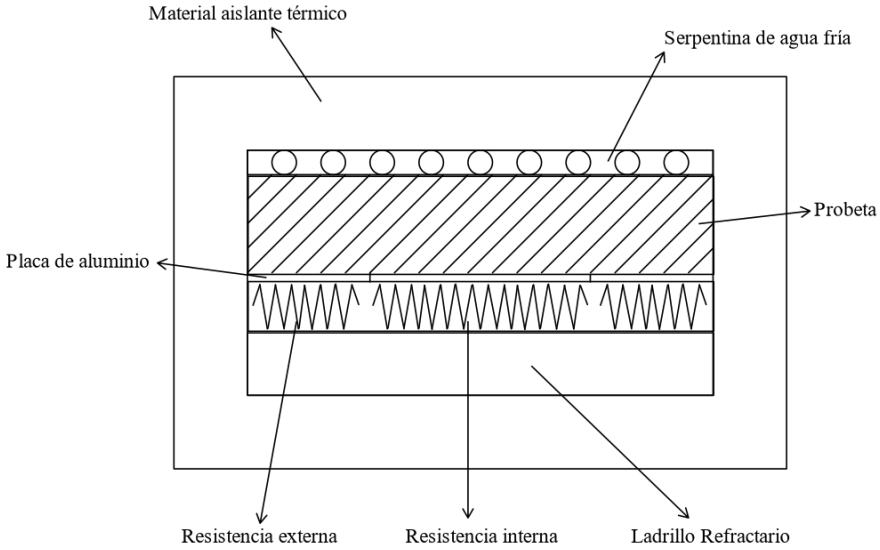


Figura 5: Esquema del medidor de conductividad térmica.

Lógica de programación

El sistema se implementa en "control_system.h" y "control_system.c". En primera instancia se crean las tareas, también los semáforos y colas necesarios para la sincronización y correcto funcionamiento del programa. Estas técnicas de sincronización son necesarias para generar un correcto orden de prioridades, como así lograr asegurar que el dato que se envía o recibe sea los más rápido y preciso posible. Además, se crea un mutex para las secciones críticas dentro del programa.

La primera de las funciones es la encargada de obtener la temperatura, para ello se utiliza get_ntc_temperature_c() y get_adc_voltage_mv_multisampling(). Para tomar correctamente el valor de temperatura se realiza dentro de una sección crítica determinada por el mutex. Luego se debe enviar el dato de temperatura obtenido al WebServer, para ello se toma un semáforo liberado por el servidor web indicando que ya se puede actualizar el valor; cuando ocurre, el mismo es enviado a través de una cola.

Cuando el valor de temperatura es obtenido, se procede a realizar la lógica de decisión para el suministro de potencia; se determina una diferencia, calculada entre el set-point deseado y el valor obtenido, determinando que cuanto mayor sea la diferencia más cantidad de onda de tensión debe ser aplicada a las resistencias, dicho en otras palabras, el dimmer debe disparar más rápidamente para que conduzca mayor parte de la onda. La curva de la figura 6 muestra los pasos entre los valores de temperatura, y cuanto delay se configura en el dimmer de acuerdo con la misma.

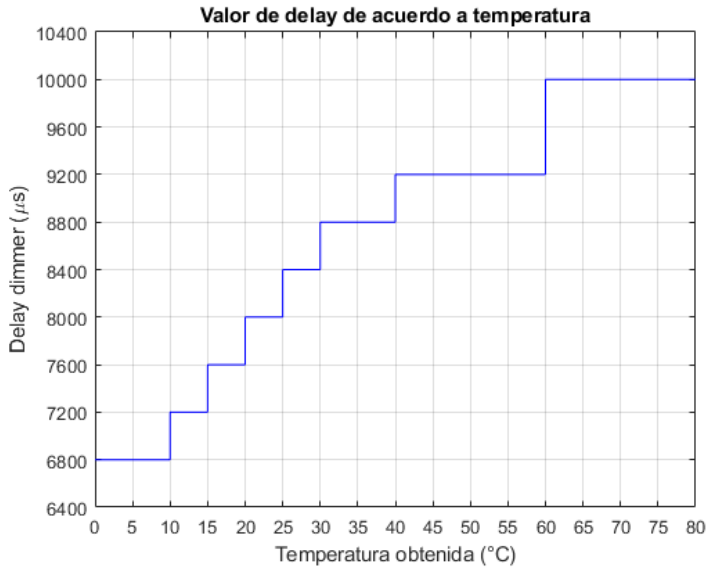


Figura 6: Gráfico en función de la temperatura obtenida con setpoint de 60 °C.

Otra función de importancia es la de envío de "steps" al cálculo de potencia, si esto no se realiza de forma correcta, la medición será errónea; en todo momento para calcular la tensión se debe conocer el valor de step (delay de dimmer), dependiendo de la temperatura.

