Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Control de una caldera con procesador ARM Cortex M4

Autor:

Cano Ortega, Pablo

Tutor:

Plaza Pérez, Francisco

Departamento de Tecnología Electrónica

Valladolid, 11 de febrero de 2023

Resumen

Los sistemas domóticos están cada día más presentes en nuestras viviendas, permitiéndonos un control más sencillo y completo de las mismas. Además, la integración de este tipo de sistemas con los dispositivos inteligentes y el Internet de las cosas posibilita su control incluso cuando estamos fuera de la vivienda.

Este proyecto busca explorar los campos anteriores y crear, a partir de un microcontrolador, un termostato inteligente que controle la caldera de una vivienda. La configuración del termostato se podrá realizar usando su pantalla táctil o con la aplicación Android desarrollada.

Palabras clave: Calefacción, Microcontrolador, Termostato inteligente, ESP8266, Aplicación Android

Abstract

The domotic systems are more present day by day in our homes, these systems enable us to have a simpler and more complete control of the houses. In addition, the integration of said systems with smart devices and the Internet of Things make the outdoors control of our home a possibility.

This project aims to explore said fields and build, from the base of a microcontroller, a smart thermostat capable of controlling the house boiler. The thermostat configuration can be done by using its touch screen or with the developed Android application.

Keywords: Heating, Microcontroller, Smart thermostat, ESP8266, Android application

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE DIAGRAMAS

GLOSARIO

A/D: Analógico/Digital.

FTP: *File Transfer Protocol.* Protocolo de transferencia de archivo.

Geofencing: se basa en la generación de una zona virtual, correspondiente a un área geográfica real, que se usa para poder controlar la localización, la entrada y la salida de dispositivos inteligentes en la zona.

GPIO: *General Purpose Input/Output*. Entrada/Salida de Propósito General.

HTML: *HyperText Markup Language.* Lenguaje de Marcas de Hipertext.

HTTP: *Hypertext Transfer Protocol.* Protocolo de transferencia de hipertexto.

I²C: *Inter-Integrated Circuit*. Circuito inter-integrado.

IDE: *Integrated Development Environment.* Entorno de desarrollo integrado.

IEEE: *Institute of Electrical and Electronics Engineers.* Instituto de Ingenieros Eléctricos y Electrónicos.

Internet de las cosas: hace referencia a sistemas que hacen uso de sensores, procesadores, *software* y que intercambian información a través de Internet u otras redes de comunicación.

IU: Interfaz de Usuario.

LQFP: *Low-profile Quad Flat Package*. Tipo de encapsulado de un circuito integrado que tiene pines por sus cuatro lados.

RAM: *Random Access Memory*. Memoria de acceso aleatorio.

RGB: *Red Green Blue*.

RISC: *Reduced Instruction Set Computing*. Conjunto de instrucciones reducido. Filosofía de diseño utilizado en microprocesadores y microcontroladores, se caracteriza por usar un número reducido de instrucciones de tamaño fijo.

RTC: *Real Time Clock*. Reloj de tiempo real.

SDRAM: *Synchronous Dynamic Random Access Memory*. Memoria de acceso aleatorio síncrona y dinámica.

SPDT *Relay*: *Single Pole Double Throw Relay*. Relé de un conmutador y dos vías.

SPI: *Serial Peripheral Interface.* Interfaz periférica serial.

SRAM: *Static Random Access Memory*. Memoria estática de acceso aleatorio.

SWD: *Serial Wire Debug*.

TCP: *Transmission Control Protocol.* Protocolo de control de transmisión.

TFT LCD: *Thin-Film-Transistor Liquid-Crystal Display*. Pantalla de cristal líquido de transistor de películas finas.

Timeout: parámetro de tiempo que tiene transcurrir para que un determinado evento tenga lugar.

UART: *Universal Asynchronous Receiver-Transmitter.* Transmisor-Receptor Asíncrono Universal.

UDP: *User Datagram Protocol.* Protocolo de datagramas de usuario.

URL: Uniform Resource Locator. Localizador de recursos uniforme.

USART: *Universal Synchronous and Asynchronous Receiver-Transmitter*. Transmisor-Receptor Asíncrono y Síncrono Universal.

USB OTG: USB *On The Go*. Conexión que permite a dispositivos, como teléfonos inteligentes o tabletas, actuar como host de otros dispositivos como ratones, *pendrives* o teclados. También permite que los teléfonos inteligentes actúen como unidades de almacenamiento ante ordenadores.

USB: *Universal Serial Bus*.

XML: *eXtensible Markup Language.* Lenguaje de Marcado Extensible.

Introducción

Introducción obtenida de (Gúzman Navarro, 2015)

> La domótica es la tecnología que automatiza las funciones e instalaciones de un edificio o vivienda para aumentar la comodidad, la seguridad y el ahorro de la energía. Los sistemas de este campo monitorizan y/o controlan servicios de una vivienda como la iluminación, la calefacción, el aire acondicionado, los sistemas de riego, etc. También pueden realizar funciones de seguridad como controlar entradas y alarmas de la casa.
>
> La domótica se basa principalmente en tres campos: la electrónica, la informática y las comunicaciones, integrándolas en un único sistema aplicado sobre una estructura arquitectónica. Cuando estos sistemas tienen conexión a Internet forman lo que se conoce como el Internet de las cosas.
>
> Los sistemas de domótica actuales buscan agrupar los distintos servicios, descritos anteriormente, en un único sistema central. Este sistema central permite integrar y controlar los diferentes sistemas, automatizados o no, cada uno de ellos de manera independiente, desde una única ubicación y con una simple actuación. Al centralizar los sistemas en una única interfaz de usuario, compartiendo información entre ellos, se puede reducir el número de sensores, el cableado necesario y el número de controladores.

En la **Figura 1** se puede ver un ejemplo de un sistema central de domótica que integra el control sobre varios servicios del hogar como la seguridad, la calefacción o la iluminación.

Figura 1: Ejemplo de un sistema central de domótica.

Fuente: (CASADOMO, 2018)

Un sistema domótico tiene que cumplir su función respetando dos principios fundamentales:

- Tiene que ser sencillo de utilizar, es decir, no puede conllevar la necesidad de que el usuario tenga conocimientos avanzados para obtener resultados adecuados.
- Su implantación debe respetar la funcionalidad habitual que el usuario está acostumbrado a realizar. El sistema debe adaptarse para tener un aspecto y uso parecido al conocido por la mayoría de los usuarios.

Ventajas e inconvenientes de la domótica

La domótica tiene ventajas y desventajas, como todas las tecnologías y campos de la ingeniería. Tanto el usuario como el fabricante deben tener en cuenta estos puntos sobre los sistemas domóticos. A continuación, se muestran algunos de ellos.

➢ Ventajas:

- **Comodidad**: los sistemas domóticos liberan a las personas de tareas repetitivas o no deseadas. Como ejemplo, automatizar persianas enrollables según la hora.
- **Gestión de la energía**: con una buena gestión se logra un ahorro energético y económico. Esto se consigue usando una correcta programación y regulación del sistema, por parte del usuario, que permita utilizar la cantidad de energía necesaria cuando se necesite. Como ejemplo, la programación del encendido de la calefacción en las horas en las que estamos en casa.
- **Seguridad**: los sistemas domóticos mejoran la seguridad de las personas y los bienes. Se pueden detectar incendios y actuar sobre ellos al instante, por ejemplo.
- **Menor coste de instalación**: hay un ahorro económico a la hora de realizar la obra, ya que se necesita menos cableado, menos sensores y controladores.

- Inconvenientes:
    - **Falta de técnicos especializados en un sistema:** los usuarios son reacios a implementar un sistema domótico en particular ante la posibilidad de carecer de técnicos, ajenos a la empresa fabricante, que sean capaces de solucionar futuros problemas. Cosa que no sucede en las instalaciones eléctricas convencionales.
    - **Coste elevado**: mayor coste de este tipo de productos respecto a los dispositivos convencionales que cumplen la misma finalidad.
    - **Sobrecoste en la vivienda**: la implementación de domótica en una vivienda ocasiona un coste muy lejos del valor real del inmueble.
    - **La fama de la tecnología**: debido al coste que supone la implementación de domótica en una vivienda, como se ha comentado en los dos puntos anteriores, esta se ha asociado con personas de gran poder adquisitivo. Por ello se han considerado estos sistemas como artículos de lujo y, por tanto, prescindibles.

Futuro de la domótica

La domótica sigue expandiéndose en el mercado, cada vez es más común ver algún sistema de este tipo en una vivienda, sobre todo porque los precios de esta tecnología se están diversificando. Muchos dispositivos siguen manteniendo precios altos que los hacen prohibitivos para muchos hogares, a cambio de tener un mayor número de funcionalidades y características de vanguardia. Pero la aparición de sistemas con precios más asequibles para personas con un menor nivel adquisitivo ha propiciado un aumento en el número de estos en los hogares españoles.

Las previsiones auguran un gran crecimiento de la domótica en España. En un artículo publicado en europapress se afirma que

> En el caso de España se espera que el 20% de las casas para 2024 tengan por lo menos un dispositivo conectado. Esto supone un crecimiento del 300% respecto a las cifras de 2018 y un mercado de más de mil millones. Esta expansión se verá reflejada especialmente en el sector de la seguridad en el hogar. (Portaltic/EP, 2020)

Aunque el mercado de esta tecnología crece en España, todavía se encuentra muy lejos de países como Estados Unidos, China, Japón o Reino Unido.

Objetivos y justificación del proyecto

Objetivos

El objetivo principal de este trabajo es desarrollar todo el *software* necesario, usando un procesador ARM Cortex M4, para el correcto funcionamiento de un termostato inteligente. El microcontrolador presente en el kit de desarrollo STM32F429I-DISC1 integra dicho procesador, este kit se usará como base para realizar el termostato que controle la caldera de una vivienda.

La configuración del termostato se podrá realizar desde una pantalla táctil, la cual integra el propio kit de desarrollo, y desde una aplicación Android. Los modos de funcionamiento que podrá tener el termostato son: modo manual y modo programa. En el modo manual, simplemente, se elegirá la temperatura deseada. En el modo programa, se elegirán dos temperaturas y, luego, se decidirá cuál de ellas será la deseada según el día y la hora. Además, el modo actual utilizado y la temperatura actual, medida por un sensor, deberán ser mostrados en la pantalla inicial de la aplicación Android y de la pantalla táctil.

Justificación

Este trabajo busca afianzar los conocimientos adquiridos durante la carrera y demostrar las capacidades obtenidas para la correcta realización de proyectos relacionados con la electrónica y la automática.

El proyecto requiere el uso de habilidades en los campos de la programación, la electrónica, la automatización, el control y las comunicaciones, todos vistos, en cierta manera, en el grado. Por lo tanto, este trabajo fin de grado es perfecto para finalizar la carrera usando una “mezcla” de las competencias propias de esta.

Descripción general del proyecto

En este apartado se va a explicar, de forma general, el proyecto final desarrollado para dar una visión completa del mismo.

Partiendo de un kit de desarrollo STM, con pantalla táctil incorporada, añadimos un sensor de temperatura y un circuito integrado ESP8266, presente en el módulo ESP-12S. El sensor de temperatura se utiliza para obtener la medida de la temperatura actual en la vivienda. El ESP-12S permite la comunicación, a través de Internet, entre el microcontrolador y la aplicación Android. Todos estos añadidos, junto con la programación necesaria, nos permiten obtener un termostato inteligente.

La configuración del termostato se puede realizar desde la pantalla táctil o desde la aplicación Android. Esta configuración permite al usuario elegir entre el modo manual y el modo programa. En el modo manual simplemente introducimos la temperatura deseada. En el modo programa podemos elegir dos temperaturas y confeccionar un programa para cada día de la semana en función de la hora.

El termostato, mediante un pin de salida digital, actúa sobre un relé que controla la caldera. Según el nivel de tensión del pin, se abre o se cierra el relé. Si el relé se cierra, la caldera se activa. Si el relé se abre, la caldera se desactiva.

En la **Figura 2** se puede ver un esquema de lo explicado.

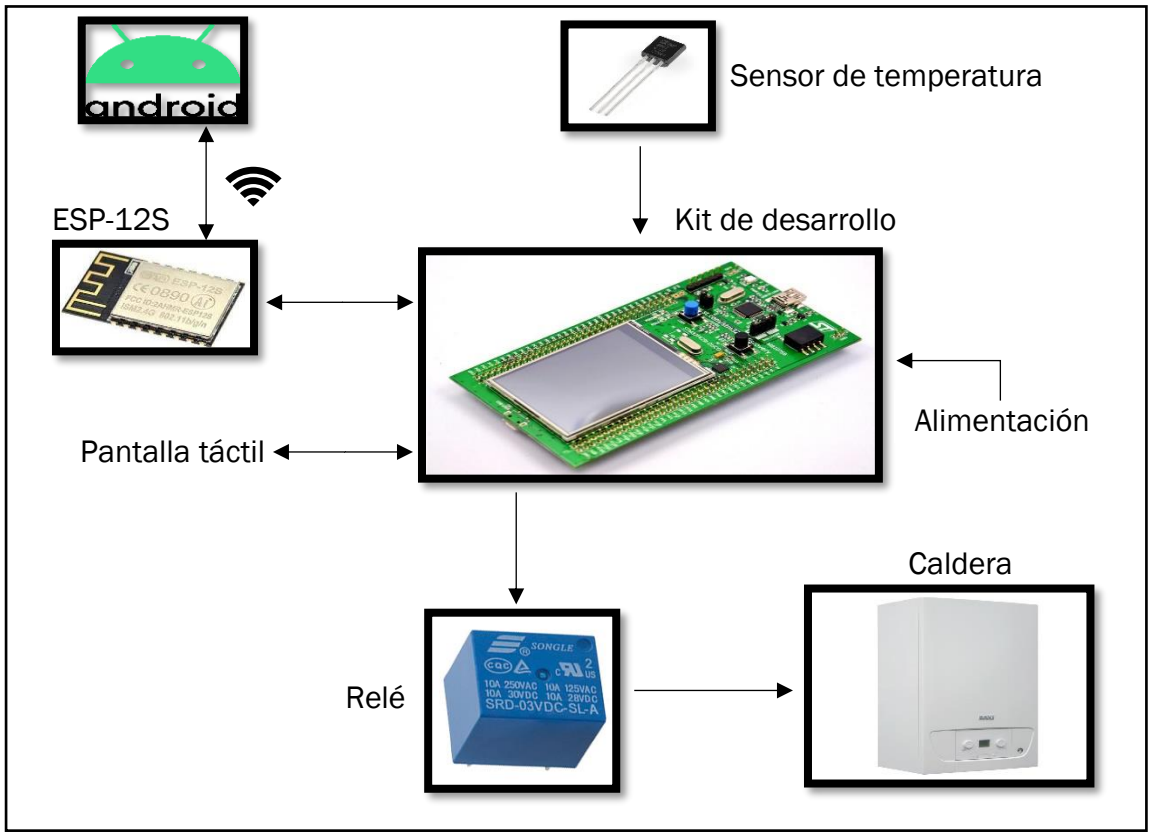

Figura 2: Esquema general del proyecto.

Fuente: Elaboración propia.

Análisis de mercado

En este apartado se van a revisar algunos termostatos inteligentes presentes en el mercado actual. Además, se han buscado productos de precios dispares para ver las diferentes funcionalidades ofrecidas en función del precio.

Sensi Touch ST75

Figura 3: Termostato Sensi Touch ST75.

Fuente: (Emerson, s.f.)

Termostato inteligente que se puede manejar a través de la pantalla táctil, con comandos de voz y con una aplicación móvil. Este termostato permite visualizar la temperatura, la humedad y diversas estadísticas sobre su uso. El dispositivo puede controlar la calefacción y el aire acondicionado de la vivienda.

Se puede configurar un programa para los 7 días de la semana y, también, se puede seleccionar el modo no programable para elegir la temperatura deseada directamente. Además, este termostato hace uso de *geofencing*, mediante el cual el sistema puede modificar sus parámetros en función de la localización en la que se el dispositivo móvil del usuario.

Precio: 169€ (Medio)

Nest Learning Thermostat

Figura 4: Termostato Nest Learning Thermostat.

Fuente: (Google, s.f.)

Termostato inteligente que se puede manejar a través de la pantalla táctil, con comandos de voz y con una aplicación móvil. Mide temperatura y humedad de la vivienda, además tiene sensores de presencia que encienden la pantalla si alguien pasa cerca del dispositivo.

Este termostato tiene las mismas funcionalidades que el anterior, tiene modo programable, modo no programable y *geofencing*. También se pueden ver estadísticas de los 10 últimos días de uso del termostato. Pero este producto tiene una característica particular, es autoprogramable. Con esta prestación, después de usarlo durante una semana, el dispositivo se adapta a las preferencias del usuario de forma automática. El sistema crea un programa a partir de la información que obtiene, pero puede ser modificado manualmente por el usuario.

Precio: 249€ (Alto)

Amazon Smart Thermostat

Figura 5: Termostato Amazon Smart Thermostat.

Fuente: (Amazon, s.f.)

Es uno de los termostatos inteligentes más asequibles del mercado, se puede manejar mediante una aplicación móvil y mediante comandos de voz. Este dispositivo puede controlar la calefacción y el aire acondicionado de la vivienda, mide, también, la temperatura y la humedad de esta.

El termostato tiene modo programable, modo no programable y estadísticas de uso. Carece de varias funcionalidades que tienen otros termostatos, como los dos anteriores. Además, la pantalla del dispositivo no es táctil, solo puede mostrar por pantalla la temperatura y el modo actual.

Precio: 60€ (Bajo)

Capítulo 1: Termostato físico

1.1 Kit de desarrollo STM32F429 Discovery

1.1.1 Introducción

En este proyecto se ha usado el kit STM32F429 Discovery (STM32F429I-DISC1) como pieza central. El STM32F429I-DISC1 es un kit de desarrollo con un precio asequible y fácil de utilizar, perfecto para empezar a trabajar en aplicaciones como la de este trabajo. Como elementos más importantes que integra el kit podemos citar: un procesador ARM Cortex M4, un microcontrolador STM32F429ZIT6 y una pantalla táctil. En la **Figura 6** se puede ver el dispositivo utilizado.

Figura 6: Kit de desarrollo STM32F429 Discovery.

Fuente: (AliExpress, s.f.)

1.1.2 Características

Información del kit obtenida de (ST, 2020).

Las características que ofrece el kit STM32F429 Discovery son las siguientes:

- Microcontrolador STM32F429ZIT6 presente en un encapsulado tipo LQFP de 144 pines, con 2 MB de memoria Flash y 256 KB de RAM.
- El circuito ST-LINK/V2 está presente en la placa. Este permite programar y depurar los microcontroladores STM8 y STM32.
- Dos formas de alimentar la placa: a través de un conector USB cable tipo-A a Mini-B o a través de una fuente externa de alimentación de 3V o 5V. La fuente externa debe estar conectada al pin de GND y al pin de 3V (o de 5V).
- SDRAM 64 Mbits (1 Mbit x 16 bit x 4 bancos) con modo de refresco automático y ahorro de energía.
- Dos pulsadores en la placa (usuario y reset).
- Pines de extensión conectados con las entradas/salidas del microcontrolador para una conexión rápida en la placa de pruebas y poder realizar aplicaciones de forma sencilla.
- TFT LCD de 2.4", 262K colores RGB, resolución de 240 x 320 pixeles.
- Capacidad de usar USB OTG a través del conector tipo Micro-AB.
- 6 LEDs:
  - LD1 (rojo/verde) para las comunicaciones a través del conector USB.
  - LD2 (rojo) para indicar que la placa tiene alimentación.
  - Dos LEDs configurables por el usuario: LD3 (verde) y LD4 (rojo)
  - Dos LEDs USB OTG: LD5 (verde) VBUS y LD6 (rojo) OC.

1.1.3 Diagrama de bloques

El STM32F429 Discovery está diseñado entorno al microcontrolador STM32F429ZIT6 que tiene un encapsulado LQFP de 144 pines. La **Figura 7** muestra el diagrama de bloques de las conexiones entre el microcontrolador y los periféricos que tiene el kit (LEDs, pulsadores, ST-LINK/V2, conexiones USB, etc.).

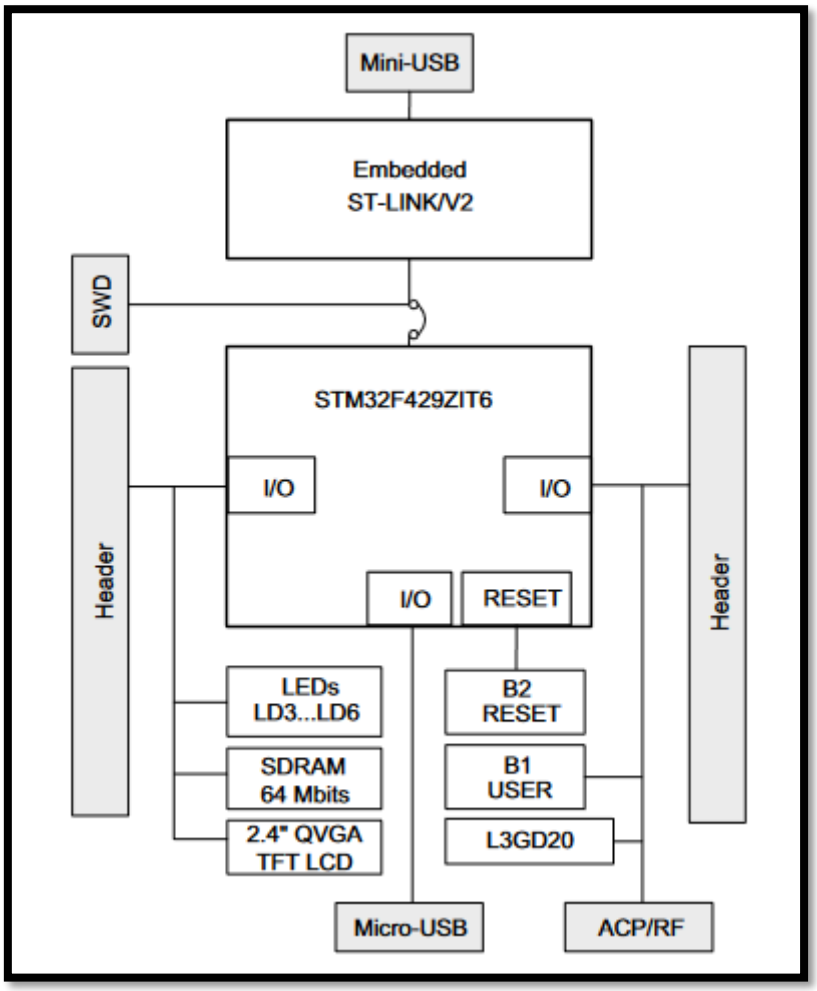

Figura 7: Diagrama de bloques del STM32F429 Discovery.

Fuente: (ST, 2020).

1.1.4 Microcontrolador

El microcontrolador STM32F429ZIT6 es un componente basado en el procesador de alto rendimiento ARM Cortex M4, el cual tiene una arquitectura RISC de 32 bits y puede llegar a funcionar a una frecuencia de 180 MHz. Este procesador tiene una unidad de coma flotante de simple precisión. Además, implementa un conjunto completo de instrucciones y una unidad de protección de memoria que aumenta la seguridad.

El STM32F429ZIT6 incorpora memorias embebidas de alta velocidad (2 Mbytes de memoria flash, 256 Kbytes de SRAM), hasta 4 Kbytes SRAM de reserva y un gran número de entradas/salidas y periféricos conectados a buses.

En la **Tabla 1** se muestran algunas de las propiedades más importantes del microcontrolador con el que trabajamos.

Microcontrolador STM32F429ZIT6	
Procesador	Cortex M4
Frecuencia de reloj	Hasta 180 MHz
Memoria flash	2 MB
SRAM	256 + 4 KB
Tensión de alimentación (Vcc)	3.3V
Vcc máxima	3.6V
Vcc mínima	1.7V
Tensión a nivel alto en pines de entrada/salida	3.3V, Rango: 1.7V – 3.6V
Corriente máxima de salida en un pin	25mA
Resolución máxima del convertidor analógico/digital y digital/analógico	12 bits
Hasta 21 pines de interfaces de comunicación	I^2C, USART, UART, SPI

Tabla 1: Propiedades del microcontrolador STM32F429ZIT6.

Fuente: Elaboración propia a partir de la información obtenida en (ST, 2018).

1.2 Desarrollo del termostato

1.2.1 Introducción

Antes de empezar con la configuración del dispositivo, vamos a recuperar parte de la **Figura 2**. En la **Figura 8** podemos ver la parte que nos interesa en este punto del proyecto, que no es otra que la del microcontrolador.

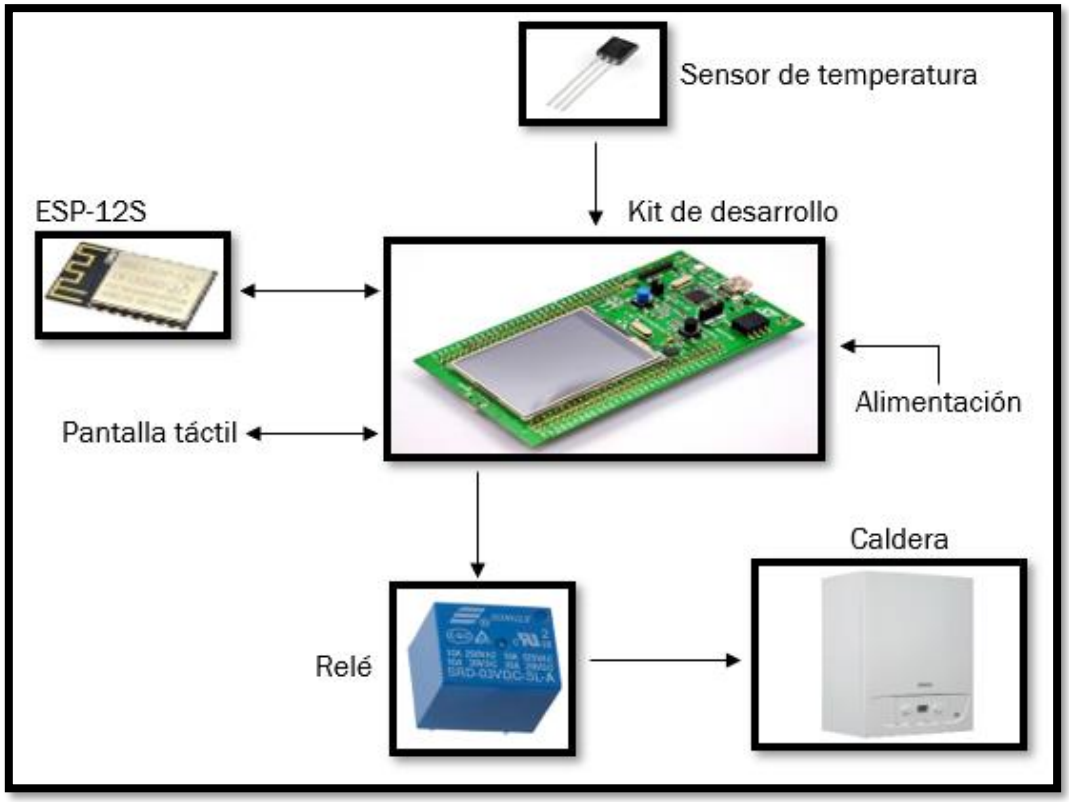

Figura 8: Esquema de la parte del microcontrolador.

Fuente: Elaboración propia.

A partir de la **Figura 8**, podemos trazar las líneas generales de los siguientes apartados. Vamos a tener que habilitar un pin que haga de convertidor analógico/digital para obtener el valor de la temperatura que mide el sensor. También tendremos que habilitar un pin de salida que pueda activar, cuando se cumplan las condiciones necesarias, el relé que controla la caldera. Por último, tendremos que habilitar un puerto UART para permitir la comunicación entre el ESP-12S y el microcontrolador. Por otro lado, para usar la pantalla táctil únicamente tendremos que incluir algunas librerías en el programa.

1.2.2 Configuración de pines y relojes

Para realizar la configuración de los pines y relojes del microcontrolador vamos a usar el programa STM32CubeMX. En este programa elegimos un microcontrolador (o una placa) y, luego, podemos habilitar pines de distinto tipo, modificar sus parámetros y establecer la frecuencia de los relojes. En nuestro caso, tenemos que elegir la placa STM32F429I-DISC1 o el microcontrolador STM32F429ZIT6.

En la **Figura 9** se muestra una captura del programa, en ella aparece el microcontrolador con los pines que hemos habilitado (aparecen en verde).

