
GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

# Calibrador de termómetros sin contacto

Autor: Gonzalo Díe Morales

Director: José Daniel Muñoz Frías

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Calibrador de termómetros sin contacto

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2023/24 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Gonzalo Díe Morales Fecha: 12/ 06/ 2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por MUÑOZ FRIAS JOSE DANIEL - ***7384** el día 11/07/2024 con un certificado emitido por AC FNMT Usuarios

Fdo.: José Daniel Muñoz Frías Fecha: 11/07/2024


GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

# Calibrador de termómetros sin contacto

Autor: Gonzalo Díe Morales

Director: José Daniel Muñoz Frías

Madrid

# Agradecimientos

A mi tutor, José Daniel, por acompañarme durante todo este proceso, a mis padres, por apoyar siempre mis estudios y a mis amigos tanto de fuera como de la universidad.

# CALIBRADOR DE TERMÓMETROS SIN CONTACTO

**Autor: Gonzalo Díe Morales.**
Director: José Daniel Muñoz Frías.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

## RESUMEN DEL PROYECTO

El proyecto consiste en un dispositivo formado por 3 placas, las cuales se ponen a temperaturas específicas de 36°C, 38°C y 40°C. De esta manera, los termómetros infrarrojos sin contacto se pueden verificar tomando la temperatura de las placas y viendo si son correctas en el termómetro.

**Palabras clave**: Temperatura, sensor, placa PCB, control

### 1. Introducción

La precisión en la medición de la temperatura es crítica en entornos médicos, donde los termómetros sin contacto son cada vez más utilizados por su comodidad y rapidez. Sin embargo, su fiabilidad ha sido cuestionada debido a la variabilidad en las mediciones. Este proyecto tiene como objetivo diseñar un dispositivo verificador que asegure la exactitud de estos termómetros, proporcionando una herramienta confiable para el personal sanitario.

### 2. Definición del Proyecto

El proyecto consiste en un verificador para termómetros de infrarrojos sin contacto. Este verificador utiliza tres placas de resistencia que se mantienen a temperaturas fijas de 36°C, 38°C y 40°C. Encima de cada placa hay 2 leds, uno rojo y uno verde, mientras la placa no esté a la temperatura que le corresponde estará en rojo, una vez llegue a la temperatura establecida se encenderán el led verde y ya se podrá usar. Cada placa está controlada por un sensor de temperatura TMP117 estos sensores tienen una sensibilidad de 0.1 grados por lo que son muy precisos. El dispositivo permite al usuario comprobar la precisión de los termómetros al compararlos con las temperaturas conocidas de las placas, asegurando así que los termómetros proporcionen lecturas precisas y consistentes. El objetivo es que sea un aparato muy cómodo, es pequeño y fácil de colocar y guardar, y también muy fácil y rápido de usar, facilitando el trabajo del personal sanitario.

### 3. Descripción del modelo/sistema/herramienta

El prototipo del calibrador está compuesto por las 3 placas de resistencia que se ponen a las temperaturas especificadas, 2 leds por cada placa, uno rojo y uno verde que indican cuando está listo para ser utilizado, 3 sensores de temperatura (TMP117), son sensores de altísima precisión que se encargan de medir la temperatura de las resistencias y el microcontrolador dsPIC33FJ32MC202 que es el que lleva toda la lógica del sistema.

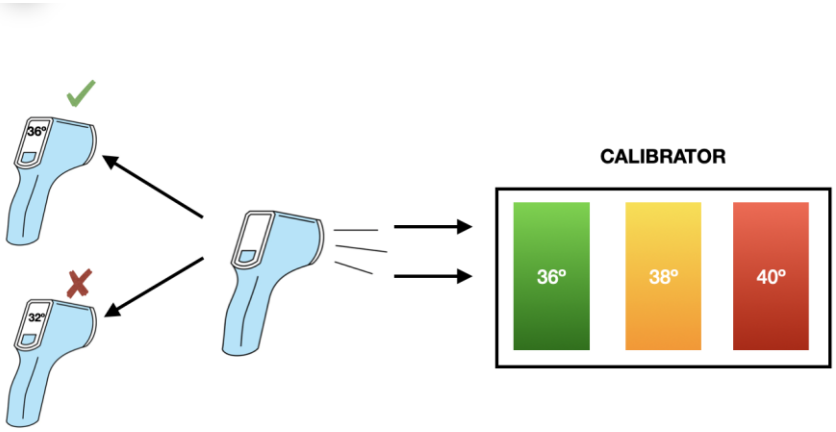



*Ilustración 1 - Esquema del funcionamiento del calibrador*

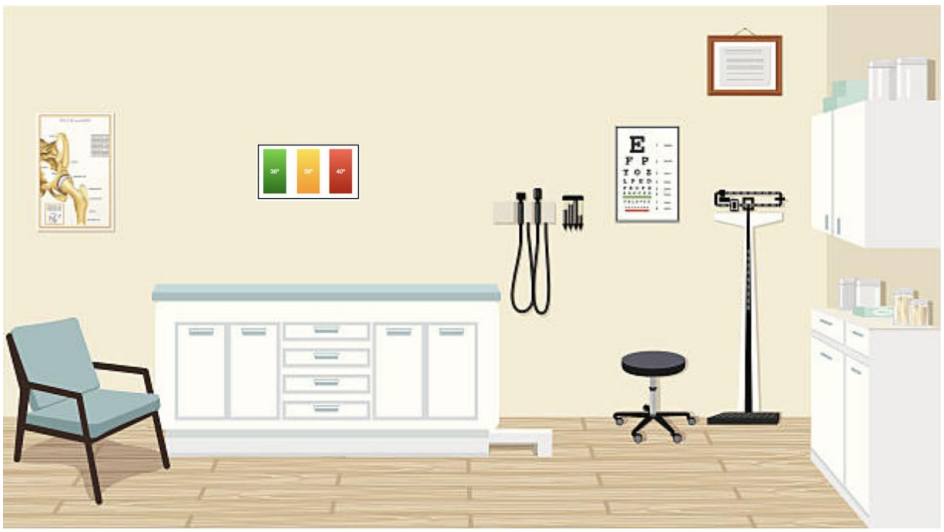


*Ilustración 2 – Utilidad y comodidad en una consulta*

## 4. Resultados

El resultado del proyecto es un prototipo funcional, que es capaz de controlar la temperatura de una placa y que se mantenga constante. Consta de un microcontrolador, un sensor, una resistencia y 2 leds.

## 5. Conclusiones

Hemos conseguido diseñar una primera versión de un controlador de temperatura para termómetros a distancia muy sencillo y fácil de usar, de momento consta de solo una placa que cuando se enciende hay que esperar un par de minutos para que ponga la placa

a la temperatura adecuada y cuando el led esta en verde ya se puede usar. Es una herramienta con un funcionamiento muy intuitivo y también pequeño y portátil por lo que se han logrado los objetivos iniciales del proyecto.

# CONTACTLESS THERMOMETER CALIBRATOR

**Author:** Gonzalo Díe Morales.
Supervisor: José Daniel Muñoz Frías.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

## ABSTRACT

The project consists of a device made up of 3 plates, which are set to specific temperatures of 36°C, 38°C, and 40°C. In this way, non-contact infrared thermometers can be calibrated by taking the temperature of the plates and verifying if the readings on the thermometer are accurate

**Keywords**: Temperature, sensor, PCB plate, control

### 1. Introduction

Accuracy in temperature measurement is critical in medical settings, where non-contact thermometers are increasingly used for their convenience and speed. However, their reliability has been questioned due to variability in measurements. This project aims to design a verification device that ensures the accuracy of these thermometers, providing a reliable tool for healthcare personnel.

### 2. Project definition

The project consists of a calibrator for non-contact infrared thermometers. This calibrator uses three resistance plates maintained at fixed temperatures of 36°C, 38°C, and 40°C. Each plate has two LEDs, one red and one green. While the plate is not at its designated temperature, the red LED will be on; once it reaches the set temperature, the green LED will light up, indicating that it is ready for use. Each plate is controlled by a TMP117 High-Precision Digital Temperature Sensor. These sensors have a sensitivity of 0.1 degrees, making them highly accurate. The device allows the user to check the precision of thermometers by comparing them with the known temperatures of the plates, thus ensuring that the thermometers provide accurate and consistent readings. The goal is to create a very user-friendly device that is small, easy to set up and store, and quick and easy to use, thereby facilitating the work of healthcare personnel.

### 3. Model description

The calibrator prototype is composed of three resistance plates set to the specified temperatures, two LEDs per plate (one red and one green) indicating when the plate is ready to be used, three temperature sensors (TMP117 High-Precision Digital Temperature Sensor), which are highly precise sensors responsible for measuring the temperature of the resistances, and the microcontroller dsPIC33FJ32MC202, which handles all the system logic.

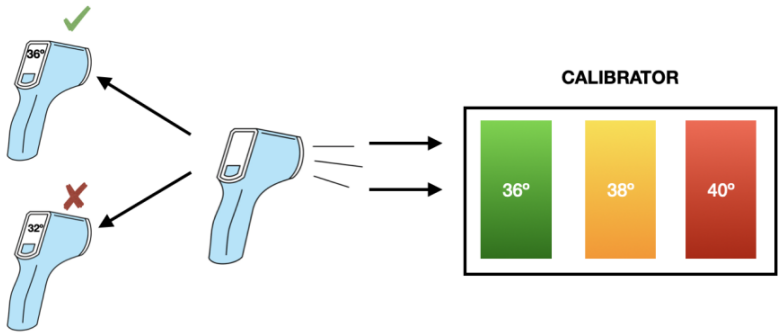



*Figure 3 – Scheme of how the calibrator works.*

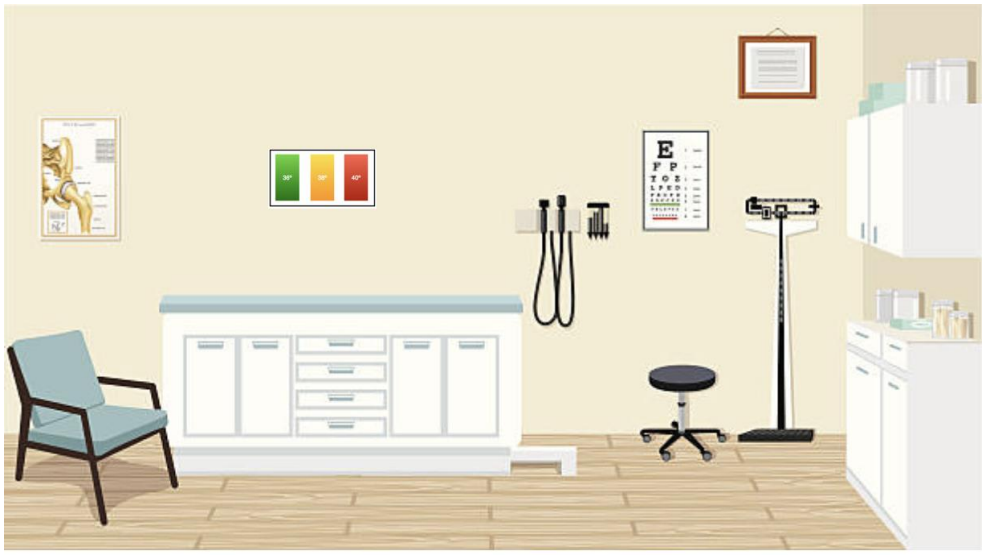


*Figure 4 – Utility and commodity in a doctors office*

## 4. Results

The result of the project is a functional prototype that can control the temperature of a plate and keep the temperature constant. It consists of a microcontroller, a sensor, a resistor, and 2 LEDs.

## 5. Conclusions

We have managed to design a first version of a temperature controller for remote thermometers that is very simple and easy to use. For now, it consists of just one heat plate, which, when turned on, needs a couple of minutes to reach the appropriate temperature. When the LED is green, it is ready for use. It is a very intuitive tool, as well as small and portable, thus achieving the intended objectives.

# *Índice de la memoria*

# *Índice de figuras*

# Capítulo 1. INTRODUCCIÓN

.

## *1.1 CONTEXTO*

En el mundo de la medicina, el bienestar de los pacientes es vital y, por ello, la precisión en todos los aspectos es fundamental. Los termómetros sin contacto son cada vez más usados por su comodidad y rapidez; sin embargo, estos termómetros se consideran poco precisos, lo que representa un problema que hay que resolver, ya que la exactitud de la medición es fundamental, pues las decisiones y diagnósticos dependen de ella.

Para intentar solucionar el problema de la inexactitud de los termómetros, se ha realizado un proyecto que consiste en diseñar un verificador de termómetros sin contacto, con el objetivo de hacer que se pueda confiar en estos termómetros y que proporcionen medidas precisas y consistentes.

El verificador es un dispositivo que busca hacerle la vida más sencilla al personal sanitario, siendo este un dispositivo compacto y fácil de usar. Será un dispositivo que contará con 3 placas que se pondrán a 36, 38 y 40 grados, cada una con un LED que muestre si está a la temperatura deseada, para que se pueda probar el termómetro con 3 temperaturas distintas: la estándar, una por debajo y una por encima. Tanto médicos como enfermeras necesitan herramientas que hagan su trabajo más sencillo, por lo que este dispositivo les será de gran utilidad.

La medición de la temperatura es extremadamente importante en entornos hospitalarios, ya que la detección de fiebre o cambios de temperatura es fundamental para los diagnósticos. Por esto, el problema de la inexactitud en los termómetros es especialmente crítico.

Se ha creado un dispositivo que ayuda a médicos y enfermeros a verificar temperaturas de una manera más sencilla y confiable, logrando así una mejor atención al paciente.

## *1.2 PROBLEMAS PARA RESOLVER*

Falta de precisión en termómetros sin contacto: Uno de los principales problemas de la actualidad en relación con la medición de las temperaturas es la falta de exactitud de los termómetros sin contacto. Un error en la medición puede hacer que ocurran fallos en los diagnósticos y esto afectaría negativamente a la atención y el tratamiento de los pacientes.

Variabilidad en las mediciones: Los termómetros sin contacto pueden proporcionar resultados inconsistentes debido a factores como la distancia entre el dispositivo y el paciente, las condiciones ambientales y la calidad del termómetro en sí. Esta variabilidad en las mediciones dificulta la detección confiable de fiebre o cambios sutiles en la temperatura corporal.

Falta de estándares de precisión: La falta de estándares uniformes en la industria de termómetros sin contacto complica aún más la garantía de mediciones precisas. La ausencia de normativas específicas puede dar lugar a una amplia variedad de dispositivos con diferentes niveles de precisión, lo que dificulta la selección de un termómetro confiable.

Confianza del personal médico: Los médicos y enfermeros se basan mucho en la temperatura corporal para realizar diagnósticos por lo que usar dispositivos poco fiables puede hacer que los pacientes pierdan confianza en las decisiones del personal sanitario.

Costos asociados a equipos ineficientes: Usar dispositivos que dan medidas exactas conlleva al uso ineficiente de estos ya que al no ser correcta la medición habría que repetir las pruebas o se tomarían medidas que no proceden malgastando tiempo y recursos.

Efectos en la toma de decisiones de tratamiento: La falta de precisión en la medición de la temperatura puede afectar la elección de tratamientos y medicamentos. Un diagnóstico erróneo debido a mediciones imprecisas puede resultar en un tratamiento inadecuado o en la falta de tratamiento necesario.

## *1.3 MOTIVACIÓN*

El principal motivo por el que se está llevando a cabo este proyecto es por la necesidad de resolver el problema de la falta de precisión en los termómetros sin contacto que se usan en clínicas y hospitales.

En primer lugar, mejorar la precisión en las mediciones reduciría el número de errores médicos y esto haría que los diagnósticos erróneos fueran más fáciles de evitar. Introducir un verificador de termómetros haría que los médicos confiasen en ellos y este proyecto busca proporcionar un medio para restaurar y fortalecer esta confianza. Otro motivo es por la optimización de los recursos médicos, usar termómetros que pueden hacer medidas inexactas puede llevar a diagnósticos erróneos y por ende a la realización de pruebas o procedimientos que no proceden malgastando recursos sanitarios.

Por último, este proyecto se enmarca en una visión más amplia de mejorar la atención al paciente, ya que la precisión en la medición de la temperatura es esencial para el diagnóstico y el tratamiento adecuados de diversas afecciones médicas, lo que tiene un impacto directo en el bienestar de los pacientes.

# Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para diseñar el dispositivo se han necesitado una serie de componentes que cumpliesen con ciertas especificaciones para conseguir los resultados esperados. Para gestionar toda la lógica se ha usado el microcontrolador dsPIC33FJ32MC202, fabricado por Microchip Technology[1], hemos elegido este ya que este dispositivo es particularmente apto para aplicaciones que requieren un alto rendimiento de procesamiento y características específicas para manejo de controladores y operaciones en tiempo real. Que es justo lo que buscamos, que mida las temperaturas constantemente y mande todas las señales necesarias según se van produciendo eventos, para mantener la placa a una temperatura constante. La resistencia que hemos elegido para conseguir las temperaturas adecuadas es de 10 Ohmios. El sensor de temperatura que tiene que medir constantemente la temperatura de la resistencia es el TMP117 High-Precision Digital Temperature Sensor[2], hemos elegido este ya que tiene una sensibilidad de 0.1 grados y comunicación i2c que es muy cómoda para integrarlo con el microcontrolador y necesitábamos un sensor de alta precisión, ya que este dispositivo esta pensado para la medicina donde la precisión es crucial. Los mosfets los necesitamos para controlar las resistencias que van a calentar las placas de temperatura, mediante una señal PWM. Como el MOS conmuta a alta frecuencia (PWM) y el micro da poca corriente, el MOS tarda bastante en conmutar porque hay que cargar la capacidad que hay entre puerta y fuente y se producen muchas pérdidas, entonces hemos tenido que usar el driver TD310[3] que concuerda con nuestras especificaciones.

Una de las partes clave del dispositivo es la comunicación entre el micro y los sensores, ya que estos tienen que medir la temperatura de las resistencias con extrema precisión y pasarle

[1] Empowering Innovation | Microchip Technology
[2] TMP117 High-Precision Digital Temperature Sensor - TI | Mouser
[3] td310-1852141.pdf (mouser.es)

la información al microcontrolador. Por eso hemos usado la comunicación i2c, con el SDA para transmitir los datos y el SCL como reloj para sincronizar

# Capítulo 3. ESTADO DE LA CUESTIÓN

En el ámbito de los termómetros clínicos infrarrojos, existen proyectos y soluciones tecnológicas que abordan desafíos similares a los que se plantean en nuestro proyecto de verificador de termómetros clínicos. Un informe destacado sobre el termómetro Caregiver es el "Termómetro Infrarrojo sin Contacto de Grado Clínico para Cuidadores", que se enfoca en la calibración y certificación de termómetros infrarrojos en un entorno clínico, cumpliendo con el estándar ASTM E1965-1998 [Calibration for Clinical Institutions, 2014].

Este informe pone de manifiesto la importancia de la calibración y la certificación en fábrica como un factor crucial para garantizar la precisión y la confiabilidad en la medición de temperatura [Calibration for Clinical Institutions, 2014]. Además, destaca el uso de sensores infrarrojos personalizados que están herméticamente sellados y carecen de partes móviles, lo que puede influir positivamente en la precisión de las mediciones.

En términos de verificación de calibración y recalibración, los fabricadores del termómetro resaltan la necesidad de realizar verificaciones periódicas y proporcionan métodos para llevar a cabo estas tareas. Las opciones que ofrecen son devolver el termómetro y lo calibran en sus fábricas o Verificar la calibración utilizando un Blackbody sumergido en un baño de agua agitada y comparar la temperatura con un termómetro de contacto de referencia trazable pero no ofrecen ningún dispositivo para verificar si funcionan bien [Calibration for Clinical Institutions, 2014].

Otra fuente de referencia valiosa es la guía "Infrared Thermometer Calibration -- A Complete Guide" (Guía de Calibración de Termómetros Infrarrojos: Una Guía Completa) [Infrared Thermometer Calibration – A Complete Guide, 2012].

Esta guía proporciona un conjunto completo de pasos y recomendaciones para llevar a cabo calibraciones precisas de termómetros infrarrojos. Se basa en estándares reconocidos, como ASTM E2847, "Standard Practice for Calibration and Accuracy Verification of Wideband

Infrared Thermometers" (Práctica Estándar para la Calibración y Verificación de Precisión de Termómetros Infrarrojos de Banda Ancha) [Infrared Thermometer Calibration – A Complete Guide, 2012]. Resalta las fuentes de incertidumbre que contribuyen significativamente a la calibración de termómetros infrarrojos, incluyendo:

- Estimación de la emisividad de la fuente de calibración.
- Campo de visión del termómetro infrarrojo.
- Gradientes de temperatura en la fuente de radiación.
- Alineación inadecuada del termómetro infrarrojo.
- Temperatura de calibración de la fuente de radiación.
- Temperatura ambiente.
- Temperatura reflejada.