Por último, se encuentra la función para eliminar las tareas cuando se termina el proceso y el archivo del servidor web es guardado. Se verifica que el handle de la tarea no sea nulo, se elimina la misma, y se configura el handle en NULL.

Medición de potencia de salida

Para la medición de potencia de salida, se comenzó con la medición de resistencia de los elementos calentadores, tanto el interno como el externo. Se obtuvo un valor de 33,5 Ω para la resistencia interna y un valor de 217,5 Ω para la externa, valor que se comprobó permanecer constante incluso a temperaturas altas de operación, de unos aproximados 100 °C.

Sabiendo que la potencia activa se obtiene a partir de la ley de Joule en su forma clásica (con una carga resistiva):

$$P = I^2 \cdot R = \frac{V^2}{R} \qquad (Ecuación\ 5)$$

Y conociendo ahora que la resistencia en el circuito es constante, se puede calcular la potencia de salida conociendo los valores eficaces de tensión o corriente.

Debido a los avances del proyecto en 2022 (Díaz & Previotto, 2022), se tenían en posesión unos módulos sensores de corriente ±30 A con el circuito integrado ACS712, con los que se realizaron pruebas en el laboratorio con distintos valores de corriente

para decidir su utilidad en el proyecto. Sin embargo, se llegó a la conclusión que, debido a su gran rango y los bajos valores de corriente del proyecto, no se implementarían en el proyecto sin un gran error de medición:

$$Corriente\ máxima\ resistencia\ interna = \frac{V}{R} = \frac{220\ V}{31{,}5\ \Omega} = 6{,}98\ A \qquad (Ecuación\ 6)$$

$$Corriente\ máxima\ resistencia\ externa = \frac{V}{R} = \frac{220\ V}{217{,}5\ \Omega} = 1{,}01\ A \qquad (Ecuación\ 7)$$

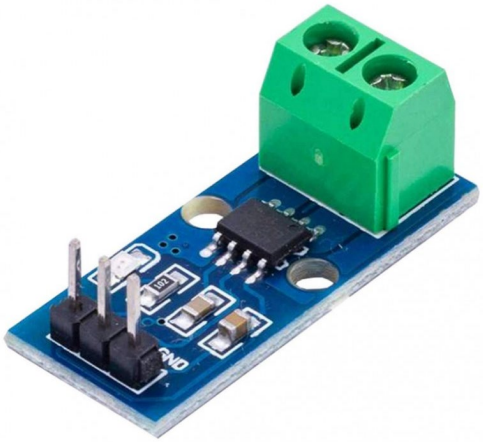


Figura 7: Módulo de sensor de corriente con el CI ACS712.

Por otro lado, se intentó realizar el cálculo de la potencia de salida tomando de referencia una señal sinusoidal pura de 311 V de amplitud, midiendo la tensión eficaz dependiendo del recorte actual del dimmer en el proyecto. Sin embargo, se obtuvieron valores erróneos, debido a que la forma de onda de la tensión de línea no es una sinusoidal pura, sino una muy deformada por armónicos, aplanando los extremos y aumentando la pendiente de la onda.

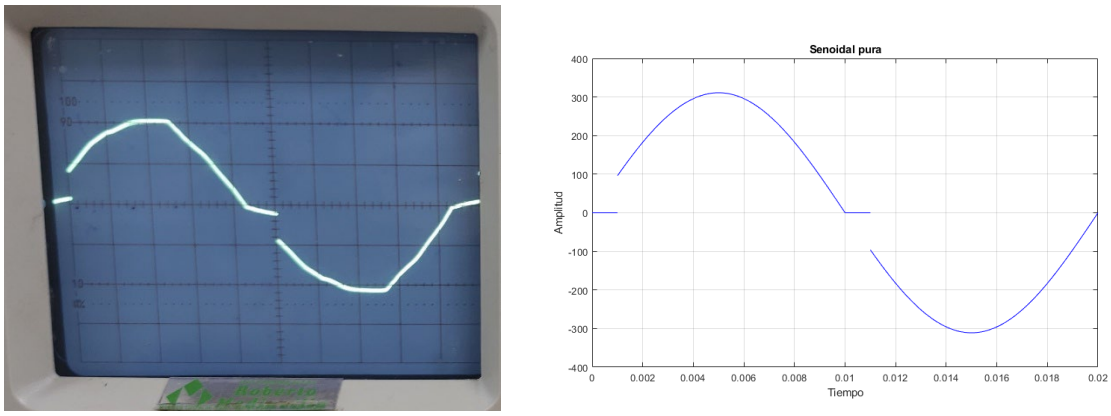


Figura 8: Señal de salida del dimmer con 1 ms sin conducción. A la izquierda, señal experimentada. A la derecha, señal ideal.

