Trabajo Fin de Grado

Termostato inteligente

Autor

Daniel Rosa Corral

Director/es

Rubén Blasco Marín
Roberto Casas Nebra

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Daniel Rosa Corral,

con nº de DNI 18451678-C en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Termostato inteligente

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 19 de Septiembre del 2016

Fdo: Daniel Rosa Corral

Resumen

En la actualidad, el sector de la domótica ha experimentado un gran crecimiento a causa de la necesidad de obtener una mayor eficiencia energética en hogares y empresas. Gracias a la tecnología inalámbrica y al gran crecimiento de dispositivos conectados a internet, la domótica es mucho más atractiva y potente, ya que nos permite tener mucha más flexibilidad, precisión y seguridad en el control de los sistemas.

Un sector que ha evolucionado debido a este crecimiento, son los sistemas de climatización. Aunque la nueva generación de termostatos inteligentes no ha llegado a la mayoría de hogares por su coste, prometen un gran ahorro gracias a sus características.

En este proyecto se ha desarrollado el prototipo de un termostato inteligente con gestión de rutinas, capaz de aprender cuando las personas están en casa y anticiparse a ello consiguiendo un mayor confort. Además, gracias a su pantalla táctil y su control inalámbrico, se consigue un interfaz más sencillo, agradable y flexible.

Para llegar a este prototipo se han investigado los actuales termostatos inteligentes y sus diversas características, y tras analizarlas se han definido las especificaciones de nuestro dispositivo. A continuación, se ha diseñado el circuito electrónico, generando los esquemáticos del mismo. Esta tarea se ha compaginado con la búsqueda y selección de los componentes necesarios para que el dispositivo cumpla las especificaciones planteadas. Una vez finalizada esta tarea se ha comenzado el diseño, fabricación, montaje y puesta a punto de la PCB del prototipo. Por último, se ha acometido la programación del firmware a partir del diseño del diagrama de flujo que describe su funcionamiento.

El resultado final del proyecto ha sido un prototipo funcional de termostato, con capacidad de conexión inalámbrica (WiFi) e interfaz basado en pantalla táctil, que aprende los hábitos (hora de llegada a casa) del usuario y se adapta a ellos. Todo ello sin perder la capacidad de funcionar como un termostato convencional cuando sea necesario.

Índice

1. Introducción

1.1. Introducción

Este proyecto se centra en el campo de la domótica y el hogar inteligente, en concreto, en la parte de climatización de la vivienda mediante termostatos inteligentes, ya que la inserción de la electrónica en el hogar está evolucionando rápidamente.

El T.F.G. se ha realizado con el grupo HOWLab[1] (Human Openware Research Lab). HOWLab es un grupo de investigación universitario, cuyo objetivo principal es la investigación y el desarrollo de tecnologías centradas en las personas y sus entornos.

El trabajo no se apoya en ningún trabajo previo, parte desde cero, siguiendo la línea de la asignatura de **Laboratorio de diseño electrónico** del grado en Ingeniería electrónica y automática, donde el objetivo es conseguir un prototipo funcional.

1.2. Objetivos y alcance

El objetivo de este T.F.G. es el diseño e implementación de un termostato inteligente, controlable inalámbricamente, que sea capaz de detectar las rutinas diarias del usuario, anticipándose a ellas.

Para alcanzar este objetivo el trabajo se ha estructurado en las siguientes tareas:

- **Búsqueda de información:**

  -Enfoque del proyecto: En esta primera fase se busca información sobre domótica y el concepto de hogar inteligente, y se decide que la parte de la domótica en la que se quiere trabajar son los termostatos inteligentes.

  -Documentación sobre termostatos: Primero se estudia el concepto de termostato convencional y su funcionamiento, y después se investigan los termostatos inteligentes que hay en el mercado y las características que cada uno ofrece.

  -Especificaciones: Una vez estudiados los termostatos inteligentes que hay en el mercado, se especifica que características principales debe tener nuestro prototipo teniendo en cuenta los recursos y el tiempo disponibles.

- **Diseño de circuito electrónico:**

  -Diagrama de bloques y elección de componentes: En esta fase se diseña un diagrama de bloques general para que el prototipo tenga las

[1] http://howlab.unizar.es/

especificaciones acordadas y se buscan los componentes que más se adapten a las necesidades teniendo en cuenta sus prestaciones y precio.

-**Circuito esquemático:** Se diseña el circuito de cada bloque, teniendo en cuenta las características de los componentes elegidos. Esta fase es una de las más importantes ya que un error en este punto tan temprano puede acarrear muchos problemas en las siguientes fases. El diseño se realiza con **CircuitMaker[2],** versión gratuita de Altium[3].

-**Diseño de la PCB (***Printed Circuit Board***):** Una vez finalizado el esquemático se exportara al editor de PCB, en el cual se realiza el diseño de la misma (ubicación de componentes, trazado y dimensionado de pistas, etc.). A continuación se generan los correspondientes archivos para su posterior fabricación.

- **Programación:** Esta fase va en paralelo con el diseño de la PCB, se trabaja con **Flyport** [1] y el kit **Grove[4]** y se programa una primera versión de prueba sin PCB.

  En esta fase también se estudia la comunicación con la pantalla y se programa el interfaz que finalmente tendrá el termostato, con el programa **Workshop** [5]de 4D Systems.

- **Preparación del prototipo:** Una vez que las fases anteriores están acabadas, se prepara el prototipo, primero se sueldan todos los componentes y se revisan las soldaduras, luego se testea la PCB para comprobar que funciona, después se preparan los conectores, se monta el prototipo completo y se transfieren los programas al microcontrolador y a la pantalla.

- **Pruebas de campo:** Una vez que está todo listo se monta en una caja y se realizan pruebas para corregir posibles errores.

- **Preparación de la documentación:** Durante todo el proyecto se documentan todas las fases para, después, recopilarlas en una memoria final.

[2] http://circuitmaker.com/
[3] http://www.altium.com/
[4] http://wiki.openpicus.com/index.php/Grove_Kits
[5] http://www.4dsystems.com.au/product/4D_Workshop_4_IDE

1.3. Planificación

El siguiente diagrama de Gantt (Ilustración 1), recoge la planificación del proyecto:

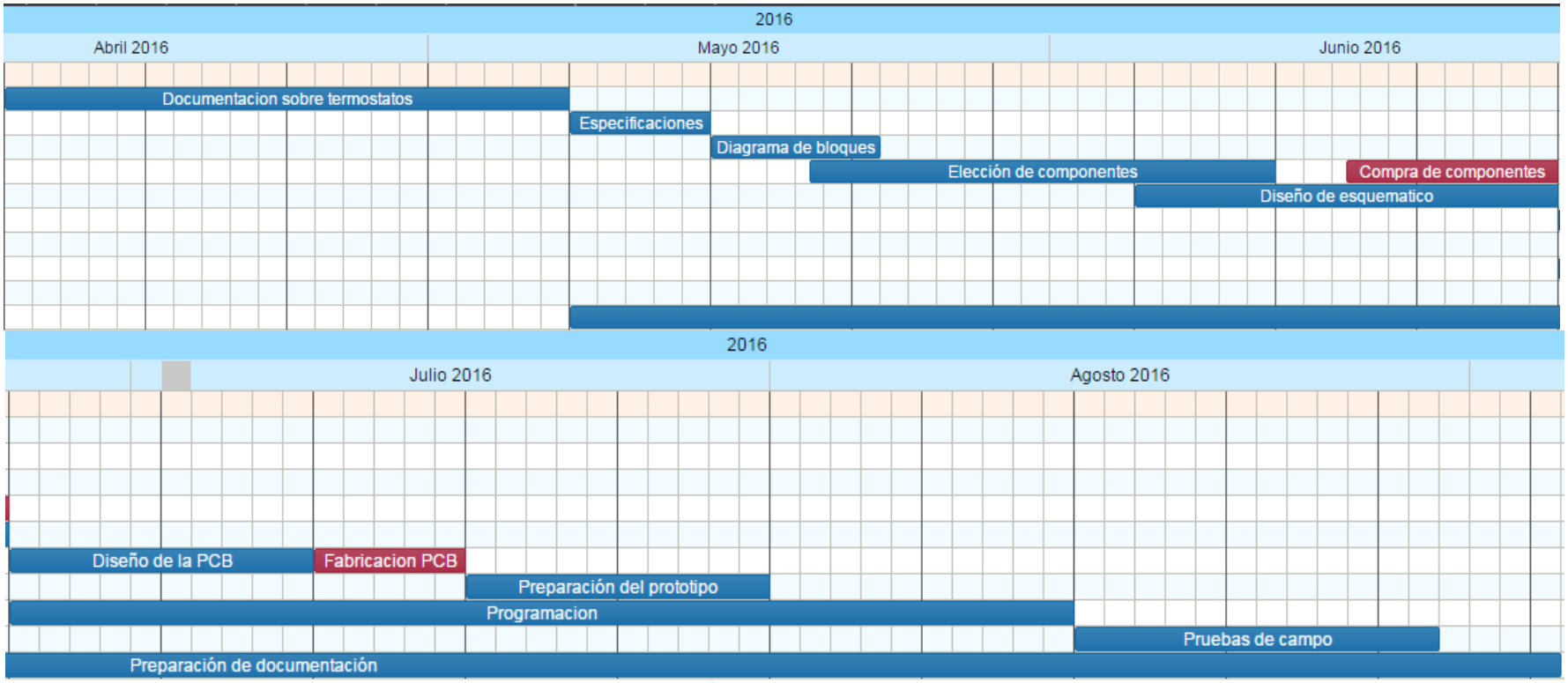


Ilustración 1: Diagrama de Gantt

1.4. Contenido de la memoria

Para documentar el T.F.G. se ha redactado la presente memoria donde se plasma todo el trabajo realizado: búsqueda de información, diseño e implementación de prototipo, así como su validación. Se ha estructurado en los siguientes capítulos:

- **Capitulo 1. Introducción:** Se describe el objetivo y alcance del proyecto, el contexto en que se realiza y la forma en la que se aborda el problema. También se describe brevemente el contenido de los demás capítulos.

- **Capitulo 2. Estado del arte:** En este primer capítulo se recopila la información en torno al concepto de termostato y su evolución, y también las alternativas que hay en el mercado actualmente**.**

- **Capitulo 3. Diseño hardware:** Se documenta como se ha diseñado el hardware del prototipo, desde el concepto inicial hasta la PCB soldada, pasando por el diagrama de bloques, diseño del circuito esquemático y diseño de la PCB.

- **Capitulo 4. Diseño firmware:** Se describe el funcionamiento del termostato bloque por bloque y las comunicaciones necesarias para su funcionamiento.

- **Capitulo 5. Conclusiones:** Se contrastan los objetivos iniciales del proyecto con lo logrado al finalizarlo, y se describen las líneas futuras en las que podría avanzar el prototipo en posteriores versiones.

- **Anexo I:** En este anexo se encuentran todos los planos referidos al diseño del hardware.

- **Anexo II:** Se detalla el coste de los materiales necesarios para fabricar una unidad.

- **Anexo III:** Se muestra el código de programación del termostato, y el interfaz de la pantalla.

2. Estado del arte

2.1. Introducción

La domótica es un conjunto de tecnologías aplicadas al control y a la automatización de la vivienda, que ayuda al usuario a mejorar su confort y seguridad [1].

El hogar digital, incorpora un sentido más amplio que la domótica. "No consiste simplemente en la instalación de dispositivos para controlar determinadas funciones en los edificios (viviendas, industrias, oficinas...) tales como alarmas, iluminación, climatización, control energético... sino que, al incorporar las tecnologías de la Información y las Telecomunicaciones, permite controlar y programar todos los sistemas, tanto en el interior de la vivienda como desde cualquier lugar, en el exterior de la misma, a través de distintas redes como Internet, mediante una interfaz apropiada"[2].

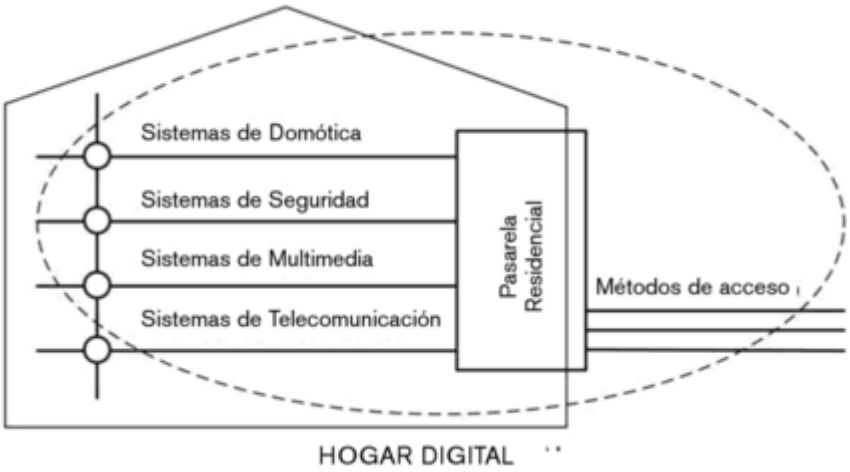


Ilustración 2: Hogar digital[3]

Este trabajo se centra en la parte de la climatización del hogar, en especial en los termostatos inteligentes. Estos equipos son la evolución al hogar digital de los termostatos programables, ya que los termostatos programables funcionan de forma autónoma.

2.2. Termostatos

Las primeras versiones de termostatos simples empezaron a salir a la luz a mediados del siglo XIX y consistían en elementos metálicos que se expandían o encogían y accionaban un dispositivo de paso. Tras el descubrimiento y difusión de la electricidad a principios del siglo XX sufren una continua evolución, pasando por los termostatos de mercurio, termostato de líquido, termostato bimetálico, termostato de gas a presión entre otros.

En la actualidad existen principalmente dos tipos de termostatos, el termostato convencional que podemos encontrar aproximadamente en el 90% de las casas y el inteligente, que poco a poco va ganando terreno [4].

El termostato convencional normalmente funciona a pilas y está conectado a la caldera con dos cables, si unimos estos dos cables la caldera empieza a funcionar, esto se consigue gracias a un relé que tiene en el interior. Cuando la temperatura es inferior a la deseada, el relé conecta los dos cables y, cuando la temperatura es superior, el relé desconecta los dos cables.

Un termostato inteligente se basa en el mismo principio de funcionamiento que uno convencional pero, además, ofrece multitud de funciones e incorporar una serie de sensores para mejorar la comodidad y la calidad de vida, y por otro lado, proporcionan un ahorro en el consumo de energía, ya que el sistema es más eficiencia [5].

Una de las grandes ventajas de los termostatos inteligentes es la conexión a la red WiFi (*wireless fidelity*), por lo que los podemos controlar desde nuestro teléfono o *tablet* desde cualquier lugar. Gracias a su conexión a internet también son capaces de detectar cuando vamos a llegar a casa, ya que el teléfono tiene nuestra posición geográfica y el termostato puede anticiparse para que la temperatura del hogar sea ideal cuando lleguemos.