La guía también identifica el equipo de calibración necesario, como la fuente de radiación térmica, el estándar de transferencia, el termómetro de temperatura ambiente, el dispositivo de montaje y el dispositivo de medición de distancia [Infrared Thermometer Calibration – A Complete Guide, 2012].

Uno de los puntos clave resaltados en esta guía es la importancia de la trazabilidad en la calibración. Proporciona información sobre la determinación de la verdadera temperatura de la fuente de calibración, clasificando dos esquemas de trazabilidad: Esquema I y Esquema II. Estos esquemas diferencian cómo se determina la temperatura real de la fuente de calibración, y se destacan las incertidumbres asociadas [Infrared Thermometer Calibration – A Complete Guide, 2012].

La guía enfatiza la necesidad de una configuración de laboratorio adecuada para minimizar incertidumbres y errores durante la calibración. La temperatura ambiental en el laboratorio debe mantenerse dentro de límites razonables y debe registrarse en el informe de calibración. Se aborda la importancia de la posición de los equipos y cómo evitar problemas relacionados con la temperatura reflejada y el flujo de aire en el laboratorio [Infrared Thermometer Calibration – A Complete Guide, 2012].

La guía detalla la preparación y los procedimientos para la calibración, destacando la necesidad de permitir que el termómetro infrarrojo alcance la temperatura del laboratorio antes de la calibración. También aborda la elección de los puntos de calibración y cómo realizar las mediciones de manera eficiente [Infrared Thermometer Calibration – A Complete Guide, 2012].

Aunque esta guía se centra en la calibración de termómetros infrarrojos, su enfoque en la trazabilidad, las fuentes de incertidumbre y las mejores prácticas de calibración ofrece una perspectiva valiosa para nuestro proyecto de verificador de termómetros clínicos infrarrojos.

# Capítulo 4. DEFINICIÓN DEL TRABAJO

## *4.1 JUSTIFICACIÓN*

A partir del análisis de los trabajos previos y del contexto establecido en capítulos anteriores, se puede deducir la necesidad de llevar este proyecto a cabo. Mientras que en la introducción se mencionó la importancia general de las soluciones e-health en la sociedad, la justificación de este proyecto se basa en argumentos técnicos y en un análisis de mercado específico. Este enfoque nos permite ver claramente por qué este verificador de termómetros de infrarrojos sin contacto es una innovación necesaria y valiosa.

### 4.1.1 NECESIDAD DE FIABILIDAD Y PRECISIÓN EN MEDICIÓN DE TEMPERATURA

En el entorno médico, la medición precisa de la temperatura es crucial para el diagnóstico y tratamiento de diversas afecciones. Los termómetros sin contacto son cada vez más utilizados debido a su comodidad y rapidez, especialmente en situaciones donde el contacto físico debe minimizarse por razones de higiene o para evitar infecciones cruzadas, como ocurrió con el covid, con los termotros tradicionales los médicos se arriesgaban en exceso al manejar estos instrumentos que habían estado en contacto con posibles infectados. Sin embargo, la fiabilidad de estos dispositivos ha sido cuestionada debido a la variabilidad en las mediciones. Factores como la distancia del termómetro al paciente, las condiciones ambientales y la calidad del termómetro pueden afectar significativamente la precisión de las lecturas. En este contexto, la necesidad de un dispositivo calibrador fiable y preciso se vuelve evidente, para que el usuario pueda estar seguro de la medición del termometro.

### 4.1.2 VACÍO EN EL MERCADO

Actualmente, el mercado carece de dispositivos accesibles y eficientes para la verificación y calibración de termómetros de infrarrojos sin contacto. Aunque existen soluciones de calibración en fábrica y guías completas para la calibración de termómetros, no hay dispositivos prácticos y fáciles de usar que permitan a los profesionales de la salud verificar la precisión de sus termómetros de manera rápida y eficaz en el lugar de uso. Este proyecto busca llenar ese vacío con un calibrador que utiliza tecnología avanzada para ofrecer resultados confiables y consistentes.

### 4.1.3 INNOVACIONES TÉCNICAS Y BENEFICIOS DEL DISPOSITIVO

- Sensores de Alta Precisión: El uso de sensores TMP117, que tienen una sensibilidad de 0.1 grados, asegura una precisión excepcional en la medición de la temperatura. Esta alta precisión es fundamental para la confiabilidad del dispositivo y para garantizar que los termómetros sin contacto puedan calibrarse con exactitud.

- Indicadores LED para Facilidad de Uso: Cada placa de resistencia está equipada con dos LEDs, uno rojo y uno verde. Este diseño intuitivo permite a los usuarios saber inmediatamente cuándo la placa ha alcanzado la temperatura deseada (cuando el LED verde se enciende), facilitando el proceso de calibración sin necesidad de verificaciones adicionales.

- Controlador Avanzado: El microcontrolador dsPIC33FJ32MC202 gestiona toda la lógica del sistema, asegurando un control preciso de la temperatura y la estabilidad de la temperatura en las placas de resistencia. Este microcontrolador es conocido por su capacidad de manejar operaciones en tiempo real, lo que es crucial para la funcionalidad del verificador.

- Diseño Compacto y Portátil: El dispositivo ha sido diseñado para ser pequeño y fácil de almacenar, lo que permite a los profesionales de la salud llevarlo y utilizarlo en

diferentes entornos sin complicaciones. Su portabilidad es una ventaja significativa en situaciones de emergencia o en lugares con espacio limitado.

### 4.1.4 IMPACTO ECONÓMICO Y OPERACIONAL

El uso de un calibrador preciso y confiable tiene un impacto económico y operacional positivo. Al asegurar que los termómetros sin contacto proporcionen lecturas precisas, se reducen los errores de diagnóstico y los costos asociados a pruebas adicionales y tratamientos incorrectos. Este ahorro de tiempo y recursos es especialmente valioso en entornos hospitalarios, donde la eficiencia operativa es muy importante.

### 4.1.5 POTENCIAL DE MERCADO Y ADOPCIÓN

El dispositivo está diseñado para ser asequible y accesible, lo que facilita su adopción en una amplia gama de entornos médicos, desde grandes hospitales hasta pequeñas clínicas. La combinación de alta precisión, facilidad de uso y costo-efectividad lo hace una inversión atractiva tanto para instituciones de salud como para proveedores individuales de servicios médicos.

## *4.2 OBJETIVOS*

Asegurar la confiabilidad de los termómetros sin contacto: El objetivo principal es asegurar la fiabilidad de los termómetros sin contacto para que el personal médico pueda usarlos sin tener dudas de su exactitud. Buscaremos alcanzar este objetivo creando un verificador de termómetros para comprobar su correcto funcionamiento.

Facilitar la verificación de termómetros para el personal médico: Este proyecto busca facilitar a los médicos y enfermeros la comprobación del correcto funcionamiento de los termómetros sin contacto. Esto hará que los médicos puedan hacer una comprobación rápida y eficaz para asegurarse de que el termómetro funciona bien.

Desarrollar un prototipo funcional y fácil de usar: Otro de los principales objetivos es hacer que el dispositivo sea muy cómodo de usar, para que los médicos no pierdan mucho tiempo verificando los termómetros a distancia y pasen a ser instrumentos prioritarios.

Preparar el terreno para futuras mejoras y aplicaciones: Un objetivo que esta más en segundo plano es diseñar un prototipo lo suficientemente bueno como para que en un futuro se le puedan aplicar cambios y mejoras para ofrecer un instrumento más accesible y cómodo para los hospitales y clínicas.

Contribuir a la resolución de problemas hospitalarios: El proyecto busca facilitar la medición precisa de la temperatura corporal. Esto no solo mejorará la atención al paciente, sino que también simplificará las tareas diarias del personal médico y reducirá costos innecesarios.

## *4.3 METODOLOGÍA*

Para llevar a cabo el proyecto hemos seguido un plan de trabajo dividido en 6 fases o etapas, en las que en cada una nos hemos enfocado en cosas distintas del trabajo para poder finalmente juntarlo todo y conseguir el prototipo funcional.

**Fase 1: Investigación y Definición de Requisitos**

Objetivo:

Definir claramente los requisitos técnicos y funcionales del calibrador, basados en un análisis de las necesidades del mercado, las limitaciones de los componentes y las tecnologías disponbles.

Actividades:

- Revisión de literatura y estado del arte para entender las tecnologías existentes y las deficiencias que nuestro proyecto pretende resolver.
- Definición de los problemas que nuestro dispositivo pretende resolver.

- Realización de los objetivos del proyecto y las motivaciones.

**Fase 2: Diseño del sistema de los sensores de temperatura**

Objetivo:

Desarrollar el sistema del sensor de temperatura para que este midiendo constantemente.

Actividades:

- Selección de componentes electrónicos adecuados, el microcontrolador dsPIC33FJ32MC202 y el sensor TMP117, que lo más importante a la hora de elegir el sensor era que tuviesen 0.1 grados de sensibilidad y comunicación i2c.
- Diseño del circuito electrónico utilizando KiCad y diseño del software para implementar la comunicación usando MPLAB.
- Fabricación y ensamblaje de la placa de circuito impreso (PCB).
- Soldadura de componentes electrónicos en la PCB.

**Fase 3: Desarrollo del sistema global**

Objetivo:

Desarrollar el sistema completo con toda su lógica e implementar el sensor de temperatura

Actividades:

- Seleccionar los componentes que nos faltaban, las resistenciasde potencia, los MOS, el driver para reducir las pérdidas de los mosfets, los leds y las otras resistencias menores.
- Diseño del circuito electrónico utilizando KiCad y diseño del software para implementar la comunicación usando MPLAB.
- Fabricación y ensamblaje de la placa de circuito impreso (PCB).

- Soldadura de componentes electrónicos en la PCB.

**Fase 4: Desarrollo de Hardware**

Objetivo:

Juntar todo el hardware para ya tener el dispositivo listo.

Actividades:

- Diseñar y planificar como vamos a integrar todos los componentes.
- Juntar todos los componentes.

**Fase 5: Pruebas y Validación**

Objetivo:

Asegurar que el calibrador funciona según lo previsto y cumple con los requisitos de precisión y fiabilidad.

Actividades:

- Realización de pruebas para verificar la precisión de las mediciones de temperatura.
- Comparación de las lecturas del calibrador con termómetros de referencia calibrados.
- Ajustes y calibración fina del dispositivo para asegurar la consistencia en las mediciones.
- Documentación de los resultados de las pruebas.

**Fase 6: Documentación y Presentación**

Objetivo:

Documentar todo el proceso de desarrollo y preparar la presentación final del proyecto.

Actividades:

- Compilación de toda la documentación técnica y de diseño en un informe final.
- Preparación de presentaciones y demostraciones del dispositivo.

## *4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA*

Para asegurar un desarrollo eficiente y estructurado del proyecto, hemos elaborado un diagrama de Gantt que detalla la planificación temporal de las actividades que hemos realizado. Este diagrama proporciona una visión clara de las fases del proyecto, las tareas específicas dentro de cada fase y la duración de cada tarea. A continuación, se presenta el diagrama de Gant con cada fase.

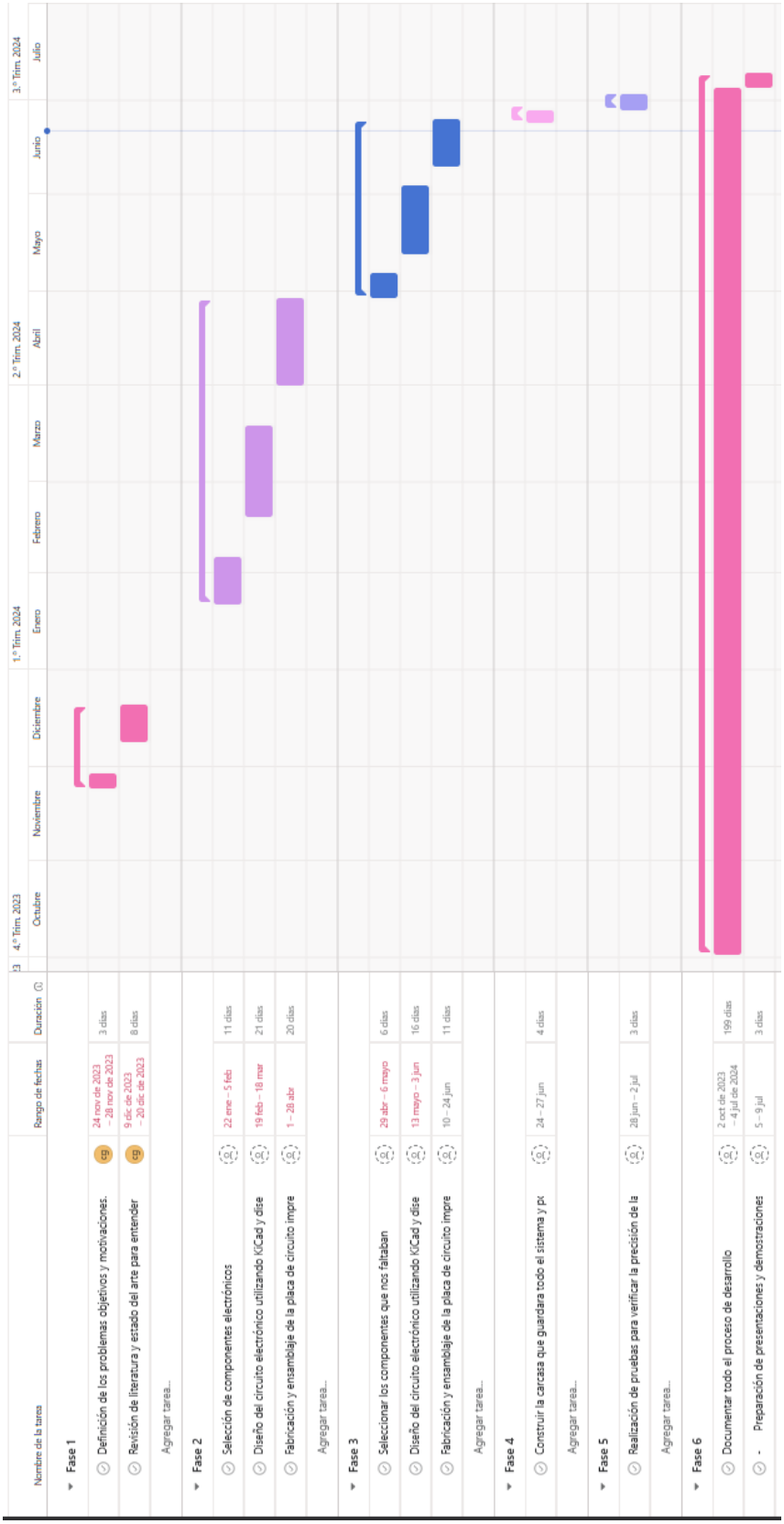

Nombre de la tarea
Rango de fechas
Duración
4.º Trim. 2023
1.º Trim. 2024
2.º Trim. 2024
3.º Trim. 2024
Octubre
Noviembre
Diciembre
Enero
Febrero
Marzo
Abril
Mayo
Junio
Julio
Fase 1
Definición de los problemas objetivos y motivaciones.
24 nov de 2023 – 28 nov de 2023
3 días
Revisión de literatura y estado del arte para entender
9 dic de 2023 – 20 dic de 2023
8 días
Agregar tarea...
Fase 2
Selección de componentes electrónicos
22 ene – 5 feb
11 días
Diseño del circuito electrónico utilizando KiCad y dise
19 feb – 18 mar
21 días
Fabricación y ensamblaje de la placa de circuito impre
1 – 28 abr
20 días
Agregar tarea...
Fase 3
Seleccionar los componentes que nos faltaban
29 abr – 6 mayo
6 días
Diseño del circuito electrónico utilizando KiCad y dise
13 mayo – 3 jun
16 días
Fabricación y ensamblaje de la placa de circuito impre
10 – 24 jun
11 días
Agregar tarea...
Fase 4
Construir la carcasa que guardara todo el sistema y pc
24 – 27 jun
4 días
Agregar tarea...
Fase 5
Realización de pruebas para verificar la precisión de la
28 jun – 2 jul
3 días
Agregar tarea...
Fase 6
Documentar todo el proceso de desarrollo
2 oct de 2023 – 4 jul de 2024
199 días
Preparación de presentaciones y demostraciones
5 – 9 jul
3 días

*Ilustración 5 – Diagrama de Gant*

**Estimación del Coste del equipo**

Para desarrollar el calibrador de termómetros de infrarrojos sin contacto, es fundamental realizar una estimación detallada del coste de los componentes necesarios. A continuación, se presenta el desglose de los costes de los componentes:

1. **Coste de la placa principal:**
    - Precio: 65,68 €
2. **Coste de los componentes de la placa principal:**
    - **Conectores:**
        - Bloques terminales conectables WR-TBL 300VAC 20A 2P Vertical: 9 unidades a 0,409 € cada una -> Total: 3,68 €
        - Bloques terminales conectables WR-TBL 300VAC 12A 2P Straight: 9 unidades a 1,23 € cada una -> Total: 11,07 €
    - **MOSFET:**
        - MOSFET IPD90N04S4L-07: 3 unidades a 2,70 € cada una -> Total: 8,10 €
    - **Resistencias:**
        - Resistencia de película: 2 unidades a 0,093 € cada una -> Total: 0,19 €
        - Resistencia de película: 6 unidades a 0,093 € cada una -> Total: 0,56 €
    - **Controlador de puerta:**

        - Controlador de puerta para transistores Triple IGBT/MOS: 1 unidad a 2,60 € -> Total: 2,60 €

    - **Total componentes de la placa principal:** 26,20 €

3. **Coste de los sensores de temperatura:**

    - 3 sensores a 2,85 € cada uno -> Total: 8,55 €

4. **Coste de la placa de los sensores:**

    - Precio: 45,00 €

5. **Coste del microcontrolador dsPIC33FJ32MC202:**

    - Precio: 30,00 €

- **Coste Total de los Componentes**

Sumando todos los costes mencionados, tenemos:

- Una placa con el sistema completo: 65,68 €
- Componentes de la placa principal: 26,20 €
- Sensores de temperatura: 11,40 €
- Placa de los sensores: 45,00 €
- Tarjeta de desarrollo PicTrainer: 30,00 €

**Coste Total Estimado de los Componentes: 175,43 €**

A esto hay que sumarle el coste del personal que ha trabajado en el proyecto. Lo que cobra por hora un ingeniero recién incorporado al IIT es más o menos 45 y he empleado alrededor de 45 horas en el proyecto. Entonces al ingeniero habría que pagarle 45 x 45 = 2025 €.

**Esto nos deja con un coste total de 2200,43 €.**

# Capítulo 5. SISTEMA/MODELO DESARROLLADO

El dispositivo consta de 3 partes, la primera es la de los sensores de temperatura, la segunda la de los LEDs y la tercera las resistencias. Para cada parte hemos tenido que desarrollar su lógica un circuito en KIKAD, haciendo tanto el esquema como la placa de circuito impreso.

Para la primera parte tuvimos que hacer un diseño aparte del esquema general de todo el proyecto. Aquí hicimos una placa PCB para cada sensor. El esquema del sistema es el siguiente:

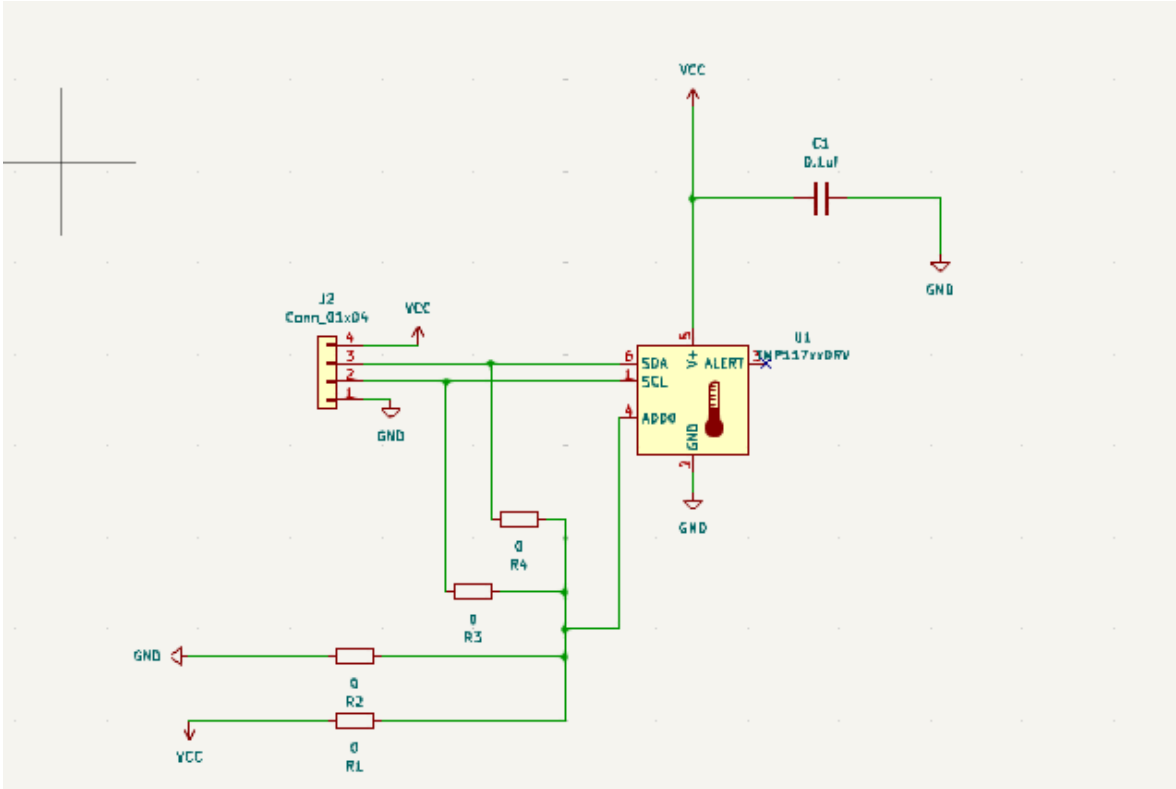



*Ilustración 6 - Esquema del sistema de los sensores de temperatura*

Como se puede ver en la ilustración 6 el cuadrado grande es el sensor, tiene 5 entradas, el Addr, que es la dirección del sensor, la tierra, el SDA y el SCL, para la comunicación i2c, es decir, el SCL es el reloj que va a poner el micro para sincronizar con el sensor y el SDA

es por donde se van a comunicar el micro y el sensor. La quinta, Alert, no está conectada a nada ya que no la necesitamos.