Se decidió entonces diseñar un circuito y un sistema de muestreo de la señal de entrada, para obtener la forma de onda real, sin importar su forma, valor pico o frecuencia. Para ello, se utilizó un transformador para reducir la tensión de entrada de 220 V a 12 V y para separar el circuito de muestreo galvánicamente del circuito de línea, un puente rectificador de onda completa para mantener los valores de tensión por encima de los 0 V, y finalmente un divisor resistivo con resistencia variable para modificar la tensión de entrada al microcontrolador de acuerdo con el ADC.

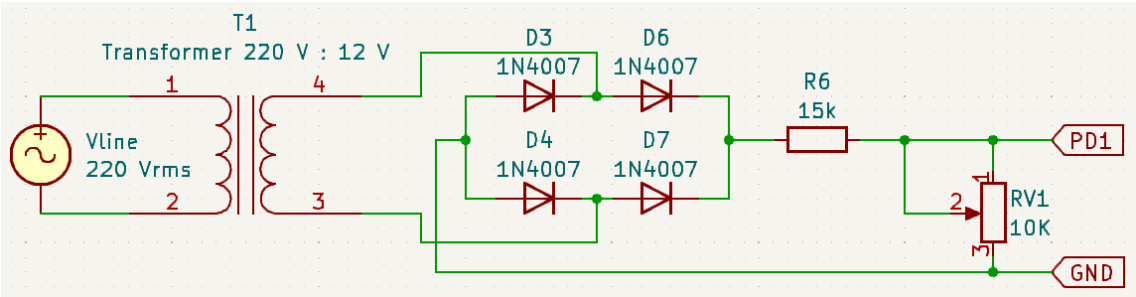


Figura 9: Circuito de muestreo de tensión de entrada.

Una vez obtenido el valor de tensión mediante el ADC, se puede obtener el valor de tensión original con la siguiente ecuación:

$$\mathrm{dV} = (V_{ADC} * r + V_F) * a \qquad (Ecuación\ 8)$$

Donde:

$dV = valor\ instantáneo\ de\ tensión\ de\ línea.$

$V_{ADC} = valor\ de\ tensión\ en\ el\ ADC.$

$r = relación\ de\ transformación\ del\ divisor\ resistivo = \frac{\hat{V}_{antes}}{\hat{V}_{después}}.$

$V_F = caída\ de\ tensión\ del\ puente\ rectificador\ (igual\ a\ la\ caída\ de\ dos\ diodos).$

$a = relación\ de\ transformación\ del\ transformador\ de\ entrada = \frac{V_p}{V_s}.$

Programación de la medición de potencia

El sistema se implementa en el código del archivo “power.h” y “power.c”. Funciona con una tarea que espera la liberación de un semáforo por parte del servidor web, la cual, una vez liberado, crea un temporizador mediante la librería “ESP Timer”, la cual ofrece temporizadores no limitados por el tickrate de FreeRTOS. Se configura un temporizador de 2500 Hz, el cual generará una interrupción cada 400 microsegundos, que se encargará de tomar constantes mediciones del ADC y guardará 50 mediciones en un arreglo, comenzando con un valor de 0 V y terminando en uno cercano a 0, guardando de esa forma los valores de un período de una onda de 50 Hz.

Una vez obtenidos todos los valores, se escalan según la ecuación 8 y se libera un semáforo para el programa del sistema de control para luego esperar la recepción de un valor de "pasos" por una cola. Una vez recibida, se calcula el valor de tensión eficaz a la salida del dimmer, junto con el valor de potencia en Watts, el cual finalmente se envía por otra cola.

$$V_{RMS} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_N^2}{N}} \qquad (Ecuación\ 9)$$

Donde:
$x_n = valor\ de\ la\ medición\ n.$
$N = cantidad\ total\ de\ mediciones.$

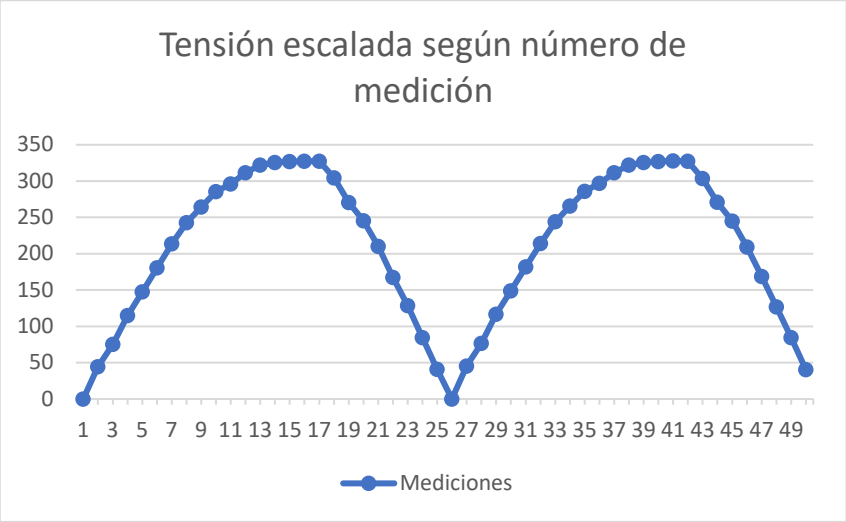


Figura 10: Arreglo de mediciones obtenidas.