Además de medir temperatura también incorporan sensores de humedad, ya que cuando es alta, la sensación de frio aumenta, sensores de luminosidad, para saber cuándo es de día o de noche, o sensores de presencia y proximidad para saber si hay alguien en casa. También incorpora funciones como el ajuste de histéresis, ya que dependiendo de qué tipo de calefacción se conecte, le cuesta calentar más o menos que a otra y así evitamos escalones de temperatura; otra función que incorpora es el aprendizaje de hábitos, el termostato aprende nuestros movimientos mediante los sensores y se adapta a nosotros por lo que puede funcionar de forma autónoma.

En el mercado actual la mayoría de termostatos son termostatos digitales. Se puede encontrar una gran gama de productos con diferentes funciones y precios. La gran ventaja que tienen respecto a los termostatos mecánicos es que se pueden programar seleccionando el día y la hora a la que queremos que se encienda la calefacción, y se podrían considerar como el antecesor de los termostatos inteligentes.

Ilustración 3: Nest. Fuente: https://nest.com/

En cuanto a termostatos inteligentes, cada vez se encuentra más variedad en el mercado, como por ejemplo el termostato **Nest**[6] **(**Ilustración 3**)** de Google que es sin duda el más conocido gracias a su elegante diseño, su app propia y sus frecuentes actualizaciones, también tenemos otras alternativas como **Tado**[7] que es la alternativa europea a Nest, y **Momit Smart**[8] creado por un grupo de ingenieros españoles. En la siguiente tabla (Tabla 1) podemos ver las características más relevantes de estos termostatos:

Termostato	Control App	Control Rutina	Presencia	Geolocalización GPS móvil	Registro consumo	Previsión meteorológica	Ahorro energético	Precio
Nest	✓	✓	✓	✓	✓	✗	12%	218€
Tado	✓	✓	✗	✓	✗	✓	31%	246€
Momit Smart	✓	✓	✓	✓	✓	✓	30%	199€

Tabla 1: Características Termostatos

En conclusión, el mercado de los termostatos inteligentes aún es pequeño, pero está en constante crecimiento, cada vez tienen mejores características que hacen que su uso sea más sencillo y eficiente y sus precios, que son su mayor inconveniente, están en torno a los 200€.

[6] https://nest.com/
[7] https://www.tado.com/es/
[8] https://www.momit.com/es/termostato/smart

2.3. Especificaciones

Tras el análisis de los termostatos inteligentes, sus funcionalidades y características más relevantes, se elaboran las especificaciones funcionales del prototipo, considerando el alcance del proyecto. El prototipo ha de ser capaz de:

- Controlar el encendido y apagado de la caldera en base a la temperatura ambiente y a la consigna introducida por el usuario.
- Fijar la temperatura de consigna (temperatura deseada por el usuario).
- Ser controlado inalámbricamente por WiFi. Por ejemplo mediante un *Smartphone*, *Tablet* o PC.
- Predecir la rutina del hogar (cuando hay personas en la vivienda) y anticiparse a ella, encendiendo la calefacción cuando sea necesario.
- Medir temperaturas en un rango de 0 a 40ºC y humedad relativa entre el 10 al 90 % (Rango habitual en viviendas).
- Funcionar a pilas, batería o conectado a un bus de continua.
- Intuitivo y fácil de manejar.

3. Diseño electrónico

3.1. Introducción.

En este capítulo, se plantea el diseño e implementación del hardware del prototipo en función de las especificaciones definidas en la sección 2.3. Para afrontar el mismo se ha optado por un planteamiento modular, donde se han identificado los principales bloques del sistema y se ha abordado su diseño de forma individual.

Para cada uno de estos bloques se han planteado distintas opciones, seleccionando los componentes más adecuados en cada caso y definiendo su esquemático.

3.2. Diagrama de bloques

El siguiente diagrama (Ilustración 4) muestra como se ha abordado el diseño, en él se puede identificar los siguientes bloques:

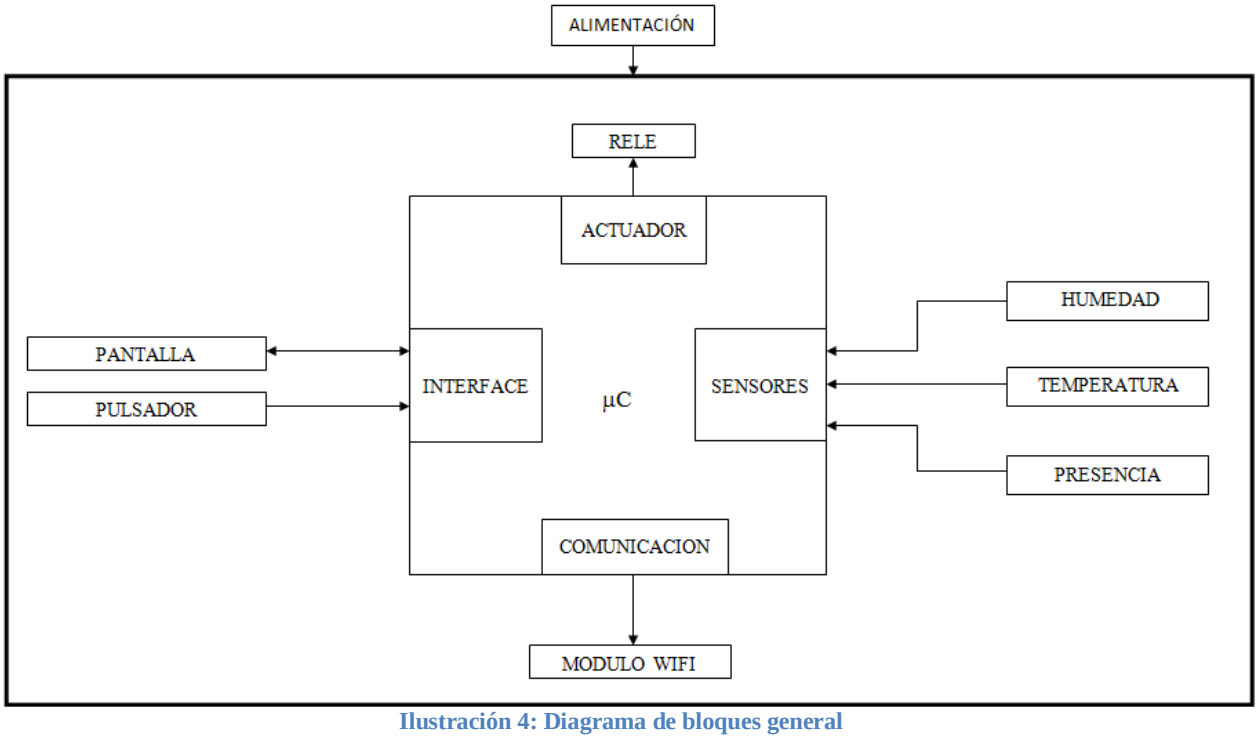


Ilustración 4: Diagrama de bloques general

- **Actuador**: Es el encargado de encender y apagar la calefacción cuando sea necesario.
- **Interface:** Se encarga de la interacción entre el usuario y el termostato.
- **Comunicación:** El modulo WiFi permite establecer la comunicación entre el termostato y los dispositivos con WiFi del usuario mediante un servidor web.

- **Sensores:** Este bloque es el encargado de proporcionar al termostato la información necesaria de su entorno.

- **Microcontrolador:** Es el núcleo del sistema y el encargado de controlar los componentes de cada bloque.

- **Alimentación:** Este bloque se encarga de suministrar el voltaje y corriente pertinente a todos los componentes electrónicos del dispositivo.

En la siguiente sección se detalla el diseño de cada uno de estos bloques, seleccionando los componentes más adecuados y diseñando el circuito electrónico.

3.2.1. Actuador

Existen dos tipos de control de caldera según su conexión: **NO** (normalmente abierto) y **NC** (normalmente cerrado) más la conexión común que tienen las dos, por lo que es necesario un relé con dos posiciones y según qué tipo de calefacción sea se conectará de una manera u otra, como se puede apreciar en la Ilustración 5.

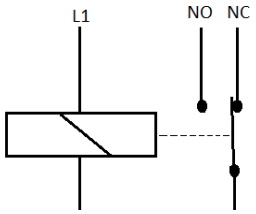


Ilustración 5. Conexión NO y NC.
Fuente: https://arrizen.wordpress.com/

Inicialmente, dentro de la gran variedad de relés que existen, se optó por un relé de **estado sólido**, ya que una de sus principales características es que no hacen ruido. Estos relés no conmutan mediante una bobina sino mediante un acoplador óptico, con lo que se puede evitar el molesto ruido de conmutación del relé que produciría en un hogar. El problema de este tipo de relé es su consumo, ya que para habilitar la conexión NO hay que aplicarle constantemente corriente, por lo que este tipo de relé se descartó, ya que el consumo es un aspecto crítico en este prototipo, como se puede apreciar en la sección 3.2.5.

Finalmente se optó por un relé de tipo ***latching,*** ya que este tipo de relés conmutan mediante un impulso eléctrico, consumiendo energía solamente en ese instante, lo que permite ahorrar energía.

En este caso se ha elegido el relé DS1E-SL2-DC3V [7] con doble bobina, *set* y *reset* para conmutar más fácilmente con la siguiente configuración:

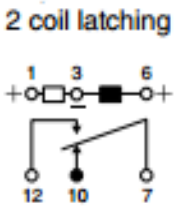


Ilustración 6. Esquema relé.
Fuente: http://www.mouser.com

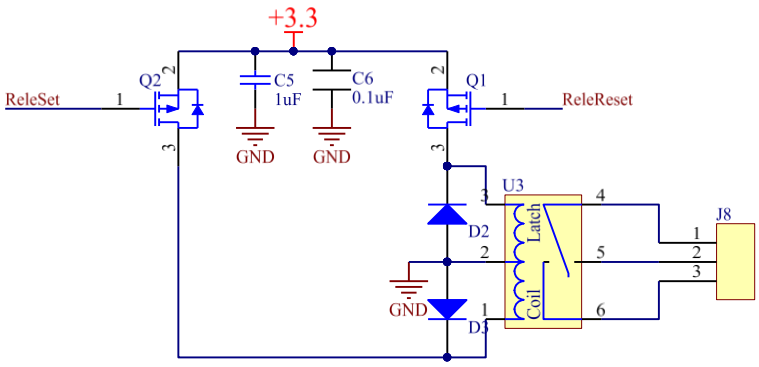


Ilustración 7: Esquemático relé

Se utilizan dos etapas transistorizadas, una para el *Set* y otra para el *Reset* del relé, ya que el microcontrolador no es capaz de proporcionar la suficiente corriente para que conmuten. En paralelo con las bobinas, se colocan dos diodos para facilitar la descarga de las mismas y, por último, un condensador para suministrar los picos de corriente que necesita el relé para conmutar.

El relé se une a un conector de tres pines, uno de los dos cables de la caldera se conecta al pin central que es el COM (común), y el otro cable se conecta al NO o NC dependiendo del tipo de caldera.

3.2.2. Sensores

3.2.2.1. PIR

El sensor PIR (*Passive Infrared Sensor*) es el que permitirá al dispositivo tener una gestión de rutinas en el hogar, ya que detecta la presencia de personas. Se ha elegido el sensor **Panasonic EKMC1601111** [7] por su sencillez y disponibilidad. Su configuración es simple ya que su salida está filtrada y tan solo hay que poner un condensador entre alimentación y masa, tal y como recomienda el fabricante.

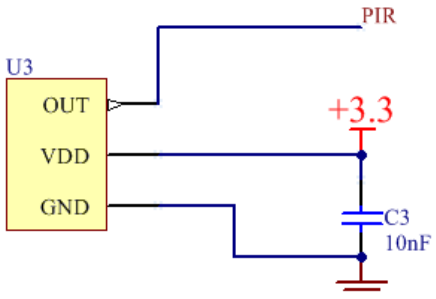


Ilustración 8. Esquemático PIR

3.2.2.2. Temperatura y humedad

Para llevar a cabo la medida de la temperatura, existen multitud de sensores con diferentes tecnologías en el mercado, como pueden ser los termopares, RTD (*Resistance Temperature Detector*), termistores y sensores integrados. La mayoría de estos últimos incorporan también la medida de humedad, la cual es un parámetro importante en el dispositivo, ya que junto a la temperatura son los dos campos principales del confort térmico en el hogar, aunque no se va actuar sobre ella.

Se han analizado dos alternativas, **Sensirion SHT71[9] y Adafruit DHT22** [7]**:**

Ilustración 9: Adafruit DHT22. Fuente:www.adafruit.com/

Ilustración 10: Sensirion SHT71. Fuente: www.sensirion.com

Tras analizar las características de cada uno de los sensores y la necesidad de especificaciones que necesita el proyecto, se llegó a la conclusión de que el sensor DHT22 era más idóneo ya que cumple todas las características necesarias, es 3 veces más barato y solo tiene una línea de datos.

Su conexión es directa ya que el integrado incorpora una resistencia de 5K1 entre alimentación y la línea de datos, recomendada por el fabricante para que la comunicación sea correcta. Por lo que solo hace falta alimentarlo y conectarlo al microcontrolador para que funcione correctamente.

[9] https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity_Sensors/Sensirion_Humidity_Sensors_SHT7x_Datasheet_V5.pdf

3.2.3. Interface

Tras buscar información sobre los termostatos comerciales, se analizaron dos alternativas:

a) Una pantalla LCD con pulsadores para poder navegar por el interfaz. Las principales ventajas de esta alternativa son su bajo coste, su fácil manejo y bajo consumo de energía comparada con una pantalla táctil. Mientras que es una alternativa poco atractiva de aspecto y con poco alcance.

b) Un módulo de pantalla táctil que es mucho más potente que la otra alternativa, ya que nos proporciona muchas más funciones, más versatilidad, posibilidad de trabajar con imagen, sonido y video, y un aspecto más moderno. Por lo contrario, es una alternativa más cara, más difícil de manejar y con un consumo superior.

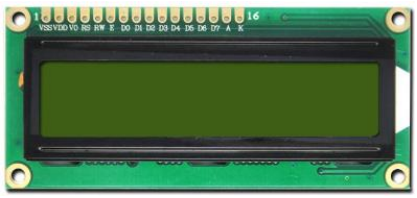


Ilustración 12. Display LCD. Fuente: www.buydisplay.com/

Ilustración 11. Modulo pantalla táctil Fuente: www.mouser.es/

Comparando las dos alternativas y los termostatos comerciales, se ha optado por la opción (b) de módulo de pantalla táctil y se selecciona **uLCD-24PTU** [7] del fabricante **4D System.** Esta es una pantalla de tipo resistivo, con una resolución de 240x320 de 2.4 pulgadas, con opción de alimentación a batería de 3.7 V.

En el Anexo I.I. Esquemático, se muestra su conexión con el resto de componentes, formada por su alimentación y bus de control, que en este caso son dos pines de una UART (*Universal Asynchronous Receiver-Transmitter*), TX y RX.