La otra parte es un conector, al que hemos conectado el SDA y el SCL del sensor ya que ese conector es el que está conectado al sistema. En la parte de abajo hemos puesto 4 resistencias con un valor de 0 ohmios ya que según las especificaciones del sensor dependiendo de a donde conectes el puerto de Addr, la dirección I2C del sensor va a variar, el Addr es la dirección del esclavo por la cual el micro sabe a donde tiene que enviar los datos. Como en un futuro el objetivo es que haya 3 sensores, necesitaremos que el micro pueda comunicarse con los 3 a la vez, por eso hemos soldado en un sensor la resistencia de 0 ohmios en VCC, en otro en SCL y en el último en SDA.

Este esquema lo implementamos en la placa PCB de esta manera:

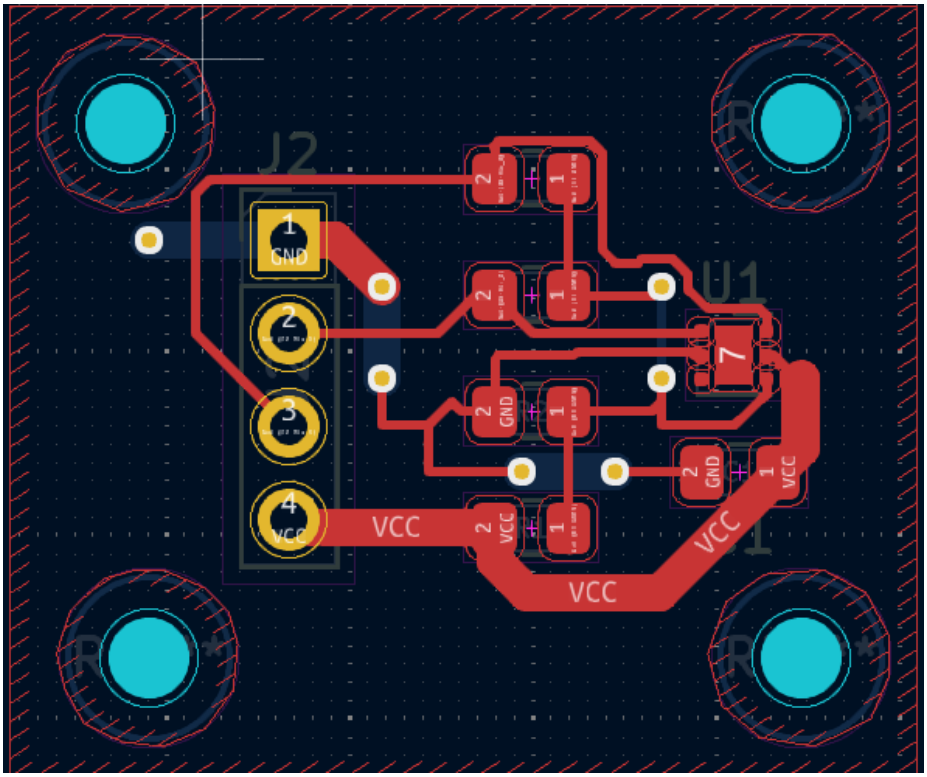



*Ilustración 7 – Placa PCB del sistema del sensor de temperatura.*

La placa está diseñada para ser lo más pequeña posible y le añadimos un plano de masa para conectar todos los puertos GND. El resultado final es el siguiente:

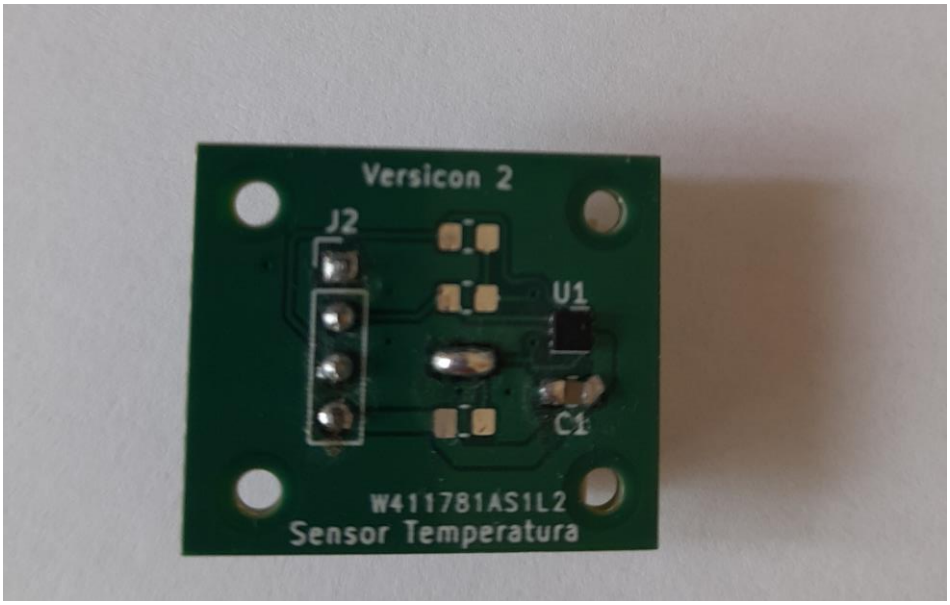



*Ilustración 8 – Imagen de la placa del sensor de temperatura.*

Así es como queda el sensor y esto es lo que se conecta al sistema completo y ya puede medir la temperatura constantemente.

Ya lo último que queda de esta parte es la comunicación i2c, lo primero que necesitamos es saber la dirección de memoria del registro y luego la dirección de memoria del byte donde está el SDA con la temperatura que está midiendo. Toda esta información está en el Data Sheet del sensor de temperatura. La dirección del registro varía según donde hayamos puesto la resistencia de 0 Ohmios, puede ser 0x48 si está conectada a tierra, 0x49 si está conectada a V+, 0x50 si está conectada a SDA y 0x51 si esta conectada a SCL. Nosotros la hemos conectado a V+, por lo que la dirección que usamos es 0x49  y la del registro donde se almacena la temperatura medida es 0x00.

**Table 7-2. Address Pin and Slave Addresses**

| DEVICE TWO-WIRE ADDRESS | ADD0 PIN CONNECTION |
|---|---|
| 1001000x | Ground |
| 1001001x | V+ |
| 1001010x | SDA |
| 1001011x | SCL |

*Ilustración 9 – Tabla con las direcciones del sensor de temperatura.*

La dirección que pone en la tabla de la ilustración 9 que hemos usado es la 1001001x que es 0x49 en hexadecimal y esa x esta porque en i2c siempre se desplaza un bit a la izquierda para indicar si se va a leer o escribir. La otra dirección que necesitamos es la de la medición de la temperatura que mirando en el Data sheet vemos que es el 0x00.

**Table 7-3. TMP117 Register Map**

| ADDRESS | TYPE | RESET | ACRONYM | REGISTER NAME | SECTION |
|---|---|---|---|---|---|
| 00h | R | 8000h | Temp_Result | Temperature result register | Go |
| 01h | R/W | 0220h(1) | Configuration | Configuration register | Go |
| 02h | R/W | 6000h(1) | THigh_Limit | Temperature high limit register | Go |
| 03h | R/W | 8000h(1) | TLow_Limit | Temperature low limit register | Go |
| 04h | R/W | 0000h | EEPROM_UL | EEPROM unlock register | Go |
| 05h | R/W | xxxxh(1) | EEPROM1 | EEPROM1 register | Go |
| 06h | R/W | xxxxh(1) | EEPROM2 | EEPROM2 register | Go |
| 07h | R/W | 0000h(1) | Temp_Offset | Temperature offset register | Go |
| 08h | R/W | xxxxh(1) | EEPROM3 | EEPROM3 register | Go |
| 0Fh | R | 0117h | Device_ID | Device ID register | Go |

*Ilustración 10 – Tabla con la dirección del registro de la temperatura medida.*

Para poder establecer la comunicación i2c primero hay que inicializar el bus, para ello hemos creado una función que según el valor de entrada que le des, define los baudios (Herzios) a los que va a funcionar:

```
void I2C1Inicializa(int velocidad)
 {
    if(velocidad == 1){
        // 400 kb/s
        I2C1BRG = 4;
        I2C1CON = 0x8000; // I2C ON, con control de slew rate
    }else{
        // 100 kb/s
        I2C1BRG = 23;
        I2C1CON = 0x8200; // I2C ON, sin control de slew rate
    }
 }
```

A continuación, hemos hecho las funciones para empezar y acabar la comunicación, para ello, el microcontrolador, tiene un bit que activa cada condición, I2C1CONbits.SEN para el start y I2C1CONbits.PEN para el stop.

```
void I2C1GeneraStart(void)
 {
    I2C1CONbits.SEN=1; //Generacondicióndestartenelbus
    while (I2C1CONbits.SEN);
    ;//Esperaelfinaldelacondición
 }

  void I2C1GeneraStop(void)
 {
    I2C1CONbits.PEN = 1; // Genera condición de stop en el bus
    while (I2C1CONbits.PEN);
    ; // Espera el final de la condición
 }
```

Además de estas 2, como vamos a estar leyendo, vamos a necesitar otra función que nos vuelva a generar la condición de start repetido.

```
  void I2C1GeneraReStart(void)
 {
    I2C1CONbits.RSEN = 1; // Genera condición de repeteaded
    // start en el bus
    while (I2C1CONbits.RSEN);
    ;//Esperaelfinaldelacondición
 }
```

Ahora ya hay que programar las funciones para leer y escribir los bytes que queramos. La de escribir es más sencilla, solo hay que poner en el registro que se envía constantemente el valor que queremos enviar y esperar al ack. En este programa estamos leyendo la temperatura, pero, primero tenemos que escribir la dirección del registro donde está conectado el sensor, luego la dirección de memoria donde este guarda el valor de la temperatura y ya finalmente otra vez la dirección del registro pero esta vez con el ultimo bit a 1 para indicarle que lo que queremos es leer. La función de leer, lo que hace es activar la recepción del dato y guardar en una variable el valor que está leyendo constantemente el registro predeterminado.

```
uint8_t I2C1LeeByte(int ack)
 {
    uint8_t dato;
    // Se activa la recepción del dato
    I2C1CONbits.RCEN = 1;
    while(I2C1CONbits.RCEN);
     // Espera el final de la recepción

    dato = I2C1RCV;
    //I2C1CONbits.RCEN = 0;
    I2C1CONbits.ACKDT = ack&1;
    I2C1CONbits.ACKEN = 1; // Se envía el ACK
    while(I2C1CONbits.ACKEN);
     // Espera el envío del ACK

    return dato;
 }

  int I2C1EscribeByte(uint8_t dato)
 {
    I2C1TRN=dato;
    //while(IFS1bits.I2C1MIF==0)
    while (I2C1STATbits.TRSTAT);
    ; //Esperaelfinaldelenvío
    //IFS1bits.I2C1MIF=0; //Borroelflag
    return I2C1STATbits.ACKSTAT;
 }
```

La última función es la que se encarga de toda la lógica para leer el byte que queremos. Para poder hacerlo bien, primero hemos tenido que leer tanto el data sheet[4] de cómo funciona el i2c en el microcontrolador dsPIC33FJ32MC202 y cuales son los pasos que hay que seguir para leer bien la temperatura en el data sheet[5] del sensor.

Para poder medir correctamente la temperatura, en el data sheet del sensor aparece una tabla con todas las direcciones de todos los campos con información que tiene el sensor.

[4] 70000195g.pdf (microchip.com)
[5] tmp117.pdf (ti.com)

### 7.6 Register Map

**Table 7-3. TMP117 Register Map**

| ADDRESS | TYPE | RESET | ACRONYM | REGISTER NAME | SECTION |
|---|---|---|---|---|---|
| 00h | R | 8000h | Temp_Result | Temperature result register | Go |
| 01h | R/W | 0220h(1) | Configuration | Configuration register | Go |
| 02h | R/W | 6000h(1) | THigh_Limit | Temperature high limit register | Go |
| 03h | R/W | 8000h(1) | TLow_Limit | Temperature low limit register | Go |
| 04h | R/W | 0000h | EEPROM_UL | EEPROM unlock register | Go |
| 05h | R/W | xxxxh(1) | EEPROM1 | EEPROM1 register | Go |
| 06h | R/W | xxxxh(1) | EEPROM2 | EEPROM2 register | Go |
| 07h | R/W | 0000h(1) | Temp_Offset | Temperature offset register | Go |
| 08h | R/W | xxxxh(1) | EEPROM3 | EEPROM3 register | Go |
| 0Fh | R | 0117h | Device_ID | Device ID register | Go |

(1) This value is stored in Electrically-Erasable, Programmable Read-Only Memory (EEPROM) during device manufacturing. The device reset value can be changed by writing the relevant code in the EEPROM cells (see the *EEPROM Overview* section).

*Ilustración 11 – Tabla con las direcciones de los registros del sensor*

Aquí solo nos tenemos que fijar en los 2 primeros, que son los únicos que nos van a importar. El primero, el 0x00, es donde se guarda la medición de la temperatura que hace el sensor (Ilustración 12) y el segundo, es el registro de configuración que contiene toda la información de los estados del sensor.

**Figure 7-13. Temperature Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| T15 | T14 | T13 | T12 | T11 | T10 | T9 | T8 |
| R-1 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| T7 | T6 | T5 | T4 | T3 | T2 | T1 | T0 |
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |

**Table 7-5. Temperature Register Field Descriptions**

| BIT | FIELD | TYPE | RESET | DESCRIPTION |
|---|---|---|---|---|
| 15:0 | T[15:0] | R | 8000h | 16-bit, read-only register that stores the most recent temperature conversion results. |

*Ilustración 12 – Tabla con la información del registro de temperatura*

Como podemos ver en el registro de configuración en la ilustración 13, hay 11 campos que cada uno da una información diferente. A nosotros solo nos importa el Data_ready en el bit 13, que esto nos va a indicar cuando el sensor haya hecho la medición y la conversión y está

lista para ser leída, si lo leemos antes de que ese bit este a 0, vamos a obtener una medición errónea.

**Figure 7-14. Configuration Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| HIGH_Alert | LOW_Alert | Data_Ready | EEPROM_Busy | MOD1(2) | MOD0(1) | CONV2(1) | CONV1(1) |
| R-0 | R-0 | R-0 | R-0 | R/W-0 | R/W-0 | R/W-1 | R/W-0 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CONV0(1) | AVG1(1) | AVG0(1) | T/nA(1) | POL(1) | DR/Alert(1) | Soft_Reset | — |
| R/W-0 | R/W-0 | R/W-1 | R/W-0 | R/W-0 | R/W-0 | R-0 | R-0 |

**Table 7-6. Configuration Register Field Descriptions**

| BIT | FIELD | TYPE | RESET | DESCRIPTION |
|---|---|---|---|---|
| 15 | HIGH_Alert | R | 0 | High Alert flag:<br>1: Set when the conversion result is higher than the high limit<br>0: Cleared on read of configuration register<br>Therm mode:<br>1: Set when the conversion result is higher than the therm limit<br>0: Cleared when the conversion result is lower than the hysteresis |
| 14 | LOW_Alert | R | 0 | Low Alert flag:<br>1: Set when the conversion result is lower than the low limit<br>0: Cleared when the configuration register is read<br>Therm mode: Always set to 0 |
| 13 | Data_Ready | R | 0 | Data ready flag.<br>This flag indicates that the conversion is complete and the temperature register can be read. Every time the temperature register or configuration register is read, this bit is cleared. This bit is set at the end of the conversion when the temperature register is updated. Data ready can be monitored on the ALERT pin by setting bit 2 of the configuration register. |
| 12 | EEPROM_Busy | R | 0 | EEPROM busy flag.<br>The value of the flag indicates that the EEPROM is busy during programming or power-up. |
| 11:10 | MOD[1:0] | R/W | 0 | Set conversion mode.<br>00: Continuous conversion (CC)<br>01: Shutdown (SD)<br>10: Continuous conversion (CC), Same as 00 (reads back = 00)<br>11: One-shot conversion (OS) |
| 9:7 | CONV[2:0] | R/W | 100 | Conversion cycle bit.<br>See Table 7-7 for the standby time between conversions. |
| 6:5 | AVG[1:0] | R/W | 01 | Conversion averaging modes. Determines the number of conversion results that are collected and averaged before updating the temperature register. The average is an accumulated average and not a running average.<br>00: No averaging<br>01: 8 Averaged conversions<br>10: 32 averaged conversions<br>11: 64 averaged conversions |
| 4 | T/nA | R/W | 0 | Therm/alert mode select.<br>1: Therm mode<br>0: Alert mode |
| 3 | POL | R/W | 0 | ALERT pin polarity bit.<br>1: Active high<br>0: Active low |
| 2 | DR/Alert | R/W | 0 | ALERT pin select bit.<br>1: ALERT pin reflects the status of the data ready flag<br>0: ALERT pin reflects the status of the alert flags |

**Table 7-6. Configuration Register Field Descriptions (continued)**

| BIT | FIELD | TYPE | RESET | DESCRIPTION |
|---|---|---|---|---|
| 1 | Soft_Reset | R/W | 0 | Software reset bit.<br>When set to 1 it triggers software reset with a duration of 2 ms<br>This bit will always read back 0 |
| 0 | — | R | 0 | Not used |

(1) These bits can be stored in EEPROM. The factory setting for this register is 0220.
(2) The MOD1 bit cannot be stored in EEPROM. The device can only be programmed to start up in shutdown mode or continuous conversion mode.