Implementación física de la placa desarrollada

Tanto el dimmer como el circuito para el muestreo de la señal de salida del mismo fueron implementados en una misma placa de circuito impreso (PCB)(Figura 11).

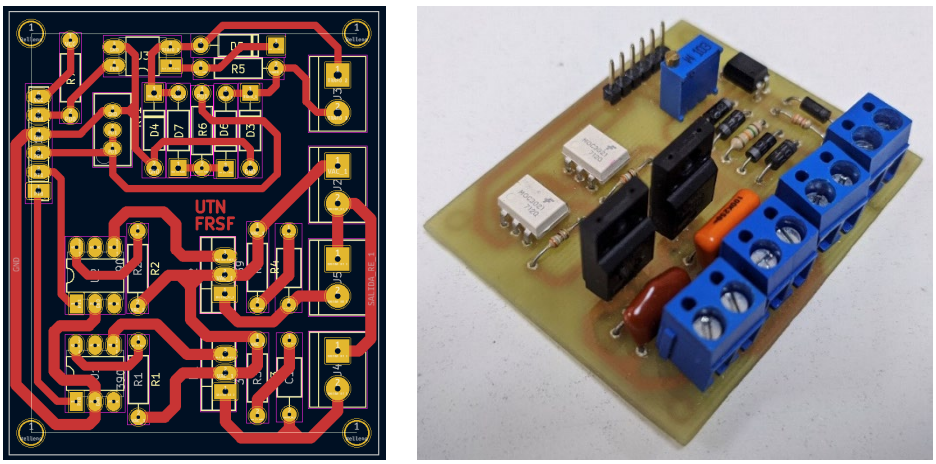


Figura 11: Placa de circuito impreso desarrollada.

Refrigeración con placas Peltier

La placa fría, al igual que la placa caliente, consiste en una placa de aluminio, que se encuentra en contacto con la probeta. La diferencia que existe radica en la temperatura constante y uniforme a la que se debe mantener, que es de 13,0 °C, dentro del 2% de la diferencia de temperatura a través de la probeta, y menor que la correspondiente a la unidad calefactora. Esto puede lograrse mediante el uso de un fluido de temperatura constante, por el uso de calefactores eléctricos, por la utilización de un aislante térmico de resistencia térmica uniforme aplicado entre las superficies externas de la unidad calefactora y de las placas de enfriamiento auxiliares, o, como se llevó a cabo en este caso, por la combinación de los tres sistemas que permita obtener la temperatura deseada en la unidad de enfriamiento.

En primera instancia, para lograr la temperatura deseada en la placa de enfriamiento, se construyó un circuito el cual constaba de tres partes: un depósito de agua aislado térmicamente, el sistema de refrigeración, y la placa fría que se coloca dentro del experimentador. Este circuito se puede observar en el diagrama de bloques de la Figura 12.

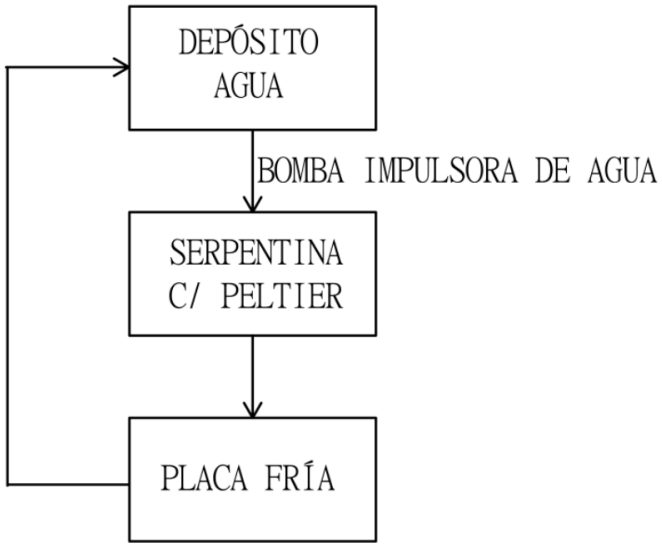


Figura 12 : Diagrama de bloques de prototipo inicial.