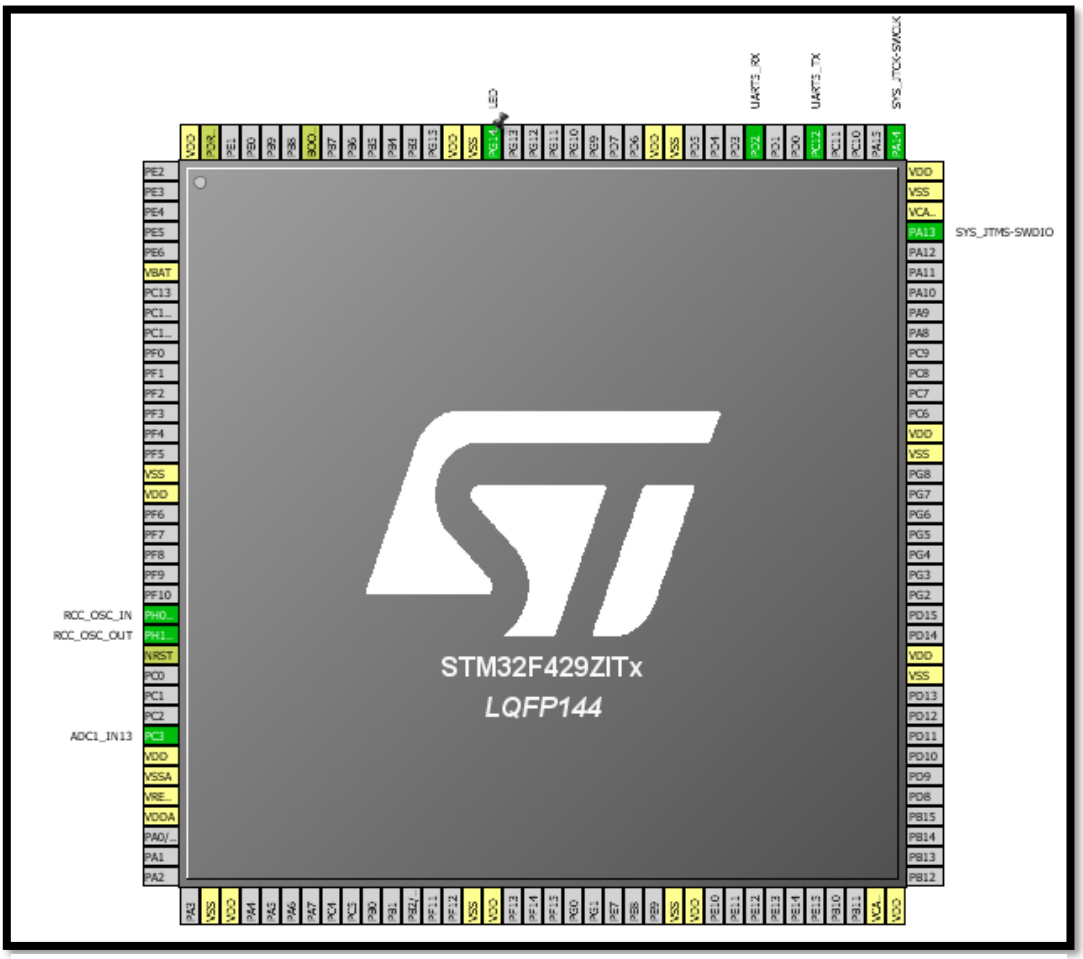

Figura 9: Captura de los pines habilitados en el microcontrolador.

Fuente: Elaboración propia.

A continuación, se van a mostrar los pines habilitados, ampliando la **Figura 9** para verlos claramente.

Como se puede ver en las **Figuras 10 y 11**, se ha habilitado el convertidor analógico/digital en el pin PC3 y el puerto UART5 en los pines PD2 (RX) y PC12 (TX).

El pin PG14, que se ha llamado LED, se ha establecido como una salida digital; este pin GPIO gobierna el LED LD4 de la placa. Se ha habilitado este pin, ya que no vamos a usar un relé en el montaje físico del proyecto, por lo que se ha decidido que el LED haga las veces de relé. Cuando el LED se enciende, quiere decir que el relé se cierra y se activa la caldera.

Los pines PH0 Y PH1 aparecen cuando habilitamos el reloj externo de alta velocidad. Mientras que los pines PA13 Y PA14 aparecen cuando habilitamos la depuración del dispositivo mediante SWD. Además de todo lo anterior, se ha activado el reloj de tiempo real del sistema.

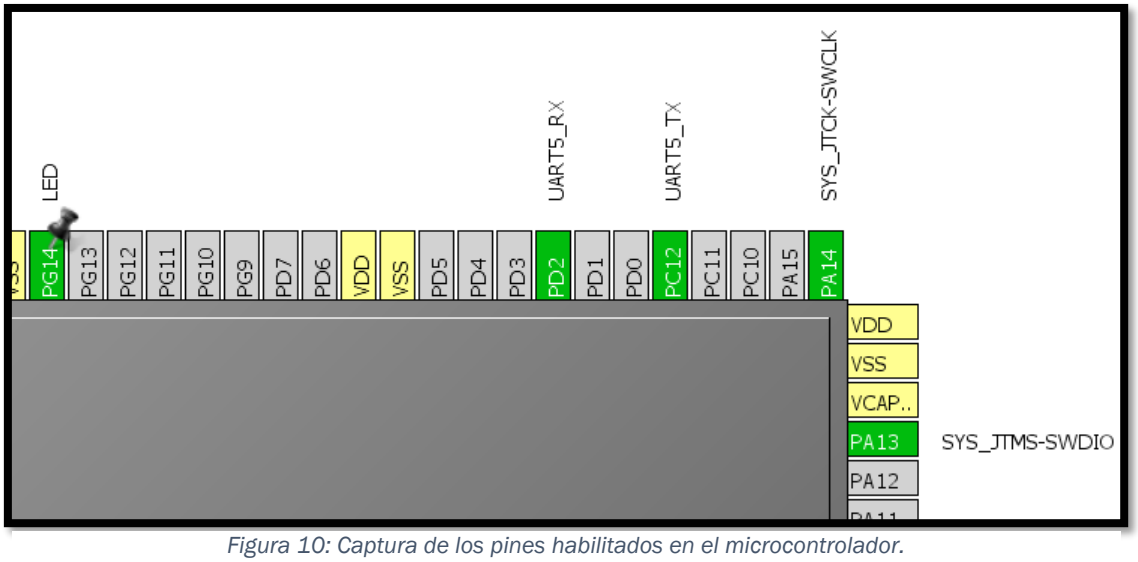

Figura 10: Captura de los pines habilitados en el microcontrolador.

Fuente: Elaboración propia.

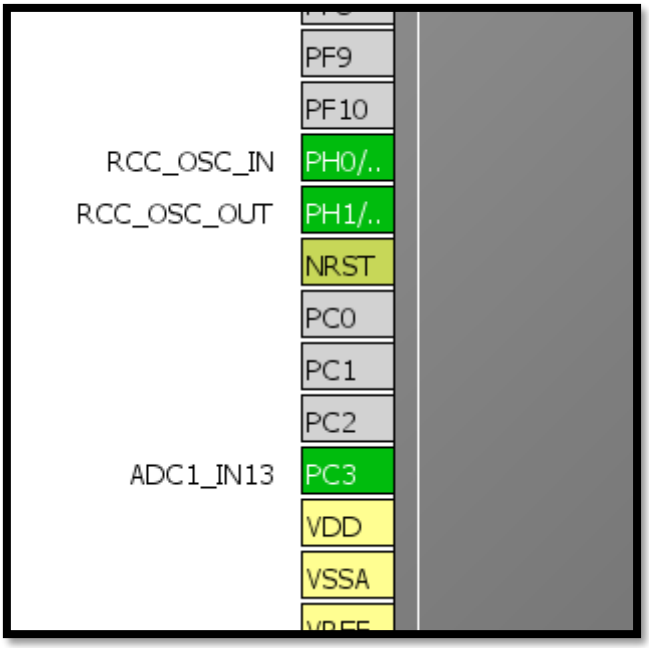

Figura 11: Captura de los pines habilitados en el microcontrolador.

Fuente: Elaboración propia.

Las frecuencias de los relojes del dispositivo se han configurado con el máximo valor posible, ya que en este proyecto no nos vamos a enfocar en reducir el consumo del termostato, ni en los costes asociados al desarrollo de este. En la **Figura 12** se pueden ver los valores de estas frecuencias.

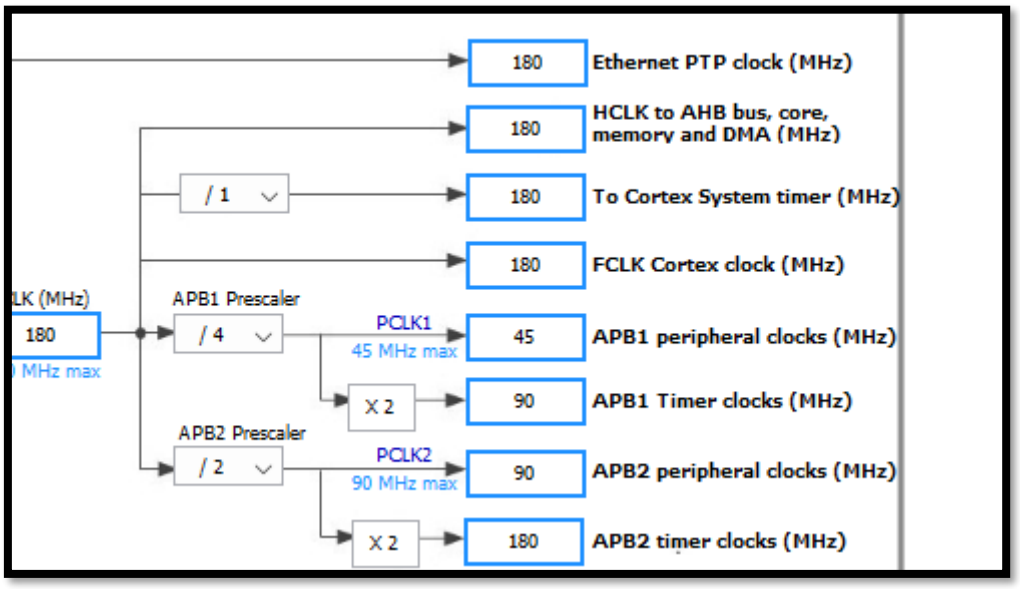

Figura 12: Frecuencias aplicadas en el microcontrolador.

Fuente: Elaboración propia.

Finalmente, se han configurado los parámetros del convertidor analógico/digital, del reloj de tiempo real y del puerto UART5. En la **Figura 13** se puede ver la configuración del convertidor. En este caso se ha puesto una resolución de 12 bits (la máxima posible) para tener mayor exactitud en la medida de la temperatura.

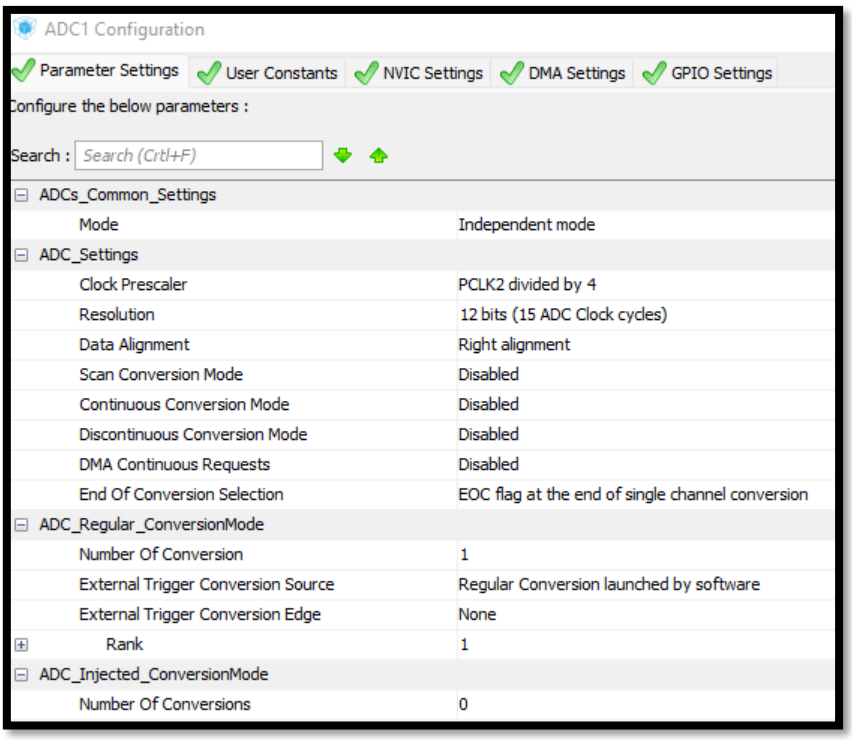

Figura 11: Parámetros del convertidor analógico/digital.

Fuente: Elaboración propia.

En la **Figura 14** se puede ver la configuración del reloj de tiempo real. Se han definido la fecha y la hora, en formato de 24 horas, iniciales al encender el dispositivo.

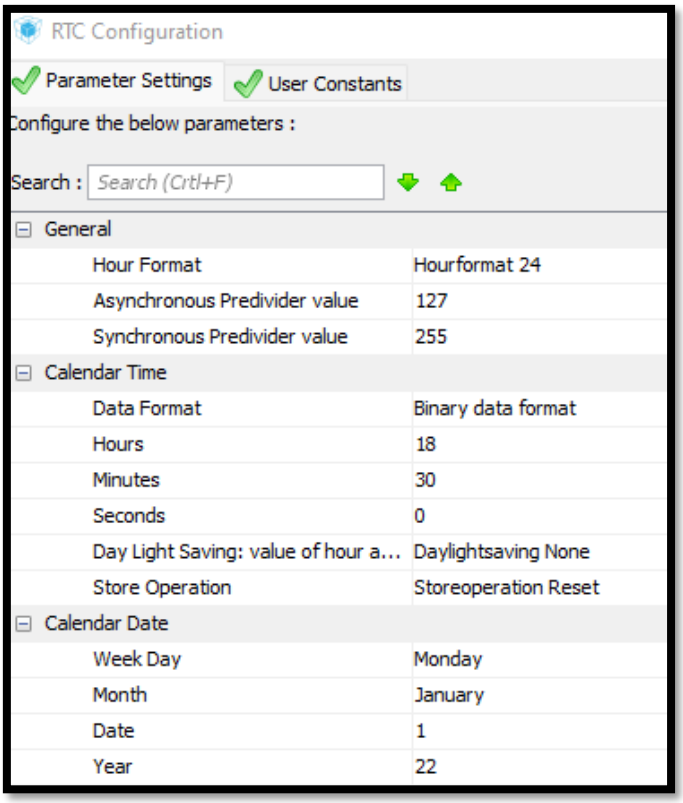

Figura 12: Parámetros del reloj de tiempo real.

Fuente: Elaboración propia.

En la **Figura 15** se puede ver la configuración del puerto UART5, este puerto será el que permita la comunicación entre el ESP8266 y el microcontrolador. Se ha establecido los baudios a 9600, ya que el ESP trabaja con ese valor de velocidad.

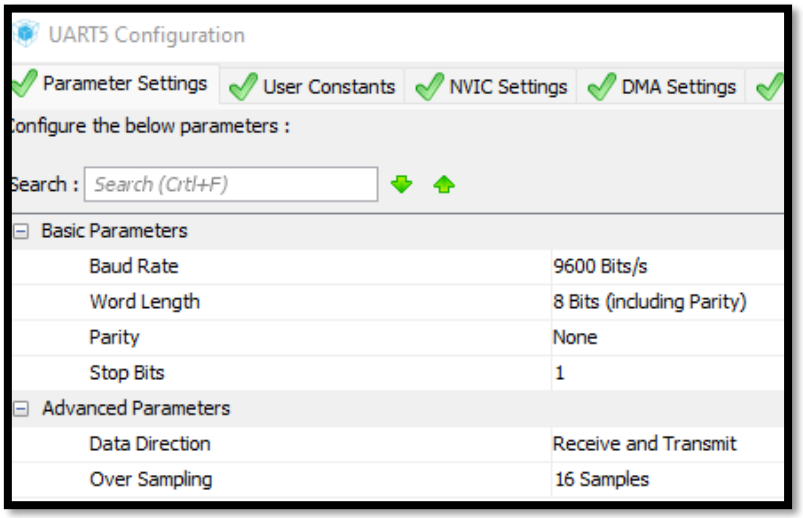

Figura 13: Parámetros del puerto UART.

Fuente: Elaboración propia.

A modo de resumen de los pines más importantes del proyecto, y que se van a ver en los apartados siguientes, se ha realizado la **Tabla 2**.

Pin	Función	Características
PC3	Convertidor Analógico/Digital	Resolución de 12 bits
PG14	Salida digital que controla el LED LD4	El LED "actúa" como relé
PD2	UART5 Receptor (RX)	Tasa de baudios = 9600 Bd Interrupción habilitada
PC12	UART5 Transmisor (TX)	

Tabla 2: Resumen de los pines habilitados.

Fuente: Elaboración propia.

1.2.3 Alimentación del dispositivo

Como se vio en el apartado **1.1.2 Características,** el kit de desarrollo STM32F429I-DISC1 se puede alimentar a través de un conector USB cable tipo-A a Mini-B o a través de una fuente externa de alimentación de 3V o 5V, a través de los pines homónimos. Durante la realización de este proyecto, el STM32F429I-DISC1 se ha alimentado usando un conector USB cable tipo-A a Mini-B desde una de las salidas USB de un ordenador.

Es importante definir cómo va a ser la alimentación del termostato inteligente final, ya que, como se verá en el apartado de Mejoras, una de las líneas futuras de desarrollo es completar el diseño del dispositivo totalmente y ofrecer un termostato inteligente en el mercado. Para acometer este objetivo hay que definir la tensión de alimentación que va a necesitar el dispositivo final. Esta tensión va a depender de los componentes presentes en el termostato final. A continuación, se presentan estos componentes y sus tensiones de alimentación.

- **Pantalla LCD**: la pantalla táctil TFT LCD ILI9341 del kit de desarrollo STM32F429I-DISC1 funciona con una tensión de alimentación de 3.3V.
- **Microcontrolador STM32F429ZIT6**: el microcontrolador funciona con una tensión de alimentación de 3.3V.
- **Sensor de temperatura TMP36**: el sensor puede funcionar con una tensión de alimentación de entre 2.7V y 5V.
- **ESP-12S**: este módulo funciona con una tensión de alimentación de 3.3V.
- **Relé SRD-03VDC-SL-C**: este relé funciona con una tensión de bobina de 3V.

Una vez vistos los componentes y sus tensiones, podemos concluir, en un primer intento de diseño, que la tensión de alimentación del dispositivo debe estar en torno a los 3.3V.

Para conseguir esta tensión de alimentación podemos usar baterías o pilas, de esta forma, podremos tener un termostato que no dependa de una conexión a la red. Además, muchos termostatos actuales funcionan usando baterías. Aparte de las pilas, usaremos un regulador de tensión para obtener los 3.3V que necesitamos y tener una tensión de alimentación estable.

En caso de que el dispositivo final consumiera más potencia de lo que la presente alimentación mediante baterías y regulador pudiera ofrecer, habría que considerar mantener la alimentación a través de conexión USB presente en el kit de desarrollo. Esta opción, en caso de tener un consumo considerable, permitiría no estar cambiando las baterías de tanto en cuanto y ofrecer una potencia adecuada a las necesidades del dispositivo, sin aumentar el tamaño de la placa de circuito impreso final. Además, el termostato físico está pensado para estar en el mismo lugar de la vivienda todo el tiempo, por lo que este tipo de alimentación sería interesante.

El regulador de tensión que se va a usar es el regulador de 800mA AMS1117-3.3 del fabricante ADMOS (*Advanced Monolithic Systems*). Las propiedades más importantes de este dispositivo se han recogido en la **Tabla 3**.

Regulador de tensión AMS1117-3.3	
Tensión de entrada (Vin)	4.75V – 12V
Tensión de salida (Vout)	3.3V
Corriente de salida (Iout)	Hasta 800mA
Voltaje de caída típico	1.1V
Temperatura máxima	125°C
Resistencia térmica unión-aire ($\emptyset_{j-a}$)	80°C/W

Tabla 3: Propiedades del regulador AMS1117-3.3

Fuente: (Advanced Monolithic Systems).

Lo más importante que se puede sacar de la **Tabla 3** es que, para obtener una tensión de salida de 3.3V, debemos aplicar una tensión de entrada de, como mínimo, 1.1V más que la salida, es decir, alrededor de 4.4V.

El circuito de alimentación ideado se presenta, simulado en el programa Proteus, en la **Figura 16.** Consta de una alimentación de 4.5V (3 pilas de 1.5V en serie), 2 condensadores, en la entrada y salida (el fabricante recomienda un valor de 22µF) del regulador, para estabilizar la tensión y el propio regulador de tensión. Cabe destacar que, como el AMS1117 no estaba presente en la librería del programa, se ha utilizado un regulador de tensión muy parecido al anterior, el LM1117. El resultado de la simulación, como se puede ver, es satisfactorio, ya que obtenemos en la salida los 3.3V deseados.

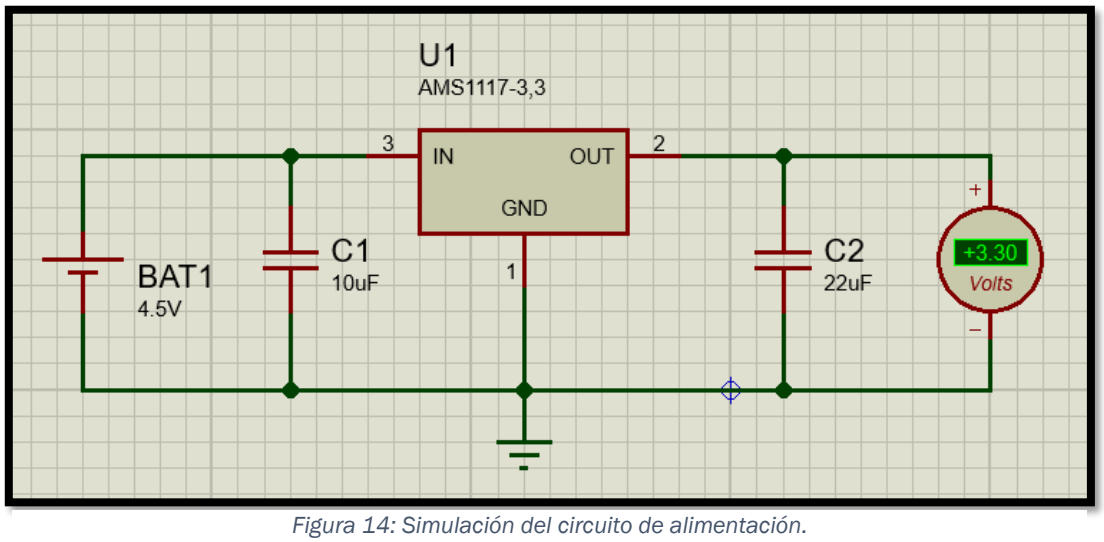

Figura 14: Simulación del circuito de alimentación.

Fuente: Elaboración propia.

En cuanto al uso de un radiador, podemos ver que, en principio, no sería necesario añadir uno, ya que tenemos margen de temperatura suficiente, teniendo en cuenta que el termostato se va a situar dentro de un domicilio. Los cálculos, suponiendo una corriente máxima de 800mA, para llegar a esta conclusión han sido los siguientes.

$$Pdisipada = (Vin - Vout) * Iout \quad [1]$$

$$Pdisipada = (4.5 - 3.3) * 0.8 = 0.96W$$

$$Temperatura\ unión - aire = \emptyset_{j-a} * Pdisipada \quad [2]$$

$$Temperatura\ unión - aire = 80\ \frac{°C}{W} * 0.96W = 76.8°C$$

$$Margen = Temperatura\ máxima - Temperatura\ unión - aire \quad [3]$$

$$Margen = 125°C - 76.8°C = 48.2°C$$

Por lo visto anteriormente, podemos decir que, suponiendo que el dispositivo consume una corriente máxima de 800mA, la temperatura ambiente del lugar de la vivienda donde se encuentre el termostato no debe superar los 48.2°C.

1.2.4 Sensor de temperatura y convertidor A/D

El sensor de temperatura que se ha utilizado en el proyecto es el TMP36 del fabricante Analog Devices. Se trata de un sensor de temperatura analógico que necesita una baja tensión de alimentación para funcionar, entre 2.7V y 5.5V, y puede medir temperaturas entre -40ºC y 125ºC. El sensor proporciona un voltaje de salida que es linealmente proporcional a la temperatura en grados centígrados. Además, tiene una precisión de ±1ºC a 25ºC y de ±2ºC en el rango de temperatura de -40ºC a 125ºC. En la **Figura 17** se puede ver la imagen del sensor TMP36 y su *pinout*.

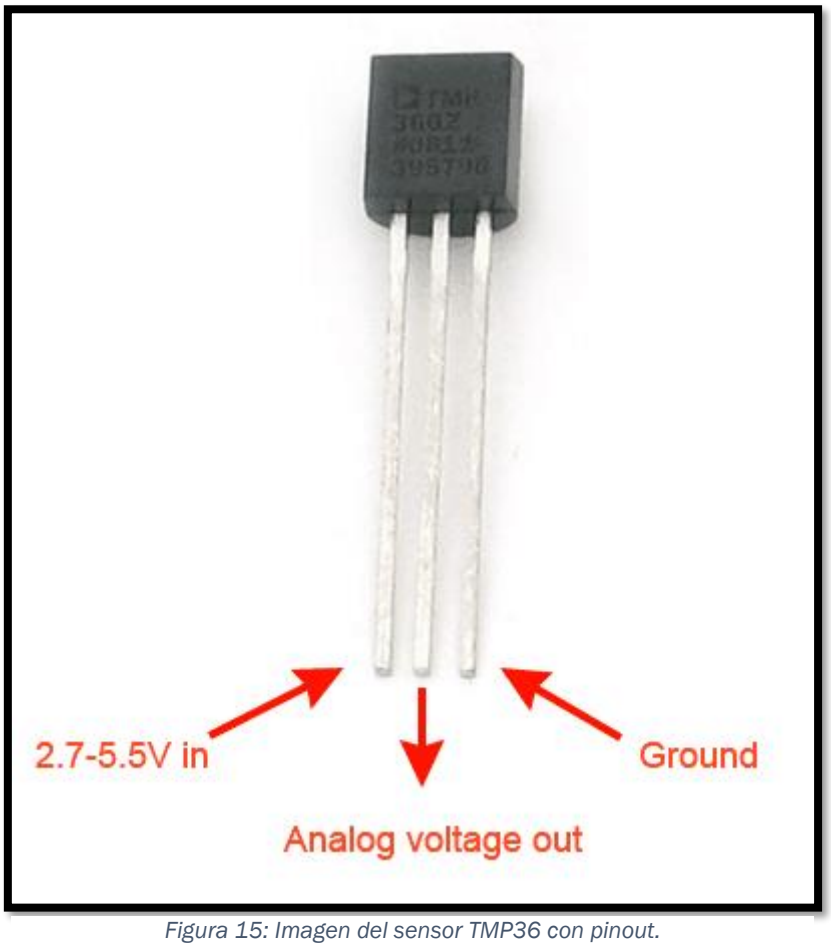

Figura 15: Imagen del sensor TMP36 con pinout.

Fuente: (ada, 2012).

Se ha elegido este sensor ya que la baja impedancia de salida, el voltaje de salida proporcionalmente lineal y su calibración precisa son muy útiles en circuitos de control de temperatura y convertidores analógico/digitales. Además, el uso de este sensor en aplicaciones con microcontroladores como Arduino, o el microcontrolador que se usa en este proyecto, es recomendable debido a su fácil manejo, buena precisión y su asequible precio.

Algunas de las propiedades más importantes del sensor TMP36 se recogen en la Tabla 4.

Sensor de temperatura TMP36	
Tensión de alimentación (Vs)	2.7V a 5.5V
Corriente de alimentación (Is)	Menos de 50μA
Rango de temperatura	-40ºC a 125ºC
Precisión	±1ºC a 25ºC ±2ºC de -40ºC a 125ºC
Voltaje de salida a 25ºC	750mV
Rango del voltaje de salida	100mV a 2V
Rango de la corriente de salida	0μA a 50μA
Error por autocalentamiento	0.1ºC
Encapsulado	TO-92 con 3 pines

Tabla 4: Propiedades del sensor TMP36.

Fuente: (ANALOG DEVICES, 2010).

El sensor tendrá su pin de tensión de alimentación conectado al pin de 5V de la placa, mientras que el pin de tierra estará conectado al pin de tierra de la placa. El pin analógico del sensor, que da el voltaje de salida, estará conectado al pin de la placa habilitado con el convertidor analógico/digital. Si observamos la Tabla 2, podemos ver que se trata del pin PC3 y que tiene una resolución de 12 bits. Para obtener la medida de la temperatura, realizada por el sensor, tenemos que hacer una serie de cálculos sobre el valor en el pin PC3.

El valor del voltaje de entrada en el pin PC3 tendrá un valor de entre 0.1V y 2V, tal y como indica la Tabla 4. El convertidor de 12 bits transforma un valor de voltaje de entre 0V y el voltaje de referencia en el pin, el cual es de 3.3V según la Tabla 1, en un número entero entre 0 y 2 ^ (12 bits) - 1 = 4095. Hay que decir que el valor del voltaje de referencia se cambió de 3.3V a 2.93V, después de ver que un pin digital de salida a nivel alto daba entre 2.9V y 2.95V al realizar la medida con el voltímetro. Además, se hicieron pruebas con el valor del voltaje de referencia en el código del programa para comprobar su buen comportamiento.