Para saber cuándo la pantalla tiene que ser encendida, se estudia la posibilidad de hacerlo mediante el PIR, y cuando el usuario se aproximara al termostato, la pantalla se encendería, pero provocaría también que la pantalla se encendiera si cualquier persona pasara por delante sin intención de usarla. Por lo que se opta por incorporar un sistema mixto entre el PIR y un pulsador, que el usuario deberá

pulsar cuando quiera visualizar e interactuar con la pantalla, mientras que el PIR detectará que el termostato está siendo usado y mantendrá la pantalla encendida.

El pulsador se conecta con una resistencia *pull-up* y un filtro para evitar rebotes.

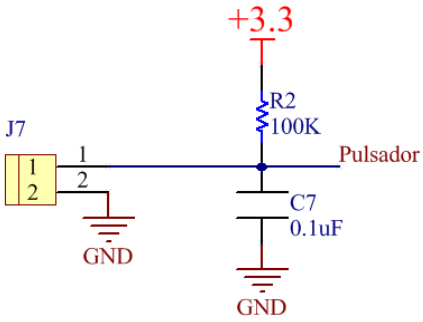


Ilustración 13. Circuito pulsador

3.2.4. Comunicación y µC

Este bloque es el más importante de todos ya que va a ser el encargado de controlar y gestionar todos los componentes del dispositivo. En un principio se planteó hacer cada bloque por separado, se seleccionó un microcontrolador PIC18 y un módulo WiFi (RN1723[10]) de Microchip y para el almacenamiento de datos una memoria EEPROM. Tras el análisis de esta configuración y su dificultad, se buscaron otras alternativas.

Ilustración 14. FlyportPro WIFI. Fuente: wiki.openpicus.com/

Finalmente se optó por el módulo FLYPORT PRO WIFI [8], basado en la tecnología de microchip que incorpora un microcontrolador PIC24 de 16 bit. El módulo WiFi MRF24WG0MB[11] y una memoria EEPROM de 64kbit integrada. De esta forma se tiene el microcontrolador, la comunicación WiFi y el almacenamiento en un mismo dispositivo, lo que permite una manera más fácil de conectar el dispositivo a internet. Además de su entorno de programación con multitud de librerías, que simplifica el desarrollo posterior del firmware.

FlyportPro incorpora 32 GPIO (*General Purpose Input/Output*) de las cuales se utilizan 7 para los distintos bloques, también incorpora 4 UARTs de las cuales se utiliza una para controlar la pantalla y otra para programar el microcontrolador directamente, sin necesidad de una plataforma de programación. Además posee un modulo WiFi para poder controlar el termostato inalámbricamente desde cualquier dispositivo con WiFi. Todos estos bloques se pueden ver en la Ilustración 15.

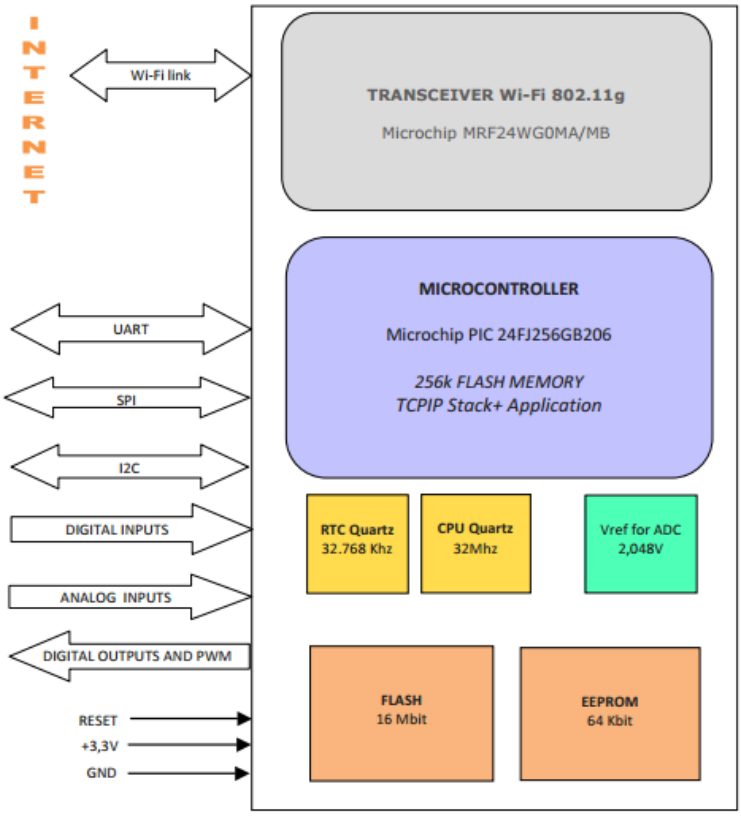


Ilustración 15: Diagrama de bloques FlyportPro. Fuente: wiki.openpicus.com

También se añade al FlyportPro un pulsador de *reset* que es compartido con la pantalla para poder resetear el dispositivo.

[10] http://ww1.microchip.com/downloads/en/DeviceDoc/70005224A.pdf
[11] http://ww1.microchip.com/downloads/en/DeviceDoc/70686B.pdf

3.2.5. Alimentación

Los consumos en este prototipo son un aspecto crítico, dado que se plantea que el mismo pueda funcionar a batería. La siguiente tabla muestra los consumos de los principales componentes del mismo:

Componente	Consumo
Temperatura: DHT22	1.5 mA (alimentado a 3.3 V)
PIR: EKMC1601111	170 µA (alimentado a 3.3 V)
Pantalla: uLCD_24PTU	Min: 10 mA, Typ: 125 mA, Max:220 mA (alimentado a 3.7 V)
FlyportPro (µC y WIFI)	Wi-Fi not connected: 35 mA, Wi-Fi connected: 150 mA , Hibernate mode: 20 mA, Sleep mode: 180 µA (alimentado a 3.3 V)

Tabla 2: Consumo de los componentes

El consumo máximo de este prototipo se ha estimado en 372 mA. No obstante no siempre va a consumir esa corriente, ya que se han definido tres modos de funcionamiento, según los componentes que están activos.

Modo	Componentes
Modo 1	Todos los componentes en funcionamiento
Modo 2	FlyportPRO con WiFi conectado, PIR y DHT22 funcionando y pantalla apagada
Modo 3	FlyportPRO dormido y todos componentes apagados

Tabla 3: Modos

El primer modo representaría la situación que el usuario estuviera utilizando la pantalla táctil para visualizar o configurar algún parámetro del termostato, el segundo modo se corresponde con la pantalla del dispositivo apagado, pero con la comunicación WiFi activa. El último modo sería de bajo consumo, en el que todos los componentes están apagados y el microcontrolador dormido. Este modo se activaría cuando el dispositivo no fuera a ser utilizado, como por ejemplo por la noche.

A continuación, se realiza una estimación de consumo medio valorando el tiempo en funcionamiento en cada uno de estos modos durante 24h, como muestra la Tabla 4. El primer y el segundo modo tienen un consumo elevado, pero están activos un breve intervalo de tiempo, mientras que el tercer modo, que es el que está activo la mayor parte del tiempo, es el que menos consume.

Modo	Consumo Típico	Tiempo activo	% Activo
Modo 1	275 mA	3 min	0.21%
Modo 2	150 mA	27 min	1.88%
Modo 3	0.35 mA	23.5 h	97.92%

Tabla 4: Calculo consumo medio

$$Consumo\ medio\ = (0.0021 * 275) + (0.0188 * 150) + (0.9792 * 0.35) = 3.728\ mA$$

Ecuación 1: Consumo medio

El consumo teórico medio del dispositivo es un tanto elevado, ya que con una batería de 5000 mAh, el termostato tendría una autonomía aproximada de 55 días, es una autonomía reducida ya que los termostatos comerciales tienen una autonomía aproximada de 1 año.

Por otro lado, en este primer prototipo funcional el dispositivo solo trabaja en el modo 1 y 2, ya que para implementar el modo 3 hacen falta estrategias avanzadas de gestión del WiFi que no son objeto del presente T.F.G.

Tras analizar los consumos del dispositivo se opta por integrar dos formas de alimentación, mediante un bus de 5 V o una batería de 3.7 V. La alimentación de 3.7 V se añade porque, aunque con este firmware no sea recomendable alimentar mediante batería, en futuras mejoras podría funcionar con batería sin necesidad de modificar el hardware. La forma de alimentación se selecciona mediante un *jumper* que habilita un circuito u otro.

Estos dos niveles de alimentación se seleccionan observando el voltaje nominal de los principales componentes del dispositivo, todos los dispositivos pueden alimentarse a 3.3 V menos la pantalla, que necesita 3.7 V.

Componente	Voltaje Nominal
Temperatura: DHT22	3.3V
PIR: EKMC1601111	3.3V
Pantalla: uLCD_24PTU	3.7V
FlyportPro (µC y WIFI)	3.3V

Tabla 5: Voltaje Nominal

En la siguiente imagen (Ilustración 16) se muestra el circuito esquemático de la alimentación con los dos conectores bus 5 V y batería, también los *jumpers* J5 y J6, y el diodo D1 (con un V_F aprox. de 1 V) para adaptar la tensión del bus de 5 V a la alimentación de la pantalla.

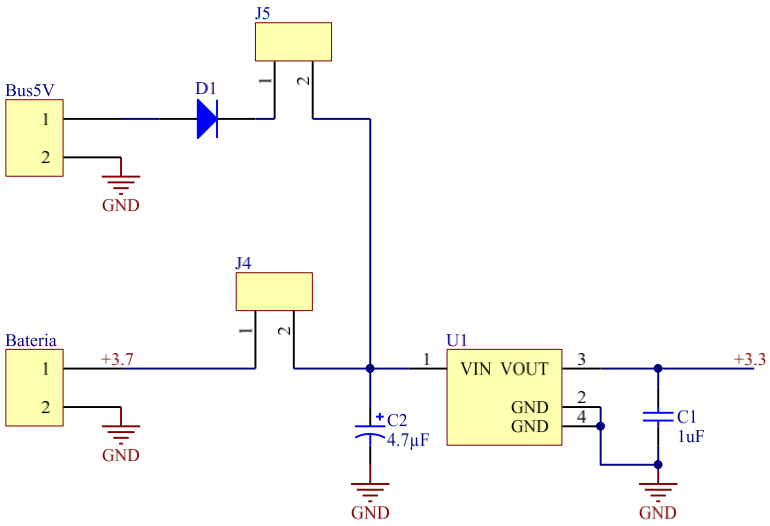


Ilustración 16: Esquemático alimentación

Por último está el LDO (*Low-Dropout regulator*): MCP1825S-3302EDB [11], con salida de 3.3 V y corriente máxima 500 mA, suficiente para alimentar todos los componentes ya que el consumo máximo del dispositivo es de aproximadamente 400 mA. Al ser un componente SMD utilizamos el propio plano de masa como disipador térmico.

En cuanto a la pantalla, está se alimenta mediante los pines de batería para que el dispositivo pueda funcionar tanto con batería como con un bus de continua de 5 V, la alimentación incorpora un circuito de *ON/OFF* para poder encender y apagar la pantalla, y reducir el consumo.

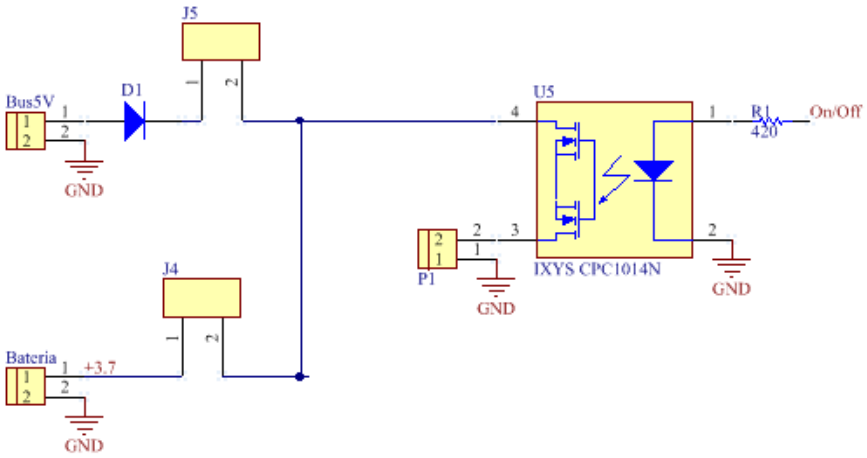


Ilustración 17. Alimentación y ON/OFF pantalla

Este es el circuito de alimentación de la pantalla (Ilustración 17), con los dos tipos de alimentación, bus de 5 V y batería de 3.7 V, en la parte del bus se añade un diodo de 1 V de caída para ajustar la tensión, ya que la pantalla se puede alimentar hasta 4.2 V que es el voltaje que tiene una batería de 3.7 V cargada. Los dos tipos de alimentación incorporan un *jumper* de protección para que solo pueda estar

habilitada una alimentación. Para controlar el ON/OFF se utiliza el relé de estado solido CPC1014N [12], al cual se le incorpora una resistencia de precisión en serie en su circuito de control para ajustar la corriente de activación, y en el otro extremo se conecta la alimentación de 3.7 V con la pantalla.

La corriente necesaria para activar el relé es 5 mA, la tensión que cae en el fotodiodo interno es 1.2 V y la tensión de salida del microcontrolador, que es encargado de activar este circuito, es 3.3 V. Por tanto, la resistencia necesaria es:

$$R = \frac{Vcc - Vd}{I} = \frac{3.3 - 1.2}{0.005} = 420\Omega$$

Ecuación 2: Calculo de resistencia R1

3.3. Diseño PCB

Para el diseño de la PCB (*Printed Circuit Board*) se ha utilizado CircuitMaker, una herramienta muy potente, ya que es la versión gratuita de Altium, uno de los softwares más utilizados y completos en el diseño de PCBs. CircuitMaker tiene una comunidad virtual de usuarios, donde los diseños se cargan en el servidor de Altium y pueden ser vistos y reutilizados por cualquier usuario.

Para la implementación del prototipo se ha optado por una PCB mixta, ya que incorpora componentes de tipo THL (*Through Hole Device*), que son insertados a través de patillas para su posterior soldadura, y SMD (*Surface Mounted Device*) que son soldados en la misma superficie.

Los componentes se han situado por zonas, dependiendo del bloque al que pertenecen, tratando de alinearlos en la medida de los posible, ya que la colocación de componentes será más rápida en caso de automatizarla y, por lo tanto, más barata. En el centro se ha colocado el microcontrolador para tener acceso a todas las partes de la PCB y facilitar su conexionado. En la parte superior se ha ubicado la alimentación con todos sus componentes y, los conectores se encuentran en el perímetro de la PCB, para facilitar la conexión con los componentes externos. También incorpora cuatro orificios para anclar la PCB, como se puede ver en el Anexo I.

Esta primera PCB se ha fabricado en el taller del Departamento de Electrónica y Comunicaciones de la EINA. Dada la tecnología de fabricación del mismo se ha optado por diseñar una PCB de clase 3[12] y a doble cara, con pistas en ambas caras, simplificando así el ruteado de la misma.