*Ilustración 13 – Tabla con la información del registro de configuración*

Entonces, antes de leer la temperatura, tenemos que primero leer este registro de configuración y esperar a que el bit 13, el del data_ready, esté a 1, cuanto esté a 1 ya leemos la temperatura. Además a la función tenemos que pasarle como argumento la dirección del sensor que estamos usando. De tal manera que la función quedaria así:

```
float LeerTemperatura(uint8_t dir_chip)
 {
    int chip_id =-1; //-1 indica error de recepción

    uint8_t conf1 = 0;
    uint8_t conf2 = 0;
    uint16_t reg_conf = 0;

    uint8_t byte1 = 0;
    uint8_t byte2 = 0;
    uint16_t entero = 0;

    I2C1GeneraStart();
    if(I2C1EscribeByte(dir_chip<<1) != 0){ //NACK
        I2C1GeneraStop(); // Abortamos
        return chip_id;
    }
    if(I2C1EscribeByte(CONF_REGISTER) != 0){ //NACK
       I2C1GeneraStop(); // Abortamos
        return chip_id;
    }
    I2C1GeneraReStart();

    if(I2C1EscribeByte(dir_chip<<1|1) != 0){ //NACK
        I2C1GeneraStop(); // Abortamos
        return chip_id;
    }

    conf1 = I2C1LeeByte(1);

    conf2 = I2C1LeeByte(0);

    I2C1GeneraStop();

    reg_conf = ((uint16_t)conf1) << 8;
    reg_conf = reg_conf | conf2;

    if((reg_conf & (1 << 13)) != 0){

        I2C1GeneraStart();
        if(I2C1EscribeByte(dir_chip<<1) != 0){ //NACK
            I2C1GeneraStop(); // Abortamos
            return chip_id;
        }
        if(I2C1EscribeByte(TEMPERATURE_REGISTER) != 0){ //NACK
           I2C1GeneraStop(); // Abortamos
```

```
            return chip_id;
        }
        I2C1GeneraReStart();

        if(I2C1EscribeByte(dir_chip<<1|1) != 0){ //NACK
            I2C1GeneraStop(); // Abortamos
            return chip_id;
        }

        byte1 = I2C1LeeByte(0);
        byte2 = I2C1LeeByte(1);

        entero = ((uint16_t)byte1) << 8;
        entero = entero | byte2;

        I2C1GeneraStop();
        return entero* 0.0078125;
    }
    else{
        I2C1GeneraStop();
        return -100;
    }

}
```

Este .c lo guardamos aparte e incluimos su .h en el .c donde tenemos el main. Además, también tenemos que incluir otros .c, estos son el de la uart, ya que vamos a usar un programa, el Coolterm, que mediante la uart le vamos a ir mandando todas las mediciones de la temperatura para que las muestre por pantalla; También necesitamos incluir el de idle.c, que este lo que hace es que cada vez que hace el while(1) espera un rato, en nuestro caso 1 ms para que no vaya demasiado rápido y no se sature. Para calentar la resistencia, lo que hacemos es poner el pin al que está conectada a 1 y cuando supera la temperatura deseada lo ponemos a 0 y cuando baja de la temperatura deseada lo ponemos a 1 para que vuelva a calentar. Este método es muy rudimentario y para futuras versiones el objetivo es mediante el archivo incluido PWM.c hacer una función que deje la temperatura siempre estable, que no tenga fluctuaciones, pero de momento no hemos hecho uso de ese .c. Para configurar el reloj del micro necesitamos la función que incluye el archivo config.c. Ya por último, lo primero quehacemos en el main es declarar los leds como salidas al igual que las resistencias, que son Pin_T_Baja, Pin_T_Media, Pin_t_Alta. De tal manera que el main queda así:

```
/*
```

```
 * File:   newmainXC16.c
 * Author: gonza
 *
 * Created on 30 de abril de 2024, 22:51
 */


#include "xc.h"
#include "pwm.h"
#include "config.h"
#include "idle.h"
#include "uart.h"
#include "i2c.h"

#define PIN_T_ALTA 15 // INC del driver
#define PIN_T_MEDIA 13 // INB
#define PIN_T_BAJA 11 // INA


#define sensor1 0x48
#define sensor2 0x49
#define sensor3 0x50
#define TEMPERATURE_REGISTER 0x00
#define CONF_REGISTER 0x01

#define TEMP1 36
#define TEMP2 38
#define TEMP3 40


int main(void){
    int n=0;


    inicializarReloj();


    TRISB &= ~(1 << 10); // led 4  ?Salida
    TRISB &= ~(1 << 7); // led 5  ?Salida

    TRISB &= ~(1 << PIN_T_ALTA); //RB15 salida
    TRISB &= ~(1 << PIN_T_MEDIA); //RB15 salida
    TRISB &= ~(1 << PIN_T_BAJA); //RB15 salida

    PORTB = 0;
    PORTB |= 1 << 7;
    PORTB |= 1 << 10;


    inicializarUART(115200);
    char cad[20];

    inicializarTareaIdle(10);
```

```
    uint16_t tmp2;


    I2C1Inicializa(I2C100KHZ);

    while(1){


        temperaturaC2 = LeerTemperatura(sensor2); // Lee la temperatura
del TMP117



        if(temperaturaC2 != -100){
            if(temperaturaC2 >= TEMP1){
                PORTB |= 1 << PIN_T_BAJA;

            }
            else{
                PORTB &= ~(1 << PIN_T_BAJA);

            }
        }

         if(temperaturaC2 != -100){
            if(temperaturaC2 >= (TEMP1-0.2) && temperaturaC2 <=
(TEMP1+0.2)) {
                PORTB |= 1 << 7;
                n++;
                n--; // estas instrucciones están porque si se hacen dos
// accesos consecutivos al PORTB, el segundo falla
                PORTB &= ~(1 << 10);
            }
            else{
                PORTB &= ~(1 << 7);
                n++;
                n--;
                PORTB |= 1 << 10;

            }
        }


        if(temperaturaC2 != -100){
            sprintf(cad, "T=%f\n",temperaturaC2);
            putsUART(cad);

        }


        tareaIdle();
    }
```

```
}
```

Por otro lado, tenemos la placa con el resto del sistema, esta contiene el conector para los sensores, los leds, las resistencias térmicas y el driver. El esquema completo se ve así:

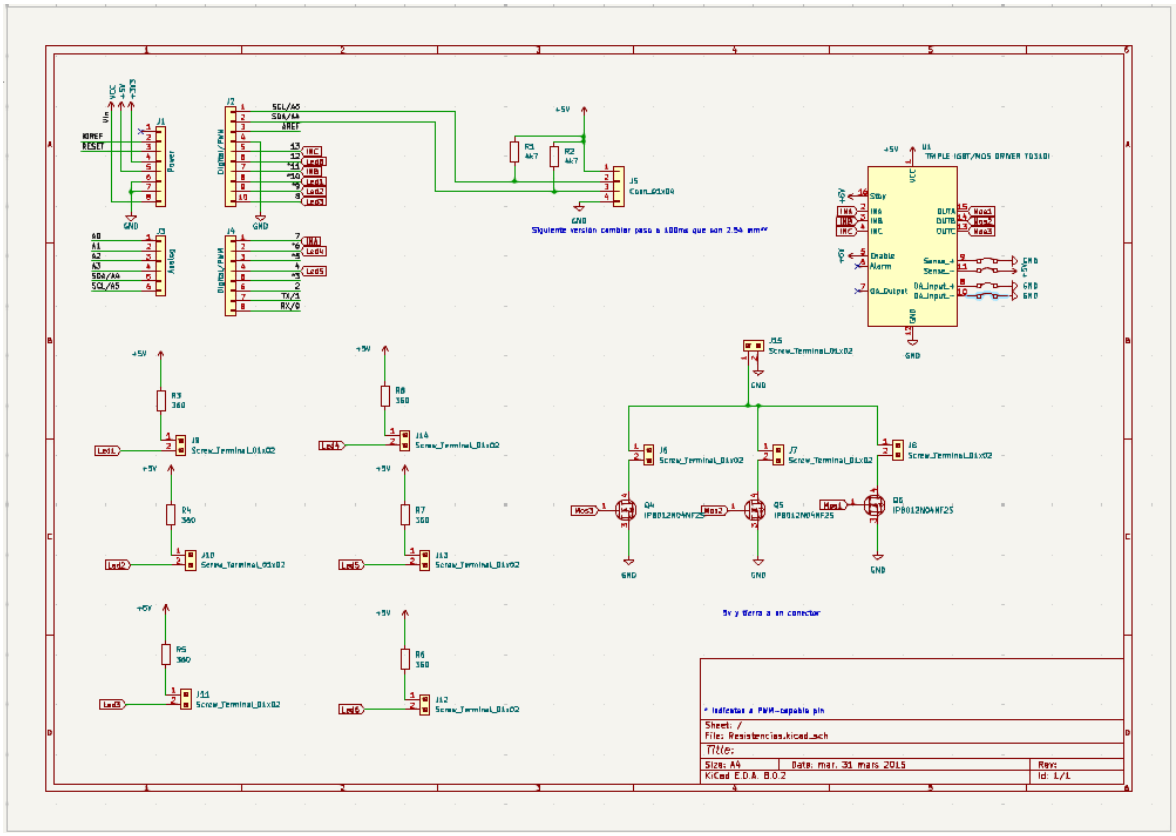



*Ilustración 14 – Esquema completo del dispositivo*

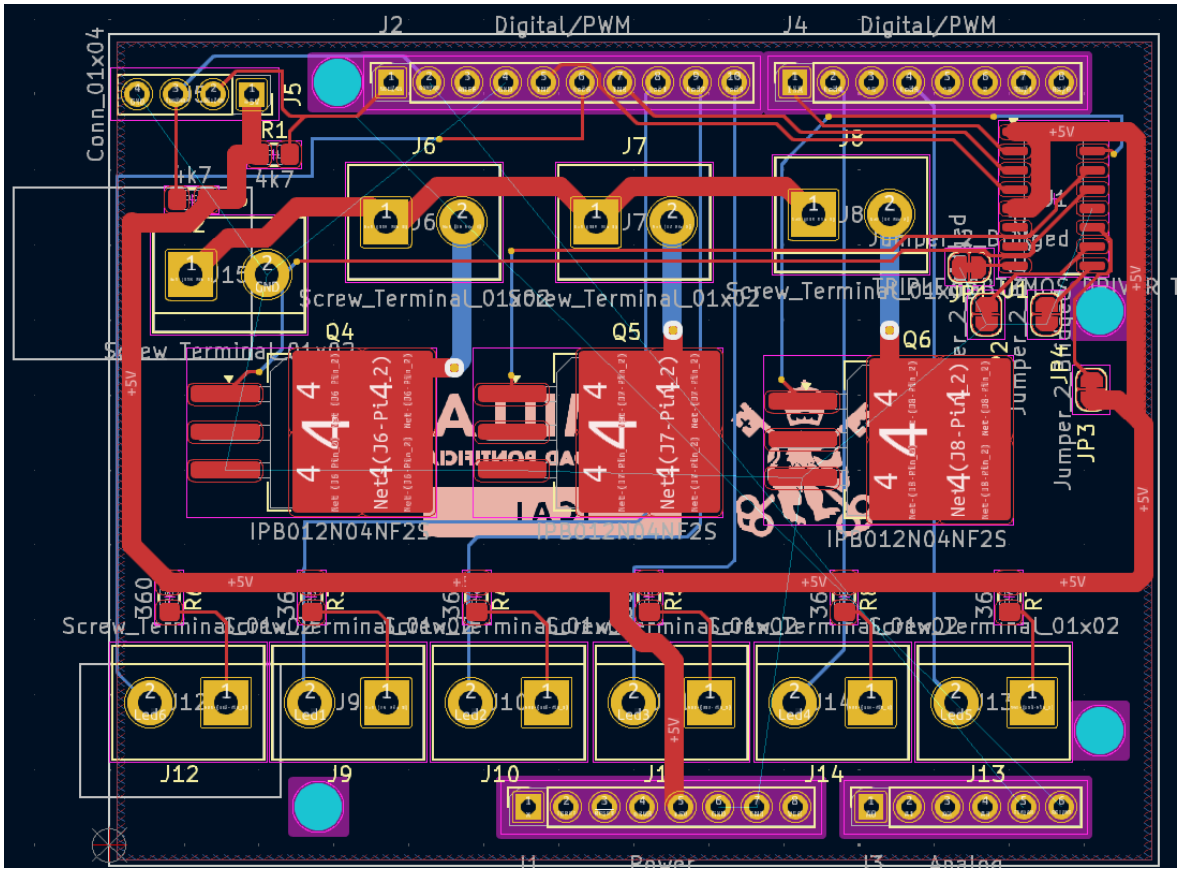



*Ilustración 15 –PCB completo del dispositivo*

Esta placa la hemos diseñado para ponerla encima del microcontrolador, de esta manera esta todo mucho más compacto y ocupa menos espacio. En el esquema completo de la ilustración 11, los conectores de arriba a la izquierda son todos los conectores del micro, que también van a estar en nuestra placa, se ve muy bien en el PCB, los conectores morados son los que van a ir enchufados a la placa.

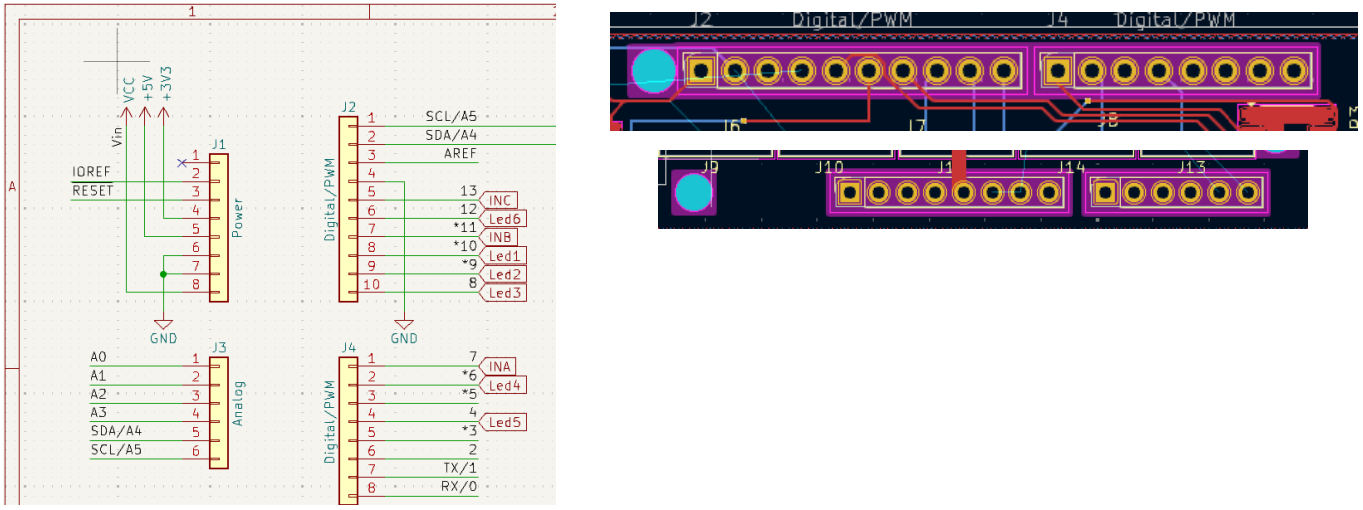



*Ilustración 16 – Conectores para conectar la placa con el micro*


*Ilustración 17 – Conectores en el micro donde conectaremos la placa*

Continuando con las distintas partes, arriba en el medio tenemos el conector al que van conectados los 3 sensores de temperatura, está conectado al SCL y SDA del micro y le hemos

puesto 2 resistencias ya que para que la comunicación i2c funcione es necesario que haya 2 resistencias de 4.7k entre SCL y SDA y 5V.

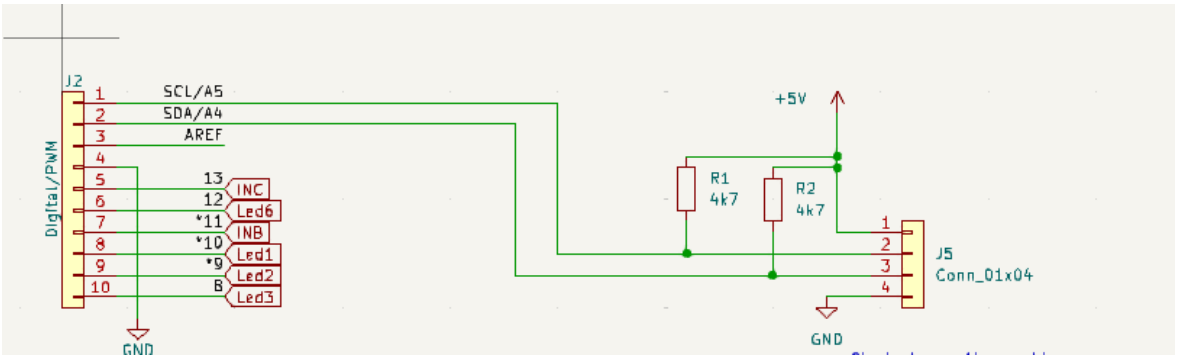



*Ilustración 18 – Conector para conectar los sensores de temperatura*

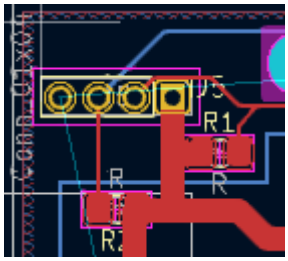



*Ilustración 19 – Conector en la PCB*

Abajo a la izquierda tenemos los circuitos de los 6 LEDs, cada led está conectado a un pin distinto del micro para que sean independientes, luego en la placa hemos puesto un screw terminal (Terminal de tornillo) en los que hemos puesto los LEDS, para que no estuviesen directamente en la placa y tuviésemos más movilidad al hacer la carcasa. Ya por último, hemos tenido que conectar una patilla del LED a 5 voltios con una resistencia de 360 Ohmios entre medias. Hay 6 iguales ya que necesitamos 6 LEDs, 3 verdes y 3 rojos para indicar cuando esta lista cada placa.

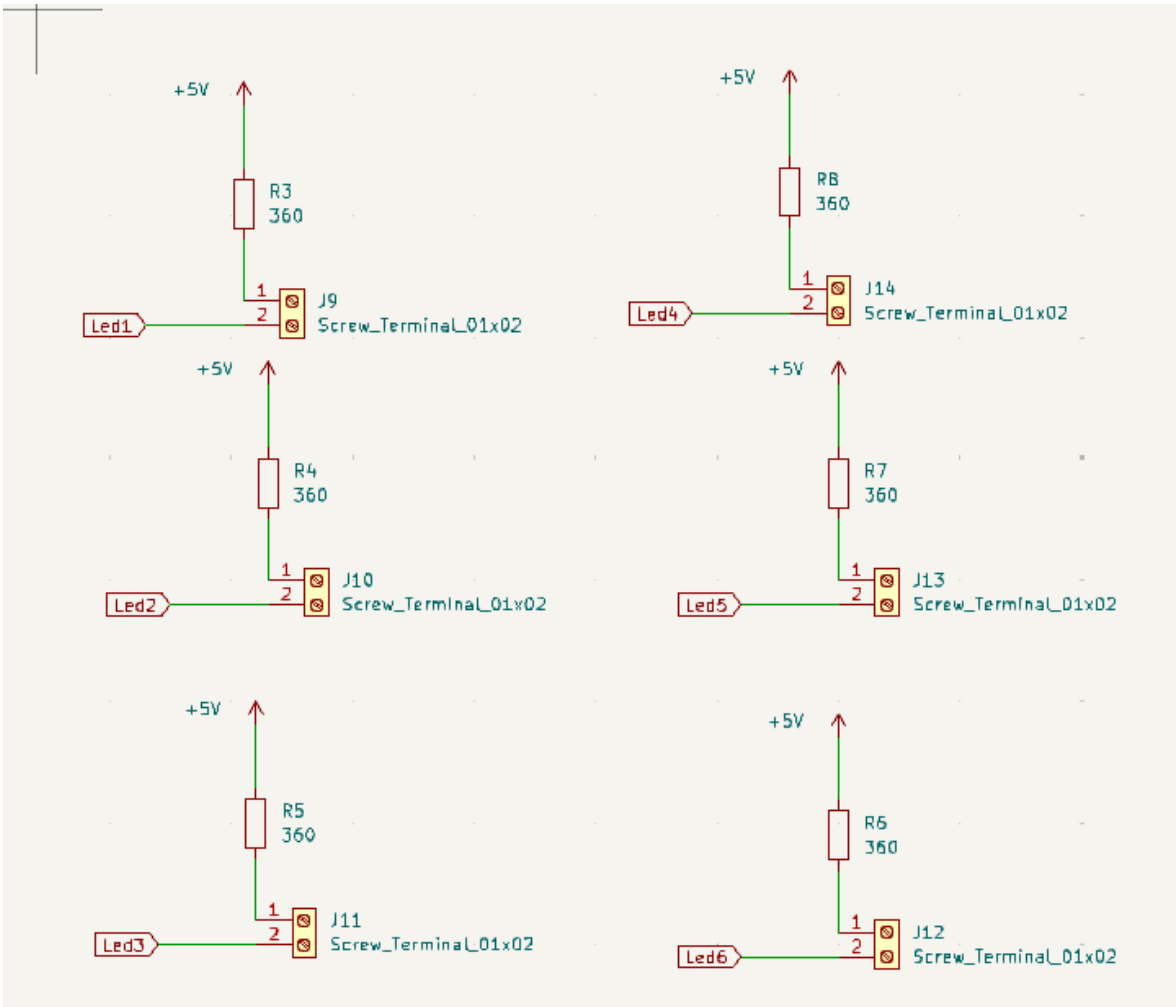



*Ilustración 20 – Esquema de los LEDs*

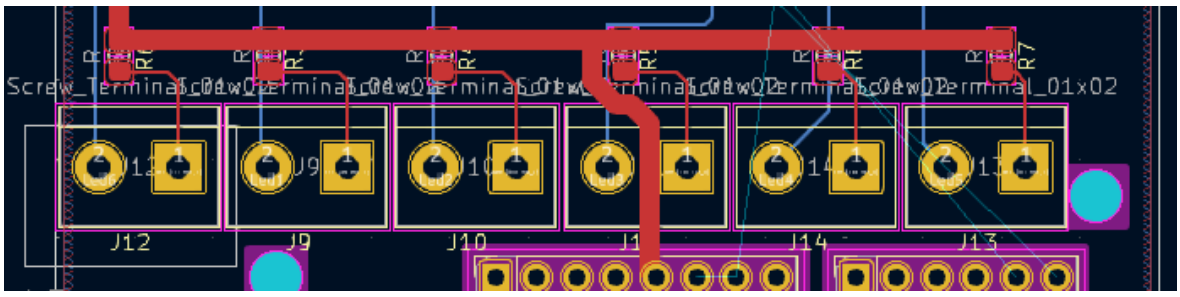



*Ilustración 21 – LEDs y resistencias en la PCB*

Continuando, abajo a la derecha tenemos la parte clave del proyecto y son las 3 placas que se van a poner a 36°C, 38°C y 40°C. Para diseñar los 3 circuitos hemos puesto un mosfet en cada resistencia, que está conectado a la placa ya que el mosfet lo necesitamos para controlar las placas de temperatura, que pase o no pase corriente, para controlar el calor que generan (PWN). Los mosfets no están conectados directamente a la placa, aquí también hemos hecho como con los LEDS, los hemos conectado a screw terminals (Terminales de tornillo), para poder conectar las placas con cables y así tener más movilidad y poder colocarlas donde mejor encajaban en la carcasa. Además, las resistencias no pueden ir conectadas directamente a la fuente de alimentación del ordenador ya que este no es capaz dealimentarlas, por lo que hemos puesto otro screw terminal (Terminal de tornillo) al que van conectadas las 3 resistencias de tal manera que este luego lo conectamos a una fuente de alimentación externa.