Una vez expuesto lo anterior, el cálculo que tenemos que realizar para recuperar el voltaje proporcionado por el sensor es el siguiente:

$$Vsalida = lectura\ pin\ PC3 * \frac{Vref}{2^{nº\ de\ bits\ de\ resolución}} \quad [4]$$

$$\boldsymbol{Vsalida = lectura\ pin\ PC3 * \frac{2.93V}{2^{12}}} \quad [5]$$

Ahora que tenemos una forma de obtener la tensión de salida del sensor de temperatura, podemos calcular la temperatura fácilmente. Como la temperatura es linealmente proporcional con el voltaje de salida, podemos usar la ecuación de la recta para calcular cualquier temperatura a partir del voltaje.

Para obtener la ecuación usamos información del *Datasheet* del sensor. En la **Tabla 4** habíamos visto que a 25ºC el voltaje de salida es 0.75V. En la **Figura 18** se puede ver la gráfica del sensor que nos da el voltaje de salida en función de la temperatura. A partir de esta obtenemos fácilmente que a 50ºC el voltaje de salida es de 1V.

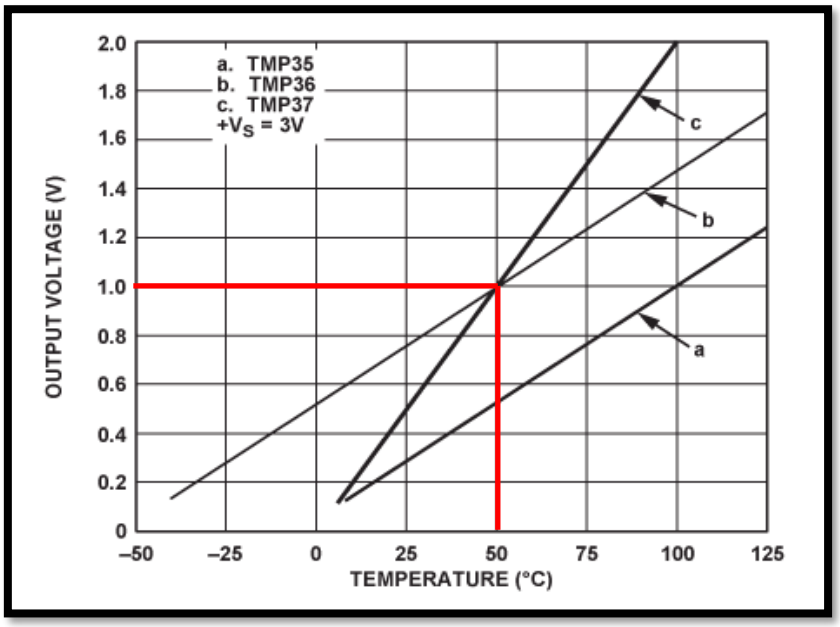

Figura 16: Gráfica del sensor TMP36 que relaciona el voltaje de salida con la temperatura.

Fuente: Elaboración propia a partir de la gráfica obtenida en (ANALOG DEVICES, 2010).

Una vez expuesto lo anterior, el cálculo que tenemos que realizar para obtener la temperatura es el siguiente:

$$y - y_1 = \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) \quad [6]$$

$$y - 0.75 = \frac{0.75 - 1}{25 - 50}(x - 25) \rightarrow y - 0.75 = \frac{1}{100}(x - 25)$$

$$Cambiando\ x\ por\ T\ e\ y\ por\ Vsalida: \boldsymbol{T = (Vsalida - 0.5) * 100} \quad [7]$$

1.2.5 Relé

A lo largo del trabajo se ha comentado que para manejar el circuito eléctrico de la caldera vamos a emplear un relé. Este dispositivo es controlado por un circuito eléctrico de baja potencia, normalmente, caracterizado por la presencia de componentes electrónicos. Al mismo tiempo, el relé controla un circuito de mayor potencia que el anterior. Básicamente, se trata de un interruptor que actúa como un punto de unión entre la electrónica y la electricidad. Con esto, se muestra la utilidad de este componente a la hora de encender o apagar la caldera de la vivienda.

Como se explicó en el apartado **1.2.2 Configuración de pines y relojes**, se ha usado el LED LD4 para que haga de relé, ya que no se ha incorporado este elemento en el montaje físico de este proyecto. El LD4 depende de la salida del pin digital PG14. Si hay una salida a nivel alto, el LED se enciende, lo que indica que, en ese caso, el relé se cerraría y la caldera se pondría en funcionamiento. Por el contrario, si hay una salida a nivel bajo, el LED se apaga, lo que indica que el relé se abriría y la caldera dejaría de funcionar.

El relé que se va a utilizar, teóricamente, en este trabajo es el SRD-03VDC-SL-C de la marca SONGLE RELAY, en la **Figura 19** se puede ver una imagen de este. Se trata de un relé SPDT que tiene una tensión de bobina de 3V en continua y una corriente nominal de 120mA, información obtenida de (SONGLE RELAY). Además, este relé soporta hasta 250V de alterna a 10A, por lo que puede controlar perfectamente una caldera eléctrica de una vivienda, ya que estas funcionan, normalmente, a una tensión alterna de 230V.

Figura 17: Imagen del relé utilizado.

Fuente: (Heschen, s.f.).

Como la corriente nominal del relé es de 120mA y, como se puede ver en la **Tabla 1**, la corriente máxima que puede ofrecer un pin del microcontrolador es de 25mA, debemos implementar un circuito *Relay Driver* para poder dar la corriente nominal suficiente y activar el relé.

Este circuito se basa en usar una resistencia limitadora y un transistor que actúe como amplificador de la corriente. También se usa el propio relé en paralelo con un diodo *flyback*, el cual conduce cuando el transistor deja de conducir, ya que se produce, de forma momentánea, un voltaje inverso en la bobina del relé.

Para poder activar el relé habría que habilitar un pin digital de salida en el microcontrolador, usando el programa STM32CubeMX. Este pin de salida y el pin de GND estarían conectados al circuito anteriormente descrito. Cuando se cumplieran las condiciones para incrementar la temperatura en la vivienda, este pin daría una salida a nivel alto, provocando el cierre del relé y la activación de la caldera. Además, como el relé tiene una tensión de bobina de 3V, puede ser activado por la tensión de alimentación prevista en el apartado **1.2.3 Alimentación del dispositivo**. El circuito indicado se puede ver en la **Figura 20**.

Hay que aclarar que, en función del modelo de la caldera y del fabricante, habría que conectar el relé a las bornas de la caldera de diferente forma. Los manuales de todas las calderas recomiendan que las conexiones de la caldera sean realizadas por un instalador autorizado.

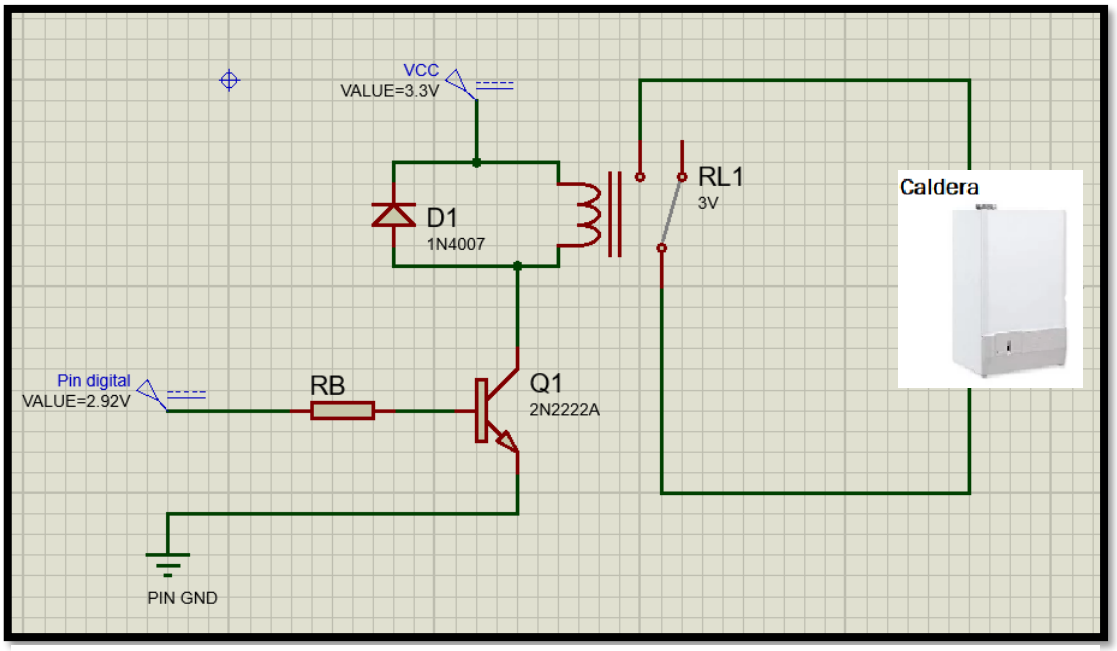

Figura 18: Circuito Relay Driver.

Fuente: Elaboración propia.

Para realizar el circuito *Relay Driver* debemos calcular y elegir los componentes de este. En primer lugar, elegimos un transistor adecuado, teniendo en cuenta que la corriente de colector debe ser de 120mA. Se ha elegido el transistor 2N2222A, ya que es un transistor típico de diseño y que, además, cuenta con una corriente de colector máxima de 800mA. En la **Tabla 5** se recogen las principales características de este.

Transistor 2N2222A	
Tipo	NPN
Tensión colector-base máxima (Vcb)	75V
Tensión colector-emisor máxima (Vce)	40V
Tensión emisor-base máxima (Veb)	6V
Corriente de colector máxima (Ic)	0.8A
Temperatura máxima	175°C
Ganancia de corriente (hfe)	100

Tabla 5: Propiedades del transistor 2N2222A.

Fuente: Elaboración propia a partir de la información obtenida en (PHILIPS, 1997).

A continuación, en la **Figura 21**, se presenta el circuito, que habíamos visto anteriormente, adaptado para calcular la resistencia de base y ser simulado en Proteus. Hay que indicar que, como se había expuesto en el apartado **1.2.4 Sensor de temperatura y convertidor A/D**, la tensión de salida de un pin digital a nivel alto está entre 2.9V y 2.95V.

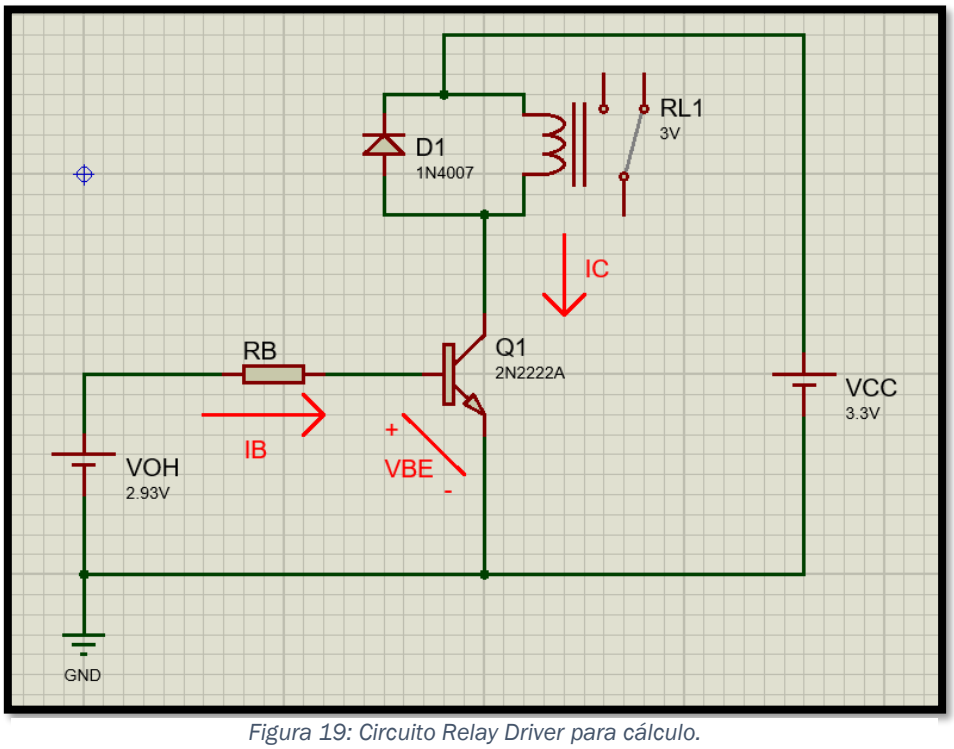

Figura 19: Circuito Relay Driver para cálculo.

Fuente: Elaboración propia.

Se proceden a realizar los cálculos pertinentes con los datos que se tienen.

$$Datos: V_{OH} = 2.93V, I_C = I_{relé} = 120mA, h_{fe} = 100$$

$$h_{fe} = \frac{I_C}{I_B} \rightarrow I_B = \frac{I_C}{h_{fe}} = \frac{0.12A}{100} = 0.0012A \quad [8]$$

Según el resultado **[8]**, la corriente de base mínima, para que el transistor pueda conducir la corriente de colector de 120mA, es de 1.2mA.

Como el transistor va a funcionar en conmutación, es decir, entre corte y saturación, debemos calcular tensión entre la base y el emisor en saturación. Queremos que el transistor, cuando se encuentre en saturación, conduzca una corriente de colector de 120mA. Acudimos al *datasheet* del transistor 2N2222A, en este encontramos la gráfica que relaciona las tensiones de saturación del transistor con la corriente de colector. En la **Figura 22** se puede ver que la tensión de saturación de base-emisor es de 0.85V.

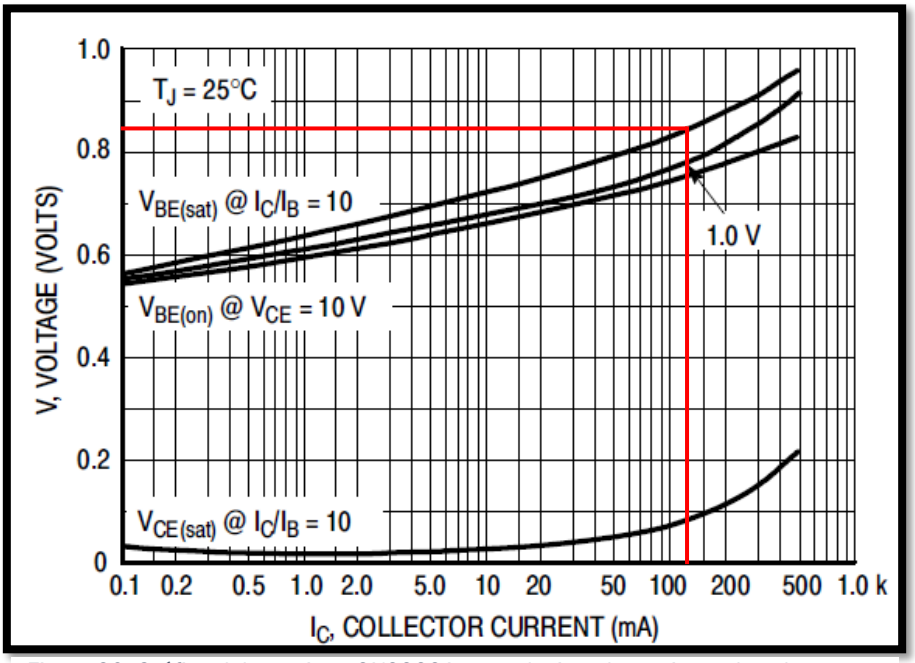

Figura 20: Gráfica del transistor 2N2222A que relaciona la corriente de colector con las tensiones.

Fuente: Elaboración propia a partir de la gráfica obtenida en (ON Semiconductor, 2013).

Ahora, con la tensión obtenida, procedemos a calcular la resistencia de base aplicando Kirchoff en la malla de entrada.

$$-V_{OH} + R_B * I_B + V_{BE(Sat)} = 0 \quad [9]$$

$$R_B = \frac{V_{OH} - V_{BE(Sat)}}{I_B} = \frac{2.93 - 0.85}{0.0012} = 1733.33\Omega \rightarrow R_B = 1800\Omega \quad [10]$$

Como se puede ver en el cálculo **[10]**, la resistencia de base tiene un valor de 1733.33Ω, que, usando un valor comercial, pasa a valer 1800Ω.

En la **Figura 23** se muestra la simulación del circuito para comprobar su funcionamiento. Podemos observar que el circuito funciona, ya que el relé pasa de abierto a cerrado ante una salida a nivel alto en el pin digital.

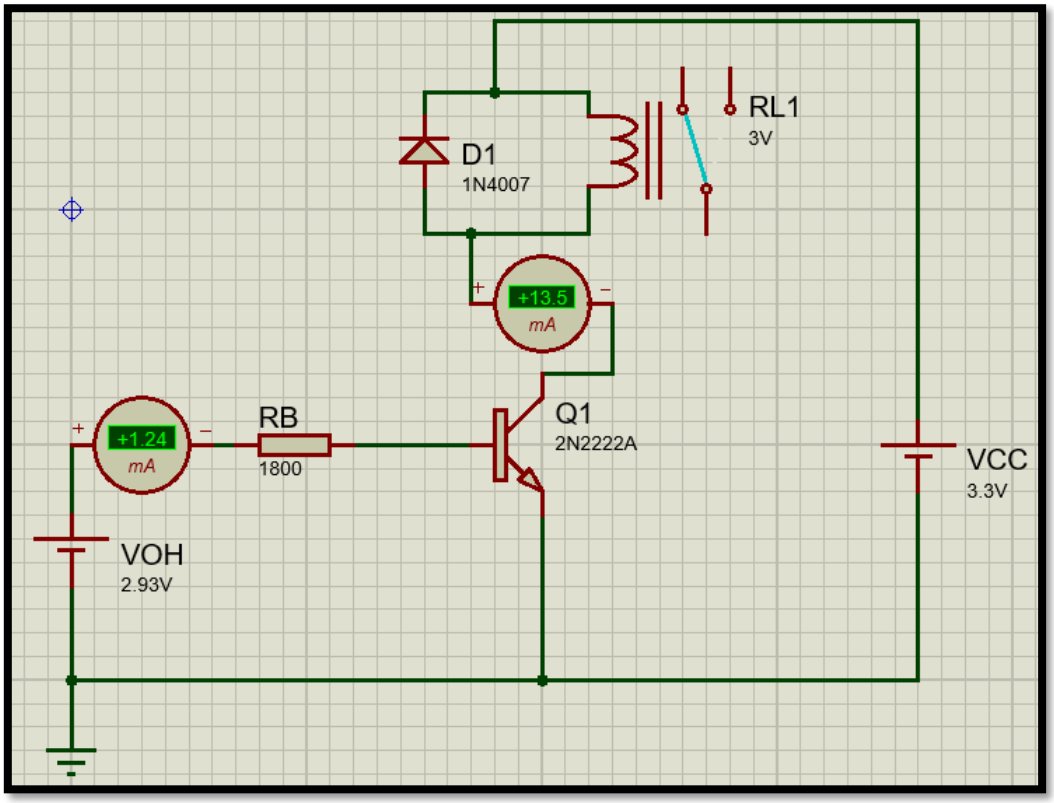

Figura 21: Simulación del circuito Relay Driver.

Fuente: Elaboración propia.

1.2.6 Reloj de tiempo real

Como se dijo en el apartado **1.2.2 Configuración de pines y relojes**, se ha activado el reloj de tiempo real del dispositivo. El RTC del microcontrolador mantiene actualizadas la fecha y hora inicialmente seleccionadas, siempre que el dispositivo se encuentre correctamente alimentado. La fecha y la hora del RTC se pueden cambiar a través de la pantalla táctil.

El reloj de tiempo real debe estar activado y correctamente sincronizado con la fecha y hora actuales. Esta es una condición necesaria para el buen funcionamiento del termostato, ya que en el modo programa se activa o desactiva la caldera en función de la temperatura seleccionada y de la fecha y hora actuales.

1.2.7 ESP8266 y comunicación UART

El ESP8266 es un microcontrolador fabricado por la empresa china Espressif, este circuito integrado permite la conexión a Internet mediante el uso de Wifi. Se trata de un componente muy utilizado en el desarrollo de aplicaciones relacionadas con el Internet de las cosas. En la **Figura 24** se puede ver una imagen de este.

Figura 22: Circuito integrado ESP8266.

Fuente: (Wikipedia, s.f.).

Las propiedades a destacar del ESP8266 se muestran en la **Tabla 6**.

Microcontrolador ESP8266	
Procesador	Tensilica L106 RISC de 32 bits
Frecuencia de reloj	Hasta 160 MHz
Tensión de alimentación (Vcc)	3.3V
Vcc máxima	3.6V
Vcc mínima	2.5V
Memoria RAM de datos	96 KBytes
Memoria RAM de instrucciones	64 KBytes
Protocolo	IEEE 802.11 b/g/n Wi-Fi
Interfaces de comunicación	UART, I^2C, SPI
Protocolos de comunicación	TCP, UDP, HTTP
Pines GPIO	17

Tabla 6: Propiedades del microcontrolador ESP8266.

Fuente: Elaboración propia a partir de la información obtenida en (Espressif Systems, 2022).

Hay un gran número de módulos que integran el ESP8266, siendo la mayoría de ellos muy parecidos. Todos se caracterizan por su bajo precio y la sencillez de su uso, diferenciándose en el número de pines habilitados. El módulo utilizado en este proyecto es el ESP-12S, el cual se puede ver en la **Figura 25**.

Figura 23: Imagen del ESP-12S.

Fuente: (WAVESHARE, s.f.).

El ESP-12S va a servir como puente para la comunicación entre el STM32F429I-DISC1 y la aplicación Android, como se pudo ver en la **Figura 2**. Esto se debe a que, como hemos visto antes, el ESP8266 ofrece la posibilidad de conexión a Internet y es compatible con el uso del protocolo de comunicación HTTP, el cual vamos a usar en este proyecto.

La comunicación entre el STM32F429I-DISC1 y el ESP-12S se va a realizar utilizando la interfaz de comunicación UART. Se ha usado UART en el proyecto porque se trata de una interfaz muy sencilla de implementar y ambos dispositivos son capaces de usarla.

Un dispositivo UART permite la comunicación serial asíncrona. Además, se pueden configurar los principales parámetros de transmisión de este (velocidad y formato de la información que se envía), como se pudo ver en la **Figura 15**. La transmisión serial asíncrona se caracteriza por el envío secuencial de los bits de un elemento de información, estando este conjunto de bits precedido por un bit de arranque y seguido de un bit de parada. El bit de arranque debe producir un cambio de estado en la línea de comunicación. El bit de parada debe tener el mismo estado que la línea cuando está inactiva. En la **Figura 26** se puede ver un ejemplo gráfico de lo explicado.

Figura 24: Ejemplo de transmisión asíncrona.

Fuente: (Pérez Turiel, 2021).

La conexión entre dos dispositivos UART se realiza conectando las tierras y cruzando sus pines de transmisión (TX) y recepción (RX) de información, es decir, TX se conecta con RX. Se puede ver en la **Figura 27** una conexión típica.

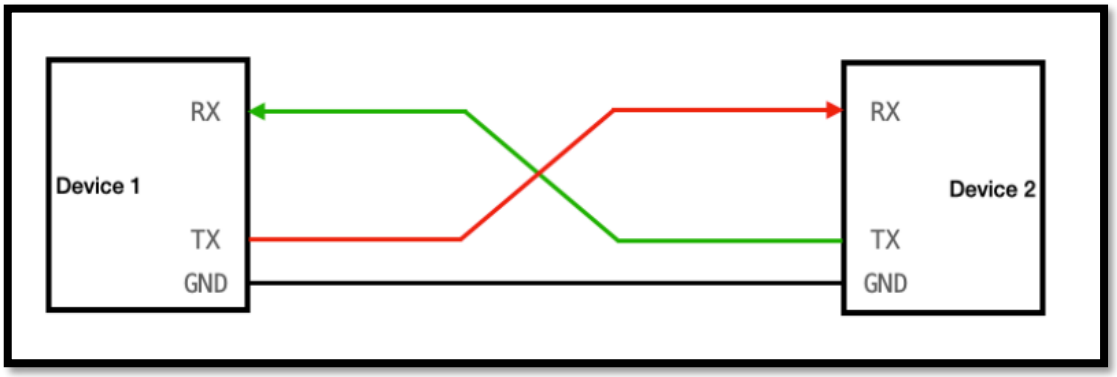

Figura 25: Conexión UART de dos dispositivos.

Fuente: (Hunter Adams, s.f.).

El puerto UART5, que se había habilitado para comunicarse con el ESP8266, se configuró a una tasa de 9600 baudios, como se puede ver en la **Tabla 2**. Se configuró a esa tasa de baudios porque es la tasa más utilizada en las aplicaciones de comunicación y la gran mayoría de los ESP8266 vienen configurados con esta tasa. Pero hay algunos dispositivos ESP8266 que usan 57600 u 115200 baudios, dependiendo de la versión de *firmware* que tengan instalada, por lo que es recomendable comprobar esta tasa para asegurarse del valor que hay que configurar.

Para realizar la conexión del ESP-12S y el STM32F429I-DISC1, primero hay que ver el *pinout* del ESP-12S, el cual aparece en la **Figura 28**. Los pines utilizados son los de UART, es decir, TXD0 Y RXD0, también se utiliza el pin de tensión de alimentación Vcc y el pin de tierra GND.

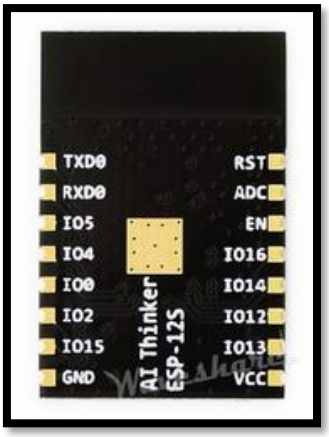

Figura 26: Pinout del ESP-12S.

Fuente: (WAVESHARE, s.f.).

Los pines utilizados del STM32F429I-DISC1 son el pin de tensión de 3V, los pines pertenecientes al puerto UART5 y el pin de tierra. Siguiendo lo establecido en la **Tabla 2**, la conexión queda como aparece en la **Figura 29**.

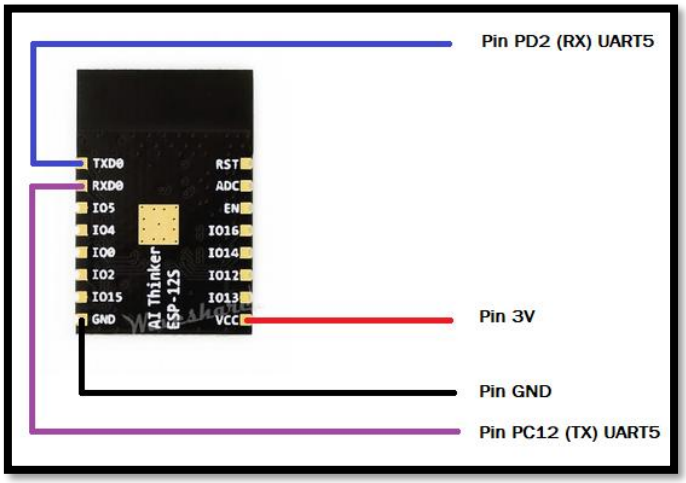

Figura 27: Conexión del ESP-12S con el microcontrolador.

Fuente: Elaboración propia.

Por último, hay que indicar que la conexión UART entre el ESP-12S y el STM32F429I-DISC1 se puede realizar sin problemas, ya que los niveles de tensión de ambos dispositivos son compatibles. La tensión de alimentación de ambos es de 3.3V, por lo que los niveles de tensión de entrada y salida en los pines de los dos dispositivos son bastante parecidos. Esta situación asegura mantener los niveles lógicos entre ambos dispositivos, es decir, que un '1' o un '0' lógico se siga entendiendo como tal en el otro dispositivo. Lo anterior permite que la comunicación UART se efectué correctamente.

Para inicializar y configurar el ESP8266 se hace uso de los comandos AT, estos se envían desde el microcontrolador, a través del puerto UART5. Hay que decir que todos los comandos AT deben estar en letras mayúsculas y terminar realizando un retorno de carro y una línea nueva (CRLF). Los comandos que se han utilizado en el proyecto son los que aparecen en la **Tabla 7**.

Comandos AT			
Comando	**Función**	**Parámetros**	**Respuesta**
AT	Test para comprobar que el ESP responde	-	OK
AT+RST	Reinicio	-	OK
AT+CWMODE=modo	Establece el modo del ESP	modo = 1, 2, 3 1: modo estación 2: modo punto de acceso 3: modo estación y punto de acceso	OK
AT+CWJAP="ssid","pass"	El ESP se conecta a una red wifi	ssid: nombre de la red wifi pass: contraseña de la red wifi	OK
AT+CIPMUX=modo	Se configura el número de conexiones posibles	modo = 0, 1 0: conexión única 1: múltiples conexiones	OK

AT+CIPSERVER=modo,puerto	Se establece el ESP en modo servidor	modo = 0, 1 0: eliminar servidor 1: crear servidor puerto: número del puerto donde está el servidor	OK
AT+CIPSEND=id,longitud	Se envían datos en la conexión con identificador "id"	id: número identificador de la conexión longitud: longitud de los datos a enviar en la conexión con número "id"	SEND OK
AT+CIPCLOSE=id	Se cierra la conexión con identificador "id"	id: número identificador de la conexión	OK

Tabla 7: Comandos AT utilizados en el proyecto.

Fuente: Elaboración propia a partir de la información obtenida en (Espressif Systems, 2021).