Las pistas de señal son de 0.5 mm y las de alimentación son de 1 mm. Las pistas son lo más cortas y directas posible. También se evitan los ángulos de 90 grados porque implica un cambio brusco de impedancia. La PCB incorpora un plano de masa en cada cara para facilitar el retorno de la corriente.

[12] http://www.fastpcb.com/pdf/Clases%20Circuitos%20Fast.pdf

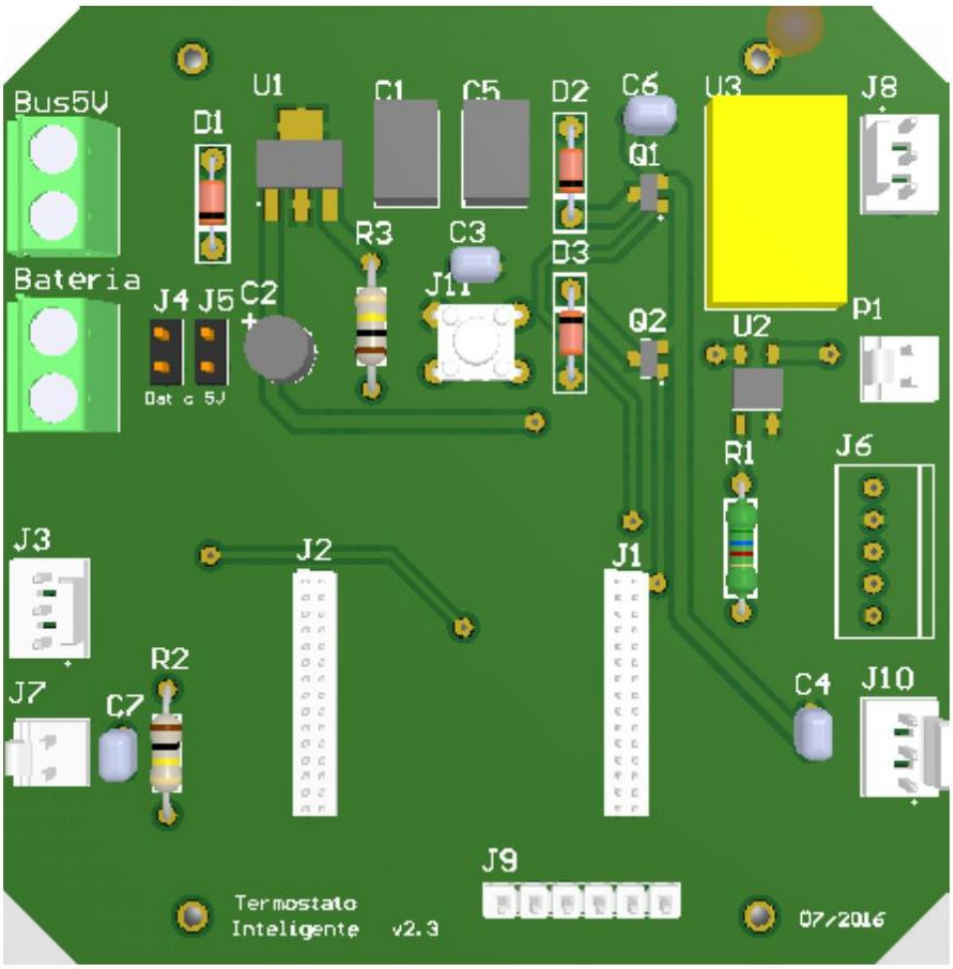


Ilustración 18: [7]Plano de PCB en 3D

La vista 3D (Ilustración 18) es muy útil para el posicionamiento de componentes y ver la distribución espacial, ya que podemos incorporar los componentes externos como la pantalla y la caja en 3D y comprobar que todo encaja correctamente.

Una vez que el diseño de la PCB está finalizado, se generan los archivos Gerber[13] necesarios para su fabricación. Una vez finalizada, se testea y sueldan todos los componentes en ella y su resultado es el siguiente (Ilustración 19, Ilustración 20):

[13] https://es.wikipedia.org/wiki/Gerber_(formato_de_archivo)

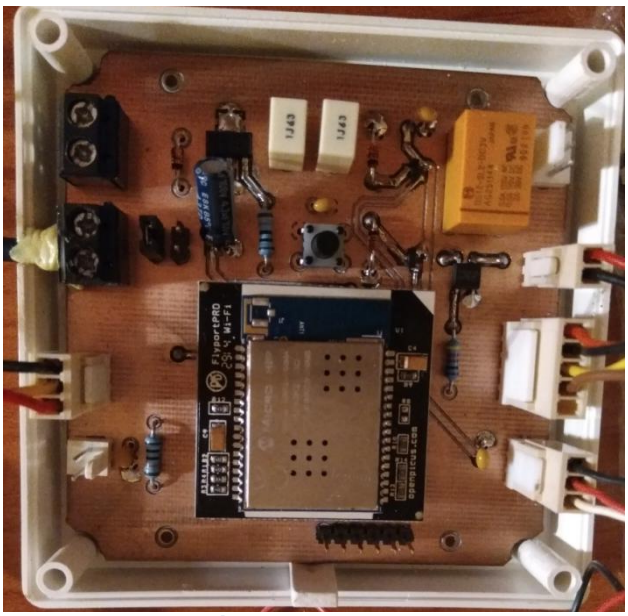


Ilustración 20: Cara top PCB soldada

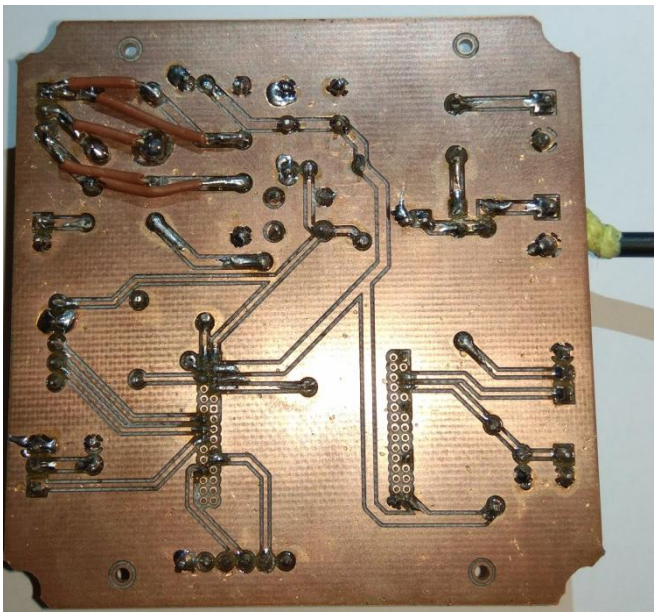


Ilustración 19: Cara bottom PCB soldada

En la Ilustración 20 se puede ver la cara top de la PCB, con todos los componentes ya soldados, aunque solo se puede apreciar la soldadura de las vías y de los componentes SMD. La mayoría de las soldaduras se encuentran en la cara bottom (Ilustración 19) ya que casi todos los componentes son THL, por lo que la mayor parte de las pistas también se encuentras en esta cara.

En este primer prototipo se ha detectado un error en la huella del relé que controla la caldera, por lo que las conexiones no son correctas y hay que corregirlo mediante cable, como se puede ver en la parte superior izquierda de la Ilustración 19. Para solucionar este problema en futuras versiones, simplemente se tendría que girar 180º la huella en la herramienta de diseño de PCB.

4. Diseño firmware

4.1. Introducción

En este capítulo se explica el diseño del firmware del primer prototipo. En primer lugar se muestra de forma resumida el funcionamiento de los procesos mediante diagramas, el código fuente se puede consultar en el Anexo III.I Código del programa principal. A continuación se explican las comunicaciones necesarias para el funcionamiento del termostato y la programación realizada para implementarlas.

4.2. Funcionamiento del sistema

En el siguiente diagrama de estados (Ilustración 21) se puede observar el funcionamiento general del termostato, con sus 3 bloques principales y sus transiciones.

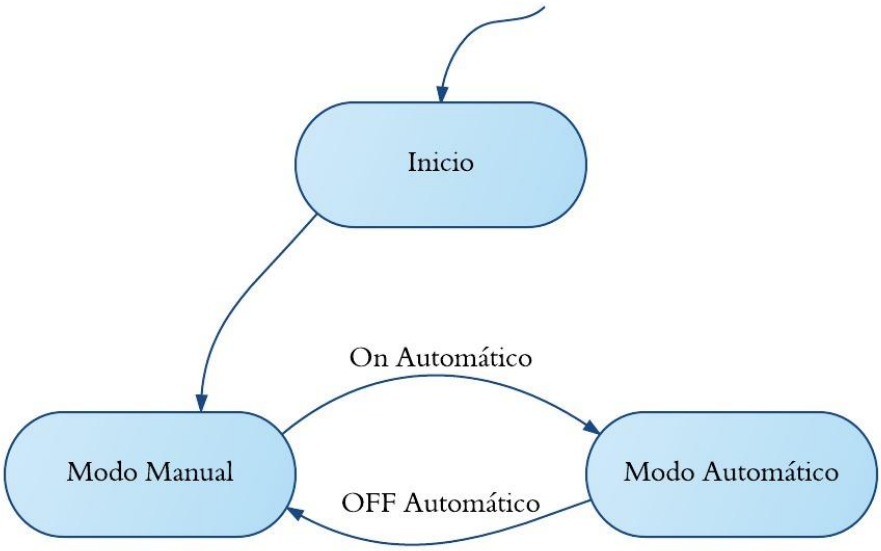


Ilustración 21: Maquina de estados general

El prototipo se inicializa con una serie de configuraciones de parámetros necesarios para su funcionamiento, una vez que este proceso termina, el dispositivo entra en el modo manual. En este modo el aparato funciona como un termostato convencional, en el que se selecciona una temperatura deseada y el termostato enciende la calefacción hasta conseguirla. El dispositivo se puede controlar de dos maneras, mediante la pantalla táctil que incorpora o desde cualquier dispositivo WiFi como un *Smartphone*, *Tablet*, ordenador, etc. Desde ambos es posible cambiar la temperatura deseada, apagar el termostato, visualizar la temperatura y humedad de la vivienda, y cambiar el modo de funcionamiento.

El modo automático es el que le proporciona el adjetivo "inteligente", ya que el dispositivo va a ir aprendiendo día a día la rutina de los usuarios del hogar, para poder tener la vivienda a una temperatura idónea justo cuando lleguen a casa, sin necesidad de programar nada.

4.2.1. Inicio

Este es el primer proceso que realiza el microcontrolador cuando se enciende. Al comenzar crea las variables globales del sistema que pueden ser vistas y modificadas por todos los métodos, como por ejemplo la temperatura seleccionada, el estado del termostato, los vectores de los días de la semana para la rutina, etc. (ver Anexo III.I Código del programa principal). El siguiente paso es configurar los pines de los distintos componentes como entradas y salidas: se configura como salida el pin de alimentación del sensor de temperatura y humedad, el pin que activa el relé que controla la alimentación de la pantalla, así se puede controlar su activación y se puede obtener un menor consumo. También se configura como salida los pines que van al relé que controlan la caldera, para poder encenderla y apagarla.

A continuación se configura la UART que va a comunicarse con la pantalla. Una vez establecida, se enciende con el interfaz de configuración del tiempo (Ilustración 40), en el cual se introduce la fecha actual para inicializar el RTC (*Real Time Clock*), y se configura el mismo para que se produzca una interrupción cada 10 segundos. También se configura como interrupción el pin donde se conecta el PIR, ya que gracias a él se controla la rutina del usuario. Por último, se inicializa y se cargan los datos de rutina almacenados en EEPROM que permite guardar los datos aunque se apague el termostato.

4.2.2. Modo manual

Modo manual (Ilustración 22**):** Como se ha comentado anteriormente, este es uno de los dos modos de funcionamiento del prototipo. Si el termostato está apagado la calefacción se mantiene apagada, mientras que si el termostato está encendido comprueba continuamente que la temperatura ambiente sea aproximadamente igual a la seleccionada mediante la pantalla o la página web, en caso de ser menor el termostato empezará a actuar. Para el control de encendido y apagado de la calefacción se utiliza un ciclo de histéresis (Ilustración 23) de un grado de temperatura, para evitar que la calefacción no esté constantemente encendiéndose y apagándose. Este sistema lo llevan incorporado la mayoría de termostatos para no dañar la calefacción.

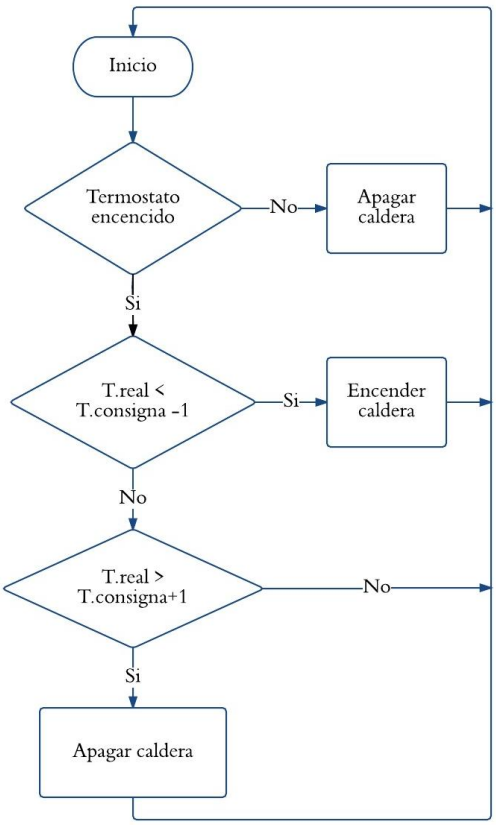


Ilustración 22: Diagrama de flujo Modo manual

Mientras que el termostato está encendido, existen dos procesos en paralelo que se ejecutan independientemente del modo seleccionado. Estos dos procesos son: (i) la interrupción RTC y (ii) el pulsador para encender la pantalla.

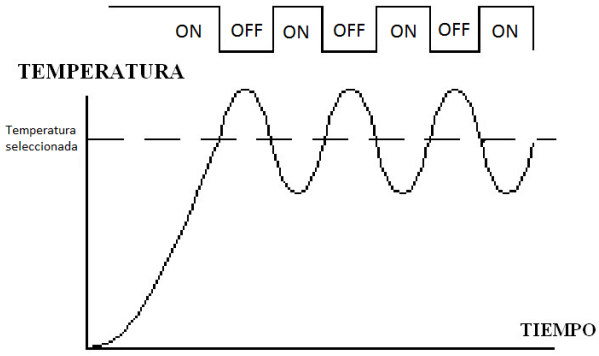


Ilustración 23: Ciclo de histéresis

La interrupción RTC, como se comenta en el estado de inicio, es configurada mediante la pantalla táctil en su primer inicio, y se configura para que se ejecute cada 10 segundos, por lo que el dispositivo cuenta con una base de tiempo para poder temporizar procesos. Así pues, cada vez que se produce esta interrupción, el microcontrolador se comunica con el sensor DHT22 para actualizar la temperatura y la humedad, y enviarlas a la página web y a la pantalla, si está encendida.