El sistema de refrigeración se componía de cuatro partes. Una placa de aluminio la cual se utilizaba para aumentar la superficie de contacto entre las placas Peltier en la parte superior, y la serpentina de cobre por la cual circulaba el agua, en la parte inferior. Una unión de la placa se enfría, la cual estará en contacto con la placa de aluminio, mientras que la otra se calienta. Para disipar el calor de esta cara de la placa se utiliza un sistema de disipación forzada compuesto por un disipador de aluminio y un cooler. Para aumentar la transferencia de temperatura, tanto de la cara fría de la

celda a la placa de aluminio, como de la cara caliente al disipador, se utiliza pasta térmica.

Cabe destacar, además, que se decide utilizar placas Peltier ya que tienen un costo y tamaño significativamente menor que otros sistemas de refrigeración, como es el caso de un sistema de refrigeración por gas.

El efecto que producen las placas Peltier ocurre cuando una corriente pasa a través de dos materiales semiconductores (tipo n y tipo p) que están conectados entre sí por puentes eléctricos. La corriente origina una transferencia de calor desde una unión hasta la otra, de tal forma que una unión se enfría mientras que la otra se calienta (Sandoval, Espinosa, & Barahona, 2007)(Figura 13).

Figura 13: Estructura de celda peltier.

Las placas Peltier que se utilizaron son las TEC1 – 12707, cuya tensión de alimentación es de 12 V, la corriente máxima es 7 A, y las dimensiones 40 x 40 x 3,8 [mm].

Por último, la placa fría es una placa de aluminio de 30 x 30 cm, la cual por la parte inferior va a estar en contacto con la muestra a experimentar, y la parte superior tiene adherida una serpentina de cobre por la cual circula el agua proveniente del sistema de refrigeración, como se puede observar en la Figura 14.

Figura 14: Placa fría.

Cada una de estas partes se encontraba conectadas entre sí por medio de mangueras aisladas para que se disipe la menor cantidad de temperatura, y por ellas circulaba agua impulsada por una bomba sumergible que se colocaba dentro del depósito de agua. Es un circuito cerrado por lo que al terminar de circular por la placa que se encontraba dentro del experimentador, volvía al depósito.

Si bien el sistema de refrigeración enfriaba el agua, no llegaba a enfriar la placa de aluminio a 13,0 °C, debido a las pérdidas en la transferencia de temperatura desde la celda Peltier a placa fría. Por lo que se decide modificar el sistema, quedando las demás partes (depósito de agua aislado y placa fría) como se lo había diseñado previamente, y se construye un depósito de aluminio en el cual ingresa el agua para ser refrigerada. Sobre este depósito se adhieren las placas Peltier con los disipadores forzados, y nuevamente para aumentar la transferencia de temperatura, tanto de la cara fría de la placa al depósito, como de la cara caliente al disipador, se utiliza pasta térmica, y por debajo del depósito se aísla térmicamente para evitar pérdidas.

En ambos diseños del sistema de refrigeración, a través de la realimentación del sensor NTC de la placa fría, se controla el encendido de las placas Peltier, como de la bomba con módulos relé actuando como llave; esto se implementa por medio de programación.

Lógica de programación

El código correspondiente al sistema de refrigeración está contenido dentro de “peltier.c” y “peltier.h”.

Al igual que para el control de temperatura de placa caliente, en primera instancia, a través de una función, se crean las tareas, los semáforos y las colas para la sincronización y correcto funcionamiento del programa. Son necesarias estas técnicas para un correcto orden de prioridades, y para asegurar que el dato que se envía o recibe, sea el último, y lo más rápido y preciso posible. Además, se utiliza un mutex para las secciones críticas dentro del programa.

En el código, con la tarea vTaskReadTemperature, se obtiene la temperatura del NTC de la placa fría a través de get_ntc_temperature_c y get_adc_voltage_mv_multisampling. Esto se hace dentro de una sección crítica para garantizar una correcta lectura.

Luego se debe enviar el dato de temperatura obtenido al WebServer, para ello, en la tarea vTaskSendData, se toma un semáforo liberado por el servidor indicando que ya se puede actualizar el valor; cuando ocurre el mismo es enviado a través de una cola.

Dentro de la tarea vTaskDecision, se utiliza este valor de temperatura para obtener la diferencia entre este último y la temperatura deseada, fijada en 13,0 °C. Teniendo en cuenta esta diferencia se plantean 3 posibles casos: en el caso 1, si la diferencia es mayor o igual a 2,0 °C se enciende el total de las Peltiers y se hace circular agua a

través del sistema por medio de la bomba; para el caso 2, si la diferencia es menor o igual a -2 °C se apaga el total de las placas, además de la bomba para que no circule agua; el caso 3, se da cuando la discrepancia se mantiene entre 2 °C y -2 °C, en este modo, solo queda una placa encendida y el agua circulando. Este proceso de medición se realiza cada 5 minutos.