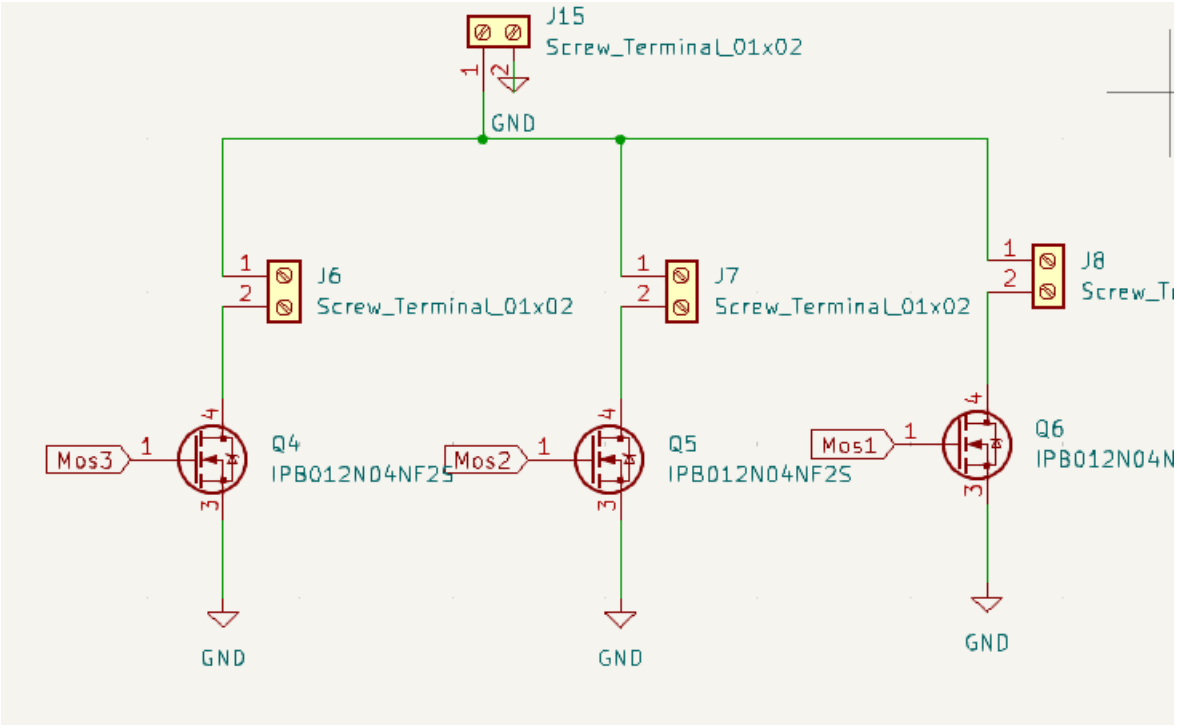



*Ilustración 22 – Esquema de las placas de temperatura*

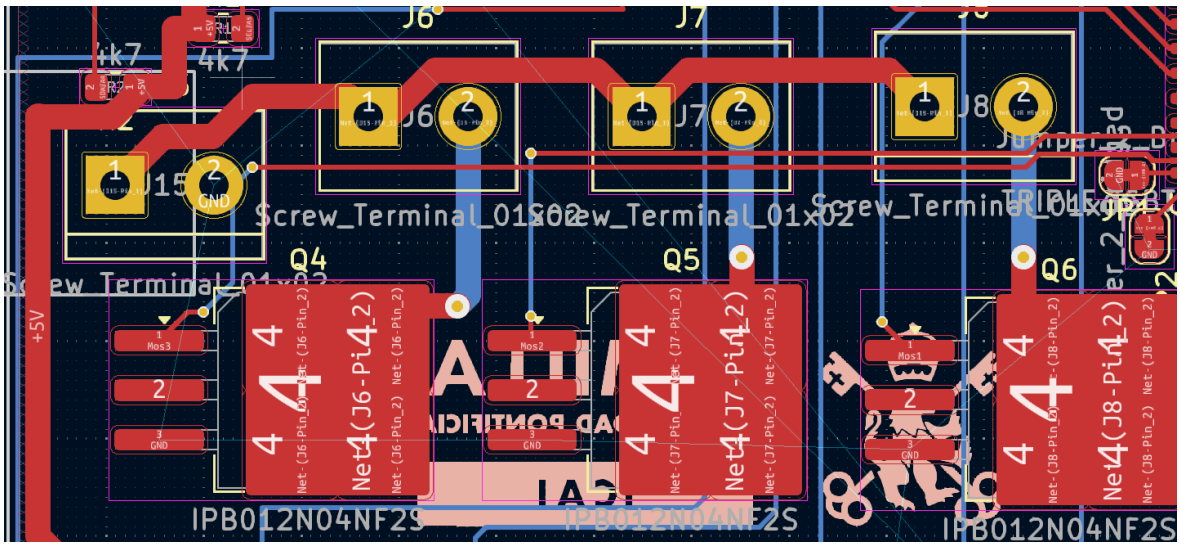



*Ilustración 23 – Placas de temperatura en la PCB*

Los mosfets no pueden ir conectados directamente al micro, es por ello por lo que hemos tenido que usar un driver, ya que como el MOS conmuta a alta frecuencia (PWM) y el micro da poca corriente, el MOS tarda bastante en conmutar porque hay que cargar la capacidad que hay entre puerta y fuente y se producen muchas pérdidas. El driver no estaba en las librerías de Kikad por lo que tuvimos que crearlo nosotros en el editor de símbolos de Kikad y luego añadirlo.

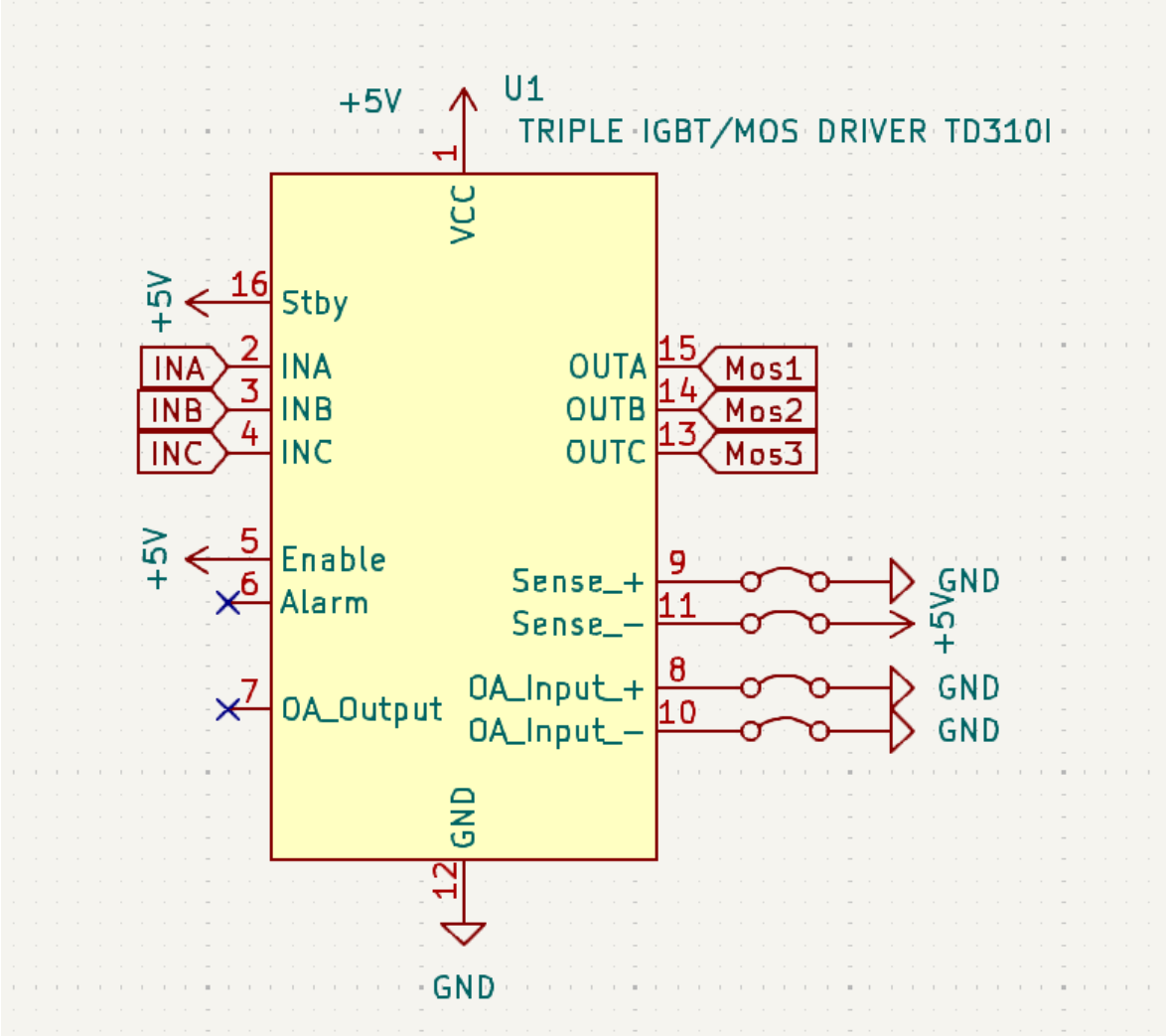



*Ilustración 24 – Esquema del driver*

Como vemos en la ilustración 24, el driver que hemos escogido tiene 16 puertos, varios de ellos no los utilizamos como el alarm, el OA_OUTPUT, y los senses y los OA_INPUT, tampoco los utilizamos, pero los hemos conectado a tierra usando puentes, menos el sense-, que tiene que estar conectado a 5V, ya que como indica el data sheet del driver[6], para que

[6] td310-1852141.pdf (mouser.es)

este funcione en modo normal, sense+ tiene que ser menor que sense-, como se puede ver en la ilustración 25.

The following table summarizes the functions of the TD310 :

| | Pin | 16 | 9/11 | 5 | 2/3/4 | 15/14/13 | 6 | 7/8/10 | Con-sumption |
|---|---|---|---|---|---|---|---|---|---|
| | Config | UVLO/ stdby | Sense+/ Sense- | Enable | In A/B/C | Out A/B/C | Alarm | Op-Amp | |
| Normal | 1 | H | + > - | X | X | L | L | OK | H (1.5mA) |
| | | | + < - | H | IN | $\overline{IN}$ | H | | |
| | | | | L | X | L | | | |
| Stdby | 2 | L | + > - | X | X | L | L | HZ | L (30µA) |
| | | | + < - | | | | H | | |
| UVLO | 3 | M | X | X | X | L | L | OK | H |

*Ilustración 25 – Tabla del funcionamiento del driver*

Esto lo hacemos por si fuese necesario usarlos en un futuro. Los pines 2,3,4 estan conectados a diferentes pines del micro, para poder controlarlo y los 13,14 y 15 a los diferentes mosfets. En la PCB queda así:

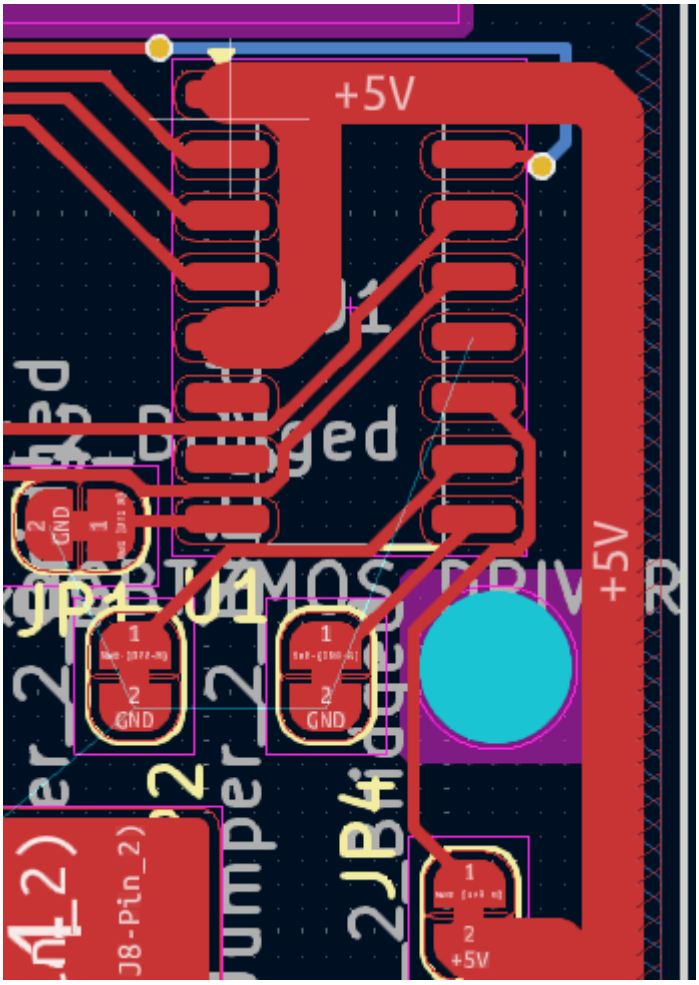



*Ilustración 26 – Driver en la PCB*

Ya con la placa diseñada, hay que mandarla fabricar y el siguiente paso, cuando tenemos ya la placa, es soldar todos los componentes. Hay que tener mucho cuidado sobre todo con el driver ya que es muy pequeño y sus patillas están muy juntas. También hay que tener cuidado

con los mosfets, ya que son muy delicados. Para soldar utilizamos estaño y el proceso de soldado se puede ver en las siguientes ilustraciones.

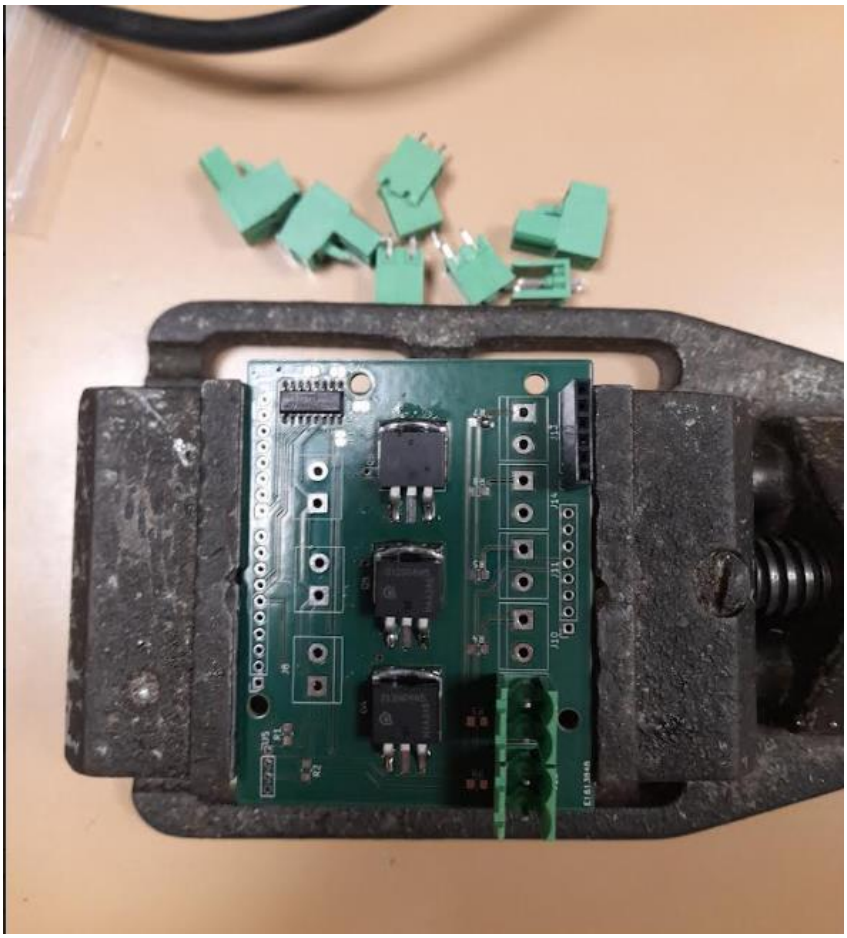


*Ilustración 27 – Proceso de soldado de la PCB*

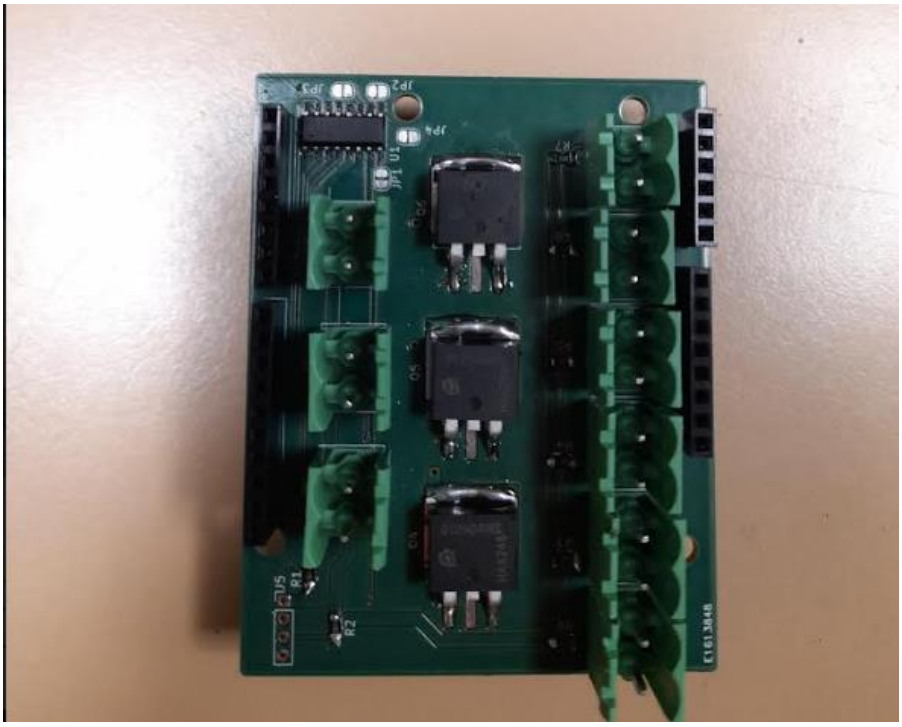


*Ilustración 28 – PCB soldada al completo*

Teniendo ya la placa lista, hay que conectarla al micro, poniéndola encima y encajando todos los conectores. Con la placa encima del micro ya solo queda poner todos los otros componentes, conectamos el sensor a la placa, para esto hay que tener especial cuidado ya que cada pin del sensor tiene que estar conectado con su correspondiente en la placa, si nos equivocamos aquí se generaría un cortocircuito y romperíamos el sensor.