1.2.8 Programación

A la hora de realizar la programación del microcontrolador, podemos generar código en lenguaje C a partir de la configuración de los pines y relojes hecha en el programa STM32CubeMX, lo cual nos ahorra bastante tiempo y facilita la programación del dispositivo. El fichero de código se genera en el IDE Keil uVision5. En este programa podemos modificar el código, depurarlo y descargarlo a la memoria flash del dispositivo, entre otras cosas. Es importante indicar que tenemos que añadir una serie de librerías, las cuales aparecen en la **Figura 30**, para activar la pantalla LCD y poder usar la funcionalidad táctil de la misma.

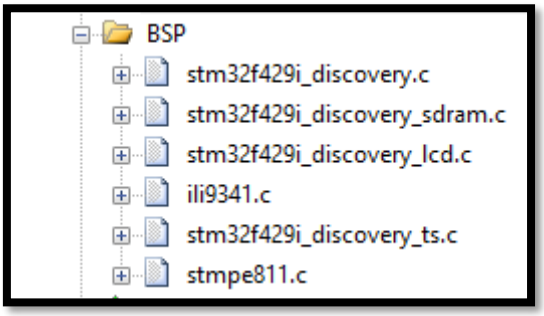

Figura 28: Librerías de la pantalla LCD y táctil.

Fuente: Elaboración propia.

En este apartado se va a hacer un recorrido general del código desarrollado mediante diagramas de flujo, los cuales son todos de elaboración propia. Además, nos enfocaremos en aclarar las partes más importantes del código. Para una explicación más completa de la programación realizada, y un mejor entendimiento de este apartado, se recomienda ver el código fuente del programa, presente en los anejos, con los comentarios realizados en él.

1.2.8.1 Diagrama general del programa

El **Diagrama 1** representa el diagrama de flujo general del programa realizado. Primero se inicializan los periféricos y los componentes (ESP y pantalla) que vamos a utilizar junto al microcontrolador STM32F429ZIT6. Además, se declaran las variables generales que se van a usar en la mayor parte del código y de las funciones.

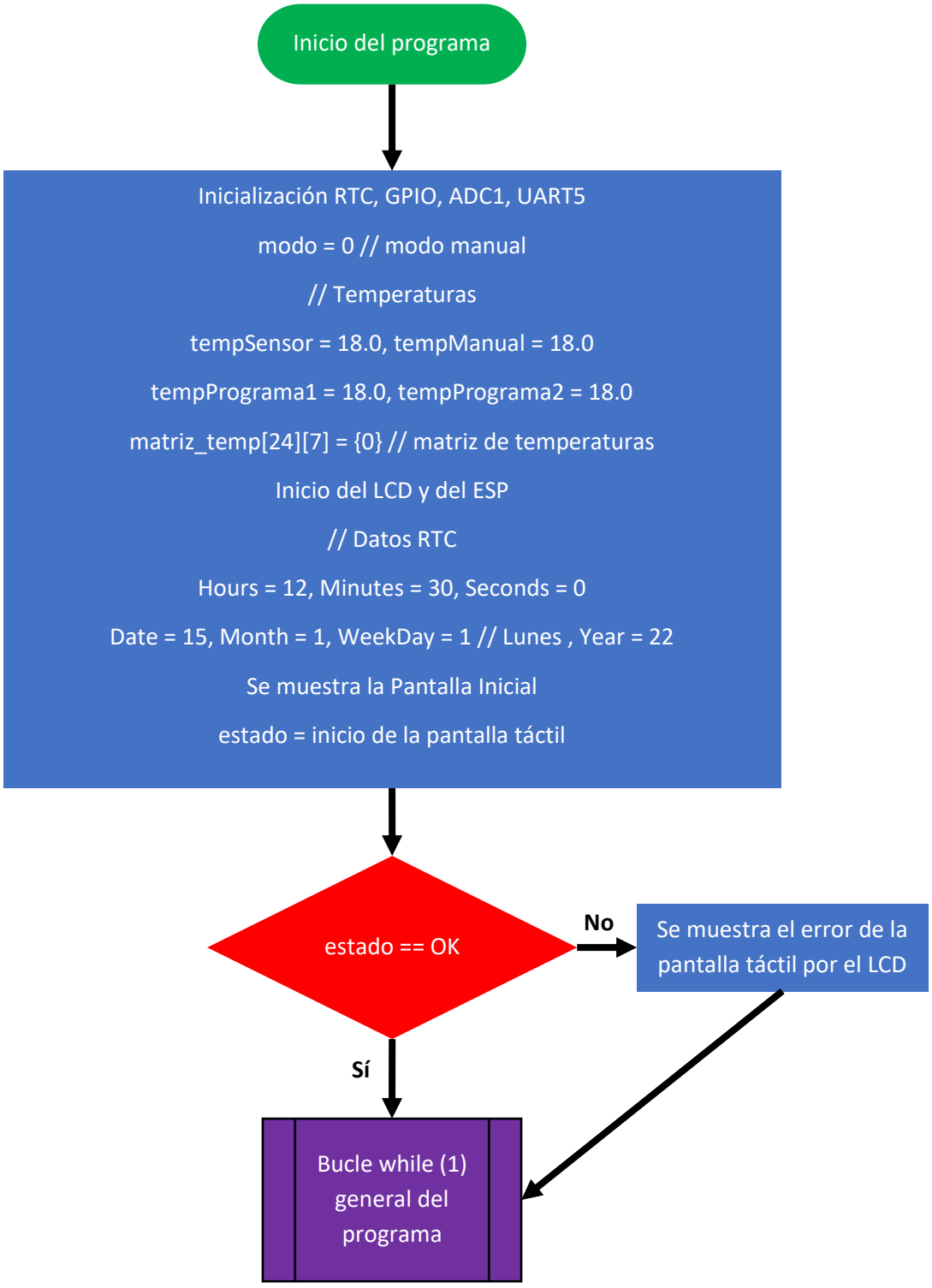

Diagrama 1: Diagrama de flujo general del programa.

- **Matriz de temperaturas del modo programa**

Como se ha indicado en apartados anteriores, el modo programa del termostato tiene dos temperaturas configurables, representadas por las variables tempPrograma1 y tempPrograma2. Dentro del modo programa, se puede elegir una de las dos temperaturas como la temperatura deseada para cada una de las 24 horas de los 7 días de la semana.

Para conseguir este propósito se ha utilizado la variable matriz_temp[24][7] = {0}. Se trata de una matriz booleana, inicializada a cero, que tiene 24 filas (enumeradas de 0 a 23), que representan las horas del día, y 7 columnas (enumeradas de 0 a 6), que representan los días de la semana. El valor booleano 0 indica que se ha elegido la temperatura de programa 1, mientras que el valor 1 indica que se ha elegido la temperatura de programa 2. Esta matriz nos permite modificar y comprobar, fácilmente, la temperatura asignada a cada hora de cualquier día de la semana.

A modo de ejemplo: el booleano en la posición matriz_temp[0][0] indica la temperatura elegida a las 12 de la noche del lunes. El booleano en la posición matriz_temp[23][6] indica la temperatura elegida a las 11 de la noche del domingo.

- **Configuración e inicio del ESP**

Para inicializar el ESP vamos a hacer uso del puerto UART5, al cual está conectado el ESP, y de parte de los comandos AT expuestos en la parte final del apartado **1.2.7 ESP8266 y comunicación UART**. Al utilizar estos comandos vamos a seguir la estructura que se puede ver en la **Figura 31**. La estructura es la siguiente:

1. Limpiamos la pantalla LCD y mostramos el paso en el que estamos, en este caso el primero, usando funciones de la librería BSP_LCD. Limpiamos la pantalla de nuevo y preparamos las variables de envío y recepción.

2. Usando las funciones HAL_UART, transmitimos el comando AT deseado. Finalmente, recibimos la respuesta del ESP y la mostramos por el LCD.

```
// AT de test
BSP_LCD_Clear(LCD_COLOR_WHITE);
BSP_LCD_DisplayStringAtLine(3, "PASO 1: AT");
HAL_Delay(1000);

BSP_LCD_Clear(LCD_COLOR_WHITE);
uint8_t mensajeAT[] = "AT\r\n";
uint8_t respuesta[10];
HAL_UART_Transmit(&huart5, mensajeAT, sizeof(mensajeAT), 100);
HAL_UART_Receive(&huart5, respuesta, sizeof(respuesta), 100);
BSP_LCD_DisplayStringAtLine(3, respuesta);
HAL_Delay(2000);
```

Figura 29: Código de programación de uno de los pasos seguidos al iniciar el ESP.

Fuente: Elaboración propia.

Las funciones HAL_UART de transmisión y recepción usan el mismo formato con los mismos argumentos, estos son: (&manejador del puerto UART, variable, tamaño, *tiemout*).

La configuración realizada sobre el ESP ha sido la siguiente:

- AT+CWMODE=1: se establece el dispositivo ESP como una estación dentro de una red.
- AT+CWJAP=ssid,pass: se conecta el ESP a la red wifi deseada, se necesita el nombre y la contraseña de la misma.
- AT+CIPMUX=1: se habilitan las conexiones múltiples, lo cual nos permite atender varias peticiones y poder crear el servidor en el paso siguiente.
- AT+CIPSERVER=1,puerto: se crea un servidor en el puerto indicado.

- **Bucles while (1)**

A lo largo del código, y de los diferentes diagramas de flujo, se pueden ver varios bucles while (1), en el **Diagrama 1** podemos ver el primero. Estos bucles se utilizan para que el microcontrolador se encuentre a la espera de la llegada de eventos. Estos eventos pueden ser: peticiones desde la aplicación móvil, activaciones temporales de procesos de control o toques realizados sobre la pantalla táctil.

- **Pantallas gráficas desarrolladas**

Para poder realizar la configuración táctil del termostato se han realizado varias pantallas gráficas que se muestran en el LCD. Para la confección de estas pantallas se han usado las funciones de la librería BSP_LCD, las cuales permiten elegir colores de representación, mostrar un texto de distintas formas o dibujar círculos y líneas. Las **Figuras 33 y 34** muestra las distintas pantallas que existen en el termostato. Además, a la hora de usar las funciones BSP_LCD, hay que tener en cuenta las dimensiones de la pantalla (240 x 320 pixeles) y el origen de coordenadas de esta, ambas se muestran en la **Figura 32**.

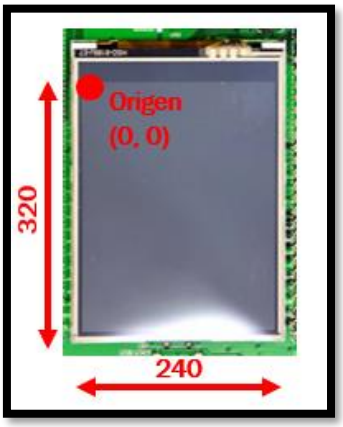

Figura 30: Medidas pantalla LCD.

Fuente: Elaboración propia.

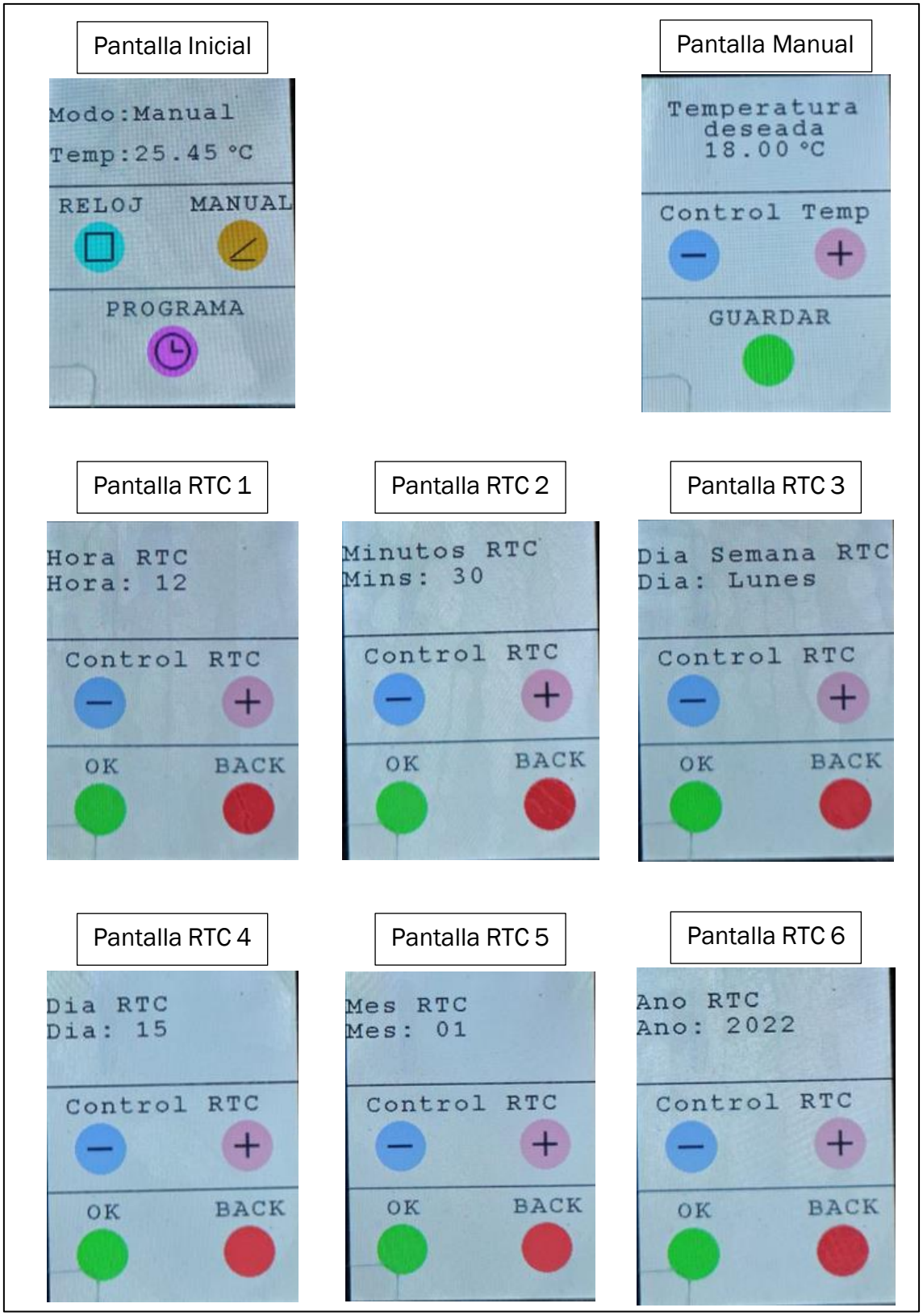

Figura 31: Pantallas Inicial, Manual y RTC del termostato.

Fuente: Elaboración propia.

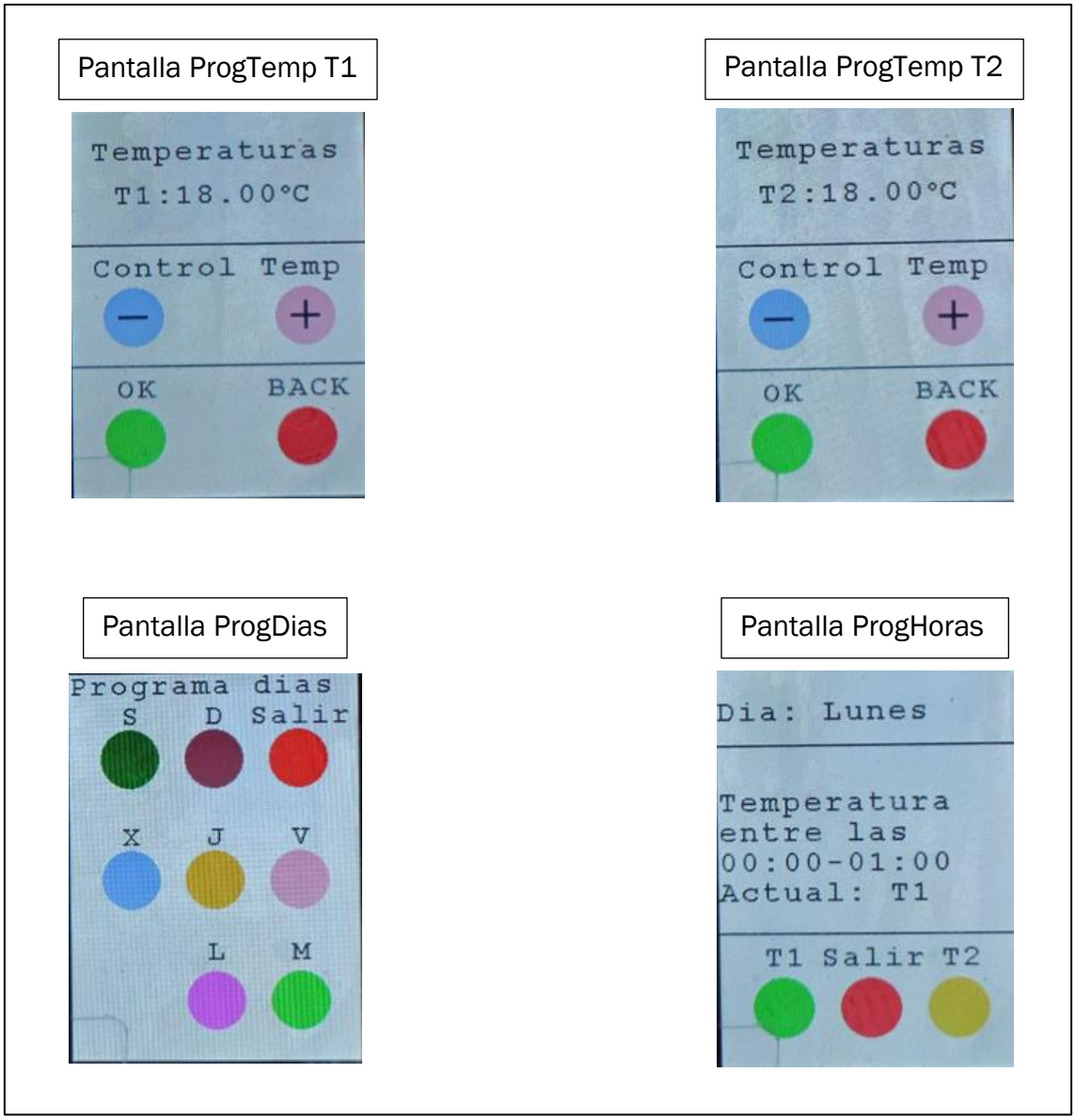

Figura 32: Pantallas ProgTemp, ProgDias y ProgHoras del modo programa del termostato.

Fuente: Elaboración propia.

1.2.8.2 Diagrama bucle while (1) general

Los **Diagramas 2 y 3** representan el diagrama de flujo del bucle while (1) que se había visto en el **Diagrama 1**. En este bucle el dispositivo se encuentra a la espera de la llegada de algún evento. Una vez se produce el evento, el sistema ejecuta el proceso adecuado para atenderlo.

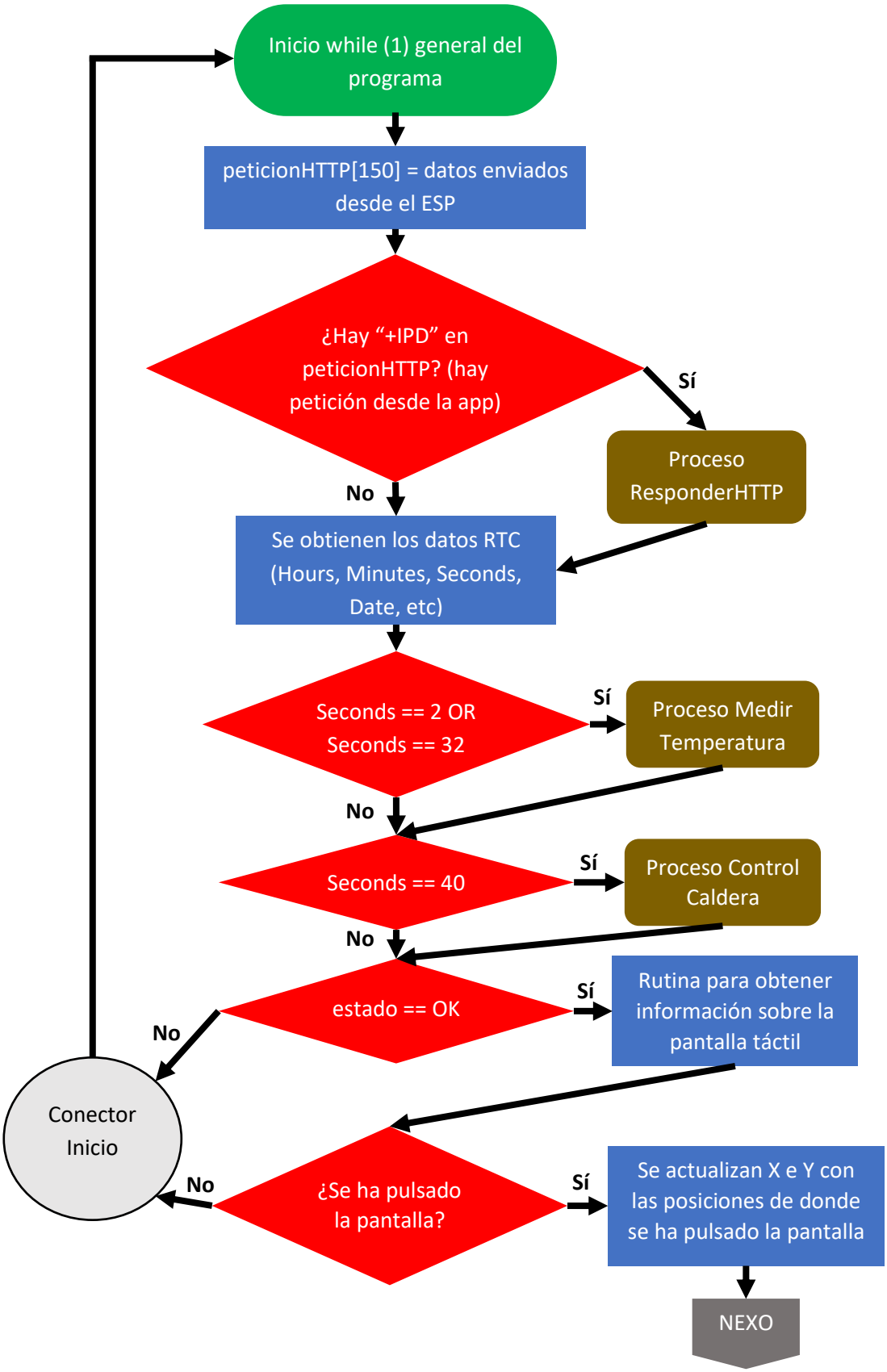

Diagrama 2: Diagrama de flujo del bucle while (1) general.

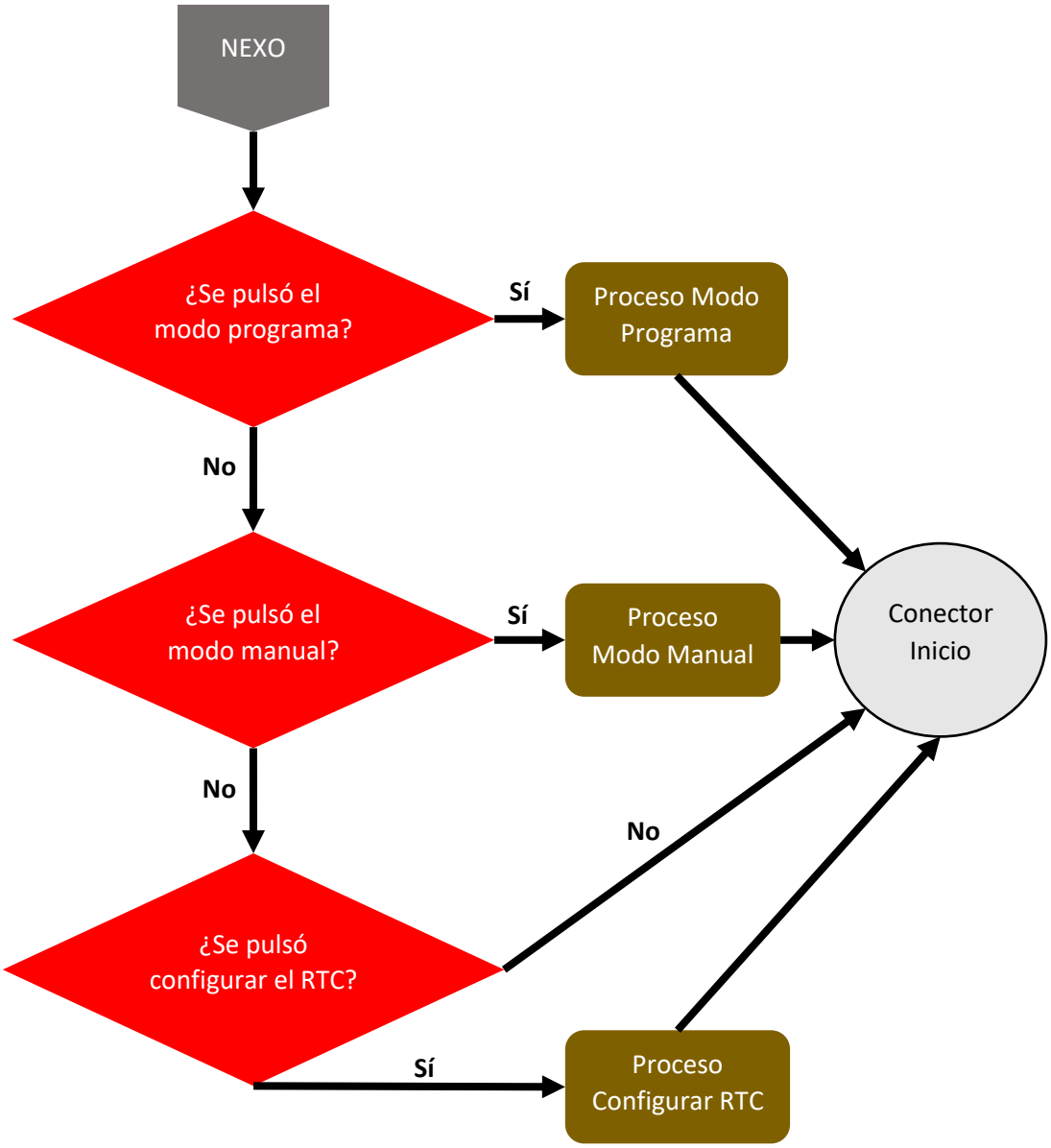

Diagrama 3: Diagrama de flujo del bucle while (1) general.

- **Pulsación sobre la pantalla táctil**

Para localizar una pulsación sobre la pantalla táctil hay que trabajar con las funciones de la librería BSP_TS, en concreto vamos a utilizar GetState y TouchDetected. La estructura seguida para detectar una pulsación sobre la pantalla táctil se muestra en la **Figura 35**. La estructura seguida es la siguiente:

1. Dentro de un while (1) se usa BSP_TS_GetState(&TS_State) para obtener el estado e información de la pantalla táctil. Siendo TS_State la variable de la pantalla táctil donde se guarda su estado.
2. Se detecta si ha habido un evento por pulsación sobre la pantalla usando if (TS_State.TouchDetected). Si la pulsación ha existido, se guardan sus coordenadas X e Y.
3. Finalmente, se discierne dónde se ha pulsado usando las variables que guardan las coordenadas.

```
BSP_TS_GetState(&TS_State); // Rutina para obtener la informacion o el estado de la pantalla tactil

if (TS_State.TouchDetected) // Se detecta una pulsacion sobre la pantalla
{
  // Se guardan las coordenadas al pulsar sobre la pantalla
  x = TS_State.X;
  y = TS_State.Y;
```

Figura 33: Código de programación utilizado para localizar una pulsación sobre la pantalla.

Fuente: Elaboración propia.

- **Petición desde la aplicación móvil**

Para atender el evento de la llegada de una petición desde la aplicación móvil, hay que filtrar la información que llega desde el dispositivo ESP al puerto UART5.

Las peticiones realizadas por la aplicación móvil van a generar un envío de información por parte del ESP al microcontrolador. Esta información va a quedar identificada por el uso de la cadena de caracteres "+IPD". Como se indica en el **Diagrama 2**, los datos que llegan desde el ESP se almacenan en la variable peticionHTTP[150] y luego se busca en ella la cadena "+IPD".

Para recibir los datos y guardarlos usamos la función HAL_UART_Receive_IT, la cual funciona mediante interrupción y no tiene un *timeout*. El formato de esta es: HAL_UART_Receive_IT(&manejador del puerto UART, variable donde se guardan los datos, tamaño de los datos).

Se usa la función strstr() para buscar si la cadena "+IPD" está entre los datos enviados desde el ESP. Esta función tiene el siguiente formato: strstr(cadena donde se busca, cadena a buscar). La función retorna un puntero con la posición donde se encuentra el comienzo de la cadena, si se encuentra. En caso de no encontrar la cadena, retorna un NULL.

1.2.8.3 Diagrama Proceso ResponderHTTP

Los **Diagramas 4 y 5** representan el diagrama de flujo del Proceso ResponderHTTP, proceso que se había visto en el **Diagrama 2**. Este proceso se basa en concretar cuál es la petición que llega desde la aplicación, realizar las acciones pertinentes asociadas a la petición, y, si procede, enviar información a la aplicación móvil. Para precisar qué información aparece en los datos guardados se va a seguir el método visto antes usando la función strstr().