Por último, se encuentra la función para eliminar las tareas cuando se termina el proceso y el archivo del servidor web es guardado.

Wi-Fi y Servidor Web HTTP

En este dispositivo se ha implementado un servidor web HTTP en el ESP32 utilizando el entorno de desarrollo ESP-IDF, que actúa como interfaz de comunicación entre la máquina y el usuario, permitiendo el control de inicio y finalización del proceso, la visualización de datos de mediciones en tiempo real, y la descarga de datos.

Inicialización y configuración del controlador de WiFi

Para poder implementar el servidor HTTP, primero se inicializa el controlador de WiFi y se establece al ESP32 como Punto de Acceso (AP). Este modo de funcionamiento crea una red de WiFi propia que funcione como punto central al que otros dispositivos puedan conectarse y de esta manera comunicarse con el ESP32. El código correspondiente a esta etapa se encuentra contenido dentro de “wifi.c” y “wifi.h”.

A continuación se detallan los pasos para la implementación de la API del controlador de WiFi en el marco ESP-IDF, los cuales se desglosan dentro de tres funciones principales **wifi_lwip_init()**, **wifi_config()**, **wifi_start()**.

La primera función en ejecutarse es **wifi_lwip_init()**, la cual inicializa la pila de red LwIP (Lightweight IP) y el controlador WiFi, mediante las siguientes funciones de la librería “esp_wifi”:

- **nvs_flash_init()**: Inicializa la memoria flash no volátil (NVS) para el almacenamiento de configuración persistente, como la configuración de WiFi.
- **esp_netif_init()**: Inicializa la interfaz de red ESP-IDF utilizada para gestionar las interfaces de red en el sistema.
- **esp_event_loop_create_default()**: Crea un bucle de eventos predeterminado para gestionar eventos de red como la conexión o desconexión del WiFi.
- **esp_netif_create_default_wifi_ap()**: Crea una interfaz WiFi en modo Punto de Acceso (AP).
- **wifi_init_config_t wifi_config** = WIFI_INIT_CONFIG_DEFAULT(): Inicializa una estructura de configuración WiFi con valores predeterminados, que posteriormente será modificada en la segunda función principal.

- **esp_wifi_init(&wifi_config)**: Inicializa el controlador WiFi con la configuración proporcionada.

La siguiente función principal es **wifi_config()**, dentro de la cual se configura la conexión WiFi en modo Punto de Acceso (AP) con los parámetros de red especificados.

- **esp_wifi_set_mode(WIFI_MODE_AP)**: Establece el modo WiFi en modo punto de acceso (AP).
- **esp_wifi_set_config(WIFI_IF_AP, &wifi_config)**: Configura los parámetros del modo punto de acceso (AP) con la configuración proporcionada

  Previo a esta función, se actualiza la estructura de configuración con los siguientes parámetros:
  - Nombre de la red WiFi (AP_SSID)
  - Contraseña para la red WiFi (AP_PASSWORD)
  - Longitud del SSID
  - Modo de autenticación
  - Número de conexiones máximas de clientes

Por último, dentro de la función **wifi_start()**, se hace un llamado a la función **esp_wifi_start(),** la cual se encarga de iniciar la conexión WiFi, en este caso, en modo AP.

Inicialización y configuración del Servidor Web HTTP

Luego de configurar el controlador WiFi como Punto de Acceso (AP), el ESP32 está listo para desempeñar el papel de servidor web. Esta funcionalidad se logra con la implementación de un servidor HTTP, que será capaz de recibir solicitudes HTTP entrantes y proporcionar respuestas a los dispositivos conectados a la red del ESP32. El código correspondiente a esta parte del proyecto se encuentra en los archivos "webserver.c" y "webserver.h".

A continuación, se detallan los pasos para implementar la API expuesta por el servidor HTTP en el marco ESP-IDF.

Primeramente, se declara el handler httpd_server para el servidor HTTP, que se utilizará para realizar operaciones relacionadas con el servidor web.

Lo siguiente es declarar los handlers para las URI raíz, post_time, get_data y saveShutdown, asignando a cada tipo de solicitud (POST o GET) una función a ejecutar.

Y finalmente, en la función start_webserver() se inicia el servidor web y se registran los handlers URI configurados previamente:

- **httpd_config_t httpd_config** = HTTPD_DEFAULT_CONFIG(): Se crea una estructura de configuración predeterminada para el servidor HTTP.
- **httpd_config.stack_size**: Se configura el tamaño de la pila para las tareas del servidor web.
- **httpd_start(&httpd_server, &httpd_config)**: Inicia el servidor web con la configuración establecida.
- **httpd_register_uri_handler**: Registra cada URI handler.