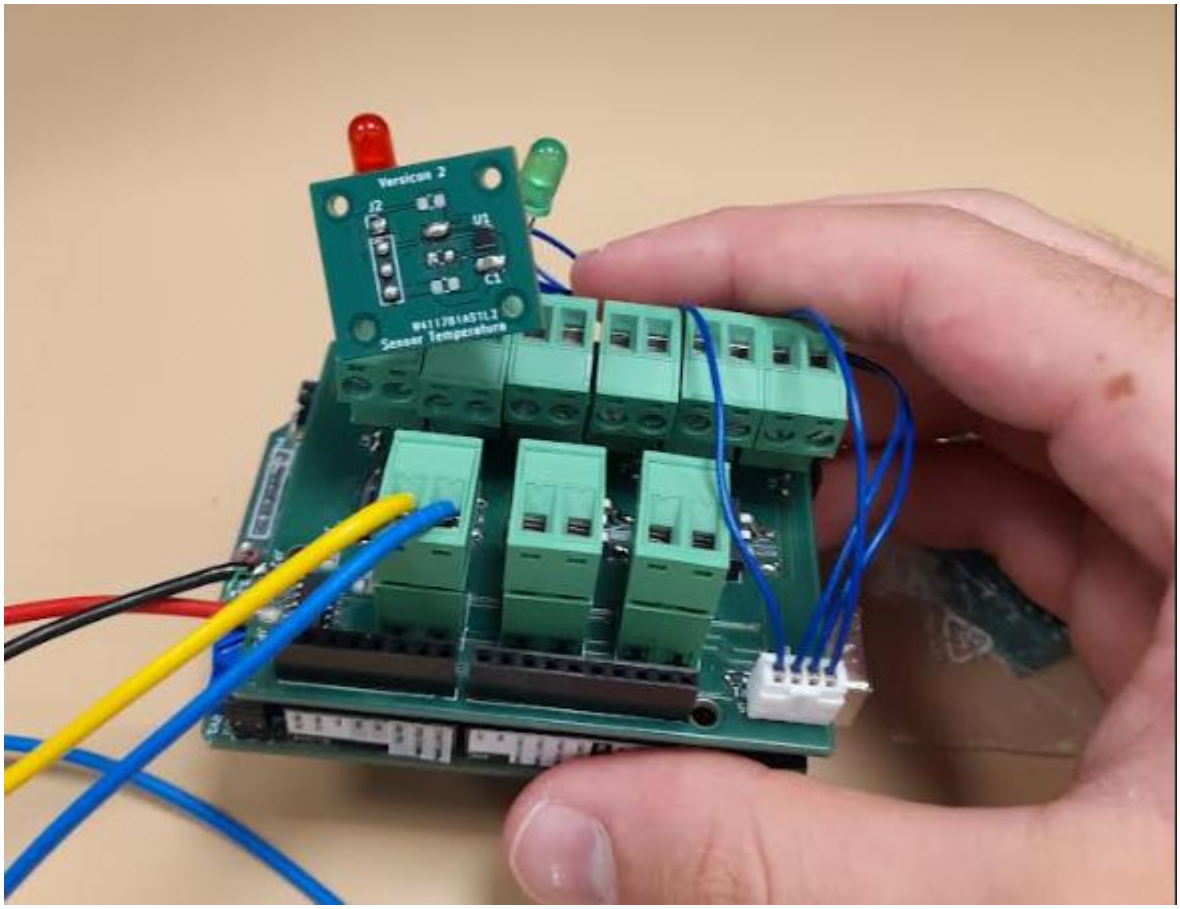



*Ilustración 29 –Sensor integrado en la placa*

También hay que poner los LEDs, uno verde y otro rojo, para que indiquen bien cuando esta lista y cuando no

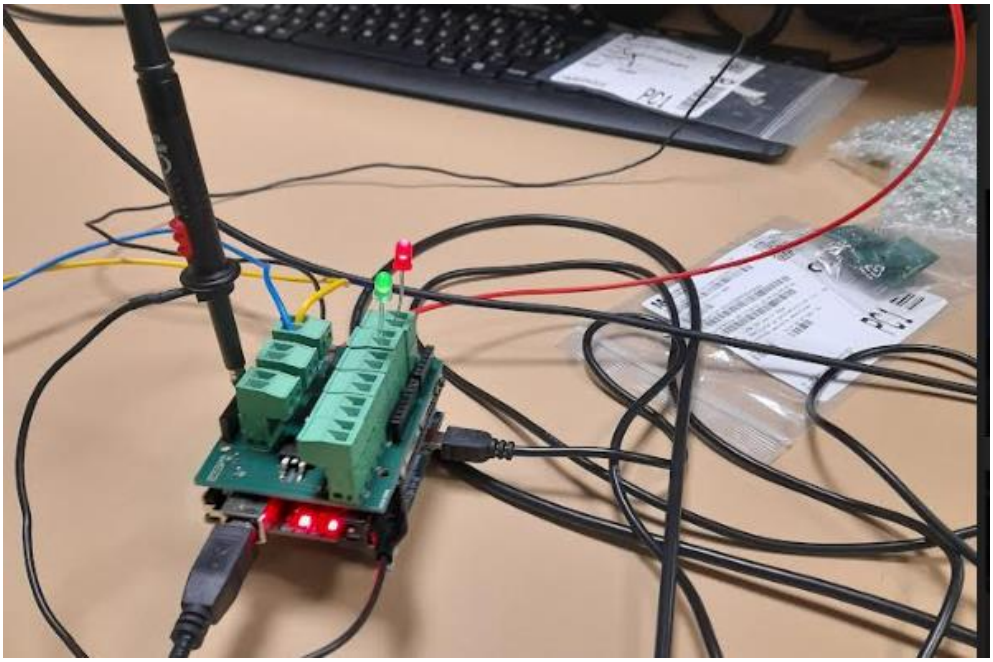


*Ilustración 30 – Leds integrados en la placa*

y por último la resistencia, que una vez colocada hay que pegarla al sensor. Para esto, primero hay que cubrir la placa del sensor con cinta aislante ya que, de lo contrario, la resistencia entraría en contacto con todos los componentes. Los juntamos con una cinta elástica y el prototipo ya está completo como se puede ver en la ilustración 31.

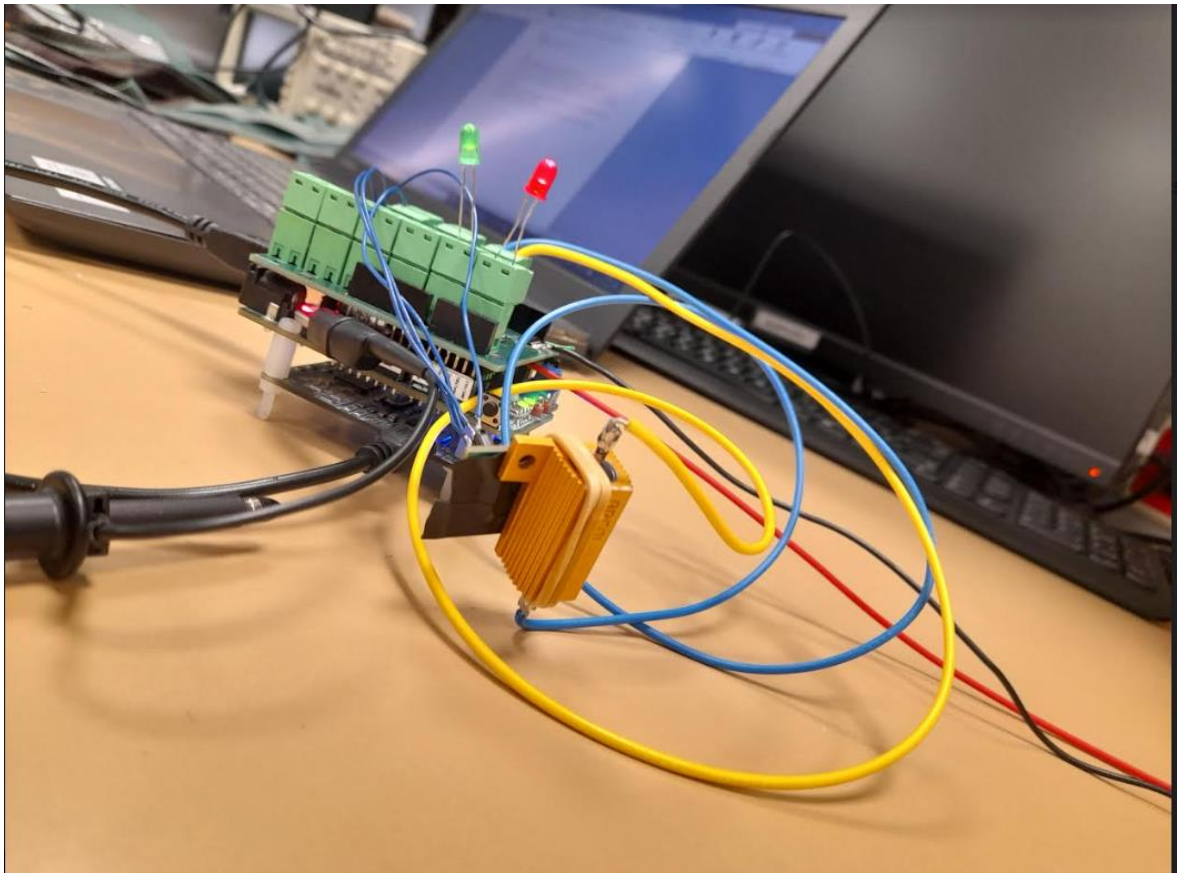


*Ilustración 31 – Prototipo montado y completo*

# Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se presentan y analizan los resultados obtenidos en la prueba de funcionamiento del prototipo de control de temperatura. La figura de la ilustración 32 muestra la gráfica de temperatura registrada durante la prueba. El objetivo de la prueba era evaluar la capacidad del sistema de mantener la temperatura alrededor de un valor objetivo utilizando un método de control básico.

El experimento consistió en calentar la resistencia controlada por el microcontrolador. La resistencia se activaba cuando la temperatura era inferior a 36 grados Celsius y se desactivaba cuando superaba ese valor. Este control se realiza de manera cíclica para mantener la temperatura alrededor del punto de referencia. La figura muestra la evolución de la temperatura a lo largo del tiempo.

La gráfica muestra claramente tres fases distintas durante el experimento. En la fase de calentamiento inicial, que abarca desde 0 hasta aproximadamente 250 segundos, la temperatura inicial era de aproximadamente 27 grados Celsius. La resistencia se activa y la temperatura comienza a aumentar rápidamente debido a la energía proporcionada por la resistencia. Este aumento rápido es típico de la fase inicial en sistemas de calentamiento controlado.

En la fase de inercia y estabilización, que se extiende de los 250 a los 350 segundos, al llegar a los 36 grados Celsius, la resistencia se desactiva. Sin embargo, debido a la inercia térmica, la temperatura sigue aumentando ligeramente, alcanzando un pico de alrededor de 36.9 grados Celsius. Este comportamiento es típico en sistemas de control térmico debido al retardo en la respuesta del sistema a los cambios en la entrada de energía.

La fase de fluctuación, que se observa de los 350 a los 850 segundos, muestra que después del pico, la temperatura comienza a disminuir hasta que la resistencia se activa nuevamente. La temperatura fluctúa alrededor de los 36 grados Celsius con pequeñas oscilaciones. Estas

oscilaciones son resultado del método de control rudimentario utilizado, que se basa en un simple encendido y apagado de la resistencia.

El método de control empleado, aunque simple, ha demostrado ser efectivo para mantener la temperatura en torno al valor deseado. Sin embargo, las oscilaciones observadas indican que este método no es el más preciso ni eficiente. Las fluctuaciones son una consecuencia esperada debido a la naturaleza del control on/off (bang-bang control), que no permite un ajuste fino de la temperatura. Las oscilaciones de temperatura indican que el método podría no ser adecuado para aplicaciones que requieren una alta precisión. La eficiencia energética también puede verse comprometida debido a los ciclos repetidos de encendido y apagado de la resistencia. Por esto para futuras mejoras, se usará un control PI con salida PWM que será capaz de mantener la temperatura estable.

El experimento ha sido un éxito en términos de demostrar la funcionalidad básica del sistema de control de temperatura. Las fluctuaciones observadas están dentro de lo esperado dado el método de control simple utilizado. El análisis muestra que el sistema es capaz de mantener la temperatura cerca del valor deseado, aunque con oscilaciones. Este resultado es un paso importante en el desarrollo del prototipo y proporciona una base sólida para futuras mejoras. La implementación de un control más sofisticado como un controlador PID permitirá alcanzar un control de temperatura más estable y eficiente, lo cual es crucial para aplicaciones médicas y otros entornos críticos donde la precisión es vital que es nuestro objetivo.

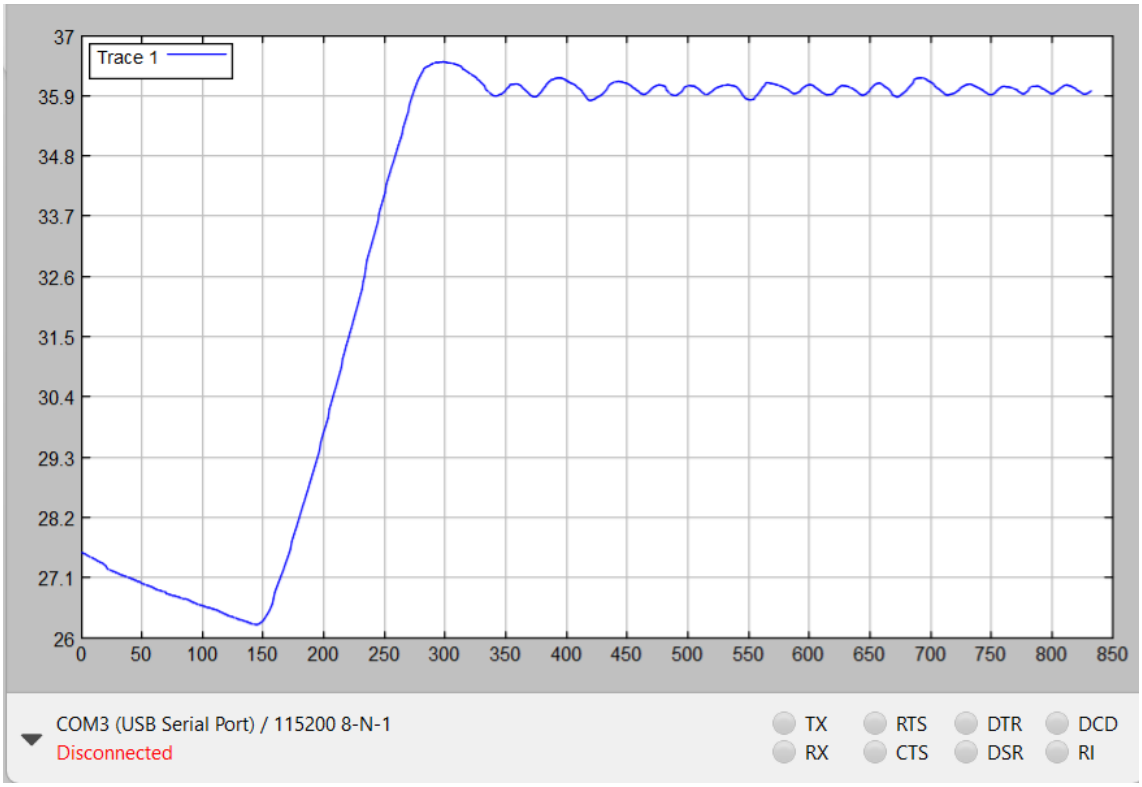



*Ilustración 32 – Gráfica con las mediciones de Temperatura*

# Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

En este proyecto, hemos logrado diseñar desde cero un prototipo básico y funcional que cumple con los objetivos iniciales planteados. El dispositivo desarrollado está compuesto por componentes simples pero esenciales: un microcontrolador, un sensor de alta precisión, una resistencia, un MOSFET y un driver. Este conjunto de elementos ha permitido construir un sistema capaz de controlar la temperatura de una placa de aluminio de manera efectiva. A lo largo del desarrollo, se ha demostrado que el prototipo es capaz de mantener la temperatura dentro de un rango deseado, activando y desactivando la resistencia de acuerdo con las lecturas del sensor. Este comportamiento cumple con los requisitos mínimos establecidos para el proyecto, confirmando la viabilidad del diseño y su aplicabilidad en escenarios donde el control térmico es crucial.

Uno de los mayores logros de este proyecto ha sido la creación de un sistema que, aunque básico, es completamente funcional y capaz de realizar la tarea para la cual fue diseñado. El éxito del prototipo radica en su simplicidad y en el uso de componentes accesibles, lo cual demuestra que se puede construir un dispositivo efectivo sin necesidad de recurrir a tecnologías complejas o costosas. Este logro es significativo ya que proporciona una base sólida sobre la cual se pueden realizar futuras mejoras y desarrollos.

El proyecto también ha demostrado la importancia de un control térmico preciso en aplicaciones médicas y otros entornos críticos. La capacidad de mantener la temperatura dentro de un rango específico es fundamental para garantizar la precisión en la medición y el diagnóstico. Este prototipo ha mostrado que, incluso con un método de control rudimentario, es posible alcanzar un nivel de precisión aceptable. No obstante, las oscilaciones observadas durante las pruebas indican que existe un amplio margen para mejorar la estabilidad y precisión del sistema.

**Trabajos Futuros:**

Si bien el prototipo desarrollado cumple con los objetivos mínimos, hemos identificado varias áreas clave para mejorar en futuras versiones. Una de las principales mejoras propuestas es la incorporación de dos placas de temperatura adicionales. Para ello, será necesario añadir dos resistencias y dos sensores más, lo que permitirá al sistema manejar tres temperaturas distintas de manera simultánea. Esta adición además de completar el sistema, proporcionará una mayor variedad en las condiciones de prueba, permitiendo una calibración más exacta de los dispositivos médicos.

Además de aumentar el número de placas y sensores, también se busca implementar un método de control más avanzado. La próxima versión del prototipo incluirá un control PI con salida PWM para excitar la resistencia. Este método de control mejorará significativamente la estabilidad del sistema, reduciendo las oscilaciones y asegurando que la temperatura se mantenga dentro de un rango estrecho alrededor del valor objetivo.

Finalmente, en términos de software, se explorarán algoritmos de control más sofisticados como el control PID (Proporcional-Integral-Derivativo). Este tipo de controlador puede ajustar de manera más precisa la potencia de la resistencia, respondiendo de forma proporcional a la diferencia entre la temperatura actual y la deseada, así como considerando la velocidad de cambio y la duración del error. La implementación de un control PID permitirá un ajuste más fino y preciso de la temperatura, reduciendo significativamente las oscilaciones y mejorando la estabilidad del sistema. Esto sería la evolución del PI, pero solo si es necesario. Si se ve que el PI cumple con los objetivos no será necesario implementar esto.

También se podrá añadir un display para mostrar la temperatura exacta de las tres placas y un lector de códigos QR e identificar cada termómetro con un código QR y así poder almacenar en una tarjeta de memoria en el aparato un histórico de las verificaciones de cada térmometro.

**Conclusión final:**

En resumen, el proyecto ha sido exitoso en demostrar la viabilidad de un prototipo básico y funcional para el control de temperatura. Los objetivos iniciales se han cumplido, y el sistema ha mostrado su capacidad para mantener la temperatura dentro de un rango deseado utilizando componentes simples y accesibles. No obstante, este es solo el primer paso en el desarrollo de un sistema de control térmico más avanzado y preciso. Las mejoras propuestas, que incluyen la adición de más placas y sensores, la implementación de métodos de control PI y la optimización tanto del hardware como del software, ofrecen un camino claro para futuras versiones del prototipo. Estas mejoras no solo aumentarán la precisión y estabilidad del sistema, sino que también lo harán más versátil y útil para aplicaciones médicas y otros entornos críticos donde el control térmico es esencial.

# Capítulo 8. BIBLIOGRAFÍA

[1] [Infrared Thermometer Calibration – A Complete Guide, 2012]. Fluke Calibration Infrared Thermometer Calibration – A Complete Guide, 2012: Infrared Thermometer Calibration – A Complete Guide | Fluke (flukecal.com)

[2] [Calibration for Clinical Institutions, 2014] Caregiver Non-contact Clinical Thermometer – The Professional Choice PRO-TF Series Calibration for Clinical Institutions, Gary O'Hara, 2014: Calibration — Clinical Non-Contact Thermometer (thermomedics.com)

[3] Microcontrolador dsPIC33FJ32MC202: 70283K.pdf (microchip.com)

[4] Comunicación i2c micro: 70000195g.pdf (microchip.com)

[5] Sensor de temperatura: *tmp117.pdf (ti.com)

[6] Driver: td310-1852141.pdf (mouser.es)

[7] MOS: td310-1852141.pdf (mouser.es)

# ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 objetivos globales adoptados por todos los Estados Miembros de las Naciones Unidas en 2015, como parte de la Agenda 2030 para el Desarrollo Sostenible. Estos objetivos buscan abordar una serie de desafíos globales, incluyendo la pobreza, la desigualdad, el cambio climático, la degradación ambiental, la paz y la justicia. Los ODS son los mostrados en la ilustración 33.


*Ilustración 33 – Todos los ODS establecidos*

Este proyecto se alinea con los siguientes objetivos:

**ODS 3: Salud y Bienestar**

El principal ODS con el que se alinea este proyecto es el ODS 3, que busca garantizar una vida sana y promover el bienestar para todos en todas las edades. La mejora en la precisión de los termómetros sin contacto tiene un impacto directo en la calidad de la atención médica.

La capacidad de medir la temperatura corporal con alta precisión es crucial para el diagnóstico y tratamiento de diversas condiciones médicas. Al asegurar que los termómetros proporcionen lecturas precisas y confiables, se mejora la calidad de los diagnósticos y, por ende, la efectividad de los tratamientos, contribuyendo significativamente al bienestar general de los pacientes.

**ODS 9: Industria, Innovación e Infraestructura**

El desarrollo de un prototipo de control de temperatura innovador también se alinea con el ODS 9, que promueve la construcción de infraestructuras resilientes, la industrialización inclusiva y sostenible, y la innovación. Este proyecto representa un avance tecnológico en el campo de la instrumentación médica, ofreciendo una solución más precisa y fiable para la medición de temperatura. La implementación de este tipo de tecnología puede impulsar la innovación en el sector de dispositivos médicos, fomentando el desarrollo de nuevos productos y soluciones que mejoren la calidad de la atención sanitaria.