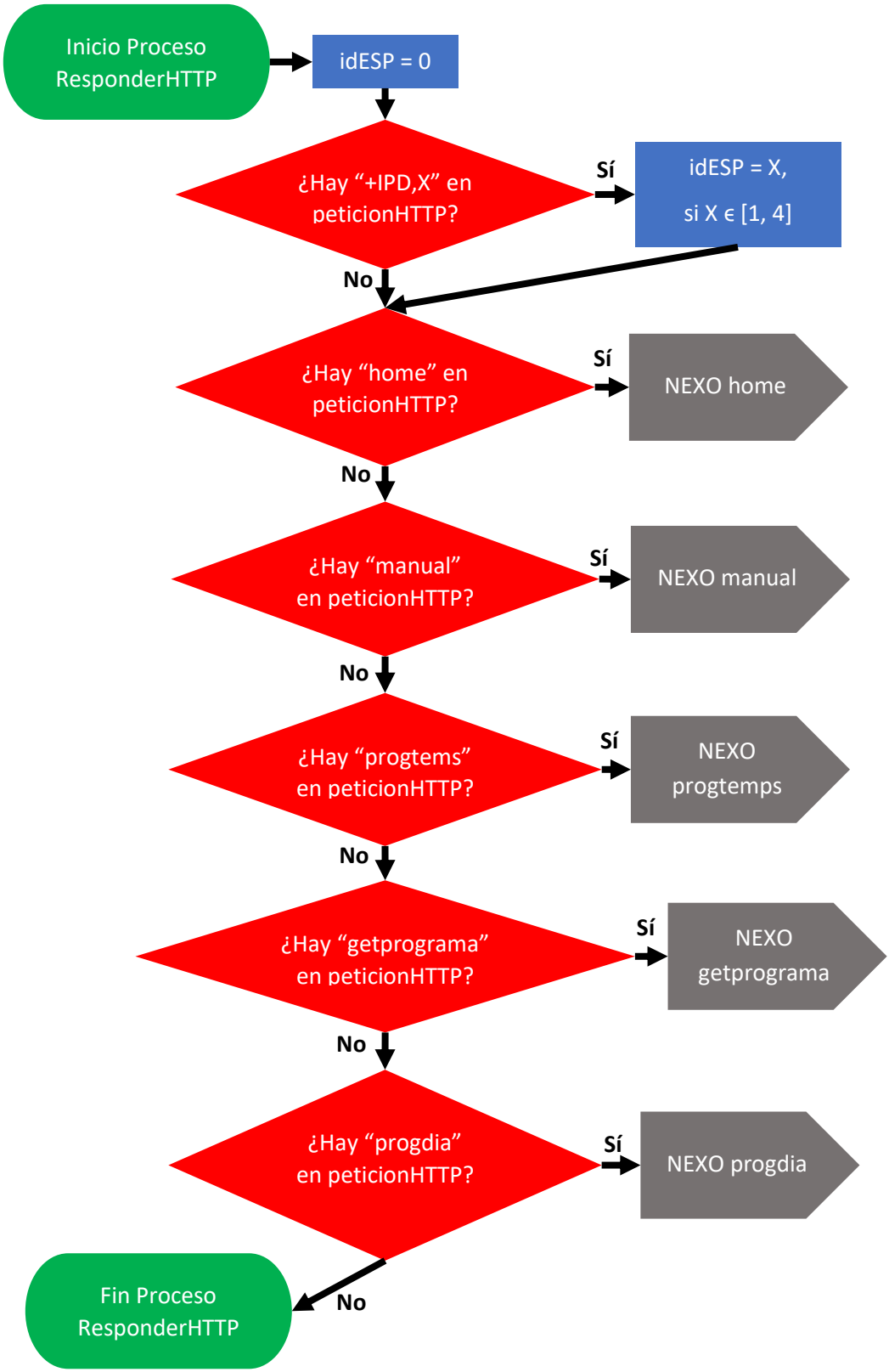

Diagrama 4: Diagrama de flujo del Proceso ResponderHTTP.

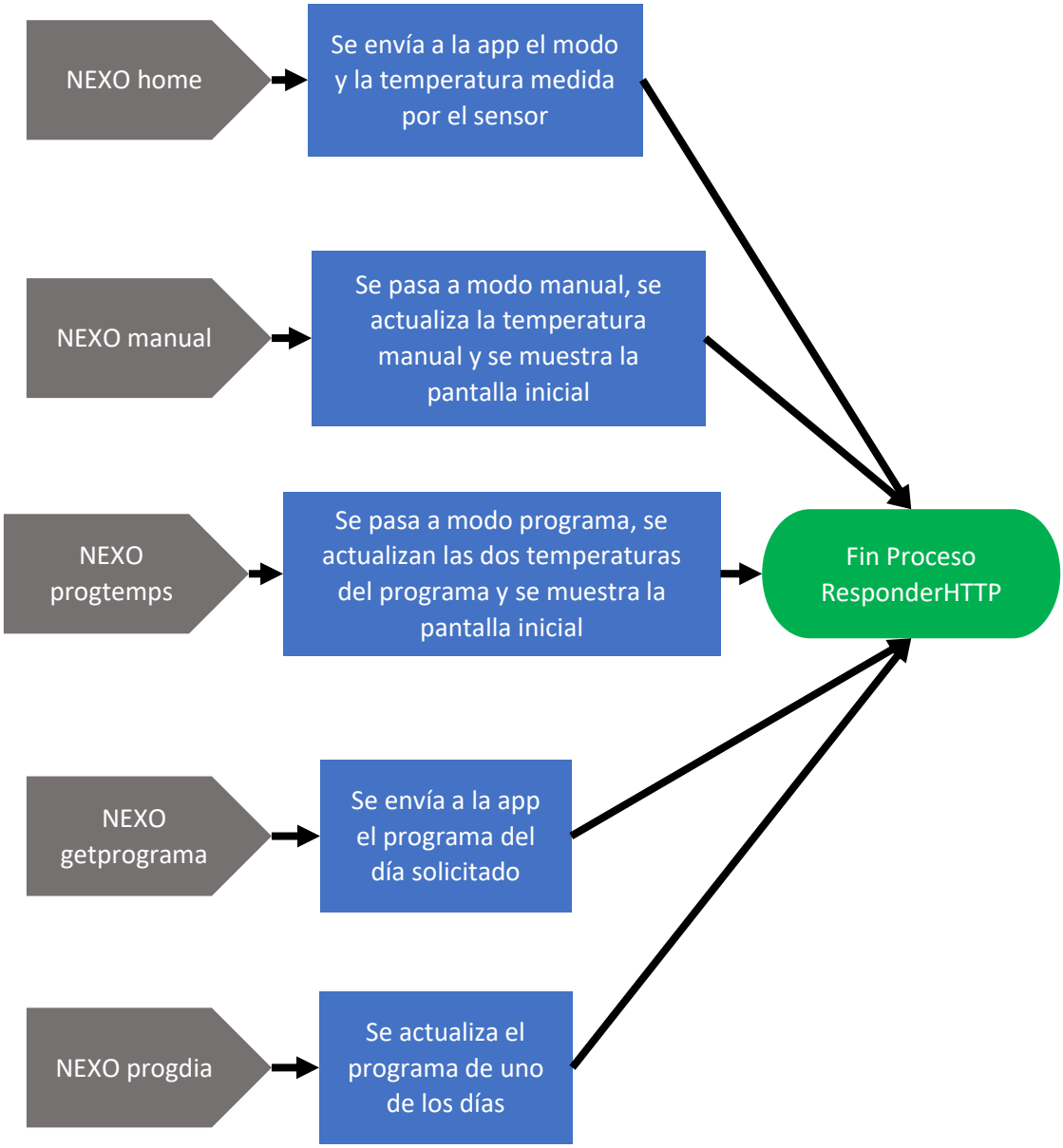

Diagrama 5: Diagrama de flujo del Proceso ResponderHTTP.

- ID de la conexión

Al habilitar las conexiones múltiples del ESP, podemos tener hasta 5 conexiones. Las conexiones se diferencian con un número de identificación que puede ser 0, 1, 2, 3 o 4. Este número es importante, ya que, si queremos realizar una transmisión de información desde el ESP a la aplicación móvil, necesitamos indicar el ID de la conexión. Además, sabemos que el ID de la conexión aparece después de la cadena de texto "+IPD,".

Usamos la función strstr() para buscar entre los datos recibidos si hay una cadena de texto del tipo de "+IPD,X", siendo X un número entre 0 y 4. Con esto podemos ajustar la variable idESP, que se ha inicializado a 0, al ID de la conexión correcto.

- Petición home

En esta petición hay que transmitir información a la aplicación móvil, hay que enviar el modo y la temperatura actuales de la vivienda. Para una correcta comunicación, hay que indicar que la respuesta a la petición pertenece al protocolo HTTP. Para hacer esto, creamos una cadena de texto como la que se muestra en la Figura 36, con esta cadena indicamos que la respuesta es de tipo HTTP.

```
uint8_t httpHome[60] = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
```

Figura 34: Cadena de texto que indica que se usa HTTP.

Fuente: Elaboración propia.

Después de realizar esto, usamos la función strcat(cadena, cadena a concatenar) para concatenar, al final de httpHome[60], las cadenas de texto que indican el modo y la temperatura actuales,.

Finalmente, transmitimos la cadena completa usando los comandos AT+CIPSEND y AT+CIPCLOSE. El comando AT+CIPSEND se usa antes de transmitir la información, sirve para indicar al ESP que se va a producir una transmisión en una conexión, hay que indicar el ID de la misma y el tamaño que tiene la información que vamos a enviar. El comando AT+CIPCLOSE se usa después de finalizar la transmisión de la información, sirve para indicar al ESP que se va a cerra una conexión, tenemos que indicar el ID de la misma.

- Petición manual

En esta petición hay que pasar al modo manual, recoger la temperatura que se ha mandado desde la aplicación y actualizar la variable de la temperatura manual.

Para recoger la temperatura usamos la función strncpy(puntero al array donde se copia la cadena, puntero al string a copiar, número de caracteres a copiar). Finalmente, pasamos la temperatura de char a float y actualizamos el valor deseado de la temperatura manual.

- Petición progtemps

En esta petición hay que pasar a modo programa, recoger las dos temperaturas que se han mandado desde la aplicación y actualizar las variables de temperatura programa.

Básicamente, se sigue el mismo proceso que en la petición manual, pero teniendo en cuenta que, en este caso, hay dos temperaturas a guardar.

- Petición getprograma

En esta petición hay que recoger el día que se ha mandado desde la aplicación y transmitir información a la aplicación móvil, se tiene que enviar el programa de temperaturas del día solicitado.

Para enviar el programa del día hay que pasar a char la columna de la matriz de temperaturas correspondiente a ese día, ya que los datos de la matriz son booleanos. Para realizar esto se recorre, mediante un bucle for, la columna de la matriz de temperaturas indicada y un array de chars. En cada paso de este proceso se va rellenando el array con los datos. Una vez realizada la conversión, se concatena el programa a una cadena de texto, que indica que se trata de una respuesta HTTP, y se realiza la transmisión de la información del mismo modo que se vio en la Petición home.

- Petición progdia

En esta petición hay que recoger el día y el programa de temperaturas de este, que se han mandado desde la aplicación, y actualizar el programa de dicho día.

En función del día recibido, se recorre, mediante un bucle for, el programa de temperaturas enviado desde la aplicación y la columna de la matriz de temperaturas indicada. En cada paso de este proceso se va actualizando la columna de la matriz.

1.2.8.4 Diagrama Proceso Medir Temperatura

El **Diagrama 6** representa el diagrama de flujo del Proceso Medir Temperatura, el cual se había visto en el **Diagrama 2**. Este proceso se basa en medir la temperatura actual en la vivienda, para ello aplicamos las fórmulas **[5]** y **[7]** del apartado **1.2.4 Sensor de temperatura y convertidor A/D**. Además, para mejorar la precisión de la medición se hacen 50 mediciones por segundo y, luego, se calcula la media. Para realizar esto usamos un bucle *for()* de 50 iteraciones e introducimos un *delay* de 20 milisegundos en cada una de ellas. En esas iteraciones se activa la conversión A/D con las funciones HAL_ADC y se realiza un cálculo de temperatura.

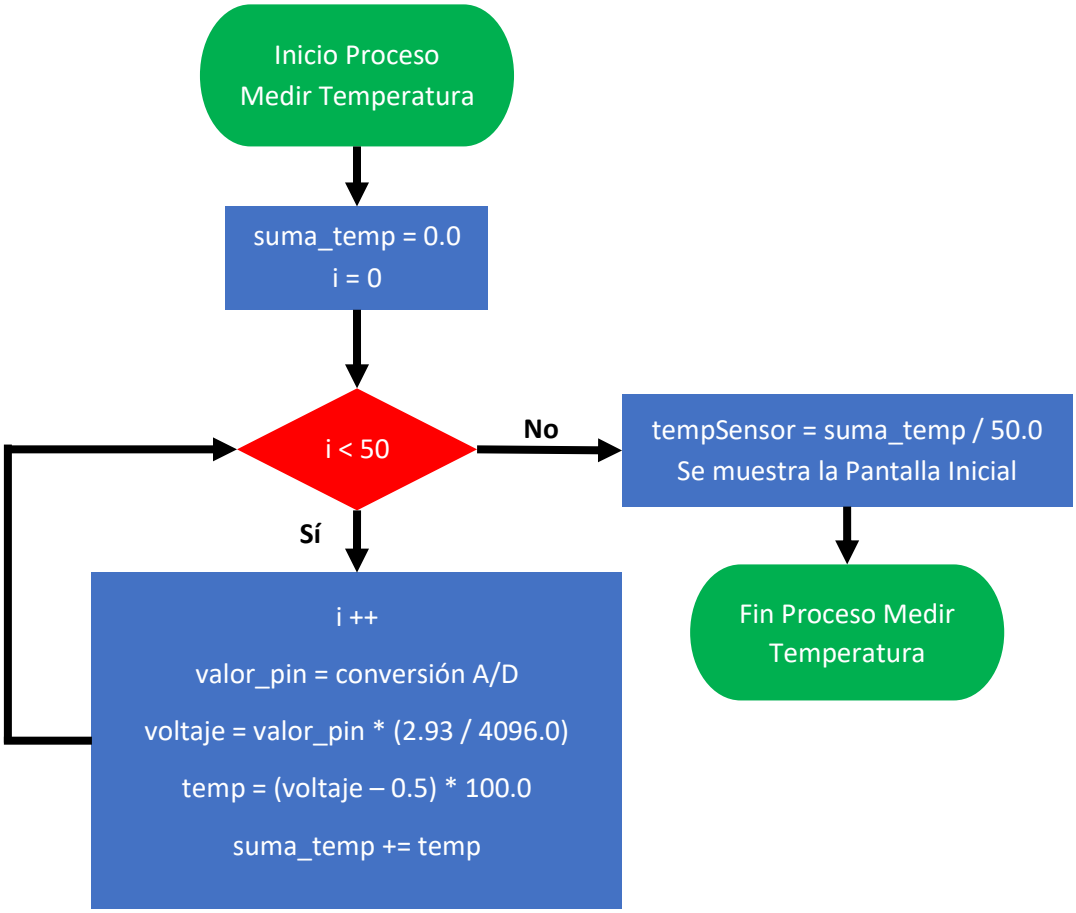

Diagrama 6: Diagrama de flujo del Proceso Medir Temperatura.

1.2.8.5 Diagrama Proceso Control Caldera

Los **Diagramas 7 y 8** representan el diagrama de flujo del Proceso Control Caldera, el cual se había visto en el **Diagrama 2**. Este proceso controla la activación de la caldera a través del relé. Como se había explicado en apartados anteriores, el LED LD4 va a actuar como si fuera el relé. El proceso de control de la caldera tiene una histéresis de medio grado sobre la temperatura manual, en caso de estar en modo manual, o sobre la temperatura programada en la hora a la que se hace el control, en caso de estar en modo programa. Con el uso de una histéresis se busca evitar el encendido y apagado continuo de la caldera.

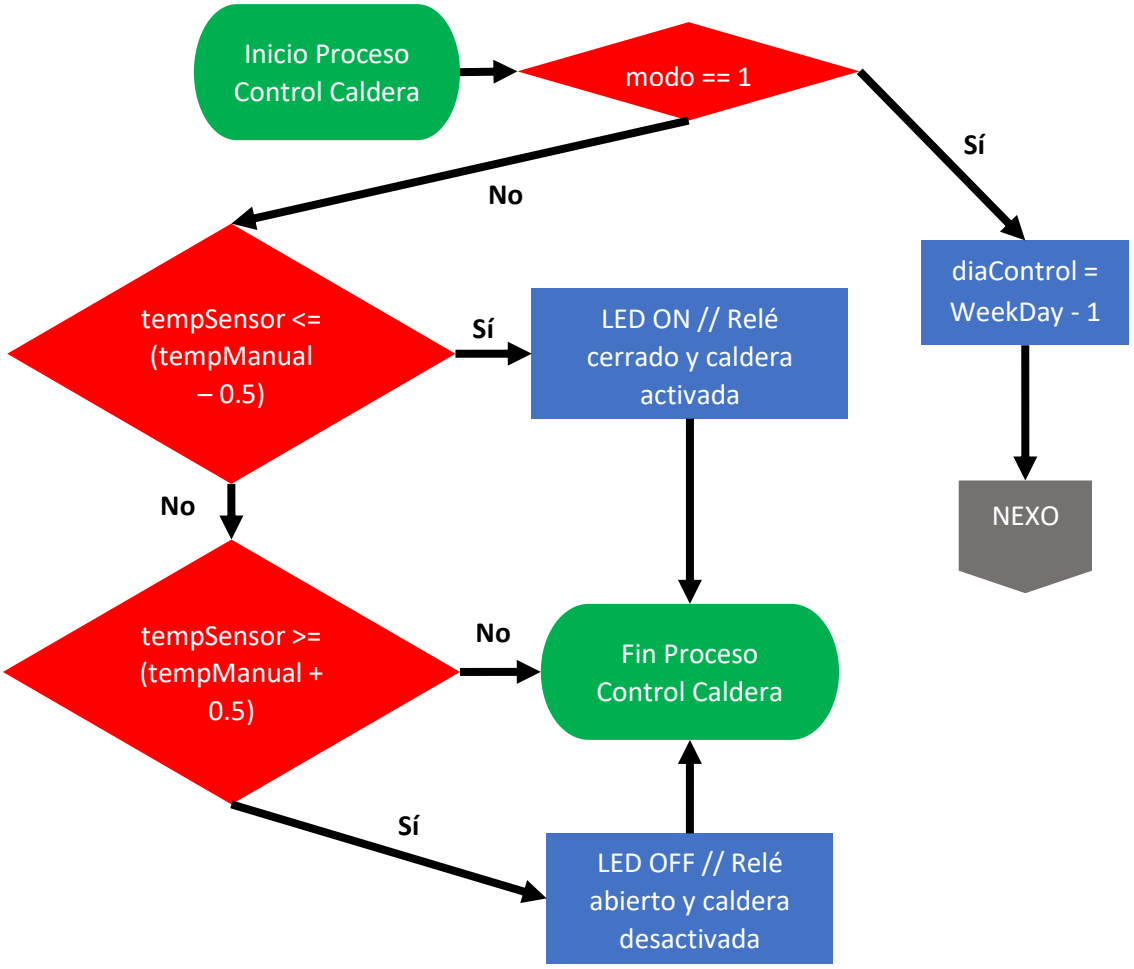

Diagrama 7: Diagrama de flujo del Proceso Control Caldera.

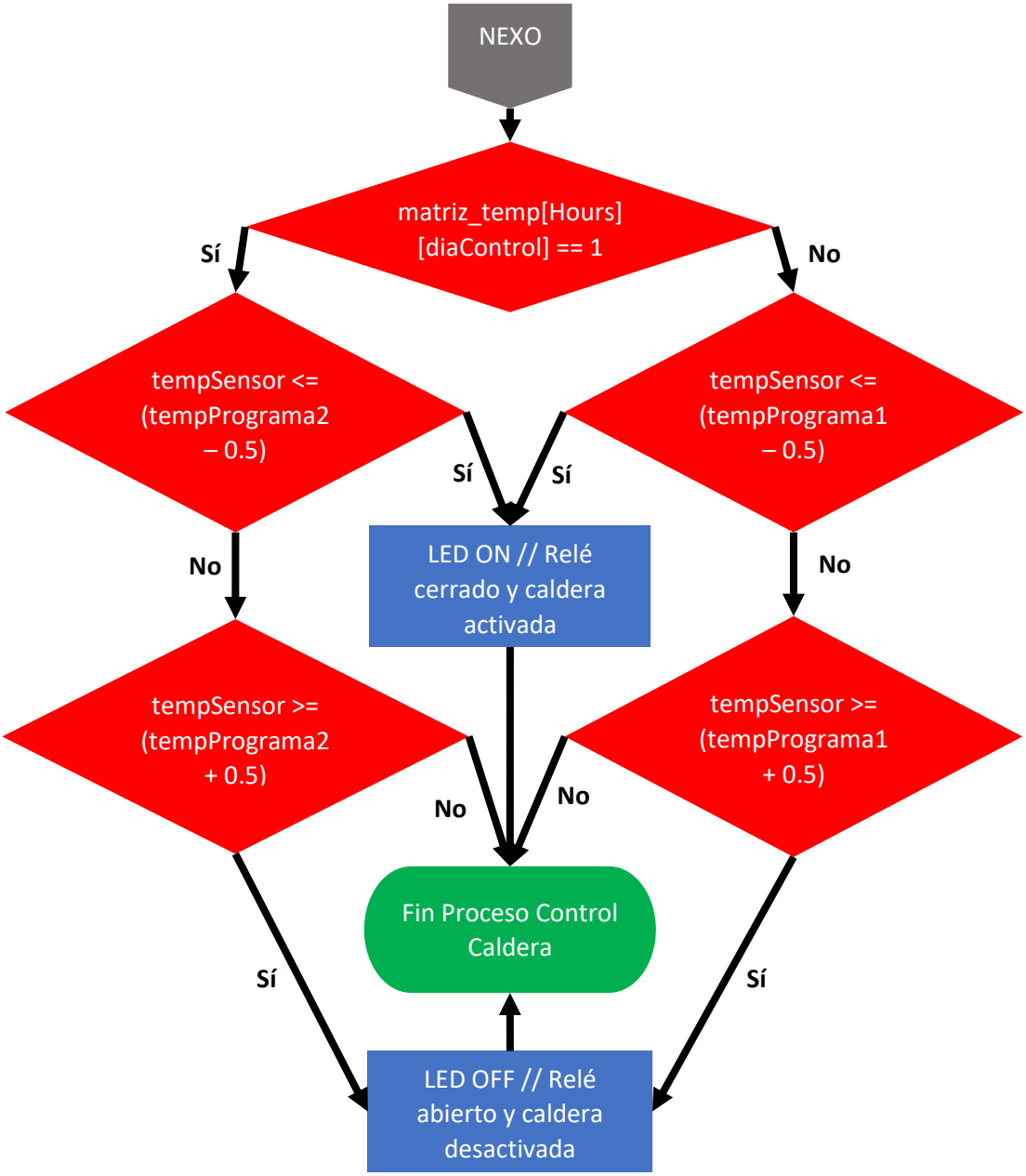

Diagrama 8: Diagrama de flujo del Proceso Control Caldera.

1.2.8.6 Diagrama Proceso Modo Programa

Los **Diagramas 9, 10, 11, 12, 13, 14, 15 y 16** representan el diagrama de flujo del Proceso Modo Programa, el cual se había visto en el **Diagrama 2**. Este proceso permite que el termostato pase a modo programa. También permite configurar las dos temperaturas del modo y los programas de los días de la semana. Todo este proceso se hace usando la pantalla táctil del termostato con las pantallas gráficas desarrolladas. Además, hay que destacar la importancia de la variable paso_programar, la cual nos permite movernos entre las pantallas y realizar la correcta configuración de los parámetros del modo.

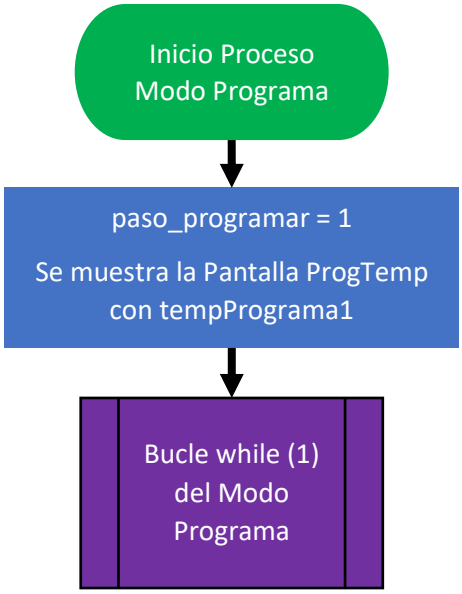

Diagrama 9: Diagrama de flujo del Proceso Modo Programa.

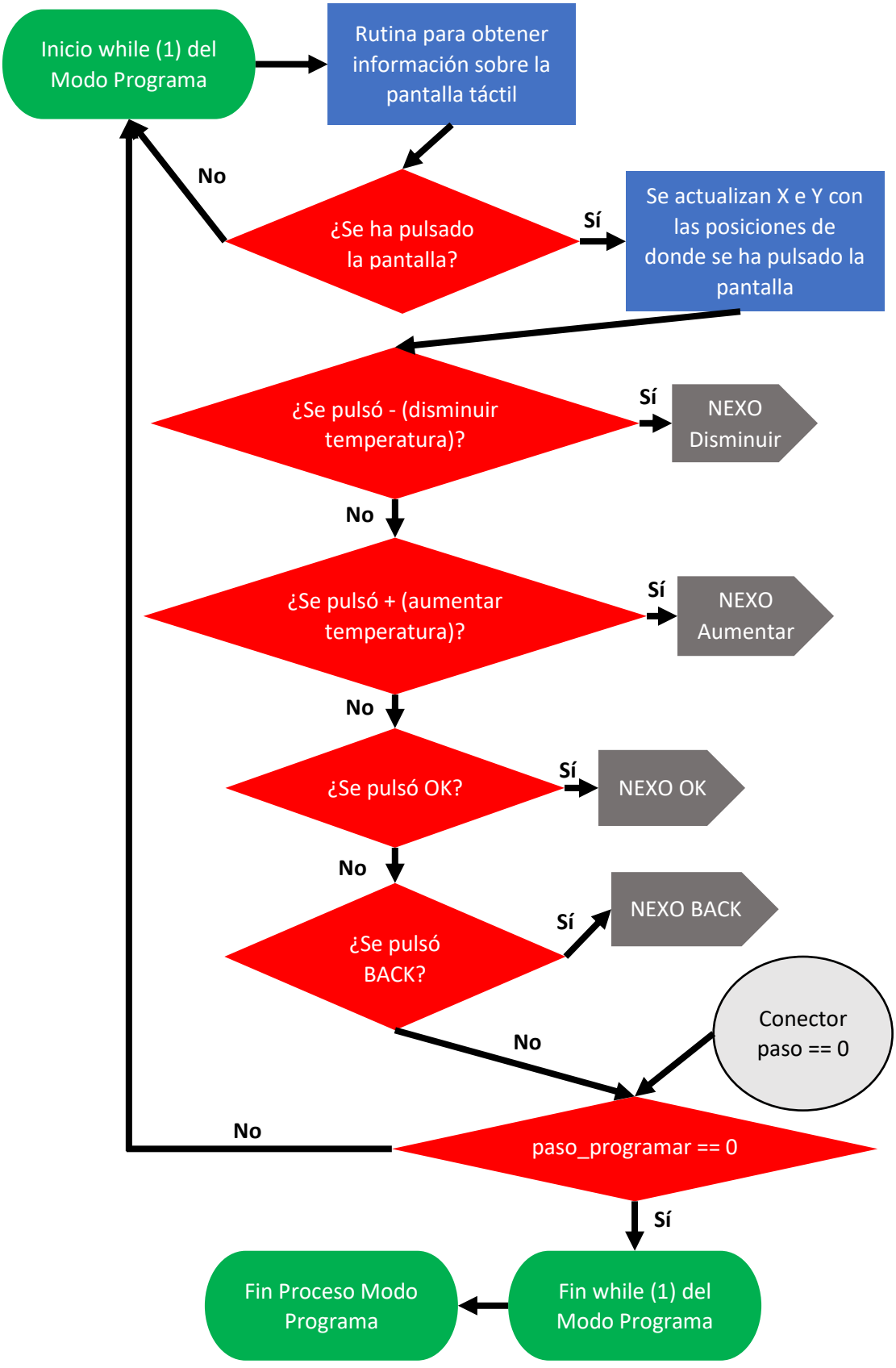

Diagrama 10: Diagrama de flujo del Proceso Modo Programa.