El otro proceso es para habilitar la pantalla y que no esté constantemente encendida, ya que tiene un consumo excesivo. Cuando el microcontrolador detecta que el pulsador ha sido presionado, la pantalla se enciende hasta que deje de detectar presencia y hayan pasado dos minutos. También se tiene en cuenta, que la pantalla al apagarse pierde toda la información introducida y recibida, por lo que es necesario restablecerla, de modo que se crea un método que envía el valor de las variables en ese momento, como por ejemplo la temperatura seleccionada, el modo en el que está funcionando el termostato, etc.

4.2.3. Modo automático

El modo automático (Ilustración 24) se apoya en un registro de rutinas como el de la Tabla 6, en el cual se van guardando la rutina del usuario conforme pasan las semanas. El registro se rellena mediante un algoritmo que intenta aprender que horas del día el usuario está en casa, cada día de la semana. Esto se consigue gracias al sensor PIR. Por lo que se recomienda dejar de 2 a 3 semanas el termostato aprendiendo las rutinas antes de utilizar este modo. De esta forma se obtendrá un funcionamiento más correcto.

Horas	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Lunes																								
Martes																								
Miércoles																								
Jueves																								
Viernes																								
Sábado																								
Domingo																								

Tabla 6: Registro de rutina

Una vez que tenemos un registro de rutinas lo suficientemente robusto, ya se puede activar este modo, que comprueba continuamente si en el registro de rutinas hay guardada presencia ese día de la semana y a esa hora. En el caso de que la lectura del registro nos indique que normalmente hay presencia, se activará el termostato a una temperatura deseada de 21 grados, ya que es la temperatura recomendada para estos dispositivos, y si la temperatura ambiente es inferior, la caldera se encenderá. En caso de que el registro desvele que normalmente no hay nadie ese día de la semana, a esa hora, la caldera permanecerá apagada.

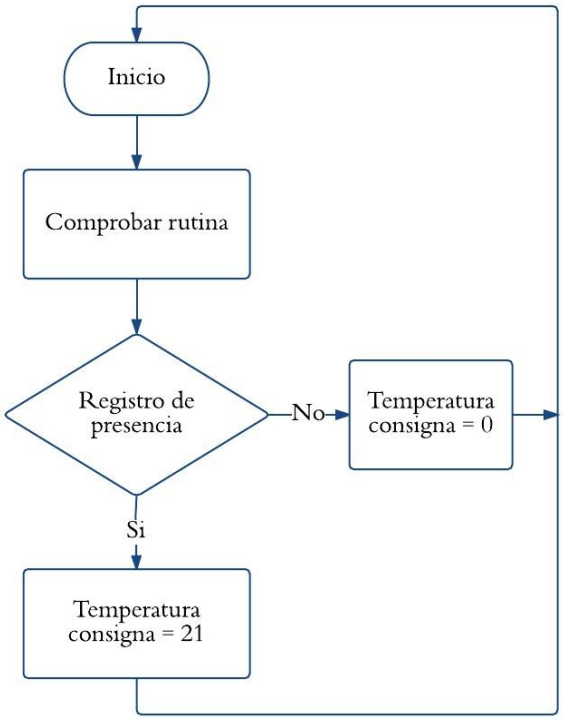


Ilustración 24: Diagrama de flujo de Modo automático

Además la pantalla incorpora un menú de visualización de rutinas, en el cual se puede ver una grafica de cada día, con los datos de presencia representados. También incorpora la opción de borrar el registro, por si el usuario cambia de hábitos, el termostato se adapte más rápido a él.

Como se ha comentado anteriormente, la interrupción del sensor PIR va a ser la que permita al dispositivo aprender la rutina diaria del usuario. Cada vez que el PIR detecte movimiento, guardará constancia de presencia en ese día, a esa hora. El algoritmo para guardar la presencia en el registro de rutinas es el siguiente:

$$R.Rutina(Dia, Hora) = \frac{R.Rutina(Dia, Hora) \times N.Semanas \ + (1 \ o \ 0)}{N.Semanas \ + \ 1}$$

Ecuación 3: Algoritmo para guardar presencia

Una vez finalizado el día, se añade al registro de rutinas, hora a hora, un 1 si en esa hora hubo presencia y un 0 si no la hubo, y se hace la media con los datos guardados. Por lo que en cada casilla tendremos un valor del 1 al 0, si el valor es mayor de 0.5 se considera que normalmente hay presencia y si es menor se considera que normalmente no hay presencia. El motivo por el que dispositivo tiene una memoria EEPROM es para poder guardar este registro de rutina, y que no se borre cuando el termostato se desconecte, ya que ha pasado un largo periodo de tiempo hasta que ha recopilado toda esa información.

4.3. Comunicaciones

4.3.1. Pantalla

Como se ha comentado previamente, la pantalla que incorpora este primer prototipo de termostato es la **uLCD_24PTU,** del fabricante **4dsystem,** controlada mediante comunicación UART. La comunicación básica de la UART (Ilustración 25) consta de tres conexiones Rx, Tx y GND, las GND de los dos componentes se unen y las otras se entrecruzan, así por el canal que una envía los datos el otro las recibe y viceversa.

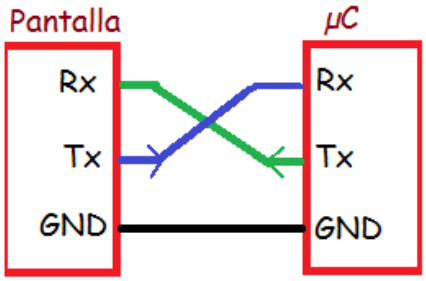


Ilustración 25: Comunicación UART

Una vez que la conexión está establecida, se analiza la interpretación de datos de la pantalla. La pantalla trabaja con cadenas da datos en hexadecimal de 6 bytes con el siguiente formato:

Command	Object Type	Object Index	Value MSB	Value LSB	Checksum

- En la primera posición (Command) se encuentra el código identificativo de las funciones (Ilustración 26), para esta aplicación se utilizan dos funciones: WRITE_OBJ y REPORT_EVENT, para enviar una orden a la pantalla el primer byte tiene que ser 0x01, y cuando el microcontrolador recibe información de la pantalla el primer byte es 0x07.

Command	Code
READ_OBJ	0x00
WRITE_OBJ	0x01
WRITE_STR	0x02
WRITE_ STRU	0x03
WRITE_ CONTRAST	0x04
REPORT_OBJ	0x05
REPORT_EVENT	0x07

Ilustración 26: Código identificativo de las funciones

- La segunda posición (Object Type) se corresponde al código de identificación del objeto (ver *datasheet*[14]) de la pantalla y la tercera posición el número de objeto que es, para poder distinguir entre los objetos del mismo tipo.

- En la cuarta y quinta posición se encuentra el valor que tiene o se le quiere dar al objeto.

- Por último, está el *Checksum*, para comprobar que la cadena de caracteres es correcta, y es igual a la XOR de los cinco primeros bytes.

[14] http://www.4dsystems.com.au/productpages/ViSi-Genie/downloads/Visi-Genie_userguide_R_1_4.pdf

Tras el análisis de la nomenclatura de trabajo de la pantalla, se implementan los métodos con los que el microcontrolador envía y recibe información a la pantalla. La función genérica para enviar datos a la pantalla se explica a continuación y su código fuente se puede ver en Anexo III.

Función	**void enviar (char identificador, char numero_objeto, valorMSB, valorLSB)**
Entradas	Identificador, numero de objeto, valor
Proceso	Calcula el *checksum* y enviar la cadena a través de la UART
Salidas	Esta función no devuelve nada

Tabla 7: Función genérica para enviar datos a la pantalla.

Y la función genérica para leer los datos recibidos desde la pantalla es:

Función	**void leerUART (char cadena[ ], int longitud)**
Entradas	Cadena de datos, longitud de la cadena
Proceso	Identificar que objeto a mandado la información, y asignarle ese valor a la variable correspondiente
Salidas	Esta función no devuelve nada

Tabla 8: Función genérica para leer los datos recibidos desde la pantalla

Una vez que se conoce la nomenclatura en la que trabaja la pantalla, y mediante estas dos funciones, ya se puede controlar por completo la pantalla (ver en Anexo III).

4.3.2. Sensor DHT22

El sensor DHT22 utilizado en el prototipo para obtener la temperatura y la humedad utiliza una comunicación 1-wire, que también es una comunicación serie compuesta por una línea de datos, un maestro, en este caso el microcontrolador, y un esclavo, que es el sensor.

Para mandarle una petición al sensor y que nos responda con los datos, se debe enviar una señal en bajo de al menos 1 ms y después ponerla en alto, tras un máximo de 40 µs el sensor toma el mando de la comunicación y envía un escalón de 80 µs en bajo y después en alto, a partir de ese momento el sensor empieza a enviar los datos.

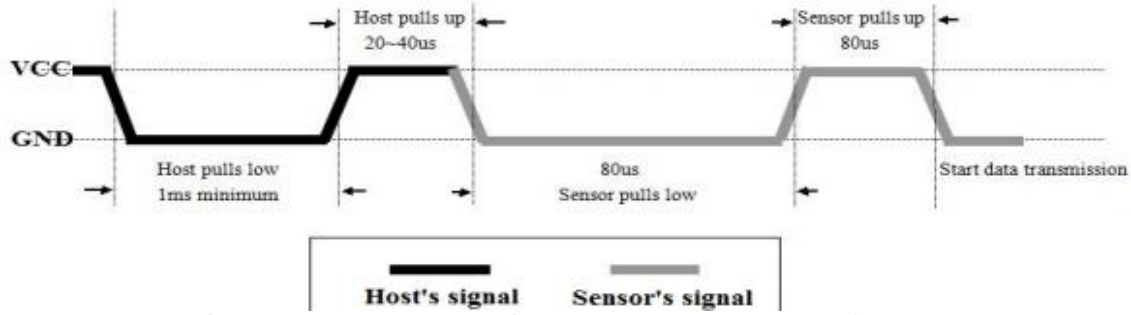


Ilustración 27: Inicio de la comunicación con DHT22. Fuente: www.adafruit.com

El sensor contesta mandando 40 bits, los 16 primeros se corresponden con la humedad relativa, los 16 siguientes con la temperatura y los 8 últimos con el *checksum* que tiene que coincidir con la suma de los 4 bytes anteriores.

Para distinguir si los bits son 1 o 0 el sensor manda una señal en bajo de 50 µs y después una en alto de 26 ~ 28 µs si el bit es un cero o 70 µs si el bit es un uno.

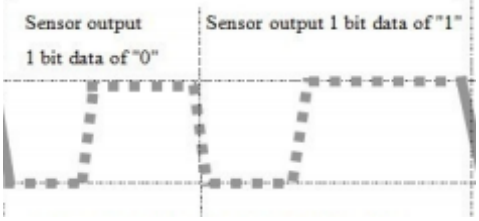


Ilustración 28: Envio de bits del DHT22. Fuente: www.adafruit.com

Para verificar y comprobar el funcionamiento y en consecuencia hacer una librería para obtener los resultados de temperatura y humedad, se establece la comunicación y se mide la señal mediante un osciloscopio:

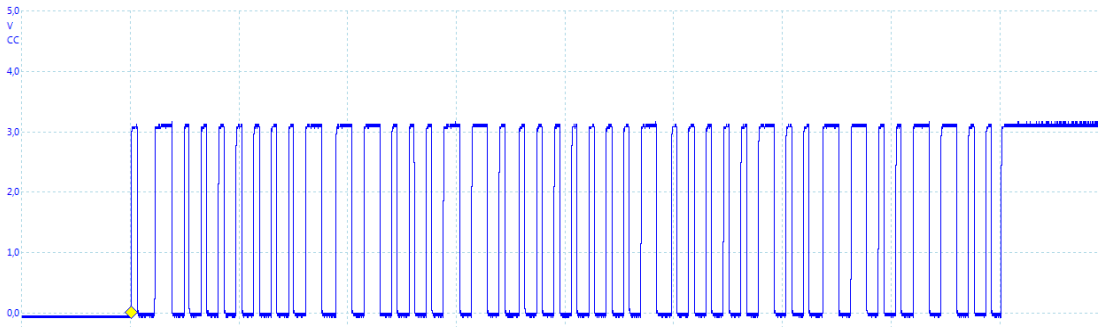


Ilustración 29: Señal real sensor DHT22

La librería realizada, basada en una librería 1-wire de **OpenPicus,** contiene principalmente 2 métodos, la comprobación *checksum* y la obtención de datos:

Función	static int checksum (unsigned long data1, unsigned long data2)
Entradas	El vector de datos con 32 bits y el vector *checksum* con 8 bits
Proceso	Comprueba que la suma de los 4 bytes de datos coincide con el vector de *checksum*
Salidas	Si los datos son correctos devuelve un 1 y si son incorrectos un -1

Tabla 9: Función Checksum

Función	void RHT (float temperatura*, float humedad*)
Entradas	Señal del sensor DHT22
Proceso	Comprueba que la comunicación con sensor es correcta, e interpreta su respuesta bit a bit entre 1 o 0. Una vez obtenidos todos los datos, se acomodan para obtener la temperatura y la humedad
Salidas	Devuelve la temperatura y la humedad, y en caso de fallar un -1

Tabla 10: Función de obtención de temperatura y humedad

4.3.3. WiFi

En esta versión de firmware, la comunicación WiFi no va a ser a través de internet, sino que el FlyporPRO WiFi va a crea una red WiFi local, en la que montará un servidor web al que se podrá acceder mediante su IP. Gracias a esto las pruebas en el laboratorio serán mucho más cómodas y fáciles, en caso de que el producto se quisiera comercializar, se debería conectar el termostato con el modem de la vivienda mediante el nombre de la red y la contraseña y configurar el router para que la IP del termostato tenga acceso remoto. De esta manera la página web sería accesible desde cualquier punto.

El servidor web de este prototipo se ha realizado modificando un ejemplo proporcionado por OpenPicus[15], el cual ha sido adaptado a las necesidades de nuestro termostato.