Posteriormente se desarrolla cada función asignada a los URI handlers. Para el URI raíz, la función asignada es root_get_handler, en la cual se accede al archivo binario de la platilla HTML y se devuelve (mediante httpd_resp_send) como respuesta a la solicitud GET de dicho URI.

Para el URI /post_time, la función asignada es time_post_handler, que manejará las solicitudes POST que surjan al presionar el botón de inicio desde la interfaz del servidor, es decir, pondrá en funcionamiento al medidor. Aquí se producirán dos sucesos: por un lado, de la solicitud se obtiene el "epoch" (una fecha y hora fijas que se utilizan como referencia a partir de las cuales una computadora mide el tiempo del sistema) del computador del cliente (mediante httpd_req_recv) a fin de sincronizar la hora y fecha del ESP32 (mediante la función set_time de "date_time.c") que luego se utilizará como uno de los datos a guardar sobre las mediciones; y por otro lado se ejecutarán las funciones create_tasks y create_data_tasks que se encargarán respectivamente de la creación de las diferentes tareas para mediciones y control del dispositivo medidor, y de una tarea vTaskUpdateData que se encargará de recolectar los datos de cada medición dentro de una estructura ("measurements") para su posterior visualización y guardado.

Para el URI /get_data, la función asignada es get_data_handler. Está función recibirá solicitudes del tipo GET, para la cual creará y devolverá un vector JSON con los distintos datos almacenados en la estructura de mediciones, a fin de visualizar dichos datos en una tabla desde la interfaz del servidor.

La función asignada al último URI, /saveShutdown, es save_shutodwn_handler. Esta será la encargada de finalizar el proceso del dispositivo medidor, eliminando las tareas creadas inicialmente (mediante delete_tasks y delete_data_task) y restaurando la estructura de mediciones (mediante resetMeasurements). Además, se encargará de generar un archivo de texto con los datos contenidos en dicha estructura, previo a la restauración de los datos, el cual será devuelto como respuesta hacia el cliente de manera sea capaz de descargar los datos de las mediciones en su computador.

Por último, como bien se nombró anteriormente, la tarea vTaskUpdateData será la encargada de actualizar periódicamente los datos de la estructura de mediciones. Para lograr el proceso descripto, esta tarea se sincroniza con las tareas encargadas de las mediciones en cuestión mediante el uso de semáforos. Dentro del bucle principal de la

tarea, se liberará un semáforo para cada tarea, las cuales al recibirlo realizarán la medición correspondiente y enviarán el dato hacia vTaskUpdateData mediante el uso de colas (queues), para luego ser almacenados dentro de la estructura de mediciones.

Plantilla HTML

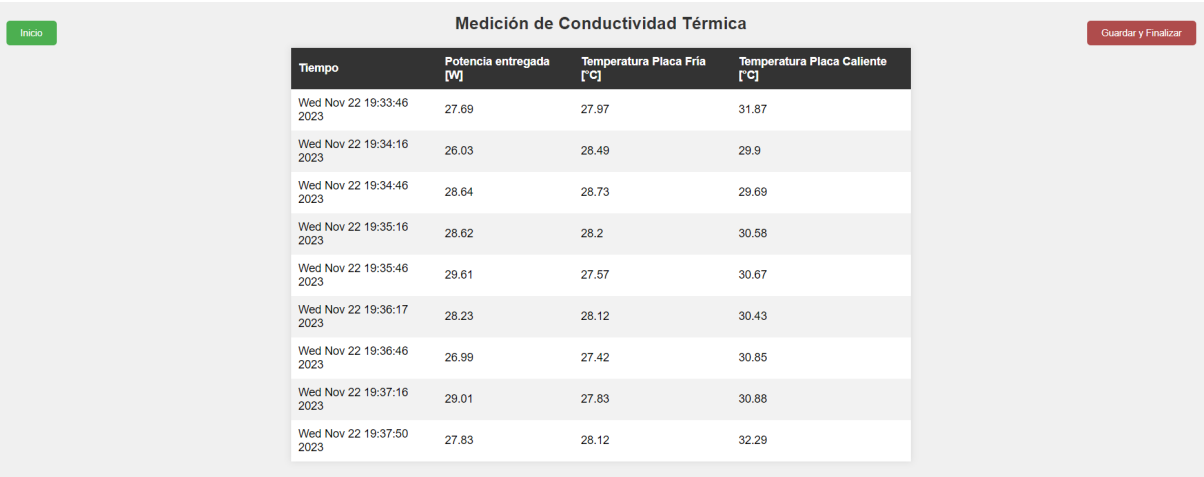
Como se detalló anteriormente, esta plantilla HTML conformará la interfaz de comunicación entre la máquina y el usuario; y estará constituida por un botón de inicio, un botón de finalización y guardado, y una tabla donde se presentarán los datos de las mediciones.