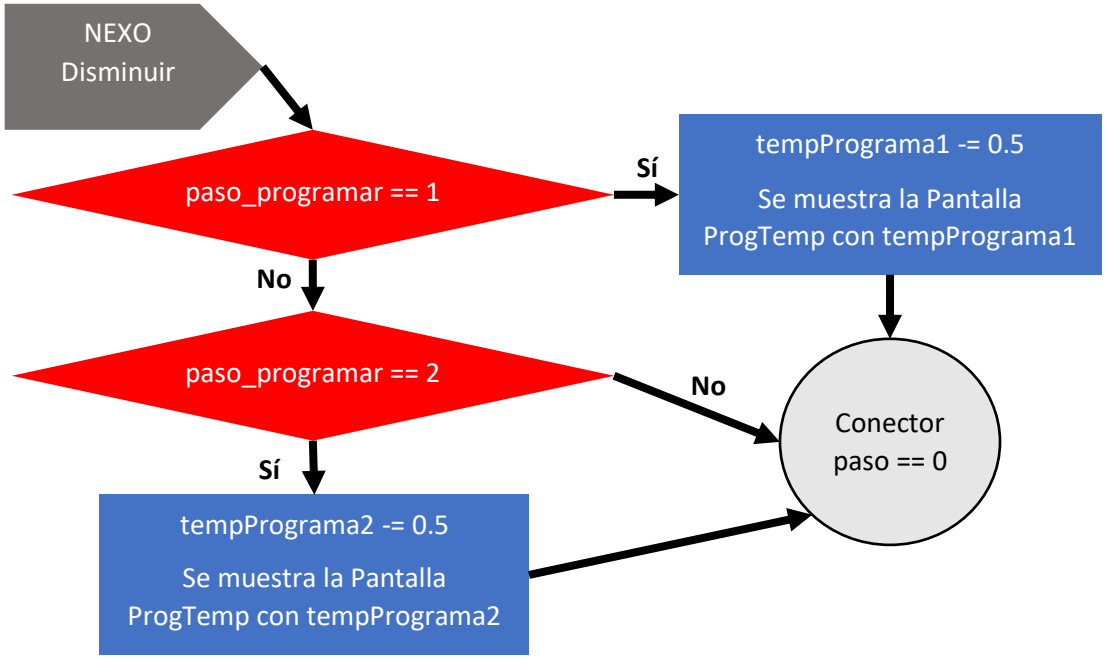

Diagrama 11: Diagrama de flujo del Proceso Modo Programa.

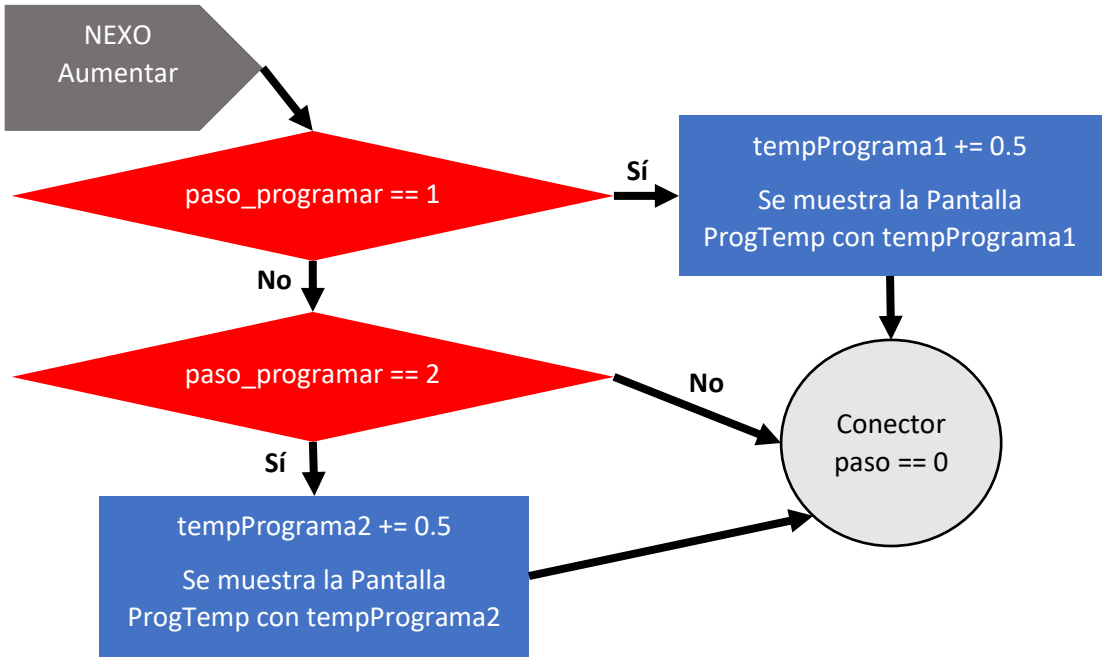

Diagrama 12: Diagrama de flujo del Proceso Modo Programa.

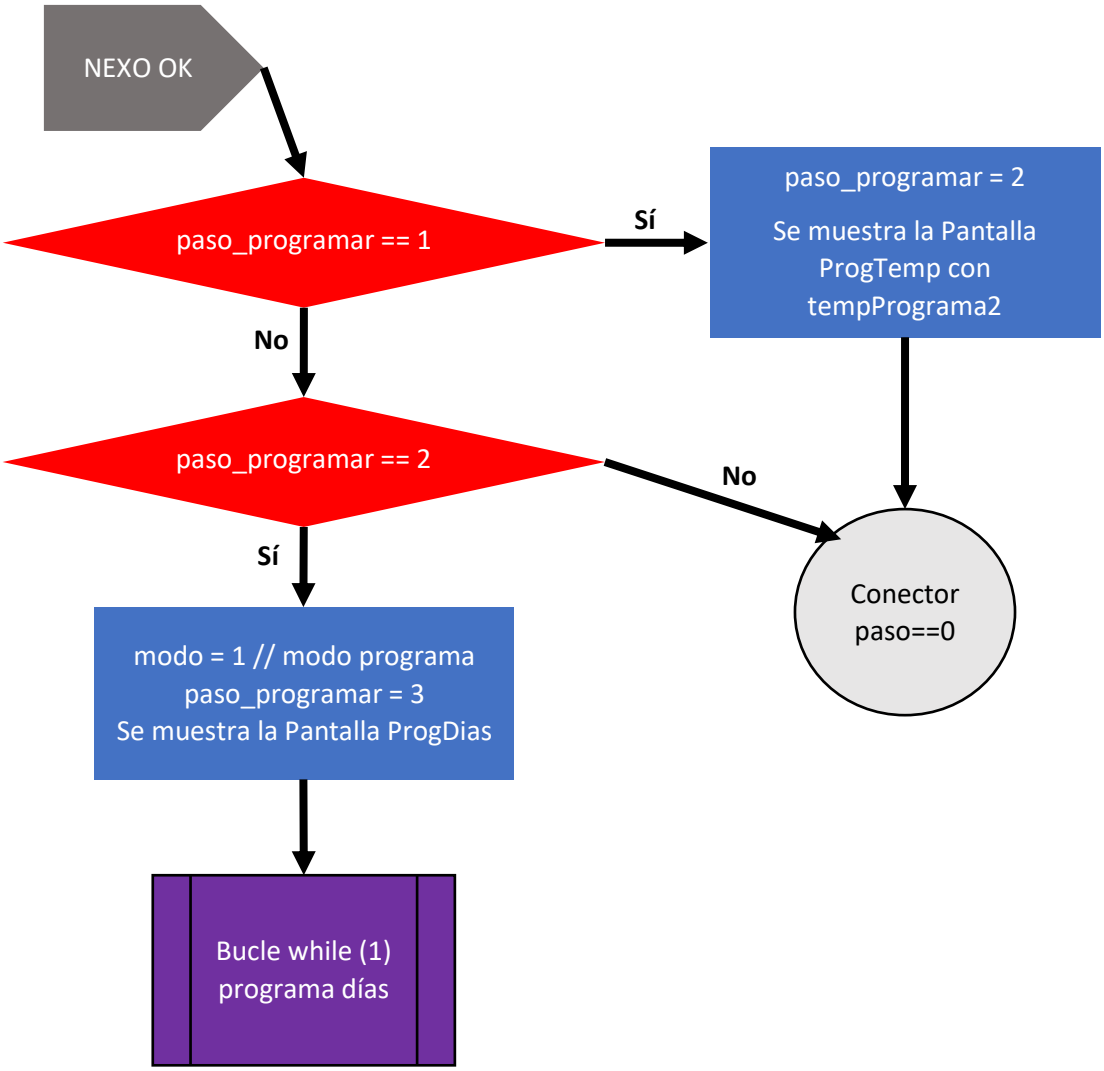

Diagrama 13: Diagrama de flujo del Proceso Modo Programa.

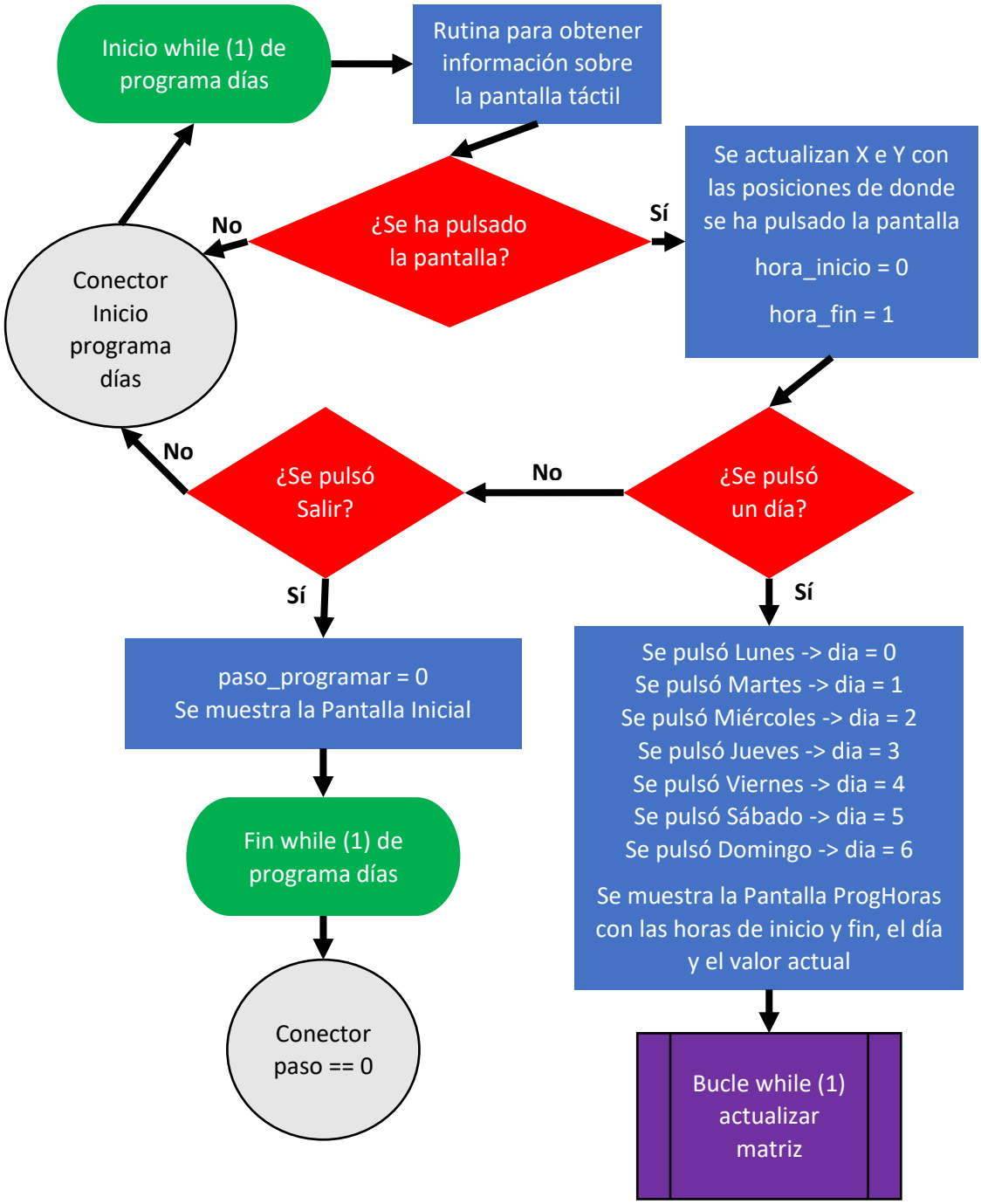

Diagrama 14: Diagrama de flujo del Proceso Modo Programa.

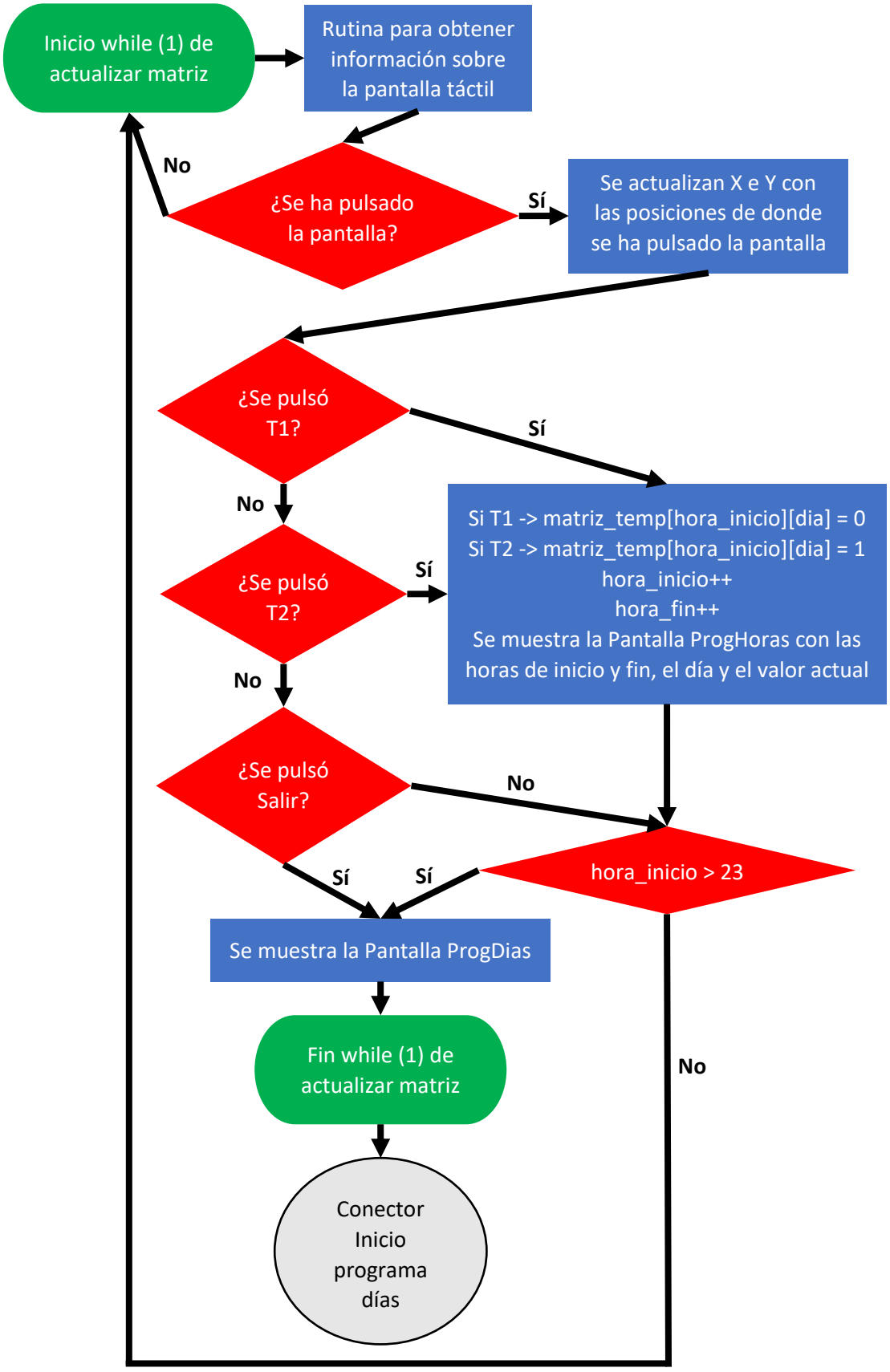

Diagrama 15: Diagrama de flujo del Proceso Modo Programa.

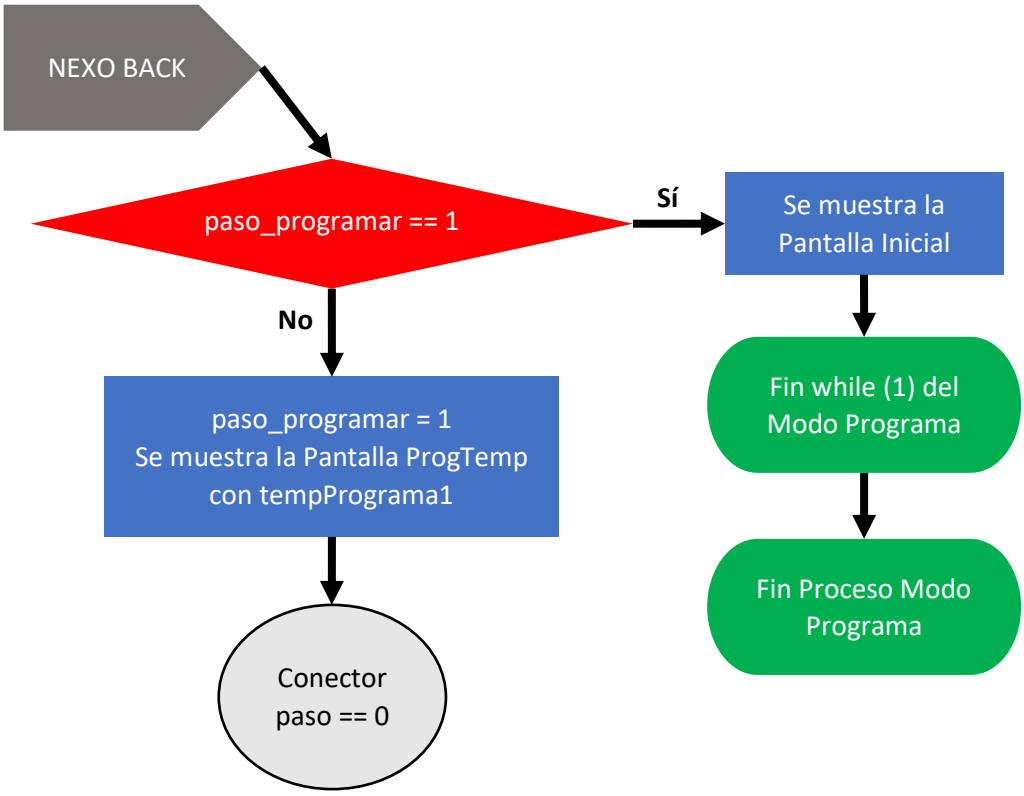

Diagrama 16: Diagrama de flujo del Proceso Modo Programa.

1.2.8.7 Diagrama Proceso Modo Manual

Los **Diagramas 17 y 18** representan el diagrama de flujo del Proceso Modo Manual, el cual se había visto en el **Diagrama 2**. Este proceso permite que el termostato pase a modo manual, configurando la temperatura deseada. Todo este proceso se hace usando la pantalla táctil del termostato con las pantallas gráficas desarrolladas.

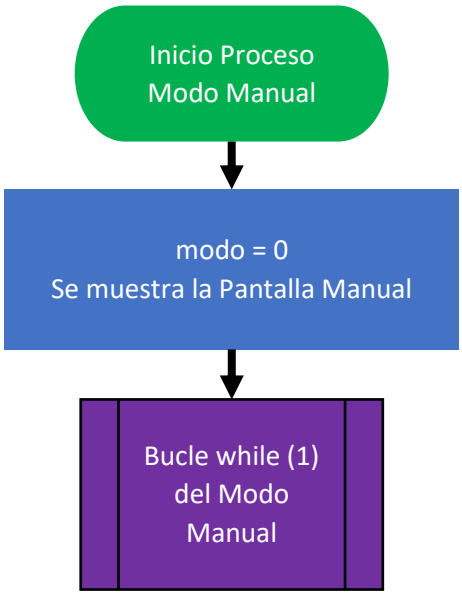

Diagrama 17: Diagrama de flujo del Proceso Modo Manual.

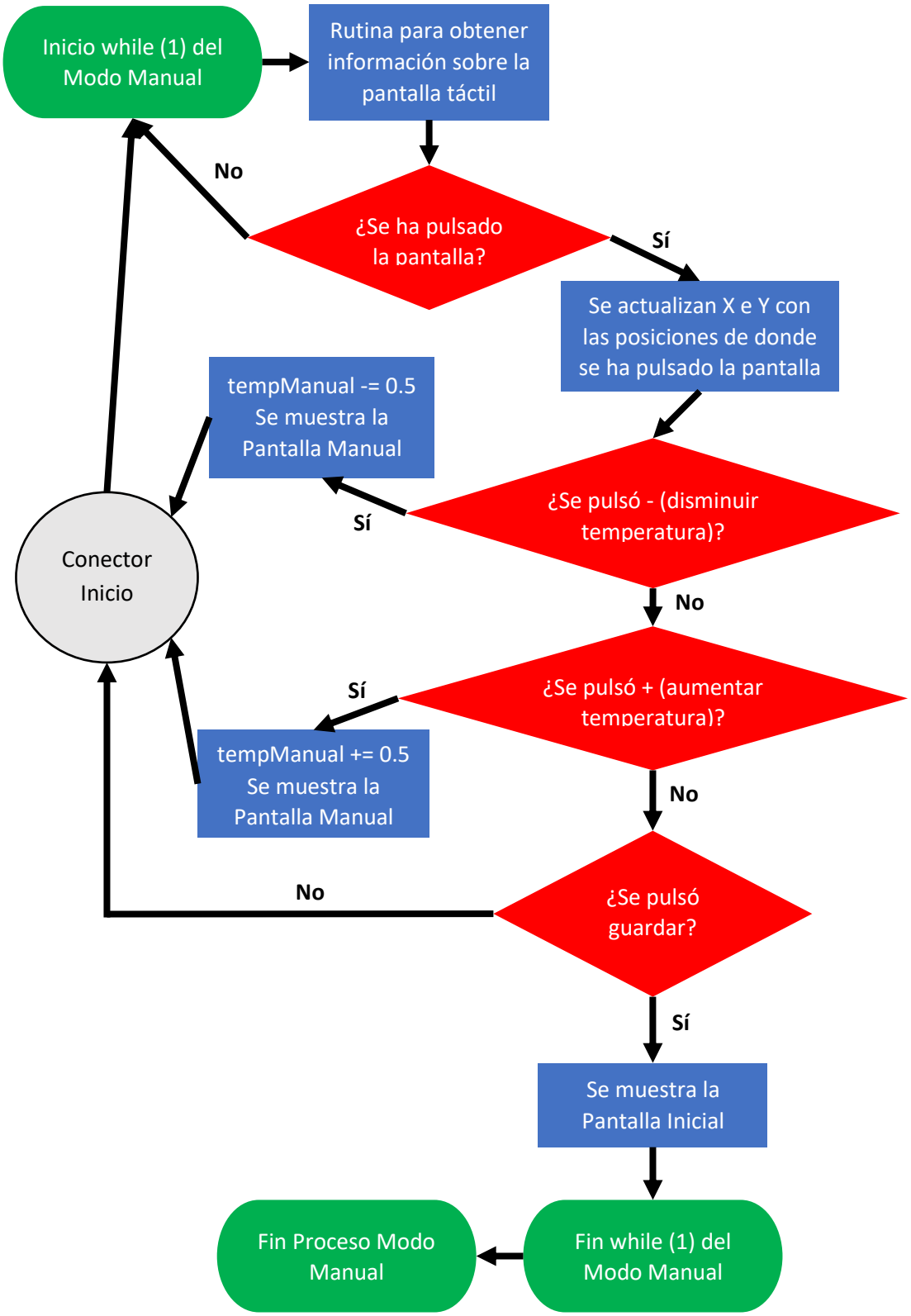

Diagrama 18: Diagrama de flujo del Proceso Modo Manual.

1.2.8.8 Diagrama Proceso Configurar RTC

Los **Diagramas 19, 20, 21, 22, 23 y 24** representan el diagrama de flujo del Proceso Configurar RTC, el cual se había visto en el **Diagrama 2**. Este proceso permite configurar la fecha y la hora actuales del termostato. Todo este proceso se hace usando la pantalla táctil del termostato con las pantallas gráficas desarrolladas. Además, hay que destacar la importancia de la variable pasoRTC, la cual nos permite movernos entre las pantallas y realizar la correcta configuración de los parámetros del reloj de tiempo real.

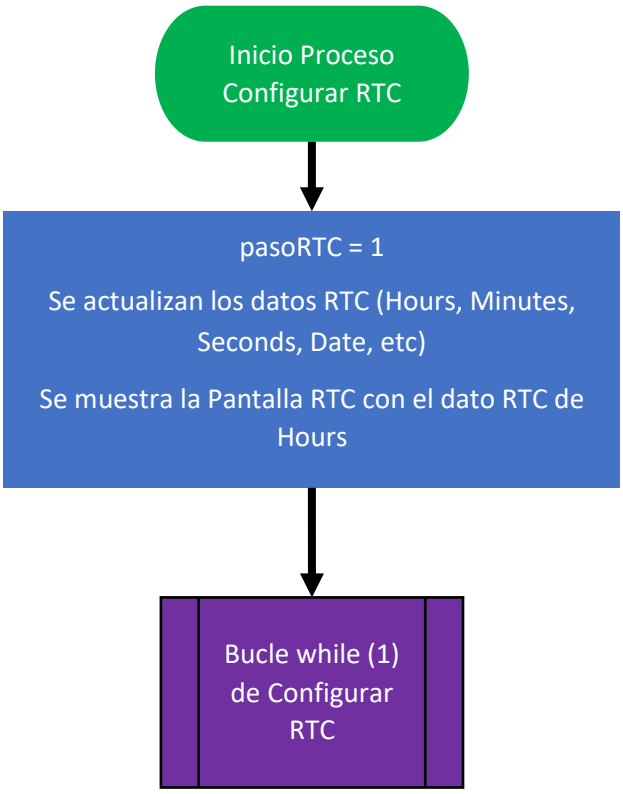

Diagrama 19: Diagrama de flujo del Proceso Configurar RTC.

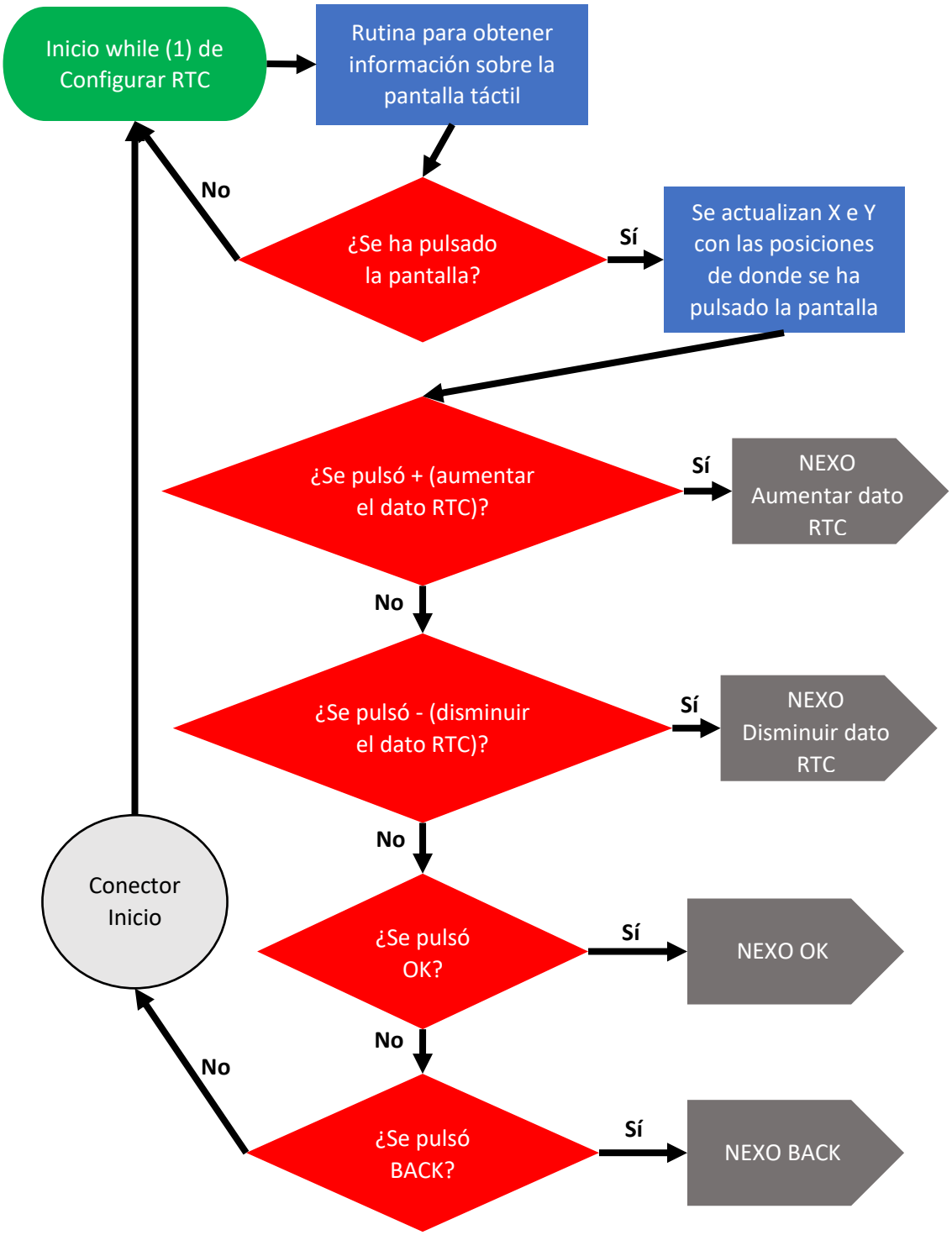

Diagrama 20: Diagrama de flujo del Proceso Configurar RTC.