**ODS 4: Educación de Calidad**

Aunque de manera indirecta, el proyecto también puede contribuir al ODS 4, que busca garantizar una educación inclusiva, equitativa y de calidad. El desarrollo de este prototipo ofrece oportunidades de aprendizaje y formación en áreas como la ingeniería electrónica, la programación de microcontroladores y el diseño de sistemas de control. Estudiantes y profesionales en formación pueden beneficiarse de este proyecto mediante la adquisición de conocimientos y habilidades prácticas que son esenciales en el campo de la tecnología médica y la ingeniería.

**ODS 12: Producción y Consumo Responsables**

El ODS 12, que promueve modalidades de consumo y producción sostenibles, también se ve reflejado en este proyecto. La creación de un dispositivo eficiente y preciso ayuda a reducir el desperdicio de recursos. Al minimizar los errores de medición y mejorar la eficiencia de los termómetros sin contacto, se contribuye a un uso más responsable y sostenible de los dispositivos médicos. Además, el diseño del prototipo con componentes accesibles y económicos también fomenta una producción más sostenible.

**ODS 17: Alianzas para Lograr los Objetivos**

Finalmente, el proyecto se alinea con el ODS 17, que busca fortalecer los medios de ejecución y revitalizar la alianza mundial para el desarrollo sostenible. La colaboración entre diferentes sectores, como la academia, la industria y el sector sanitario, es crucial para el éxito de este proyecto. El desarrollo y la implementación de tecnologías innovadoras en el ámbito médico requieren la cooperación y el intercambio de conocimientos entre diversas disciplinas y organizaciones. Este proyecto, al fomentar dichas alianzas, contribuye a la creación de un entorno más colaborativo y eficiente para alcanzar los ODS.

# ANEXO II: CÓDIGO FUENTE

## Main.c:

```
/*
 * File:   newmainXC16.c
 * Author: gonza
 *
 * Created on 30 de abril de 2024, 22:51
 */


#include "xc.h"
#include "pwm.h"
#include "config.h"
#include "idle.h"
#include "uart.h"
#include "i2c.h"

#define PIN_T_ALTA 15 // INC del driver
#define PIN_T_MEDIA 13 // INB
#define PIN_T_BAJA 11 // INA

#define sensor1 0x48
#define sensor2 0x49
#define sensor3 0x50
#define TEMPERATURE_REGISTER 0x00
#define CONF_REGISTER 0x01

#define TEMP1 36
#define TEMP2 38
#define TEMP3 40


int main(void){
    int n=0;


    inicializarReloj();


    TRISB &= ~(1 << 10); // led 4  ?Salida
    TRISB &= ~(1 << 7); // led 5  ?Salida

    TRISB &= ~(1 << PIN_T_ALTA); //RB15 salida
    TRISB &= ~(1 << PIN_T_MEDIA); //RB15 salida
    TRISB &= ~(1 << PIN_T_BAJA); //RB15 salida

    PORTB = 0;
```

```
    PORTB |= 1 << 7;
    PORTB |= 1 << 10;


    inicializarUART(115200);
    char cad[20];

    inicializarTareaIdle(10);


    uint16_t tmp2;



    I2C1Inicializa(I2C100KHZ);

    while(1){


        temperaturaC2 = LeerTemperatura(sensor2); // Lee la temperatura
del TMP117



        if(temperaturaC2 != -100){
            if(temperaturaC2 >= TEMP1){
                PORTB |= 1 << PIN_T_BAJA;

            }
            else{
                PORTB &= ~(1 << PIN_T_BAJA);

            }
        }

         if(temperaturaC2 != -100){
            if(temperaturaC2 >= (TEMP1-0.2) && temperaturaC2 <=
(TEMP1+0.2)) {
                PORTB |= 1 << 7;
                n++;
                n--;
                PORTB &= ~(1 << 10);
            }
            else{
                PORTB &= ~(1 << 7);
                n++;
                n--;
                PORTB |= 1 << 10;

            }
        }


        if(temperaturaC2 != -100){
            sprintf(cad, "T=%f\n",temperaturaC2);
```

```
            putsUART(cad);

        }


        tareaIdle();
    }


}
```

## I2c.c:

```
#define I2C400KHZ 1
#define I2C100KHZ 0



void I2C1Inicializa(int velocidad)
 {
    if(velocidad == 1){
        // 400 kb/s
        I2C1BRG = 92;
        I2C1CON = 0x8000; // I2C ON, con control de slew rate
    }else{
        // 100 kb/s
        I2C1BRG = 389;
        I2C1CON = 0x8200; // I2C ON, sin control de slew rate
    }
 }


void I2C1GeneraStart(void)
 {
    I2C1CONbits.SEN=1; //Generacondicióndestartenelbus
    while (I2C1CONbits.SEN);
    ;//Esperaelfinaldelacondición
 }

  void I2C1GeneraStop(void)
 {
    I2C1CONbits.PEN = 1; // Genera condición de stop en el bus
    while (I2C1CONbits.PEN);
    ; // Espera el final de la condición
 }


  void I2C1GeneraReStart(void)
```

```
{
    I2C1CONbits.RSEN = 1; // Genera condición de repeteaded
    // start en el bus
    while (I2C1CONbits.RSEN);
    ;//Esperaelfinaldelacondición
}


 uint8_t I2C1LeeByte(int ack)
{
    uint8_t dato;
    // Se activa la recepción del dato
    I2C1CONbits.RCEN = 1;
    while(I2C1CONbits.RCEN);
     // Espera el final de la recepción

    dato = I2C1RCV;
    //I2C1CONbits.RCEN = 0;
    I2C1CONbits.ACKDT = ack&1;
    I2C1CONbits.ACKEN = 1; // Se envía el ACK
    while(I2C1CONbits.ACKEN);
     // Espera el envío del ACK
    //I2C1CON &= ~0x1F; // Se envía el ACK

    return dato;
}

 int I2C1EscribeByte(uint8_t dato)
{
    I2C1TRN=dato;
    //while(IFS1bits.I2C1MIF==0)
    while (I2C1STATbits.TRSTAT);
    ; //Esperaelfinaldelenvío
    //IFS1bits.I2C1MIF=0; //Borroelflag
    return I2C1STATbits.ACKSTAT;
}


float LeerTemperatura(uint8_t dir_chip)
 {
    int chip_id =-1; //-1 indica error de recepción

    uint8_t conf1 = 0;
    uint8_t conf2 = 0;
    uint16_t reg_conf = 0;

    uint8_t byte1 = 0;
    uint8_t byte2 = 0;
    uint16_t entero = 0;

    I2C1GeneraStart();
    if(I2C1EscribeByte(dir_chip<<1) != 0){ //NACK
        I2C1GeneraStop(); // Abortamos
        return chip_id;
    }
```

```
    if(I2C1EscribeByte(CONF_REGISTER) != 0){ //NACK
       I2C1GeneraStop(); // Abortamos
        return chip_id;
    }
    I2C1GeneraReStart();

    if(I2C1EscribeByte(dir_chip<<1|1) != 0){ //NACK
        I2C1GeneraStop(); // Abortamos
        return chip_id;
    }

    conf1 = I2C1LeeByte(1);

    conf2 = I2C1LeeByte(0);

    I2C1GeneraStop();

    reg_conf = ((uint16_t)conf1) << 8;
    reg_conf = reg_conf | conf2;

    if((reg_conf & (1 << 13)) != 0){

        I2C1GeneraStart();
        if(I2C1EscribeByte(dir_chip<<1) != 0){ //NACK
            I2C1GeneraStop(); // Abortamos
            return chip_id;
        }
        if(I2C1EscribeByte(TEMPERATURE_REGISTER) != 0){ //NACK
           I2C1GeneraStop(); // Abortamos
            return chip_id;
        }
        I2C1GeneraReStart();

        if(I2C1EscribeByte(dir_chip<<1|1) != 0){ //NACK
            I2C1GeneraStop(); // Abortamos
            return chip_id;
        }

        byte1 = I2C1LeeByte(0);
        byte2 = I2C1LeeByte(1);

        entero = ((uint16_t)byte1) << 8;
        entero = entero | byte2;

        I2C1GeneraStop();
        return entero* 0.0078125;
    }
    else{
        I2C1GeneraStop();
        return -100;
    }

}
```

## Config.c:

```
// ------------------------------------------------------------------------
-------
// ------------------------- MÃ□DULO DE CONFIGURACIÃ□N  ---------------------
---------
// ------------------------------------------------------------------------
-------

/**
 * @file      config.c
 *
 * @author    Jaime Boal MartÃn-Larrauri
 *
 * @version   2.0.0
 *
 * @date      26/08/2016
 *
 * @brief     Funciones de configuraciÃ³n del microcontrolador.
 */

// ------------------------------------------------------------------------
-------

#include <xc.h>  // Incluye el microcontrolador seleccionado en el
proyecto
#include "config.h"

// ------------------------------------------------------------------------
-------
// --------------------------- BITS DE CONFIGURACIÃ□N -----------------------
--------
// ------------------------------------------------------------------------
-------

// FBS
#pragma config BWRP = WRPROTECT_OFF  // Permitir la escritura del
segmento de arranque flash
#pragma config BSS = NO_FLASH        // Eliminar el segmento de arranque
flash

// FGS
#pragma config GWRP = OFF            // No proteger la memoria de
programa contra escritura
#pragma config GSS = OFF             // No proteger el cÃ³digo

// FOSCSEL
#pragma config FNOSC = FRC           // Utilizar el oscilador interno
(FRC) en el arranque
#pragma config IESO = OFF            // Arrancar directamente con el
oscilador seleccionado
```

```
// FOSC
#pragma config POSCMD = NONE           // Desactivar el oscilador primario
#pragma config OSCIOFNC = ON           // Utilizar el oscilador secundario
como entrada y salida digital
#pragma config IOL1WAY = OFF           // Permitir mÃºltiples remapeos de
los pines
#pragma config FCKSM = CSECMD          // Permitir la conmutaciÃ³n del
reloj y deshabilitar el control de fallos

// FWDT
#pragma config FWDTEN = OFF            // Watchdog timer controlado por
software (Inicialmente deshabilitado)
#pragma config WDTPOST = PS32768       // Postescalado del watchdog
(1:32,768)
#pragma config WDTPRE = PR128          // Preescalado del watchdog (1:128)
#pragma config WINDIS = OFF            // Permitir resetear el watchdog en
cualquier momento

// FPOR
#pragma config FPWRT = PWR128          // Esperar 128 ms y resetear el
microcontrolador al enchufar la alimentaciÃ³n
#pragma config ALTI2C = OFF            // Utilizar los pines estÃ¡ndar
(SDA1 y SCL1) para el I2C
#pragma config LPOL = ON               // Los pines PWM low estÃ¡n activos
a nivel alto
#pragma config HPOL = ON               // Los pines PWM high estÃ¡n activos
a nivel alto
#pragma config PWMPIN = ON             // Controlar los pines de PWM desde
el registro PORTx al arrancar

// FICD
#pragma config ICS = PGD1              // Programar y depurar a travÃ©s de
los pines PGEC1 y PGED1
#pragma config JTAGEN = OFF            // Desactivar la interfaz para JTAG

// ------------------------------------------------------------------------
-------
// ---------------------------- FUNCIONES PÃBLICAS ----------------------
--------
// ------------------------------------------------------------------------
-------

void inicializarReloj(void) {
    CLKDIVbits.PLLPRE  =  0;  // Preescalado del PLL:   N1 = 2
    PLLFBD             = 41;  // Multiplicador del PLL: M = PLLFBD + 2 =
43
    CLKDIVbits.PLLPOST =  0;  // Postescalado del PLL:  N2 = 2

    // Funciones para desbloquear la escritura del registro OSCCON
    __builtin_write_OSCCONH(0x01);           // Nuevo reloj: FRC w/ PLL
    __builtin_write_OSCCONL(OSCCON | 0x01);  // Iniciar el cambio de
reloj

    while (OSCCONbits.COSC != 1);  // Esperar al cambio de reloj
```

```
    while (OSCCONbits.LOCK != 1);  // Esperar a que se sincronice el PLL
}
```

## Idle.c:

```
/**
 * @file      idle.c
 *
 * @author    JosÃ© Daniel MuÃ±oz FrÃas
 *
 * @version   1.0.1
 *
 * @date      08/09/2015
 *
 * @brief     Funciones de manejo de la tarea Idle para un sistema basado
en
 *            bucle de scan. Dicha tarea usa el timer 1 para llevar a cabo
la
 *            temporizaciÃ³n del bucle.
 */

#include "xc.h"
#include "config.h" // Se define aquÃ la frecuencia del micro FCY
#include "idle.h"

void inicializarTareaIdle(unsigned int t_s)
{
    unsigned char prescaler;
    TMR1 = 0;

    if(t_s<16){// no es necesario prescaler
        prescaler = 0; // Sin prescaler
        PR1 = ((unsigned long) t_s) * FCY / 10000L;
    }else if(t_s < 132){ // Se necesita un prescaler a 8
        prescaler = 1;
        PR1 = ((unsigned long) t_s) * (FCY/8) / 10000;
    }else if(t_s < 1058){ // Se necesita un prescaler a 64
        prescaler = 2;
        PR1 = ((unsigned long) t_s) * (FCY/64) / 10000;
    }else if(t_s < 4235){ // Se necesita un prescaler a 64
        prescaler = 3;
        PR1 = ((unsigned long) t_s) * (FCY/256) / 10000;
    }
    IFS0bits.T1IF = 0; // Borra el flag
    T1CON = 0x8000|(prescaler<<4); // Timer on, Prescaler
}

void tareaIdle(void)
{
    while (IFS0bits.T1IF == 0)
        ; // Espera fin del timer
    IFS0bits.T1IF = 0; // Borra el flag
}
```

## Pwm.c:

```
// ------------------------------------------------------------------------
-------
// --------------------------------- MÃ□DULO DE PWM  --------------------
--------
// ------------------------------------------------------------------------
-------

/**
 * @file      pwm.c
 *
 * @author    Jaime Boal MartÃn-Larrauri, JosÃ© Daniel MuÃ±oz FrÃas
 *
 * @version   1.0.1
 *
 * @date      07/09/2015
 *
 * @brief     MÃ³dulo encargado de generar seÃ±ales de PWM usando el
mÃ³dulo PWM
 *            del microcontrolador.
 */

// ------------------------------------------------------------------------
-------
#include <xc.h>
#include "pwm.h"
#include "config.h" // Para la definiciÃ³n de FCY

// ------------------------------------------------------------------------
-------
// ---------------------------------- FUNCIONES -------------------------
-------
// ------------------------------------------------------------------------
-------

void inicializarPWM(unsigned int bit_map, unsigned int frecuencia)
{
    P1TCONbits.PTSIDL   = 0;  // Continuar funcionando en modo suspensiÃ³n
    P1TCONbits.PTOPS    = 0;  // Postescalado: 1:1 (aumentar el tiempo
entre interrupciones)
    P1TCONbits.PTCKPS   = 0;  // Preescalado: 1:1
    P1TCONbits.PTMOD    = 0;  // Modo de funcionamiento: Free-running
(sÃ³lo subida)
    PWM1CON1 = 0x0700;        // Pares de pines independientes y
deshabilitados como MCPWM

    // Se configuran los bits de "override" para que se conecte la salida
del
```

```
    // generador de PWM con las salidas correspondientes del puerto B.
    if(bit_map & (1<<15)){ // PWM1L1
        P1OVDCONbits.POVD1L = 1;
    }
    if(bit_map & (1<<14)){ // PWM1H1
        P1OVDCONbits.POVD1H = 1;
    }
    if(bit_map & (1<<13)){ // PWM1L2
        P1OVDCONbits.POVD2L = 1;
    }
    if(bit_map & (1<<12)){ // PWM1H2
        P1OVDCONbits.POVD2H = 1;
    }
    if(bit_map & (1<<11)){ // PWM1L3
        P1OVDCONbits.POVD3L = 1;
    }
    if(bit_map & (1<<10)){ // PWM1H3
        P1OVDCONbits.POVD3H = 1;
    }

    setFrecuencia(frecuencia);

    // Factor de servicio inicial nulo
    P1DC1 = 0;
    P1DC2 = 0;
    P1DC3 = 0;

    PWMCON2bits.IUE    = 0;     // Sincronizar el cambio de factor de
servicio con el periodo
    PWMCON2bits.OSYNC  = 1;     // Sincronizar cambios en las salidas a
travÃ©s de OVDCON con el periodo
    PWMCON2bits.UDIS   = 0;     // Permitir la modificaciÃ³n del factor
de servicio y del periodo

    P1TCONbits.PTEN    = 1;  // Encender el mÃ³dulo generador de PWM
}

void setFrecuencia(unsigned int frecuencia)
{
   unsigned long ul_ptper;
    unsigned char uc_preescalado = 0;

    do {
        // PTPER = FCY * PREESCALADO * Periodo - 1 (mÃ¡x. 32767)
        ul_ptper = (unsigned long)(FCY/frecuencia)/(1 <<
2*uc_preescalado) - 1;
        uc_preescalado++;
    } while (ul_ptper > 32767 && uc_preescalado <= 3);

    if (ul_ptper > 32767)
        ul_ptper = 32767;

    P1TCONbits.PTCKPS = uc_preescalado - 1;
    P1TPER = ul_ptper & 0x7FFF;
}
```

```
// ---------------------------------------------------------------------
-------

void setDcPWM(unsigned int bit_map, unsigned int dc)
{
    unsigned int p1dcx;

    if (dc > 10000){
    dc = 10000;
    }
    // PTPER se multiplica por 2 porque un DC igual al
    // periodo implica un factor de servicio del 50%.
    p1dcx = (unsigned int)(2L*PTPER*dc/10000);

    if(bit_map & ( (1<<15)|(1<<14) )){ // se desea modificar el dc de un
pin del
        // canal 1
        P1DC1 = p1dcx;
    }
    if(bit_map & ( (1<<13)|(1<<12) )){ // se desea modificar el dc de un
pin del
        // canal 2
        P1DC2 = p1dcx;
    }
    if(bit_map & ( (1<<11)|(1<<10) )){ // se desea modificar el dc de un
pin del
        // canal 2
        P1DC3 = p1dcx;
    }
}

// ---------------------------------------------------------------------
-------

void activarPWM(unsigned int bit_map)
{
    int n;

    for (n=15;n>9;n-=2){ // Recorremos los bit cuyas salidas pueden tener
PWML
        if(bit_map &(1<<n)){
            PWM1CON1 |= ( 1<<((15-n)>>1) );
        }
    }
    for (n=14;n>9;n-=2){ // Recorremos los bit cuyas salidas pueden tener
PWML
        if(bit_map &(1<<n)){
            PWM1CON1 |= ( 1<<(((14-n)>>1)+4) );
        }
    }
}
void desactivarPWM(unsigned int bit_map)
{
    int n;
```

```
    for (n=15;n>9;n-=2){ // Recorremos los bit cuyas salidas pueden tener
PWML
        if(bit_map &(1<<n)){
            PWM1CON1 &= ~( 1<<((15-n)>>1) );
        }
    }
    for (n=14;n>9;n-=2){ // Recorremos los bit cuyas salidas pueden tener
PWML
        if(bit_map &(1<<n)){
            PWM1CON1 &= ~( 1<<(((14-n)>>1)+4) );
        }
    }
}
```