El código correspondiente a la plantilla en el archivo "view.html". En este se encuentran definidos los dos botones donde se le asigna el URI /post_time al botón de inicio, que a su vez estará enlazado con una función que enviará el “epoch” del cliente hacia el servidor HTTP mediante una solicitud POST.

El botón de finalización y guardado, asociado al URI /saveShutdown, realizará una solicitud del tipo GET a fin de obtener, por parte del servidor, el archivo de texto con los datos de las mediciones y generar un enlace temporal de descarga del mismo.

Finalmente, la función restante, asociada al URI /get_data, se corresponde a aquella que generará la tabla con los datos de las mediciones, que serán obtenidos mediante solicitudes de tipo GET.

En la Figura 15 se puede observar una captura de la interfaz de usuario del servidor HTTP implementado, donde se encuentran tanto los botones de “Inicio” y de “Guardar y Finalizar”, como las mediciones realizadas periódicamente, presentadas en una tabla para una mayor comprensión por parte del usuario.



Tiempo	Potencia entregada [W]	Temperatura Placa Fría [°C]	Temperatura Placa Caliente [°C]
Wed Nov 22 19:33:46 2023	27.69	27.97	31.87
Wed Nov 22 19:34:16 2023	26.03	28.49	29.9
Wed Nov 22 19:34:46 2023	28.64	28.73	29.69
Wed Nov 22 19:35:16 2023	28.62	28.2	30.58
Wed Nov 22 19:35:46 2023	29.61	27.57	30.67
Wed Nov 22 19:36:17 2023	28.23	28.12	30.43
Wed Nov 22 19:36:46 2023	26.99	27.42	30.85
Wed Nov 22 19:37:16 2023	29.01	27.83	30.88
Wed Nov 22 19:37:50 2023	27.83	28.12	32.29

Figura 15: Servidor html con datos de ejemplo.

Fecha y hora en el microcontrolador

La configuración y programación de la fecha y hora en el microcontrolador se encuentra realizada en los archivos "date_time.c" y "date_time.h". En los mismos, se configura el reloj de hardware con la zona horaria correspondiente, y se programan funciones para sincronizar el reloj con un tiempo EPOCH dado y una función que devuelva la fecha y hora. Además, se programa una tarea que muestra en consola la fecha y hora cada un segundo.

Este programa es de gran importancia, debido a que se necesita grabar en cada medición la hora exacta. Si bien se podría utilizar un módulo de reloj RTC con pila externa, se decidió aprovechar del servidor web implementado para sincronizar la fecha y hora antes de comenzar con un ensayo, obteniendo la misma del dispositivo que se conecta al microcontrolador.

Configuración de "SDKCONFIG"

Como se mencionó, en este proyecto se utiliza el *framework ESP-IDF*, en el cual para el correcto funcionamiento deben configurarse parámetros internos del ESP-32.

- Flash memory, 4 MB.
- configTICK_RATE_HZ, 1000.
- CPU frequency, 240 MHz.
- Max HTTP Request Header Length, 8192.
- High-resolution timer task stack size, 5500.
- Interrupt level, 1.

CONCLUSIÓN

- Se construyó un prototipo funcional de un medidor de conductividad térmica, con una interfaz web, la cual le permite al usuario tener control de inicio y finalización del proceso de medición, así como también provee la visualización de los datos necesarios para calcular la conductividad térmica, con fecha y hora. Al finalizar la medición se da la posibilidad de descargar un archivo con los datos obtenidos.
- Se lograron comprender y aplicar los conocimientos adquiridos en las cátedras Técnicas Digitales III y Medidas Electrónicas II.

REFERENCIAS

BetaTHERM. (Octubre de 2020). NTC Thermistor theory . Dinamarca & Noruega.

Díaz, M., & Previotto, S. (2022). *Control de potencia mediante TRIAC.* San Francisco.

Dominguez Vega, C., & Culzoni, C. (2008). *Primer informe de desarrollo experimental de equipo de placa caliente para medición de conductividad térmica.* Rafaela.

Espressif Systems. (2023). *ESP - IDF Programming Guide.* Obtenido de https://docs.espressif.com/projects/esp-idf/en/latest/esp32/about.html

Sandoval, A., Espinosa, E., & Barahona, J. (2007). *Celdas Peltier: Una alternativa para sistemas de enfriamiento con base.* Huajuapan de León. Obtenido de https://d1wqtxts1xzle7.cloudfront.net/53429800/PLACA_PELTIER-libre.pdf?1496902184=&response-content-disposition=inline%3B+filename%3DCeldas_Peltier_Una_alternativa_para_sist.pdf&Expires=1700690395&Signature=V09tvGBjWvZQWfUhpxiKnmLTGg7745~br-6~FBHlk0vTA5r2