En primer lugar se crea una página web en formato HTML (HyperText Markup Language), que servirá de interfaz para el usuario:

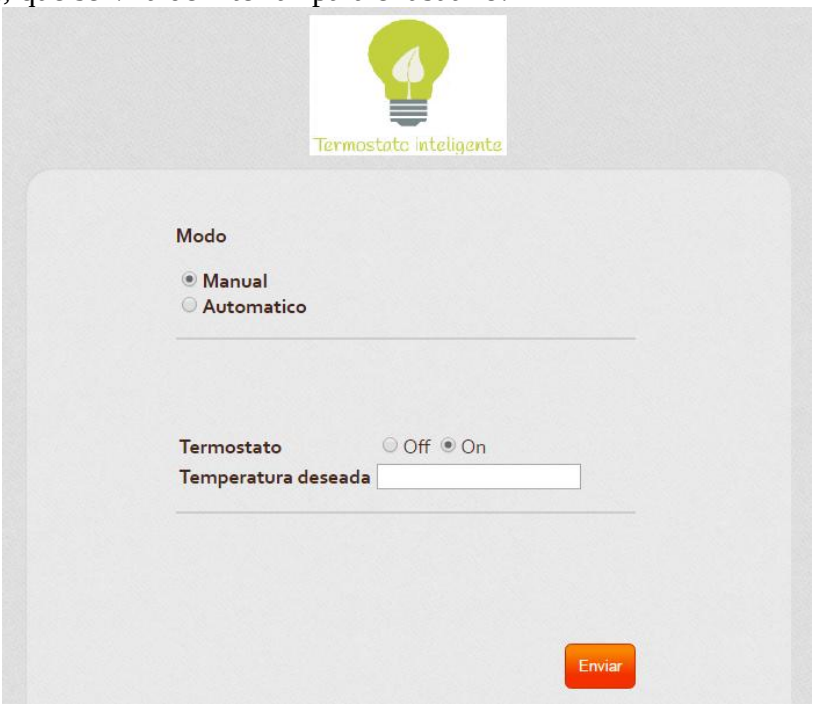


Ilustración 30: Interfaz de la web

En la página web se puede seleccionar el modo de funcionamiento del dispositivo, manual o automático, se muestra la temperatura y la humedad real del hogar, se puede encender y apagar el termostato y también podemos seleccionar la temperatura deseada, por lo que se puede controlar de igual manera el termostato desde la pantalla que desde la web.

Una vez creada la página web, el siguiente paso es recibir lo que el usuario ha seleccionado y enviar la temperatura y la humedad.

[15] http://wiki.openpicus.com/index.php/Smart_Network_Configuration

Cuando se pulsa el botón enviar, se envía una cadena de *String* con toda la información, el método para recibir los datos, busca el identificador de cada selección y copia el valor en una variable, una vez que el valor está guardado solo hace falta interpretarlo correctamente.

Para enviar la temperatura y la humedad se crea una cadena de caracteres en la que se guardan las variables con un determinado formato para que su visualización sea correcta y se envía al servidor web.

5. Conclusiones y líneas futuras

En este apartado, se plantean las conclusiones del trabajo realizado en este T.F.G. valorando el grado de complimiento de los objetivos planteados a su inicio.

En la fase de búsqueda de información, se ha aprendido el funcionamiento básico de un termostato y las características principales de los termostatos inteligentes actuales. Gracias a estos conocimientos las funciones de nuestro prototipo son lo más reales posibles.

En cuanto al diseño del hardware, todos los componentes y sus circuitos han funcionado correctamente, por lo que la puesta a punto de la PCB ha sido rápida y sin grandes contratiempos. Los errores detectados se han documentado para el diseño de nuevas versiones.

En lo referente al firmware del dispositivo, se ha conseguido crear un algoritmo para que el termostato aprenda las rutinas, se pueda controlar mediante WiFi a través de una página web, se pueda manejar mediante una pantalla táctil y pueda apagar y encender la caldera.

Por lo tanto, el objetivo principal ha sido alcanzado, ya que se ha diseñado un prototipo funcional de termostato inteligente capaz de ser controlado inalámbricamente y que aprende la rutina del usuario de forma automática.

Aun así, el prototipo se podría mejorar en muchos aspectos, y estas son algunas de las mejoras que se podrían aplicar en futuras versiones:

- **Mejora PCB:** Como se ha comentado, la huella del componente U3 se modificaría, también se podría modificar la distribución espacial de los bloques en la PCB para mejorar la conectividad WiFi.

- **Consumo:** Sería recomendable desarrollar el modo 3 (3.2.5.Alimentación) de consumo en el que el microcontrolador permanece dormido, para que el dispositivo bajara drásticamente su consumo y pudiera funcionar con una mayor autonomía con batería.

- **Conexión a internet:** Gracias a la conexión a internet mediante el modem de casa, el termostato podría ser controlado desde cualquier dispositivo con conexión a internet. Además también podría actualizar la fecha, cargar la previsión meteorológica y saber cuándo el usuario va a llegar a casa.

- **Mejora del algoritmo:** También se podría desarrollar un algoritmo más complejo de aprendizaje de la rutina, para que fuera más rápido y preciso, y se adaptara mejor al usuario.

6. Bibliografía

[1] OpenPicus, «OpenPicus,» [En línea]. Available: http://www.openpicus.com/.

[2] ASOCIACIÓN ESPAÑOLA DE DOMÓTICA E INMÓTICA, «cedom,» [En línea]. Available: http://www.cedom.es/. [Último acceso: 2016].

[3] Hörmann, «Domótica en el hogar».

[4] X. P. D. V. Stefan Junestrand, Domótica y hogar digital, Thomson Paraninfo.

[5] F. Sandoval, «La evolución del termostato,» *Cero Grados*, 2016.

[6] domboo, «domboo,» [En línea]. Available: https://domboo.es.

[7] Panasonic, «Datasheet DS1E-SL2-DC3V,» [En línea]. Available: https://www.panasonic-electric-works.com/cps/rde/xbcr/pew_eu_en/ds_61005_en_ds.pdf.

[8] Panasonic, «Datasheet EKMC1601111,» [En línea]. Available: https://www3.panasonic.biz/ac/e_download/control/sensor/human/catalog/bltn_eng_pir.pdf.

[9] Adafruit, «Datasheet DHT22,» [En línea]. Available: https://cdn-shop.adafruit.com/datasheets/DHT22.pdf.

[10] 4D Systems, «Datasheet uLCD-24PTU,» [En línea]. Available: http://www.4dsystems.com.au/productpages/uLCD-24PTU/downloads/uLCD-24PTU_datasheet_R_1_8.pdf.

[11] OpenPicus, «Datasheet FlyportPRO WiFi,» [En línea]. Available: http://space.openpicus.com/u/ftp/datasheet/datasheet_flyportpro_wifi.pdf.

[12] Microchip, «Datasheet MCP1825S-3302EDB,» [En línea]. Available: http://ww1.microchip.com/downloads/en/devicedoc/22056b.pdf.

[13] IXYSIC, «Datasheet CPC1014N,» [En línea]. Available: http://www.ixysic.com/home/pdfs.nsf/www/CPC1014N.pdf/$file/CPC1014N.pdf.

[14] Microchip, «Microchip,» [En línea]. Available: http://www.microchip.com/.

[15] «La evolución del termostato,» *Cero Grados*, 2016.

[16] 4D Systems, «4D Systems,» [En línea]. Available: http://www.4dsystems.com.au/.

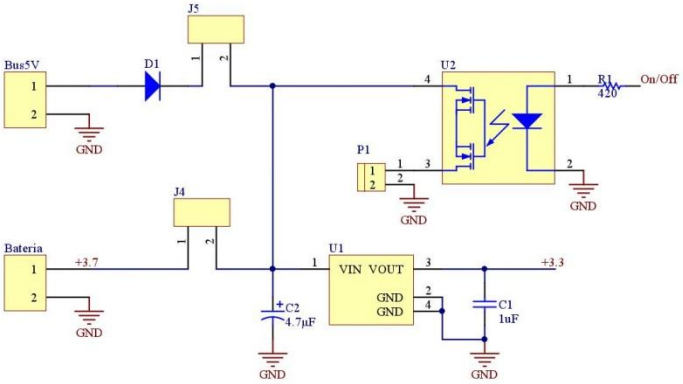
[17] CircuitMaker, «CircuitMaker,» [En línea]. Available: http://circuitmaker.com/.

Anexos

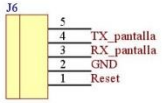
Anexo I. Planos

Anexo I.I. Esquemático

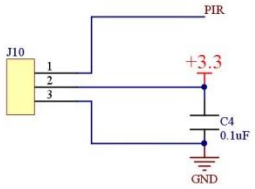
Alimentación



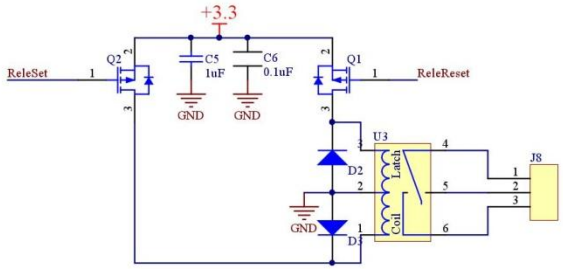
Pantalla



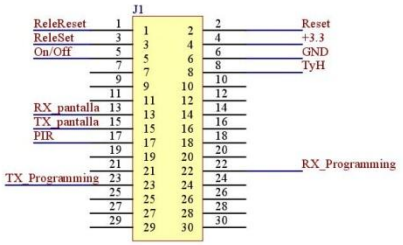
PIR

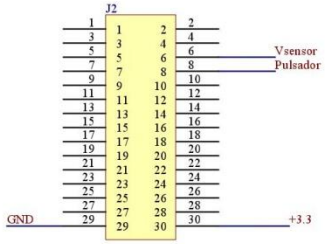


Relé

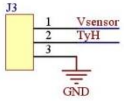


Microcontrolador

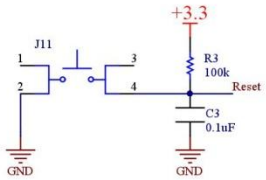




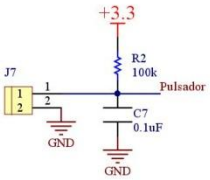
Temperatura y humedad



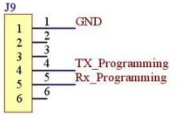
Reset



Pulsador



Programación



Title: Termostato inteligente		
Size: A3	Number: v2.3	Revision: 13/07/2016
Date: 17/08/2016		Sheet of
File: TermostatoSchDoc.SchDoc		Drawn By: Daniel Rosa

Anexo I.II. Cara top

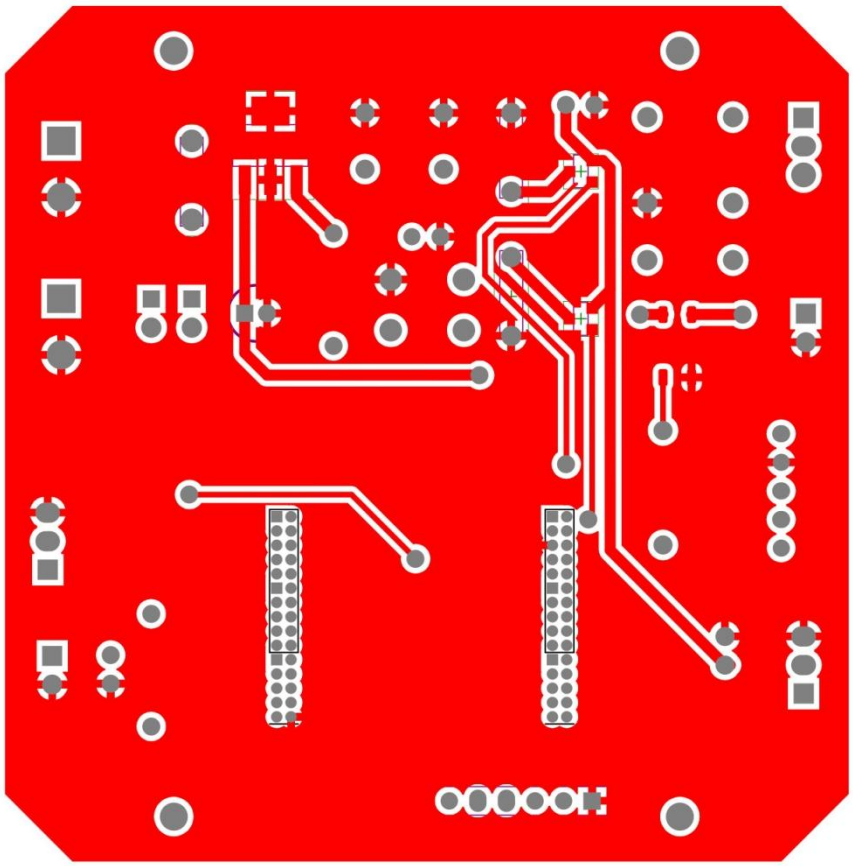


Ilustración 31: Cara top PCB

Anexo I.III. Cara bottom

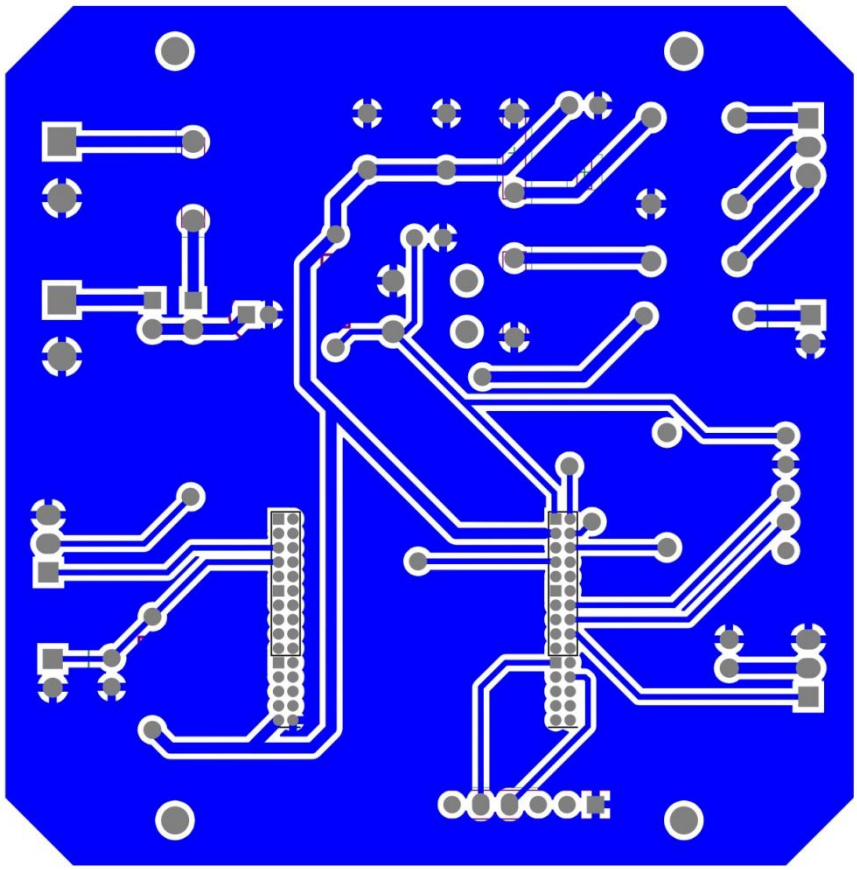


Ilustración 32: Cara bottom PCB

Anexo II. Coste estimado de componentes

Descripción	Cantidad	Precio unit.	Precio total
Pantalla uLCD_24PTU	1	53.96 €	53.96 €
FlyportPRO WiFi	1	35.00 €	35.00 €
Carcasa	1	1.50 €	1.50 €
Conectores alimentación	2	0.40 €	0.80 €
Condensador 1µF	2	0.49 €	0.98 €
Condensador 0.1µF	4	0.27 €	1.08 €
Condensador 4.7µF	1	0.15 €	0.15 €
Diodos	3	0.09 €	0.27 €
Conectores FlyportPro WiFi	2	2.54 €	5.08 €
Conector 3 posiciones	3	0.35 €	1.05 €
Conectores 5 posiciones	1	0.47 €	0.47 €
Conectores 2 posiciones	3	0.10 €	0.30 €
Conector 6 posiciones	1	0.66 €	0.66 €
Transistor mosfet	2	0.39 €	0.78 €
Resistencia 420 Ω	1	0.09 €	0.09 €
Resistencia 100kΩ	2	0.10 €	0.20 €
LDO	1	0.51 €	0.51 €
Relé estado solido	1	1.47 €	1.47 €
Relé latching	1	7.76 €	7.76 €
Pulsador de panel	1	3.85 €	3.85 €
Pulsador montaje PCB	1	0.19 €	0.19 €
		Total	115.68 €