## Uart.c:

```
// ------------------------------------------------------------------------
-------
// --------- MÃ□DULO UART (Universal Asynchronous Receiver Transmitter) -
--------
// ------------------------------------------------------------------------
-------

/**
 * @file      uart.c
 *
 * @author    Jaime Boal MartÃn-Larrauri, JosÃ© Daniel MuÃ±oz FrÃas
 *
 * @version   1.1.0
 *
 * @date      08/09/2015
 *
 * @brief     MÃ³dulo encargado de gestionar las comunicaciones USB.
 */

// ------------------------------------------------------------------------
-------

#include <xc.h>
#include "uart.h"
#include "config.h" // Se define aquÃ FCY, necesario para configurar el
baudrate


// ------------------------------------------------------------------------
-------
```

```
// ------------------------------- PARÃMETROS -------------------------
--------
// --------------------------------------------------------------------
-------

/// TamaÃ±o de los vectores y colas
#define TAM_TR_UART  250  // Cola de transmisiÃ³n
#define TAM_REC_UART 250  // Cola de recepciÃ³n

/// Prioridad de las interrupciones (mÃ¡x. 7 - mÃn. 1)
#define PR_INT_TX 5  // TransmisiÃ³n
#define PR_INT_RX 6  // RecepciÃ³n

// --------------------------------------------------------------------
-------
// ---------------------- VARIABLES GLOBALES AL MÃDULO ----------------
--------
// --------------------------------------------------------------------
-------

static unsigned char uc_cola_transmision[TAM_TR_UART];
static unsigned char uc_cola_recepcion[TAM_REC_UART];
static unsigned int ui_icabeza_tr  = 0;
static unsigned int ui_icola_tr    = 0;
static unsigned int ui_icabeza_rec = 0;
static unsigned int ui_icola_rec   = 0;

// --------------------------------------------------------------------
-------
// ------------------- PROTOTIPOS DE LAS FUNCIONES PRIVADAS ------------
-------
// --------------------------------------------------------------------
-------

void __attribute__((interrupt, no_auto_psv)) _U1TXInterrupt(void);
void __attribute__((interrupt, no_auto_psv)) _U1RXInterrupt(void);
void ponerEnColaTransmisionUART(unsigned char uc_caracter);
void transmitirUART(void);

// --------------------------------------------------------------------
-------
// ---------------------------- FUNCIONES PÃBLICAS --------------------
--------
// --------------------------------------------------------------------
-------

void inicializarUART(unsigned long baudrate)
{
    U1BRG = (FCY/baudrate)/16 - 1;  // Velocidad de transmisiÃ³n

    U1MODEbits.STSEL   = 0;   // Bits de parada: 1
    U1MODEbits.PDSEL   = 0;   // Bits de datos: 8, Paridad: Ninguna
    U1MODEbits.ABAUD   = 0;   // Deshabilitar la detecciÃ³n de velocidad
    U1MODEbits.BRGH    = 0;   // Low Speed mode
    U1MODEbits.UEN     = 0;   // Usar sÃ³lo U1TX y U1RX
```

```
    U1MODEbits.USIDL    = 0;    // Mantener el mÃ³dulo activo en modo
reposo

    U1STAbits.UTXISEL0 = 0;    // Interrupcion de Tx: Cuando se vacÃa
U1TXREG
    U1STAbits.UTXISEL1 = 1;
    U1STAbits.URXISEL  = 0;    // Interrupcion de Rx: Al recibir un
caracter

    IPC3bits.U1TXIP = PR_INT_TX; // Prioridad de la interrupciÃ³n de
transmisiÃ³n
    IPC2bits.U1RXIP = PR_INT_RX; // Prioridad de la interrupciÃ³n de
recepciÃ³n

    IFS0bits.U1TXIF     = 0;    // Borrar la bandera de la interrupciÃ³n Tx
    IFS0bits.U1RXIF     = 0;    // Borrar la bandera de la interrupciÃ³n Rx

    IEC0bits.U1TXIE     = 0;    // Deshabilitar las interrupciones de
transmisiÃ³n
    IEC0bits.U1RXIE     = 1;    // Habilitar las interrupciones de
recepciÃ³n

    U1MODEbits.UARTEN   = 1;    // Habilitar el mÃ³dulo UART
    U1STAbits.UTXEN     = 1;    // Habilitar la transmisiÃ³n (SÃ³lo si
UARTEN = 1)

    // Se configuran los pines del micro por los que se conecta la UART
USB
    __builtin_write_OSCCONL(OSCCON & 0xBF);  // Desbloquear el PPS
    RPINR18bits.U1RXR = 5;  // Asignar U1RX al pin 14 que es RP5
    RPOR2bits.RP4R    = 3;  // Asignar U1TX al pin 11 que es RP4
    __builtin_write_OSCCONL(OSCCON | 0x40);  // Bloquear el PPS

    TRISB |= 1<<5; // Configura RB5 como entrada, ya que es el U1RX
}

// ---------------------------------------------------------------------
-------

void procesarUART(void)
{
    // Mientras haya datos por procesar
    while (ui_icola_rec != ui_icabeza_rec) {
        // A rellenar por la aplicaciÃ³n.
    }

    transmitirUART();
}

void putsUART(char *pcad){
    while(*pcad != '\0'){
        ponerEnColaTransmisionUART(*pcad);
        pcad++;
    }
    transmitirUART();
```

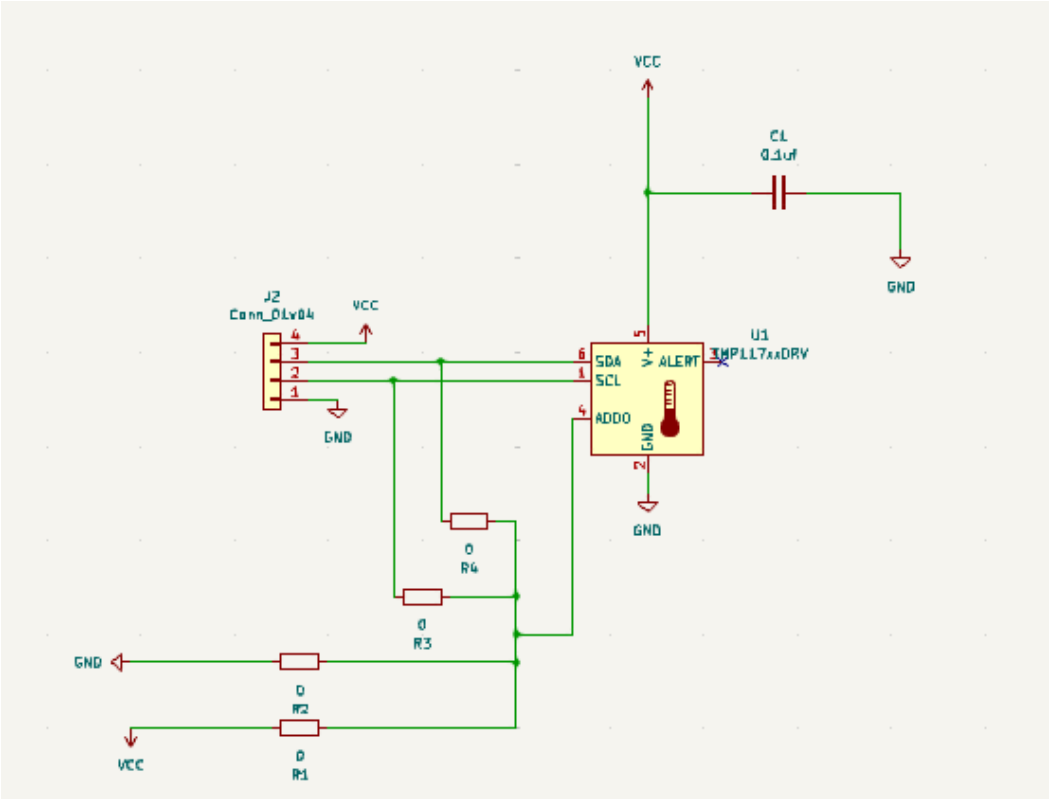
```
}

char getcharUART(void)
{
    char char_ret;

    if(ui_icola_rec != ui_icabeza_rec){ // Hay un carÃ¡cter nuevo
        char_ret = uc_cola_recepcion[ui_icola_rec];
        ui_icola_rec++;
        if(ui_icola_rec == TAM_REC_UART){
            ui_icola_rec=0;
        }
    }else{ // Cola vacÃa
        char_ret = '\0';
    }

    return char_ret;
}
// ---------------------------------------------------------------------------
-------
// ---------------------------- FUNCIONES PRIVADAS ----------------------
-------
// ---------------------------------------------------------------------------
-------

/**
 * Rutina de atenciÃ³n a la interrupciÃ³n de la UART asociada a la
transmisiÃ³n.
 */
void __attribute__((interrupt,no_auto_psv)) _U1TXInterrupt(void)
{
    IFS0bits.U1TXIF = 0;  // Borrar la bandera de la interrupciÃ³n

    if (ui_icola_tr != ui_icabeza_tr){ // Hay datos en la cola
        U1TXREG = uc_cola_transmision[ui_icola_tr++];

        if (ui_icola_tr == TAM_TR_UART){
            ui_icola_tr = 0;
        }
    }else{ // Cola vacÃa. Se deshabilita la interrupciÃ³n de transmisiÃ³n
para
           // que no estemos continuamente interrumpiendo al no tener
datos
           // que transmitir.
        IEC0bits.U1TXIE = 0;
    }
}

// ---------------------------------------------------------------------------
-------

/**
 * Rutina de atenciÃ³n a la interrupciÃ³n de la UART asociada a la
recepciÃ³n.
 *
```

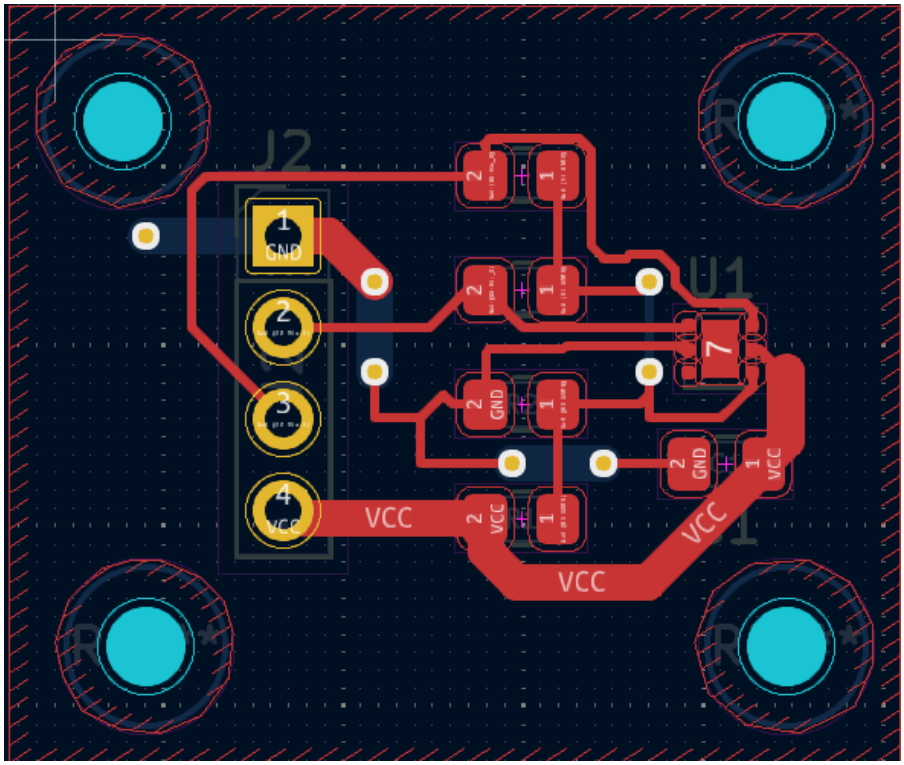
```
 * La rutina introduce el caracter recibido en la cola sÃ³lo si Ã©sta no
estÃ¡
 * llena. Si lo estÃ¡ enciende el LED RB12 de la tarjeta para avisar al
usuario.
 * en este caso el caracter recibido por la interrupciÃ³n se pierde.
 */
void __attribute__((interrupt,no_auto_psv)) _U1RXInterrupt(void)
{
    IFS0bits.U1RXIF = 0;  // Borrar la bandera de la interrupciÃ³n

    if( (ui_icabeza_rec+1 == ui_icola_rec) ||
        (ui_icabeza_rec+1 == TAM_REC_UART && ui_icola_rec == 0) ){
        // Cola llena
        PORTB &= ~(1<<12); // Enciende led RB12
    }else{
        uc_cola_recepcion[ui_icabeza_rec] = U1RXREG;
        ui_icabeza_rec++;
        if (ui_icabeza_rec == TAM_TR_UART){
            ui_icabeza_rec = 0;
        }
    }
}

// ------------------------------------------------------------------------
-------

/**
 * Coloca un dato en la cola de transmisiÃ³n.
 *
 * La funciÃ³n introduce el carÃ¡cter en la cola sÃ³lo si Ã©sta no estÃ¡
llena. Si
 * lo estÃ¡ enciende el LED RB12 de la tarjeta para avisar al usuario. En
este
 * caso el caracter enviado a la funciÃ³n se pierde.
 *
 * @param[in]  uc_caracter  Caracter que se quiere poner en cola.
 */
void ponerEnColaTransmisionUART(unsigned char uc_caracter)
{
    if( (ui_icabeza_tr+1 == ui_icola_tr) ||
        (ui_icabeza_tr+1 == TAM_TR_UART && ui_icola_tr == 0) ){
        // Cola llena
        PORTB &= ~(1<<12); // Enciende led RB12
    }else{
        uc_cola_transmision[ui_icabeza_tr] = uc_caracter;
        ui_icabeza_tr++;
        if (ui_icabeza_tr == TAM_TR_UART){
            ui_icabeza_tr = 0;
        }
    }
}

// ------------------------------------------------------------------------
-------
```

```
/**
 * EnvÃa todos los datos almacenados en la cola de transmisiÃ³n.
 */
void transmitirUART(void)
{
    IEC0bits.U1TXIE = 1;  // Habilitar las interrupciones de transmisiÃ³n
    IFS0bits.U1TXIF = 1;  // Provocar una interrupciÃ³n para emprezar a
transmitir
}
```

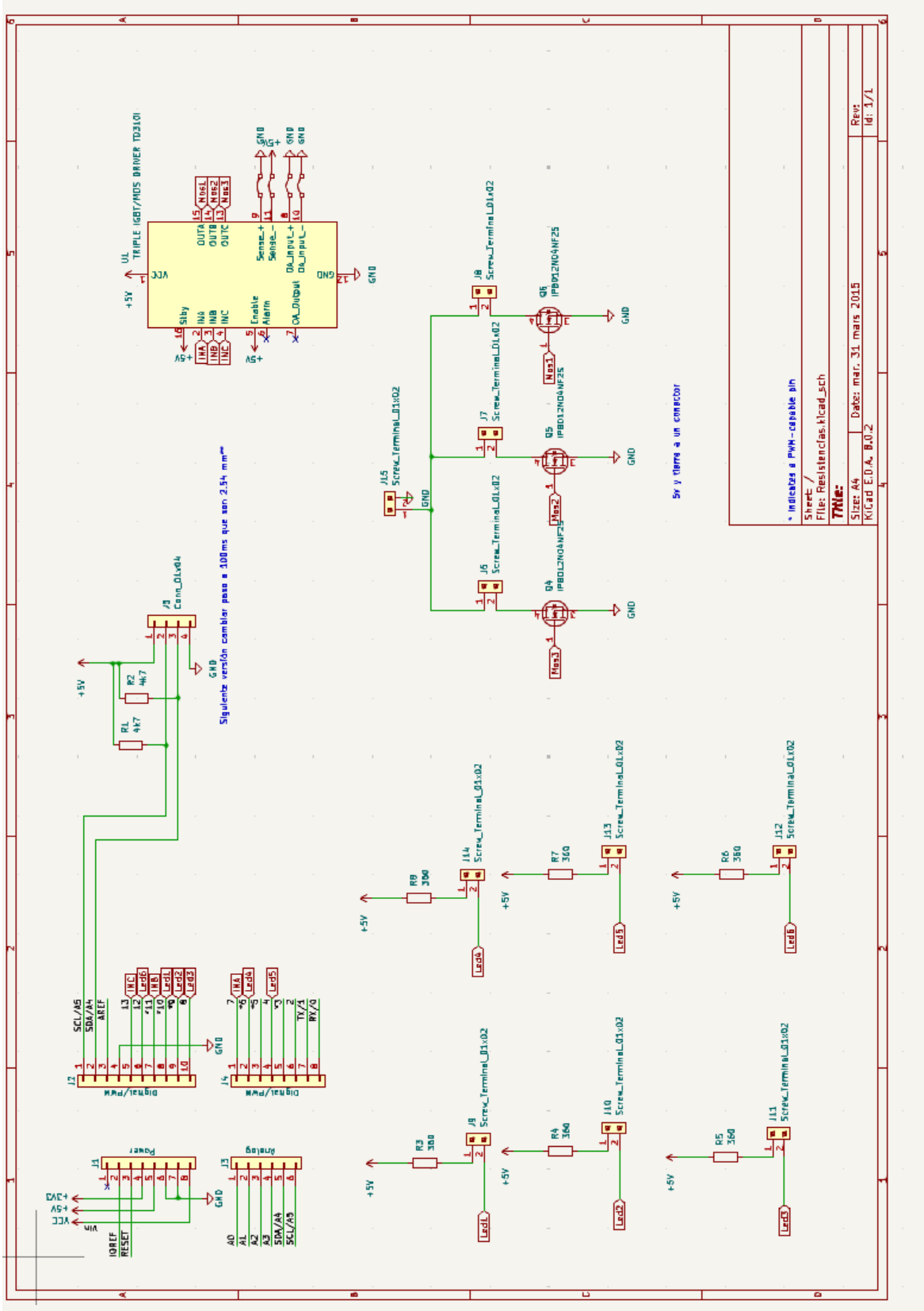
# ANEXO III: PLANOS DEL CIRCUITO

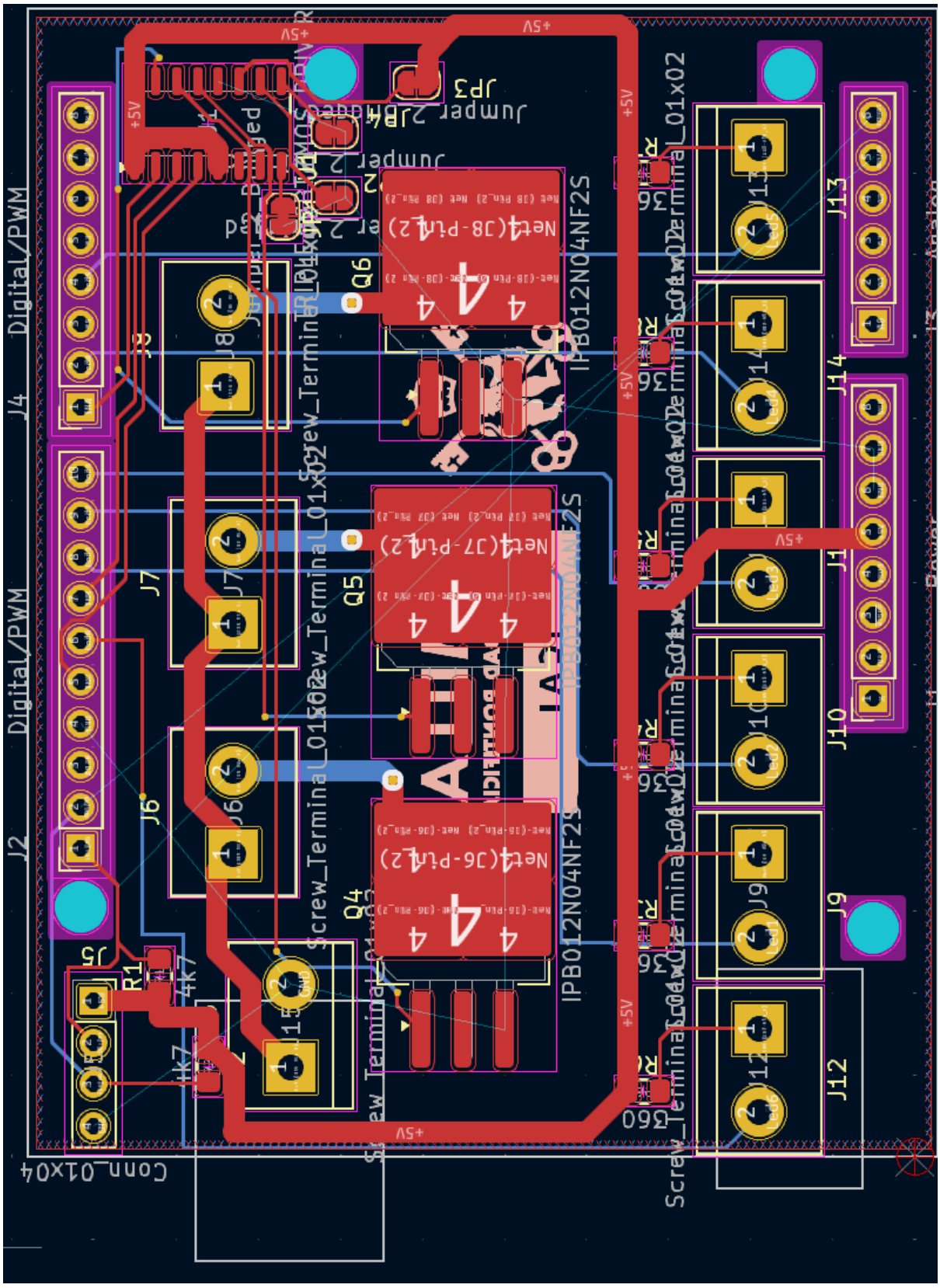
Sensor de Temperatura:






Circuito principal:


TRIPLE IGBT/MOS DRIVER TD310I
Siguiente versión cambiar paso a 100ms que son 2.54 mm
5v y tierra a un conector
* Indicadas a PWM-capable pin
Sheet: /
File: Resistencias.kicad_sch
Title:
Size: A4
Date: mar, 31 mars 2015
KiCad E.D.A. 8.0.2
Rev:
Id: 1/1


J4 Digital/PWM
J2 Digital/PWM
JP3
Jumper
Q6
Q5
Q4
IPB012N04NF2S
J13
J14
J1
J10
J9
J12
J7
J6
J5
R1
4K7
Conn_01x04
+5V