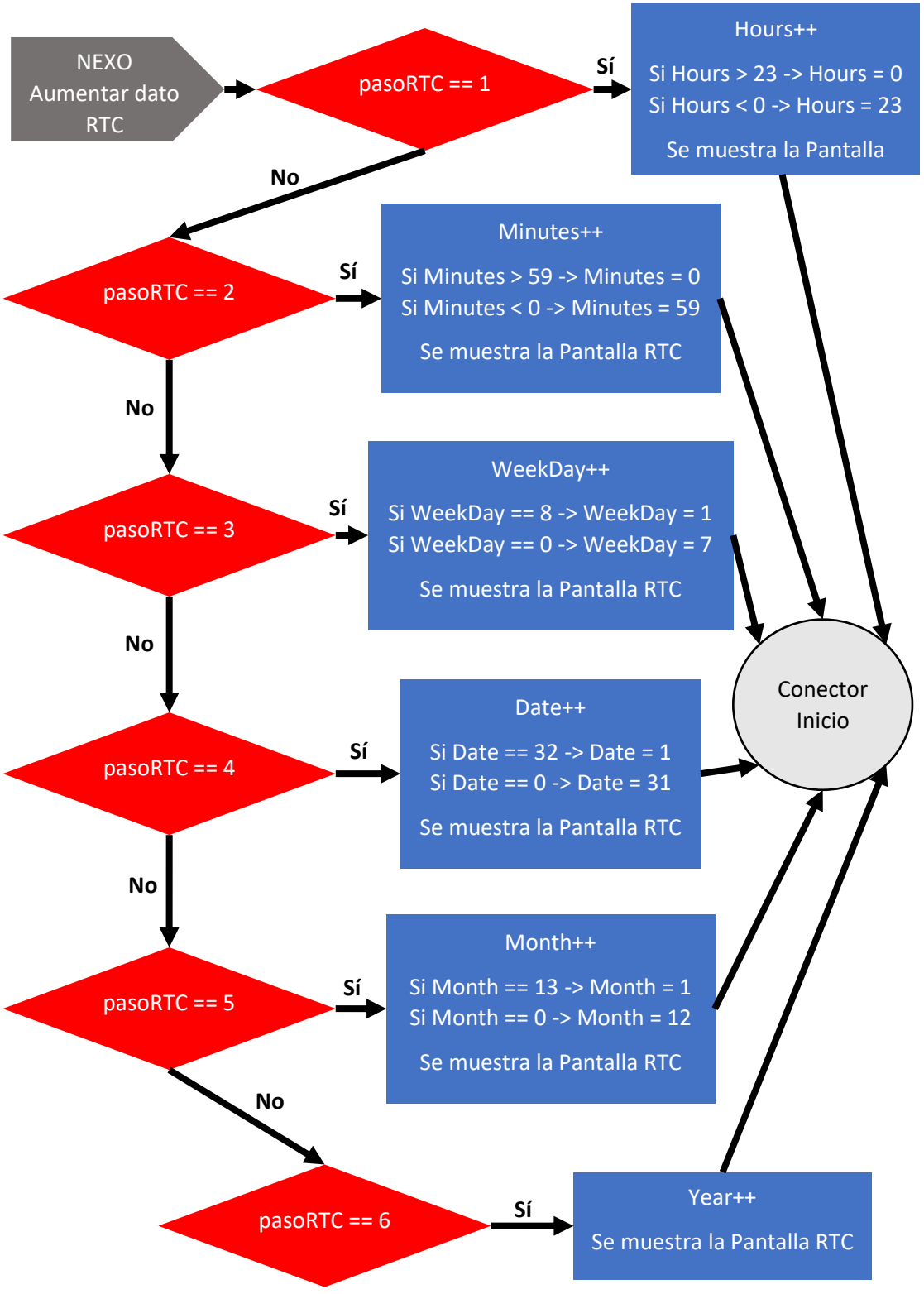

Diagrama 21: Diagrama de flujo del Proceso Configurar RTC.

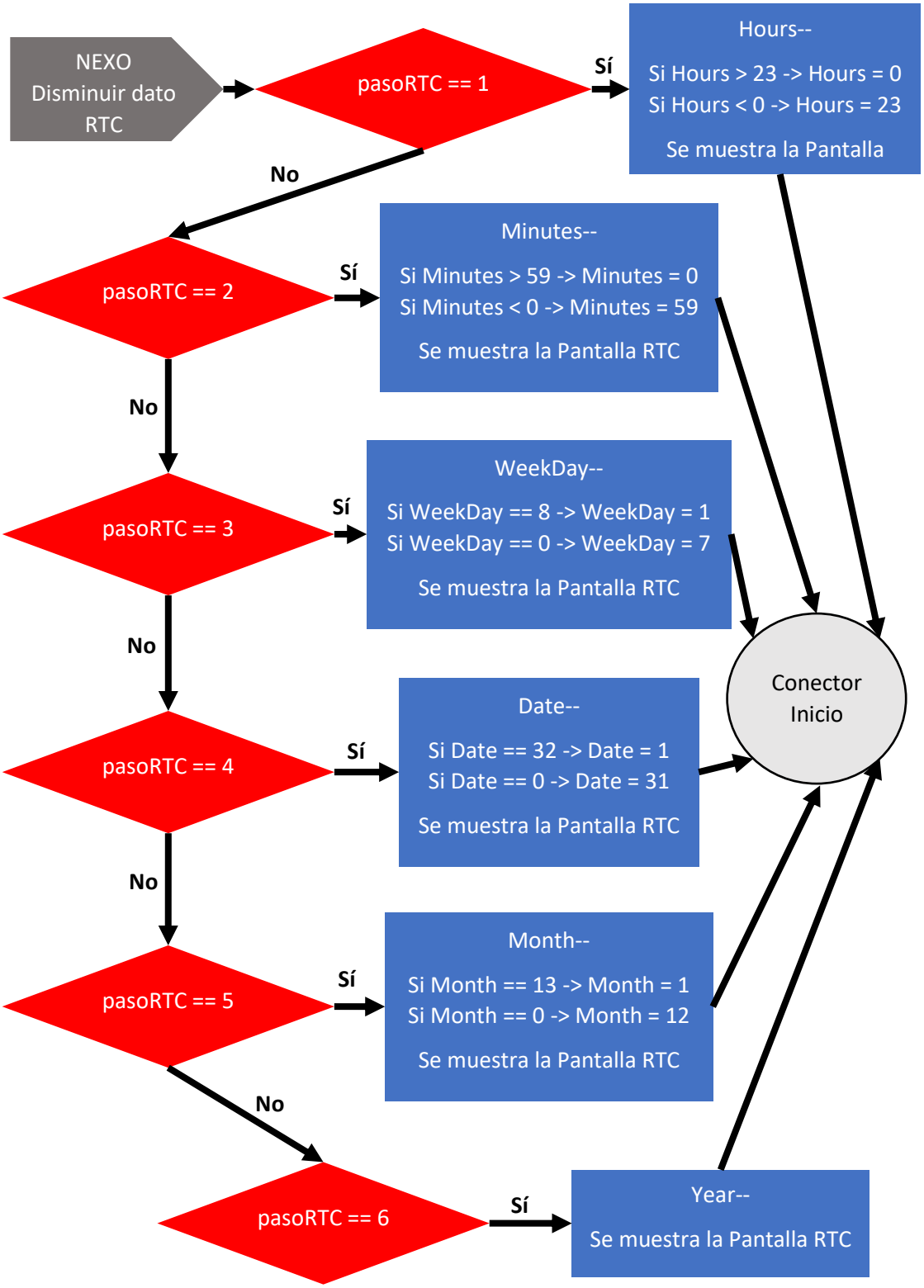

Diagrama 22: Diagrama de flujo del Proceso Configurar RTC.

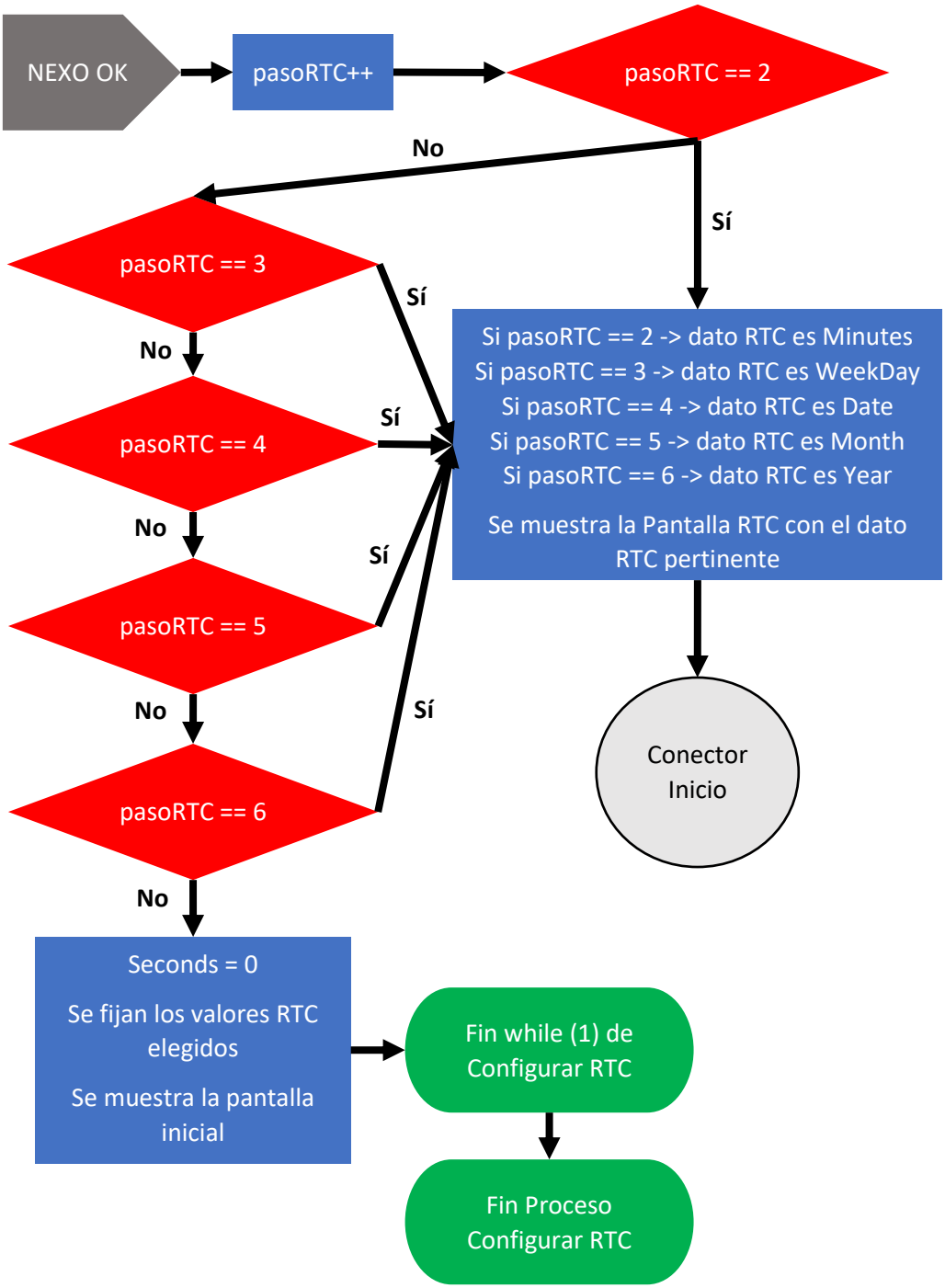

Diagrama 23: Diagrama de flujo del Proceso Configurar RTC.

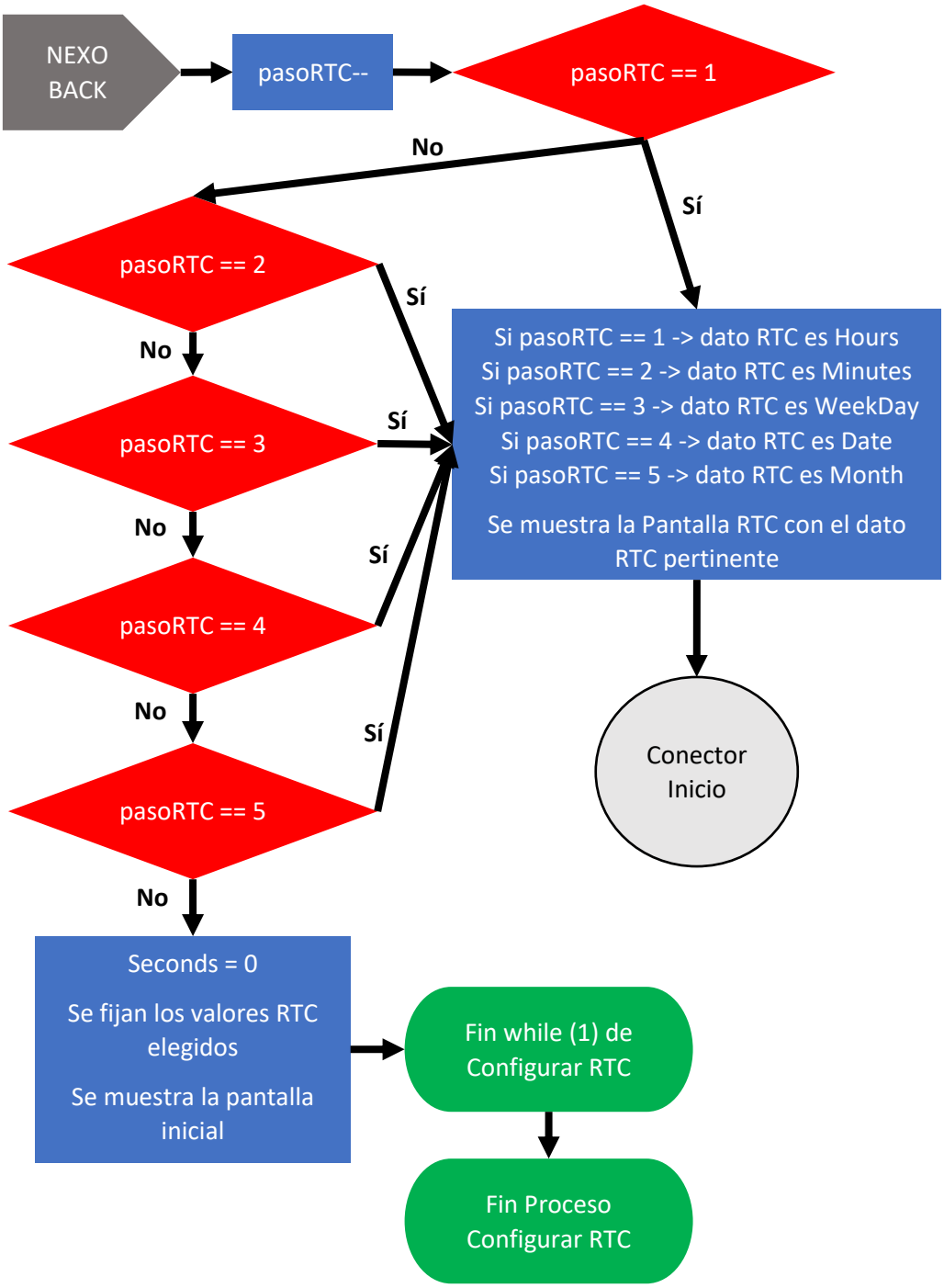

Diagrama 24: Diagrama de flujo del Proceso Configurar RTC.

Capítulo 2: Aplicación Android HomeT

2.1 Introducción

En este punto, tenemos desarrollado un termostato totalmente funcional, pero falta finalizar la parte *smart* del dispositivo. El desarrollo de la aplicación móvil, junto al uso del dispositivo ESP y la programación realizada en el STM32F429I-DISC1, completa la faceta *smart* de nuestro termostato.

HomeT es el nombre dado a la aplicación Android de nuestro termostato inteligente, esta aplicación nos permite ver la temperatura actual de la vivienda y el modo de funcionamiento en el que se encuentra el termostato. Además, la aplicación ofrece la opción de poder cambiar a modo manual y elegir la temperatura deseada. También podemos cambiar a modo programa, elegir las dos temperaturas deseadas y completar los programas de los días que deseemos.

Para realizar la aplicación móvil se ha utilizado Android Studio, el cual es el entorno de desarrollo integrado de la plataforma Android. Es el principal IDE para el desarrollo de aplicaciones Android, ya sean para móviles, relojes inteligentes o televisiones.

2.2 Pantallas gráficas desarrolladas

Android Studio nos ofrece la posibilidad de confeccionar las pantallas gráficas de dos formas distintas, pero, a su vez, complementarias. Podemos desarrollar la interfaz de usuario de la aplicación mediante el editor de diseño o mediante el lenguaje de marcado XML. En este proyecto se han usado ambas opciones.

La interfaz de usuario, presente en la aplicación, se puede ver en las **Figuras 37 y 38**. En la **Figura 37** se encuentran las pantallas referentes al manejo de la cuenta de usuario de la aplicación. En estas pantallas podemos iniciar sesión con una cuenta ya registrada, actualizar la contraseña de esta o crear una nueva cuenta. En la **Figura 38** se encuentran las pantallas principales de la aplicación, a las cuales accedemos una vez hemos iniciado sesión. En estas pantallas podemos ver la temperatura y el modo actuales, pasar a modo manual eligiendo la temperatura deseada, pasar a modo programa eligiendo las dos temperaturas deseadas y ver o configurar el programa de los días de la semana. Todo esto se consigue usando elementos gráficos como botones, campos de texto editables o botones de grupo.

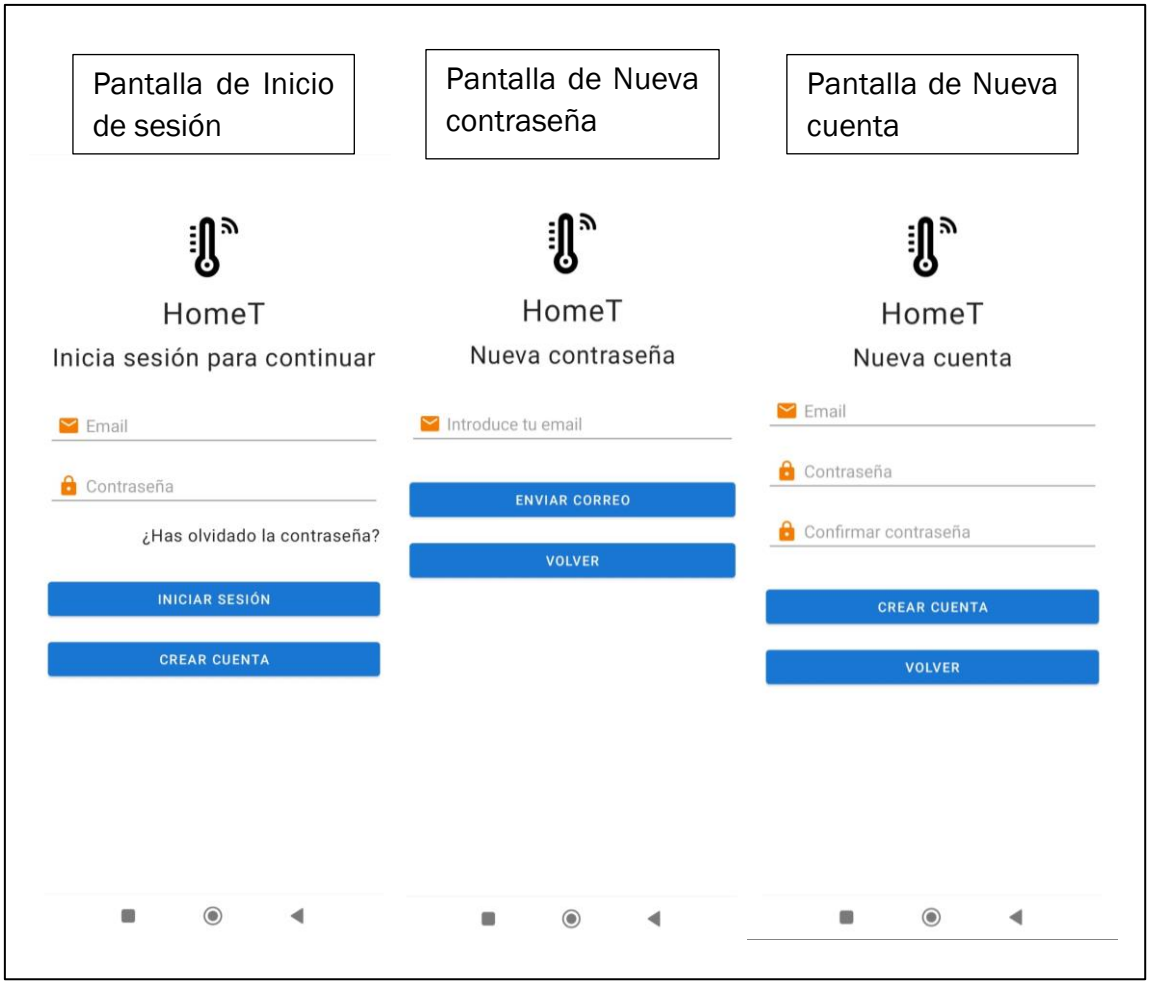

Figura 35: Pantallas de Inicio de sesión, Nueva contraseña y Nueva cuenta.

Fuente: Elaboración propia.

Figura 36: Pantallas de Inicio, Modo manual, Modo programa y Programa.

2.3 Programación

La programación de la aplicación móvil se ha realizado usando el lenguaje Kotlin. Este lenguaje se convirtió, por anuncio de Google en 2019, en el de uso preferencial para el desarrollo de aplicaciones en Android. Además, dada su mayor modernidad, se espera que, en un futuro, sustituya por completo, en este tipo de desarrollos, a Java.

En este apartado se va a hacer un recorrido general del código desarrollado. Además, nos enfocaremos en aclarar las partes más importantes del código. Para una explicación más completa de la programación realizada, y un mejor entendimiento de este apartado, se recomienda ver el código fuente del programa, presente en los anejos, con los comentarios realizados en él.

Hay que puntualizar que, en principio, se va a tener una actividad por cada pantalla gráfica presente en la aplicación móvil. Más adelante, veremos que el dogma anterior no se ha aplicado a todas las pantallas. Dando una definición simple, se podría decir que, básicamente, las pantallas de una aplicación Android son actividades. En realidad, una actividad está formada por dos archivos de programación indispensables. Uno es el archivo Kotlin (también puede ser Java), que representa la parte lógica. El otro es el archivo XML, que representa la parte gráfica.

En los siguientes apartados, se van a presentar los diagramas de flujo y ciertos detalles de importancia en la programación de la aplicación móvil.

2.3.1 Firebase Authentication

Para dar a nuestra aplicación HomeT cierta seguridad, se ha integrado la plataforma Firebase, creada por Google, con el proyecto desarrollado en Android Studio. Esta plataforma es gratuita, siempre que la aplicación no sobrepase un número de usuarios activos al mes, y nos ofrece diversas herramientas para el desarrollo de aplicaciones Android.

Algunos servicios ofrecidos por Firebase son los siguientes:

- Autenticación de la identidad de usuarios en aplicaciones.
- Almacenamiento en la nube.
- Bases de datos en tiempo real.
- Uso de Google Analitycs.

En la aplicación HomeT se ha usado el servicio Firebase Authentication para la autenticación de usuarios. Para poder usar HomeT, primero debemos registrarnos en Firebase usando un email y una contraseña. A continuación, se recibirá un correo en el email para la verificación del usuario. Una vez realizado este proceso, podemos acceder a la aplicación iniciando la sesión.

Cabe destacar que, para que el uso de la autenticación mediante correo y contraseña de una seguridad completa a la aplicación, hay que añadir un proceso de “enlace” entre la aplicación HomeT y el termostato. El proceso se explicará, de forma general, en el apartado de **Mejoras**.

2.3.2 Actividad Authentication

La actividad Authentication se corresponde con la pantalla de Inicio de sesión de la **Figura 37**. Esta actividad nos permite iniciar nuestra sesión de usuario y, con ello, poder acceder a la aplicación HomeT. Si existe algún tipo de error al iniciar la sesión en la aplicación, se avisará por pantalla al usuario con un cuadro de diálogo dando cierta información sobre lo ocurrido. En caso de que el error sea que el email no está verificado, se ofrecerá la posibilidad de recibir un nuevo correo de verificación en el email. Además, desde esta actividad podemos navegar a la actividad de creación o registro de una cuenta y a la de cambio de contraseña.

Los **Diagramas 25 y 26** representan el diagrama de flujo de la actividad Authentication.

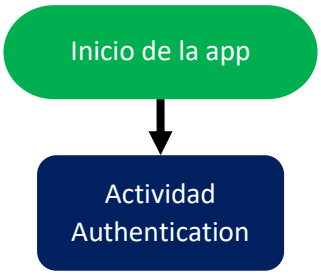

Diagrama 25: Diagrama de flujo de la actividad Authentication.

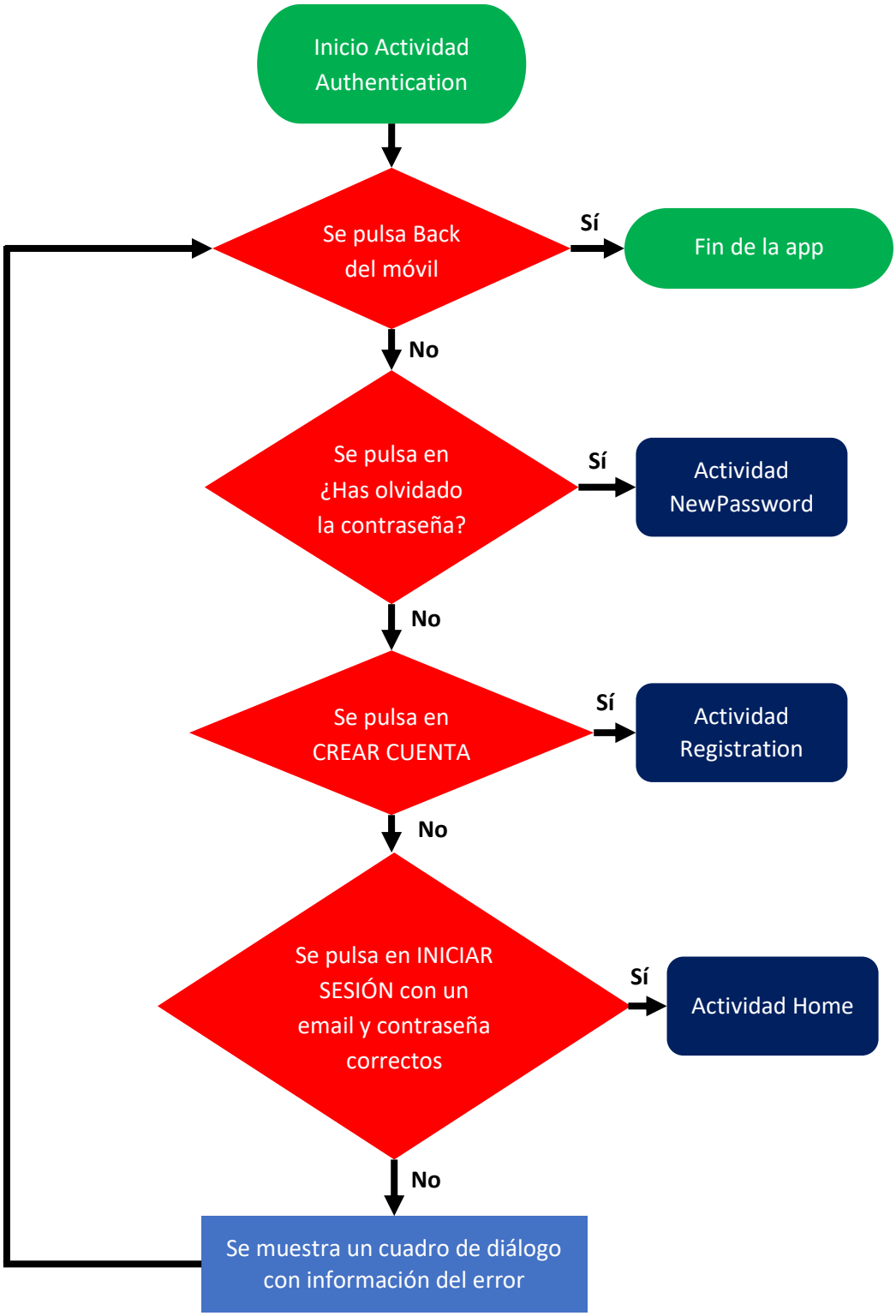

Diagrama 26: Diagrama de flujo de la actividad Authentication.

2.3.3 Actividad NewPassword

La actividad NewPassword se corresponde con la pantalla de Nueva contraseña de la **Figura 37**. Esta actividad nos permite cambiar la contraseña de nuestra cuenta. Si existe algún tipo de error con el email introducido, se avisará por pantalla al usuario con un cuadro de diálogo. En caso de que no exista error, se enviará un correo para cambiar la contraseña al email indicado.