Anexo III. Firmware

Anexo III.I Código del programa principal

```
#include "taskFlyport.h"
#include "rht03.h"


//Variables Globales
char modoAutomatico=0;
char temperaturaConsigna=0;
char estadoTermostato=1;
struct tm mytime;
float temperaturaReal, humedadReal;
char intRTC=0;
char temporizadorPantalla=0;
char flagOn=1;
char flagOff=1;
char lunes[23],martes[23],miercoles[23],jueves[23],viernes[23],sabado[23],domingo[23];
char diaActual[23]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int diaAnterior;
int Nsemanas[1];
BOOL ParamSet = FALSE;

void enviarOn(){

    char enviarled[6]={0x01,0x13,0x00,0x00,0x01,0x13}; //Led on
    UARTWriteBytes(2, enviarled,6);
    vTaskDelay(10);
    char enviarsound[6]={0x01,0x16,0x00,0x00,0x00,0x17}; //sonido
    UARTWriteBytes(2, enviarsound,6);
    flagOn=0;

}

void enviarOff(){

    char enviarled[6]={0x01,0x13,0x00,0x00,0x00,0x12}; //Led on
    UARTWriteBytes(2, enviarled,6);
    vTaskDelay(10);
    char enviarsound[6]={0x01,0x16,0x00,0x00,0x00,0x17}; //sonido
    UARTWriteBytes(2, enviarsound,6);
    flagOff=0;

}

void enviarForm13(){ // Habilitar pantalla de configuracion del tiempo


    char enviar[6]={0x01,0x0A,0x0D,0x00,0x00,0x06}; //Led on
    UARTWriteBytes(2, enviar,6);
    vTaskDelay(10);
}
```

```
49.
50. void enviarTemperatura(float temperatura){
51.     unsigned int temperaturaHex = (unsigned int)(temperatura * 100);
52.     int i;
53.     char enviar[6]={0x01,0x0F,0x00,0x00,0x00,0x00};
54.     enviar[3]=temperaturaHex>>8;
55.     enviar[4]=temperaturaHex;
56.     for(i=0;i<5;i++){
57.         enviar[5]^=enviar[i];
58.     }
59.     UARTWriteBytes(2, enviar,6);
60. }
61.
62. void enviarHumedad(float humedad){
63.     unsigned int humedadHex = (unsigned int)(humedad * 100);
64.     int i;
65.     char enviar[6]={0x01,0x0F,0x01,0x00,0x00,0x00};
66.     enviar[3]=humedadHex>>8;
67.     enviar[4]=humedadHex;
68.     for(i=0;i<5;i++){
69.         enviar[5]^=enviar[i];
70.     }
71.     UARTWriteBytes(2, enviar,6);
72. }
73.
74. void actualizarPantalla(){ // Restaurar los valores seleccionados, ya
    que cuando la pantalla se apaga son borrados
75.     int i;
76.     char enviar[6]={0x01,0x04,0x00,0x00,0x00,0x00};
77.     enviar[4]=temperaturaConsigna;
78.     for(i=0;i<5;i++){
79.         enviar[5]^=enviar[i];
80.     }
81.     UARTWriteBytes(2, enviar,6);
82.     vTaskDelay(10);
83.     char enviar1[6]={0x01,0x0F,0x02,0x00,0x00,0x00};
84.     enviar1[4]=temperaturaConsigna;
85.     for(i=0;i<5;i++){
86.         enviar1[5]^=enviar1[i];
87.     }
88.     UARTWriteBytes(2, enviar1,6);
89.
90.     vTaskDelay(10);
91.     char enviar3[6]={0x01,0x1E,0x00,0x00,0x00,0x00};
92.     enviar3[4]=estadoTermostato;
93.     for(i=0;i<5;i++){
94.         enviar3[5]^=enviar3[i];
95.     }
96.     UARTWriteBytes(2, enviar3,6);
97.     vTaskDelay(10);
98.     char enviar4[6]={0x01,0x1E,0x02,0x00,0x00,0x00};
99.     enviar4[4]=modoAutomatico;
100.        for(i=0;i<5;i++){
101.            enviar4[5]^=enviar4[i];
102.        }
103.        UARTWriteBytes(2, enviar4,6);
```

```
104.          vTaskDelay(10);
105.          UARTWrite(1,"Pantalla Actualizada \n");
106.      }
107.
108.      void enviarRutina(){ // Enviamos la rutina de todos los dias par
    a viualizar por pantalla
109.              int i=0;
110.              int j=0;
111.              for(i=0;i<24;i++){
112.                  char dia1[6]={0x01,0x19,0x00,0x00,0x00,0x18};
113.                  if(lunes[i]>0.5){
114.                      dia1[4]=0x64;
115.                      dia1[5]=0x7C;
116.                  }
117.                  for(j=0;j<10;j++){
118.                      UARTWriteBytes(2, dia1,6);
119.                  }
120.              }
121.
122.
123.              for(i=0;i<24;i++){
124.                  char dia2[6]={0x01,0x19,0x01,0x00,0x00,0x19};
125.                  if(martes[i]>0.5){
126.                      dia2[4]=0x64;
127.                      dia2[5]=0x7D;
128.                  }
129.                  for(j=0;j<10;j++){
130.                      UARTWriteBytes(2, dia2,6);
131.
132.                  }
133.              }
134.
135.              for(i=0;i<24;i++){
136.
137.                  char dia3[6]={0x01,0x19,0x02,0x00,0x00,0x1A};
138.                  if(miercoles[i]>0.5){
139.                      dia3[4]=0x64;
140.                      dia3[5]=0x7E;
141.                  }
142.                  for(j=0;j<10;j++){
143.                      UARTWriteBytes(2, dia3,6);
144.
145.                  }
146.              }
147.
148.              for(i=0;i<24;i++){
149.
150.                  char dia4[6]={0x01,0x19,0x03,0x00,0x00,0x1B};
151.                  if(jueves[i]>0.5){
152.                      dia4[4]=0x64;
153.                      dia4[5]=0x7F;
154.                  }
155.                  for(j=0;j<10;j++){
156.                      UARTWriteBytes(2, dia4,6);
157.
158.                  }
```

```
159.                }
160.
161.                for(i=0;i<24;i++){
162.
163.                    char dia5[6]={0x01,0x19,0x04,0x00,0x00,0x1C};
164.                    if(viernes[i]>0.5){
165.                        dia5[4]=0x64;
166.                        dia5[5]=0x78;
167.                    }
168.                    for(j=0;j<10;j++){
169.                        UARTWriteBytes(2, dia5,6);
170.
171.                    }
172.                }
173.
174.                for(i=0;i<24;i++){
175.
176.                    char dia6[6]={0x01,0x19,0x05,0x00,0x00,0x1D};
177.                    if(sabado[i]>0.5){
178.                        dia6[4]=0x64;
179.                        dia6[5]=0x79;
180.                    }
181.                    for(j=0;j<10;j++){
182.                        UARTWriteBytes(2, dia6,6);
183.
184.                    }
185.                }
186.
187.                for(i=0;i<24;i++){
188.
189.                    char dia7[6]={0x01,0x19,0x06,0x00,0x00,0x1E};
190.                    if(domingo[i]>0.5){
191.                        dia7[4]=0x64;
192.                        dia7[5]=0x7A;
193.                    }
194.                    for(j=0;j<10;j++){
195.                        UARTWriteBytes(2, dia7,6);
196.
197.                    }
198.                }
199.        }
200.
201.        void leerUart(char read_uart[],int longitud){
202.            int i;
203.            for(i=0;i<longitud;i++){
204.                if(read_uart[i]==0x07){  //Report_event
205.                    if(read_uart[i+1]==0x04){
206.                            temperaturaConsigna=read_uart[i+4];//Tempera
    tura Seleccionada
207.                    }
208.                    if(read_uart[i+1]==0x1E){
209.                        if(read_uart[i+2]==0x00){
210.                            estadoTermostato=read_uart[i+4];
211.                        }
212.                        if(read_uart[i+2]==0x01){ //Reset rutinas
213.                            Nsemanas[0]=0;
```

```
                        }
                        if(read_uart[i+2]==0x02){ //Modo automatico
                            modoAutomatico=read_uart[i+4];
                        }
                    }
                }

            }
        }

        void  rise_alarm() //interrupcion RTC
        {
            intRTC=1;
            if(temporizadorPantalla>0){
                temporizadorPantalla=temporizadorPantalla-1;
            }else IOPut(p5, off);
        }



        void external_interrupt_function() //PIR
        {
            UARTWrite(1,"Interrupcion PIR \n");
            struct tm my_time;
            RTCCGet(&my_time);
            int i=0;
            int j=0;

            if(my_time.tm_wday!=diaAnterior){ //Cuando termina el dia lo guardamos en la "base de datos"
                UARTWrite(1,"Cambio de dia \n");

                if(diaAnterior==MONDAY){
                    for(i=0;i<24;i++){
                        lunes[i]=(lunes[i]*Nsemanas[0]+diaActual[i])/(Nsemanas[0]+1);
                    }
                    if(EESaveData (0, lunes, 24, EE_BYTE)==0) UARTWrite(1,"The writing (DATA) is completed\r\n");
                }
                if(diaAnterior==TUESDAY){
                    for(i=0;i<24;i++){
                        martes[i]=(martes[i]*Nsemanas[0]+diaActual[i])/(Nsemanas[0]+1);
                    }
                    if(EESaveData ( (24*1), martes, 24 , EE_BYTE)==0) UARTWrite(1,"The writing (DATA) is completed\r\n");
                }
                if(diaAnterior==WEDNESDAY){
                    for(i=0;i<24;i++){
                        miercoles[i]=(miercoles[i]*Nsemanas[0]+diaActual[i])/(Nsemanas[0]+1);
                    }
```

```
264.                    if(EESaveData ( (24*2), miercoles, 24, EE_BYTE)==0) UARTWrite(1,"The writing (DATA) is completed\r\n");
265.                }
266.                if(diaAnterior==THURSDAY){
267.                    for(i=0;i<24;i++){
268.                        jueves[i]=(jueves[i]*Nsemanas[0]+diaActual[i])/(Nsemanas[0]+1);
269.                    }
270.                    if(EESaveData ( (24*3), jueves, 24, EE_BYTE)==0) UARTWrite(1,"The writing (DATA) is completed\r\n");
271.                }
272.                if(diaAnterior==FRIDAY){
273.                    for(i=0;i<24;i++){
274.                        viernes[i]=(viernes[i]*Nsemanas[0]+diaActual[i])/(Nsemanas[0]+1);
275.                    }
276.
277.                    if(EESaveData ( (24*4), viernes, 24, EE_BYTE)==0) UARTWrite(1,"The writing (DATA) is completed\r\n");
278.                }
279.                if(diaAnterior==SATURDAY){
280.                    for(i=0;i<24;i++){
281.                        sabado[i]=(sabado[i]*Nsemanas[0]+diaActual[i])/(Nsemanas[0]+1);
282.                    }
283.                    if(EESaveData ( (24*5), sabado, 24, EE_BYTE)==0) UARTWrite(1,"The writing (DATA) is completed\r\n");
284.                }
285.                if(diaAnterior==SUNDAY){
286.                    for(i=0;i<24;i++){
287.                        domingo[i]=(domingo[i]*Nsemanas[0]+diaActual[i])/(Nsemanas[0]+1);
288.                    }
289.                    if(EESaveData ( (24*6), domingo, 24, EE_BYTE)==0) UARTWrite(1,"The writing (DATA) is completed\r\n");
290.                }
291.
292.
293.                for(i=0;i<24;i++){
294.                    diaActual[i]=0;  //Borrar el dia una vez que lo hemos guardado
295.                }
296.            }
297.
298.            if(diaAnterior==SUNDAY && my_time.tm_wday==MONDAY){ //Detectamos cuando acaba la semana
299.                Nsemanas[0]++;
300.                EESaveData ( 200, Nsemanas, 1, EE_BYTE);
301.                UARTWrite(1,"Semana acabada \n");
302.            }
303.
304.            if(my_time.tm_min<30){ //Guardamos la presencia en dia actual
305.                diaActual[my_time.tm_hour]=1;
306.            }else diaActual[my_time.tm_hour+1]=1;
307.
```

```
308.          diaAnterior=my_time.tm_wday;
309.
310.          if(temporizadorPantalla>0 && temporizadorPantalla<12){ //Man
   tenemos la pantalla encendida si detectamos presencia
311.                  temporizadorPantalla++;
312.          }
313.      }
314.
315.      void FlyportTask()
316.      {
317.
318.          //Alimentar sensor t y h
319.          IOInit(p36,out);
320.          IOPut(p36, on);
321.
322.          //Alimentacion pantalla
323.          IOInit(p5,out);
324.          IOPut(p5, on);
325.
326.          //UART Pantalla
327.          // First of all: REMAP THE PINS
328.          IOInit(p15,UART3RX); // remap the p15 pin as UART3RX
329.          IOInit(p13,UART2TX); // remap the p13 pin as UART2TX
330.          //remember to enable the UARTs in the wizard
331.          UARTInit(2,9600);  //init the UART2
332.          UARTOn(2);          //enable the UART2
333.          UARTInit(3,9600); //init the UART3
334.          UARTOn(3);         //enable the UART3
335.
336.          int msglen = 0;
337.          char rcv_uart[200],uart_msg[200];
338.
339.       //Intruduccion manual de la fecha en el primer inicio
340.          char flagRTC=1;
341.          char barra=0;
342.          vTaskDelay(200);
343.          enviarForm13();
344.          while(flagRTC){
345.
346.
347.              msglen = UARTBufferSize(3); //check buffer of UART3
348.              UARTRead(3, rcv_uart, msglen);
349.              vTaskDelay(100);
350.              int i;
351.              for(i=0;i<msglen;i++){
352.                  if(rcv_uart[i]==0x07){  //Report_event
353.                      if(rcv_uart[i+1]==0x0D){//KeyBoard
354.                          if(rcv_uart[i+5]==0x07){//Enter
355.                              flagRTC=0;
356.                          }else
357.                          if(rcv_uart[i+5]==0x65){ // Barra/
358.                              barra++;
359.                          }else{
360.                              if(barra==0) mytime.tm_year=mytime.tm_ye
   ar*10+(rcv_uart[i+4]-0x30);//aÃ±o
```

```
361.                                 if(barra==1) mytime.tm_mon=mytime.tm_mon
    *10+(rcv_uart[i+4]-0x30);//mes
362.                                 if(barra==2) mytime.tm_mday=mytime.tm_md
    ay*10+(rcv_uart[i+4]-0x30);//Dia
363.                                 if(barra==3) { //Dia de la semana
364.                                     if(rcv_uart[i+4]==0x031) mytime.tm_w
    day=MONDAY;
365.                                     if(rcv_uart[i+4]==0x032) mytime.tm_w
    day=TUESDAY;
366.                                     if(rcv_uart[i+4]==0x033) mytime.tm_w
    day=WEDNESDAY;
367.                                     if(rcv_uart[i+4]==0x034) mytime.tm_w
    day=THURSDAY;
368.                                     if(rcv_uart[i+4]==0x035) mytime.tm_w
    day=FRIDAY;
369.                                     if(rcv_uart[i+4]==0x036) mytime.tm_w
    day=SATURDAY;
370.                                     if(rcv_uart[i+4]==0x037) mytime.tm_w
    day=SUNDAY;
371.                                 }
372.                                 if(barra==4) mytime.tm_hour=mytime.tm_ho
    ur*10+(rcv_uart[i+4]-0x30);// hora
373.                                 if(barra==5) mytime.tm_min=mytime.tm_min
    *10+(rcv_uart[i+4]-0x30);// minutos
374.
375.
376.
377.                             }
378.                         }
379.                     }
380.
381.                 }
382.
383.             }
384.             IOPut(p5, off);
385.             //Ajustar tiempo
386.             mytime.tm_year=mytime.tm_year+100;
387.             mytime.tm_mon=mytime.tm_mon-1;
388.             diaAnterior=mytime.tm_wday;
389.
390.             RTCCSet(&mytime); //Set time
391.             vTaskDelay(100);
392.
393.             RTCCGet(&mytime);//get actual data/time
394.             mytime.tm_sec=mytime.tm_sec+10;
395.
396.
397.             RTCCAlarmConf(&mytime,REPEAT_INFINITE,EVERY_TEN_SEC,rise_ala
    rm);//set alarm cada 10segundos
398.             RTCCAlarmSet(ON);//Enables alarm
399.
400.             //Interrupcion PIR
401.             IOInit(p17, indown);
402.             IOInit(p17, EXT_INT2);
403.             INTInit(2, external_interrupt_function, 1); // initializes e
    xternal interrupt 2 and associate a function
```

```
404.          INTEnable(2); // enables external interrupt 2.
405.
406.          // Pulsador Pantalla
407.          IOInit(p38, in);
408.
409.
410.          //Ini memoria
411.          EEInit(EE_ADDR_DEF, HIGH_SPEED, EE_SIZE_DEF); //Solo en Flyp
   ort Pro
412.          //Cargar datos EEPROM
413.          if(EELoadData (0, lunes, 24, EE_BYTE)==0) UARTWrite(1,"The r
   eading (DATA) is completed\r\n");
414.          if(EELoadData ( (24*1), martes, 24 , EE_BYTE)==0) UARTWrite(
   1,"The reading (DATA) is completed\r\n");
415.          if(EELoadData ( (24*2), miercoles, 24, EE_BYTE)==0) UARTWrit
   e(1,"The reading (DATA) is completed\r\n");
416.          if(EELoadData ( (24*3), jueves, 24, EE_BYTE)==0) UARTWrite(1
   ,"The reading (DATA) is completed\r\n");
417.          if(EELoadData ( (24*4), viernes, 24, EE_BYTE)==0) UARTWrite(
   1,"The reading (DATA) is completed\r\n");
418.          if(EELoadData ( (24*5), sabado, 24, EE_BYTE)==0) UARTWrite(1
   ,"The reading (DATA) is completed\r\n");
419.          if(EELoadData ( (24*6), domingo, 24, EE_BYTE)==0) UARTWrite(
   1,"The reading (DATA) is completed\r\n");
420.
421.          if(EELoadData ( 200, Nsemanas, 1, EE_BYTE)==0) UARTWrite(1,"
   The reading (DATA) is completed\r\n");
422.
423.
424.          //Configuracion rele
425.          IOInit(p1,out);
426.          IOPut(p1, off);
427.          IOInit(p3,out);
428.          IOPut(p3, off);
429.
430.
431.          //WIFI
432.          WFConnect(WF_DEFAULT);
433.          while(1)
434.          {
435.
436.
437.              vTaskDelay(100);
438.              msglen = UARTBufferSize(3); //check buffer of UART3
439.              if (msglen>2 && temporizadorPantalla>0)
440.              {
441.                  UARTRead(3, rcv_uart, msglen); // read from UART3
442.
443.                  leerUart(rcv_uart,msglen);
444.
445.                  sprintf(uart_msg, "TemperaturaDeseada %i\r\n",temper
   aturaConsigna);
446.                  UARTWrite(1, uart_msg);
447.                  vTaskDelay(10);
448.                  sprintf(uart_msg, "Estado del Termostato %i\r\n",est
   adoTermostato);
```

```
449.                    UARTWrite(1, uart_msg);
450.                    vTaskDelay(10);
451.                    sprintf(uart_msg, "Modo automatico %i\r\n",modoAutom
    atico);
452.                    UARTWrite(1, uart_msg);
453.                    vTaskDelay(10);
454.
455.
456.                }
457.                if(estadoTermostato==1){
458.                    if(temperaturaReal < temperaturaConsigna-
    1){ //Ciclo de histeresis
459.
460.                        flagOff=1;
461.                        if(flagOn==1){
462.                            enviarOn();
463.                            IOPut(p1, on);//Caldera a on
464.                            Delay10us(500);
465.                            IOPut(p1, off);
466.                        }
467.                    }
468.                    if(temperaturaReal > temperaturaConsigna+1 ){
469.
470.                        flagOn=1;
471.                        if(flagOff==1){
472.                            enviarOff();
473.                            IOPut(p3, on);//Caldera a off
474.                            Delay10us(500);
475.                            IOPut(p3, off);
476.                        }
477.                    }
478.                }
479.                if(estadoTermostato==0){
480.                    if(flagOff==1){
481.                    IOPut(p3, on);//Caldera a off
482.                    Delay10us(500);
483.                    IOPut(p3, off);
484.                    enviarOff();
485.                    }
486.                }
487.                if(modoAutomatico==1){ //Si ha esa hora tenemos guardado
     presencia habitual ponemos 21 grados de temepratura consigna
488.                    if(mytime.tm_wday==MONDAY && lunes[mytime.tm_hour]==
    1) temperaturaConsigna=21; else temperaturaConsigna=0;
489.                    if(mytime.tm_wday==TUESDAY && martes[mytime.tm_hour]
    ==1) temperaturaConsigna=21; else temperaturaConsigna=0;
490.                    if(mytime.tm_wday==WEDNESDAY && miercoles[mytime.tm_
    hour]==1) temperaturaConsigna=21; else temperaturaConsigna=0;
491.                    if(mytime.tm_wday==THURSDAY && jueves[mytime.tm_hour
    ]==1) temperaturaConsigna=21; else temperaturaConsigna=0;
492.                    if(mytime.tm_wday==FRIDAY && viernes[mytime.tm_hour]
    ==1) temperaturaConsigna=21; else temperaturaConsigna=0;
493.                    if(mytime.tm_wday==SATURDAY && sabado[mytime.tm_hour
    ]==1) temperaturaConsigna=21; else temperaturaConsigna=0;
494.                    if(mytime.tm_wday==SUNDAY && domingo[mytime.tm_hour]
    ==1) temperaturaConsigna=21; else temperaturaConsigna=0;
```

```
495.
496.                 }
497.
498.                 if(intRTC==1){
499.
500.                     RHT(&temperaturaReal,&humedadReal);
501.
502.
503.
504.                     char out_uart_msg[200];
505.                     RTCCGet(&mytime);//get actual data/time
506.                     sprintf ( out_uart_msg, "Alarm rised at: %s\r\n", as
    ctime (&mytime) );
507.                     UARTWrite(1, out_uart_msg);
508.
509.                     if(temporizadorPantalla>0){
510.                         enviarTemperatura(temperaturaReal);
511.                         vTaskDelay(10);
512.                         enviarHumedad(humedadReal);
513.                         vTaskDelay(10);
514.                     }
515.                     intRTC=0;
516.                 }
517.
518.                 if(!IOGet(p38)){
519.                     temporizadorPantalla=12; //120 segundos
520.                     IOPut(p5, on);
521.                     vTaskDelay(150);
522.                     UARTWrite(1,"Interrupcion pulsador");
523.                     vTaskDelay(50);
524.                     enviarTemperatura(temperaturaReal);
525.                     vTaskDelay(50);
526.                     enviarHumedad(humedadReal);
527.                     vTaskDelay(50);
528.                     enviarRutina();
529.                     vTaskDelay(50);
530.                     actualizarPantalla();
531.
532.                 }
533.
534.             }
535.         }
```

Anexo III.II. Interfaz de la pantalla

- **Home**

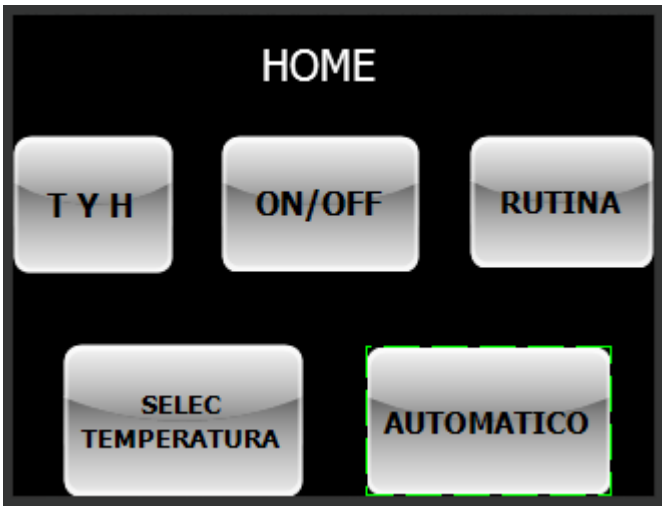


Ilustración 33: Pantalla Home

Esta es la pantalla del menú principal del termostato, desde la cual se puede acceder a todas las pantallas del interfaz, y se puede volver desde cualquiera de ellas a esta.

- **Temperatura y humedad**

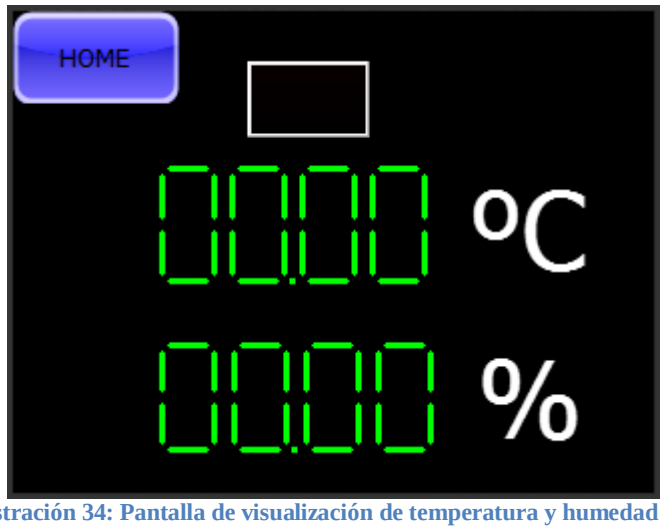


Ilustración 34: Pantalla de visualización de temperatura y humedad

En esta pantalla se visualiza la temperatura y la humedad, que se actualiza cada 10 segundos.

- **Selección de temperatura**

Ilustración 35: Pantalla de selección de temperatura

En esta pantalla se selecciona la temperatura deseada deslizando el indicador de la barra.

- **Encender y apagar termostato**

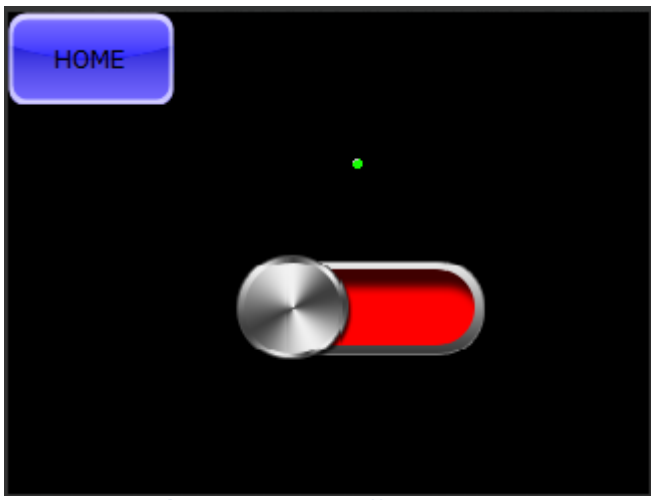


Ilustración 36: Pantalla on/off del termostato

En esta pantalla se enciende y se apaga el termostato.

- **Menú de visualización de rutina**

Ilustración 37: Pantalla de menú de visualización de rutinas

Desde esta pantalla podemos acceder a la pantalla de rutina de todos los días de la semana.

- **Visualización rutina lunes**

Ilustración 38: Pantalla de visualización de rutina del lunes

En esta pantalla se puede visualizar una grafica con la rutina del lunes, se representa de forma binaria, valor alto cuando hay presencia y en bajo cuando no, el eje horizontal se corresponde con las horas del dia.

- **Selección modo automático**

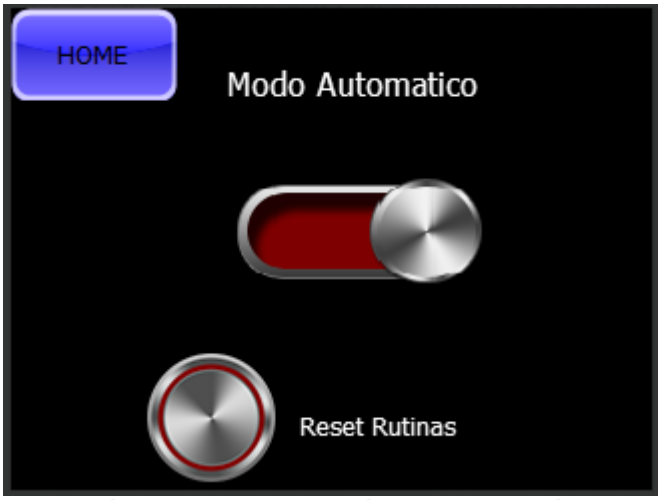


Ilustración 39: Pantalla de selección de modo automático

Desde esta pantalla pasa al modo automático del termostato y además se puede resetear el registro de rutinas.

- **Introducción de fecha**

Ilustración 40: Pantalla de introducción de fecha

Esta es la pantalla que aparece la primera vez que se enciende el termostato, y sirve para poner en fecha al termostato.