El **Diagrama 27** representa el diagrama de flujo de la actividad NewPassword.

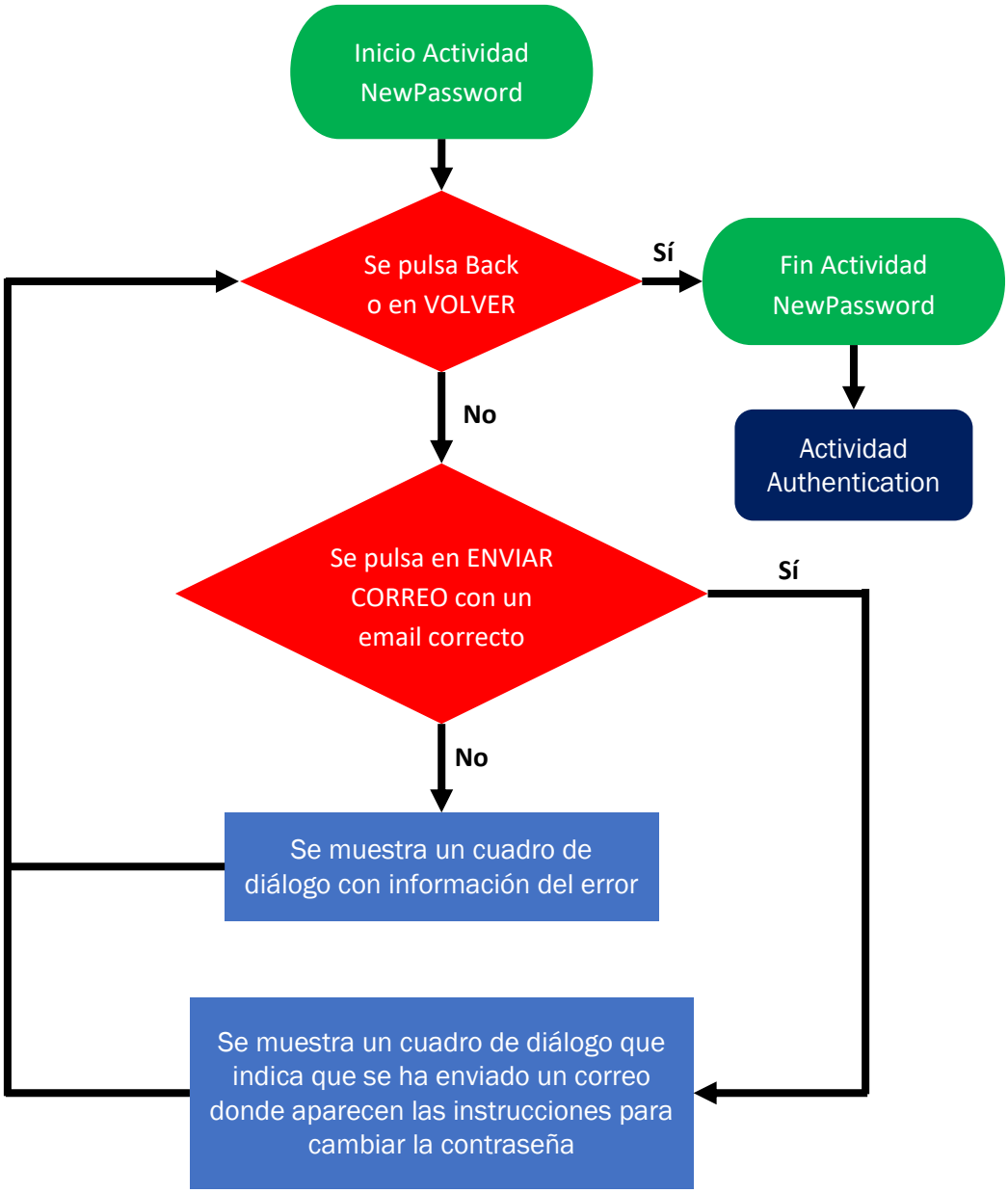

Diagrama 27: Diagrama de flujo de la actividad NewPassword.

2.3.4 Actividad Registration

La actividad Registration se corresponde con la pantalla de Nueva cuenta de la **Figura 37**. Esta actividad nos permite registrar un email (crear una cuenta) en Firebase para poder acceder a la aplicación HomeT. Si existe algún tipo de error al crear la cuenta, se avisará por pantalla al usuario con un cuadro de diálogo. En caso de que no exista error, se enviará un correo para verificar la cuenta creada.

El **Diagrama 28** representa el diagrama de flujo de la actividad Registration.

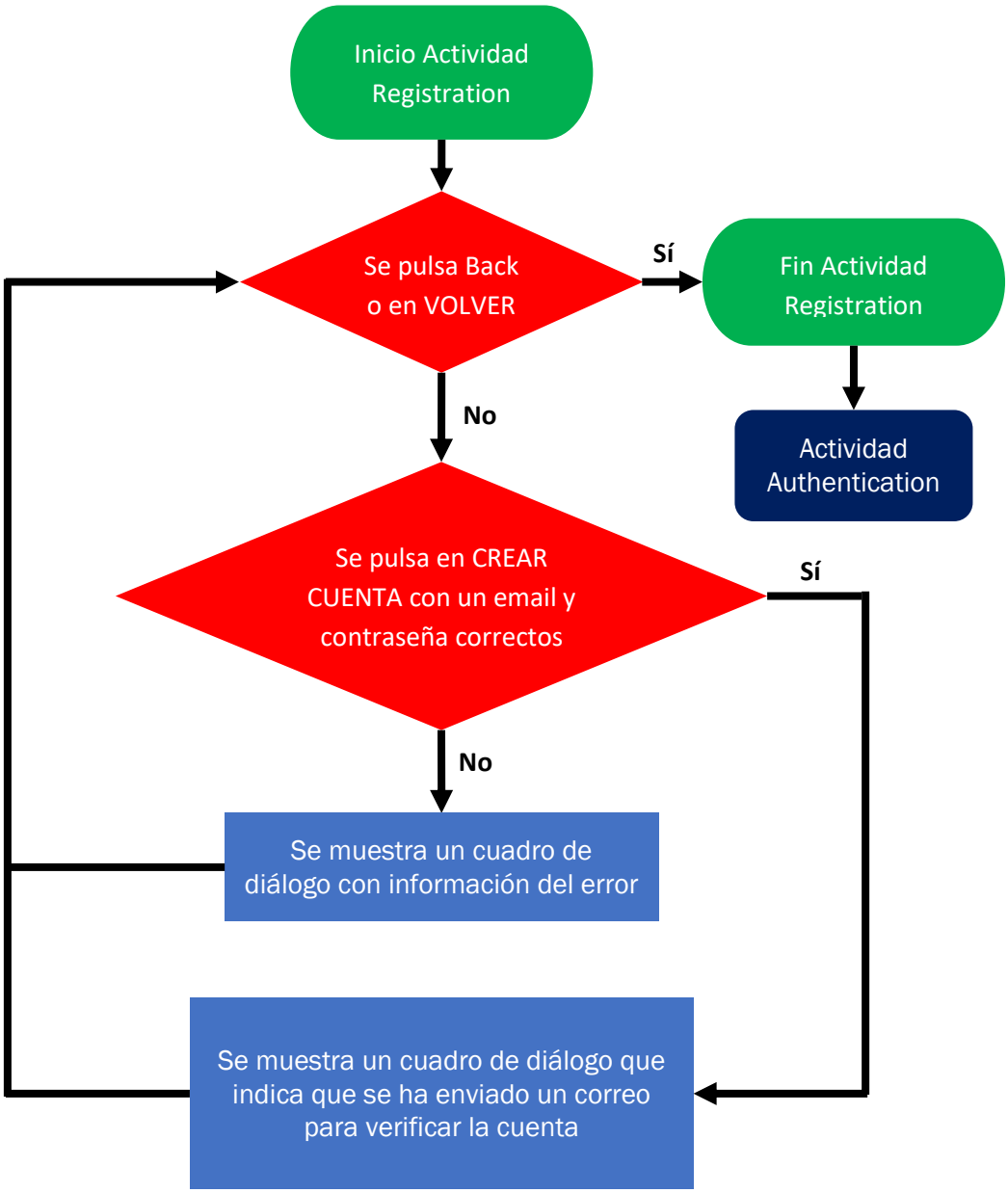

Diagrama 28:Diagrama de flujo de la actividad Registration.

2.3.5 Actividad Home

La actividad Home se corresponde con las pantallas de Inicio, Modo manual y Modo programa de la **Figura 38**, pero, en este caso, de una forma diferente a lo visto en las otras actividades. Esta actividad contiene el menú y navegador de la aplicación en la parte baja de la pantalla, pero, además, también tiene un contenedor de fragmentos, encargado de mostrar un fragmento en el resto de la pantalla no ocupada por el menú. En la **Figura 39** se puede ver la división mencionada.

Hasta este punto, se había trabajado únicamente con actividades, pero, en este apartado, vamos a usar fragmentos. Estos, básicamente, funcionan como las actividades, ya que llevan a cabo la misma función, mostrar una pantalla. Además, están formados por los dos mismos archivos de programación que las actividades, uno de lógica (en Kotlin) y otro gráfico (en XML). La diferencia principal reside en que un fragmento representa una parte reutilizable de la IU de una aplicación, es decir, una parte de la pantalla que puede cambiar visualmente. Los fragmentos son más "ligeros" que las actividades, pero no pueden existir por sí solos, deben estar alojados por una actividad u otro fragmento. Estas características hacen que los fragmentos sean una herramienta perfecta a la hora de crear una aplicación que utilice un menú.

Por lo tanto, la actividad Home va a mostrar por pantalla, en todo momento, el menú, que va a estar presente en la parte inferior. En el resto de la pantalla se podrá ver el contenedor de fragmentos, este mostrará el fragmento correspondiente a la selección realizada en el menú.

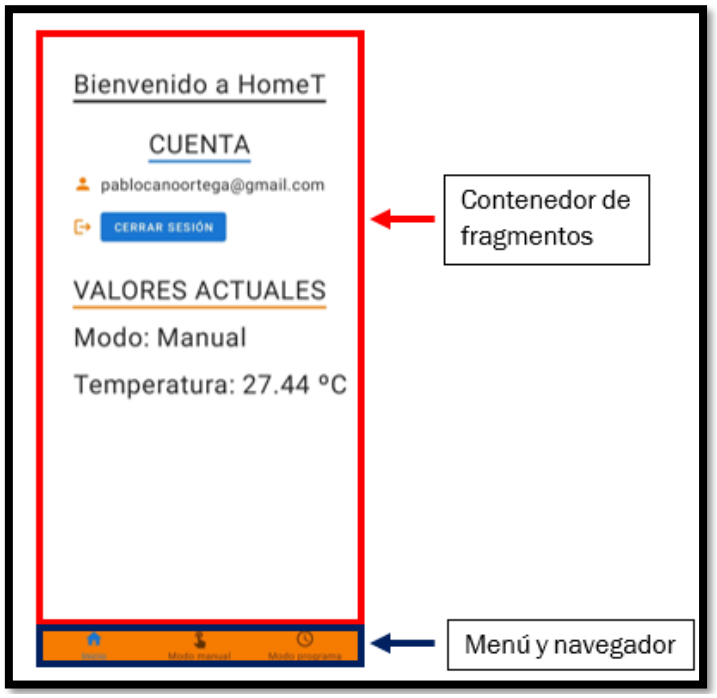

Figura 37: División de la pantalla de la actividad Home.

Fuente: Elaboración propia.

2.3.5.1 Menú y Navegador

Para conseguir la división de la pantalla presentada en la **Figura 39**, hace falta realizar varias cosas.

Primero, hay que crear dos archivos XML, uno que albergue el menú y otro que albergue el gráfico de navegación. Para crear el archivo del menú basta con añadir un nuevo archivo de recurso, de tipo Menu, al proyecto en Android Studio. En este archivo creamos lo que es el aspecto visual del menú, es decir, se añaden los iconos a mostrar y los títulos de cada uno de ellos. Para crear el archivo del gráfico de navegación basta con añadir un nuevo archivo de recurso, de tipo Navigation, al proyecto. En el gráfico de navegación, simplemente, añadimos los fragmentos por los que queremos navegar. En nuestro caso, estos fragmentos son Home, Manual Mode y Program Mode; los cuales se corresponden con la parte del contenedor de fragmentos vista en las pantallas de Inicio, Modo manual y Modo programa de la **Figura 38**.

Para acabar, hay que enlazar el menú y el navegador con la actividad Home. Para realizar esto, primero, hay que ir al archivo XML de la actividad Home y añadir una vista de tipo BottomNavigation y otra de tipo FragmentContainer. En la vista de BottomNavigation, navegador en la parte inferior de la pantalla, se le añade o asocia el menú realizado. En la vista de FragmentContainer, contenedor de fragmentos, se le añade el navegador realizado. Finalmente, acudimos al archivo Kotlin de la actividad Home para poder mostrar el fragmento deseado en el contenedor de fragmentos, es decir, se enlaza el contenedor con el navegador.

2.3.5.2 Fragmento Home

El fragmento Home se corresponde con la pantalla de Inicio, menos la parte del menú, de la **Figura 38**. Este fragmento nos permite visualizar el modo y la temperatura actuales del termostato. Esta información se muestra al iniciarse el fragmento, no hace falta presionar sobre ninguna parte de la pantalla. Para recibir esta información debemos comunicarnos con el microcontrolador a través del ESP. Esta comunicación se realiza empleando el protocolo HTTP, por ello hacemos uso de la biblioteca OkHttp en Android Studio.

A la hora de realizar la comunicación HTTP en este fragmento, seguimos el siguiente proceso:

1. Usando la dirección IP del ESP, creamos una URL, http://Direccion IP/, para poder comunicarnos con el ESP usando HTTP.
2. Se crea un cliente de OkHttp, usando la función OkHttpClient().
3. Se añade un string identificativo en la URL creada. Con esta cadena indicamos al STM32F429I-DISC1 el fragmento en el que estamos y la acción o acciones que tiene que realizar.
4. Se define el objeto con el que vamos a realizar la llamada HTTP, para ello usamos la función Request.Builder(). Con esta función pasamos la URL y añadimos una cabecera que indica que la conexión debe cerrarse al finalizar el proceso de comunicación.
5. Se realiza la llamada HTTP de forma asíncrona, ya que la aplicación no puede estar esperando a la llegada de unos datos que no sabemos cuándo van a llegar. Si ocurre algún error mostramos la excepción ocurrida por consola.
6. En caso de recibir una respuesta correcta, obtenemos el cuerpo de esta y filtramos la información para obtener, en este caso, el modo y la temperatura enviados desde el STM32F429I-DISC1. Para actualizar la pantalla del móvil con la información recibida, tenemos que hacerlo en el hilo donde se está ejecutando la interfaz de usuario. Para conseguir esto usamos la función runOnUiThread.

Los **Diagramas 29 y 30** representan el diagrama de flujo del fragmento Home.

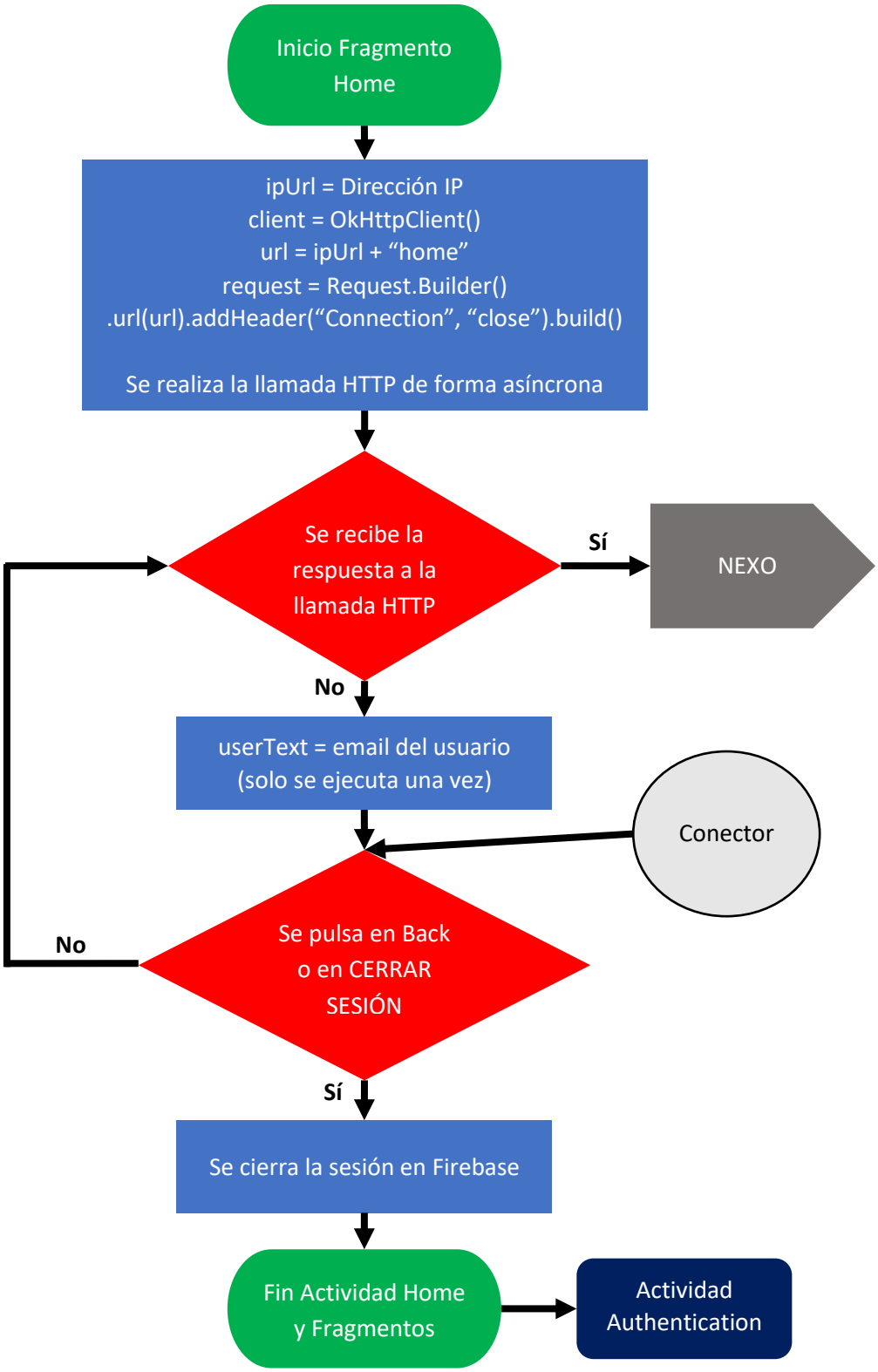

Diagrama 29: Diagrama de flujo del fragmento Home.

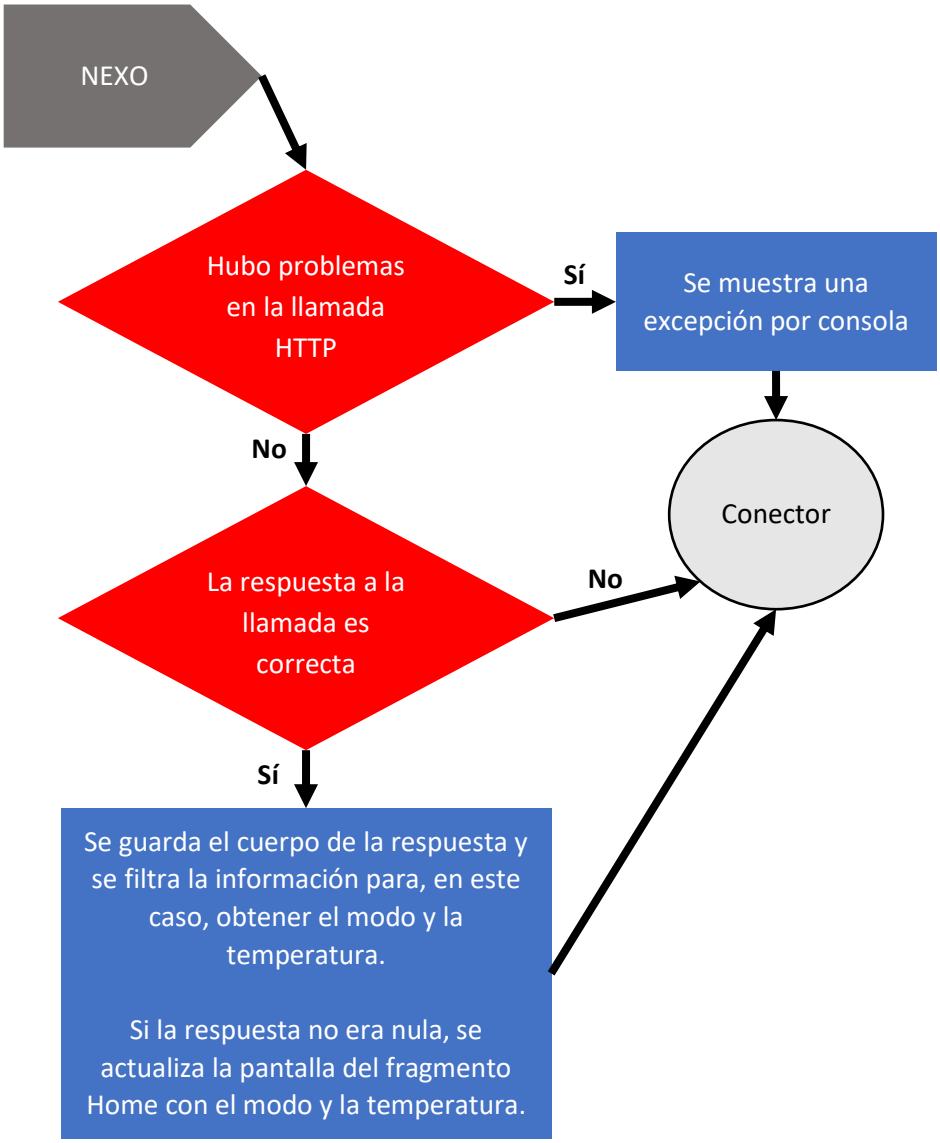

Diagrama 30: Diagrama de flujo del fragmento Home.

2.3.5.3 Fragmento ManualMode

El fragmento ManualMode se corresponde con la pantalla de Modo manual, menos la parte del menú, de la **Figura 38**. Este fragmento nos permite elegir la temperatura deseada y, con ello, pasar a modo manual. Si existe algún tipo de error al introducir la temperatura, se avisará por pantalla al usuario. Para enviar la temperatura manual que deseamos en la vivienda debemos comunicarnos con el microcontrolador a través del ESP. Esta comunicación se realiza empleando el protocolo HTTP, por ello hacemos uso de la biblioteca OkHttp en Android Studio.

A la hora de realizar la comunicación HTTP en este fragmento, seguimos el siguiente proceso:

1. Usando la dirección IP del ESP, creamos una URL, http://Direccion IP/, para poder comunicarnos con el ESP usando HTTP.
2. Se crea un cliente de OkHttp, usando la función OkHttpClient().
3. Se añade un string identificativo en la URL creada. Con esta cadena indicamos al STM32F429I-DISC1 el fragmento en el que estamos y la acción o acciones que tiene que realizar. Además, en este caso, se añade en la URL la temperatura manual deseada.
4. Se define el objeto con el que vamos a realizar la llamada HTTP, para ello usamos la función Request.Builder(). Con esta función pasamos la URL y añadimos una cabecera que indica que la conexión debe cerrarse al finalizar el proceso de comunicación.
5. Se realiza la llamada HTTP de forma asíncrona. Como no tenemos ningún dato que recibir, no hacemos nada en caso de tener respuesta.

El **Diagrama 31** representa el diagrama de flujo del fragmento ManualMode.

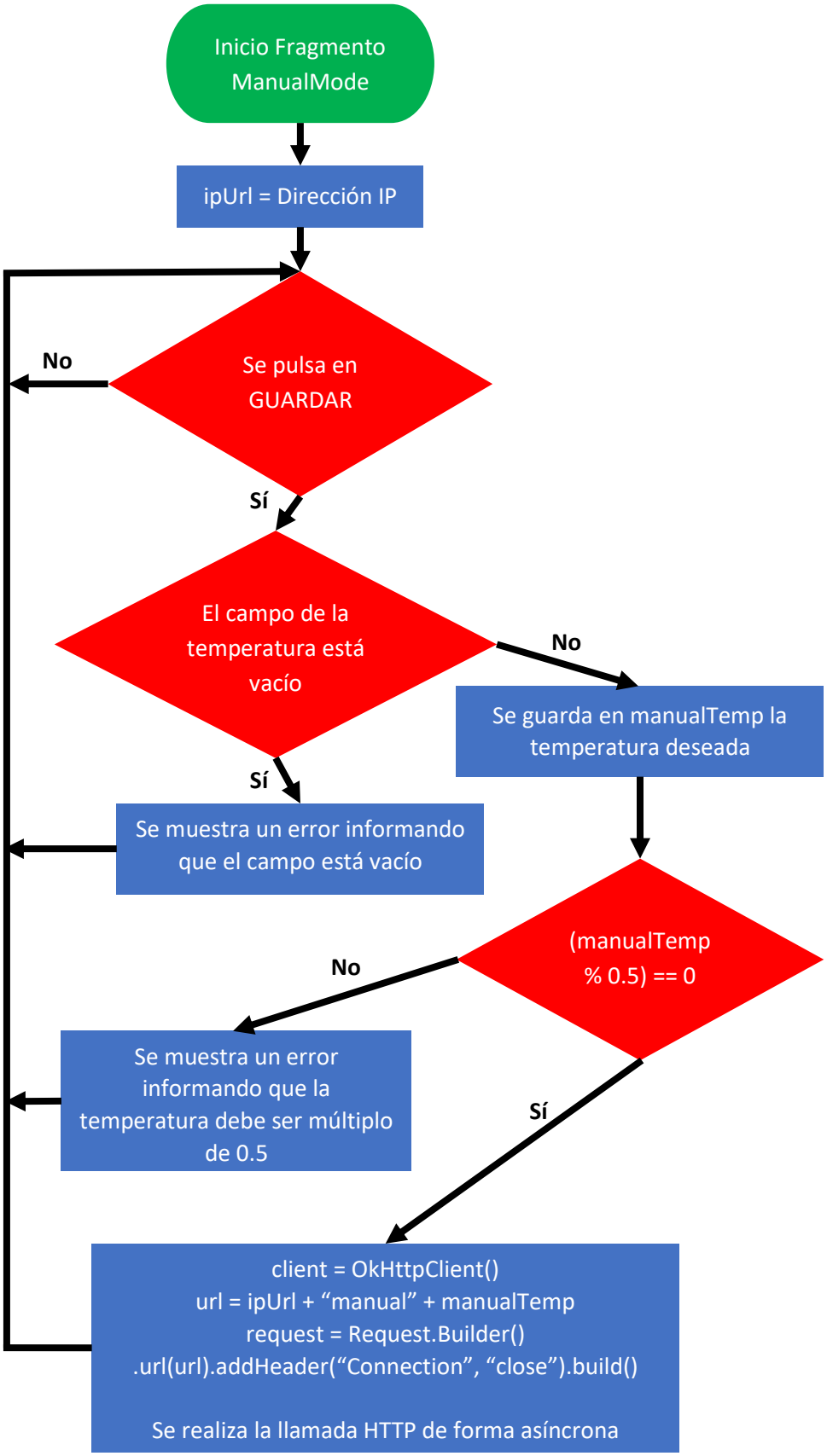

Diagrama 31: Diagrama de flujo del fragmento ManualMode.

2.3.5.4 Fragmento ProgramMode

El fragmento ProgramMode se corresponde con la pantalla de Modo programa, menos la parte del menú, de la **Figura 38**. Este fragmento nos permite elegir las dos temperaturas de programa deseadas y, con ello, pasar a modo programa. Además, desde este fragmento, podemos acceder a las actividades que controlan la configuración de los programas de cada día de la semana. Si existe algún tipo de error al introducir la temperatura, se avisará por pantalla al usuario. Para enviar las temperaturas que deseamos en los programas, debemos comunicarnos con el microcontrolador a través del ESP. Esta comunicación se realiza empleando el protocolo HTTP, por ello hacemos uso de la biblioteca OkHttp en Android Studio.

A la hora de realizar la comunicación HTTP en este fragmento, seguimos el siguiente proceso:

1. Usando la dirección IP del ESP, creamos una URL, http://Direccion IP/, para poder comunicarnos con el ESP usando HTTP.
2. Se crea un cliente de OkHttp, usando la función OkHttpClient().
3. Se añade un string identificativo en la URL creada. Con esta cadena indicamos al STM32F429I-DISC1 el fragmento en el que estamos y la acción o acciones que tiene que realizar. Además, en este caso, se añaden a la URL las dos temperaturas deseadas en el modo programa.
4. Se define el objeto con el que vamos a realizar la llamada HTTP, para ello usamos la función Request.Builder(). Con esta función pasamos la URL y añadimos una cabecera que indica que la conexión debe cerrarse al finalizar el proceso de comunicación.
5. Se realiza la llamada HTTP de forma asíncrona. Como no tenemos ningún dato que recibir, no hacemos nada en caso de tener respuesta.

Los **Diagramas 32 y 33** representan el diagrama de flujo del fragmento ProgramMode.

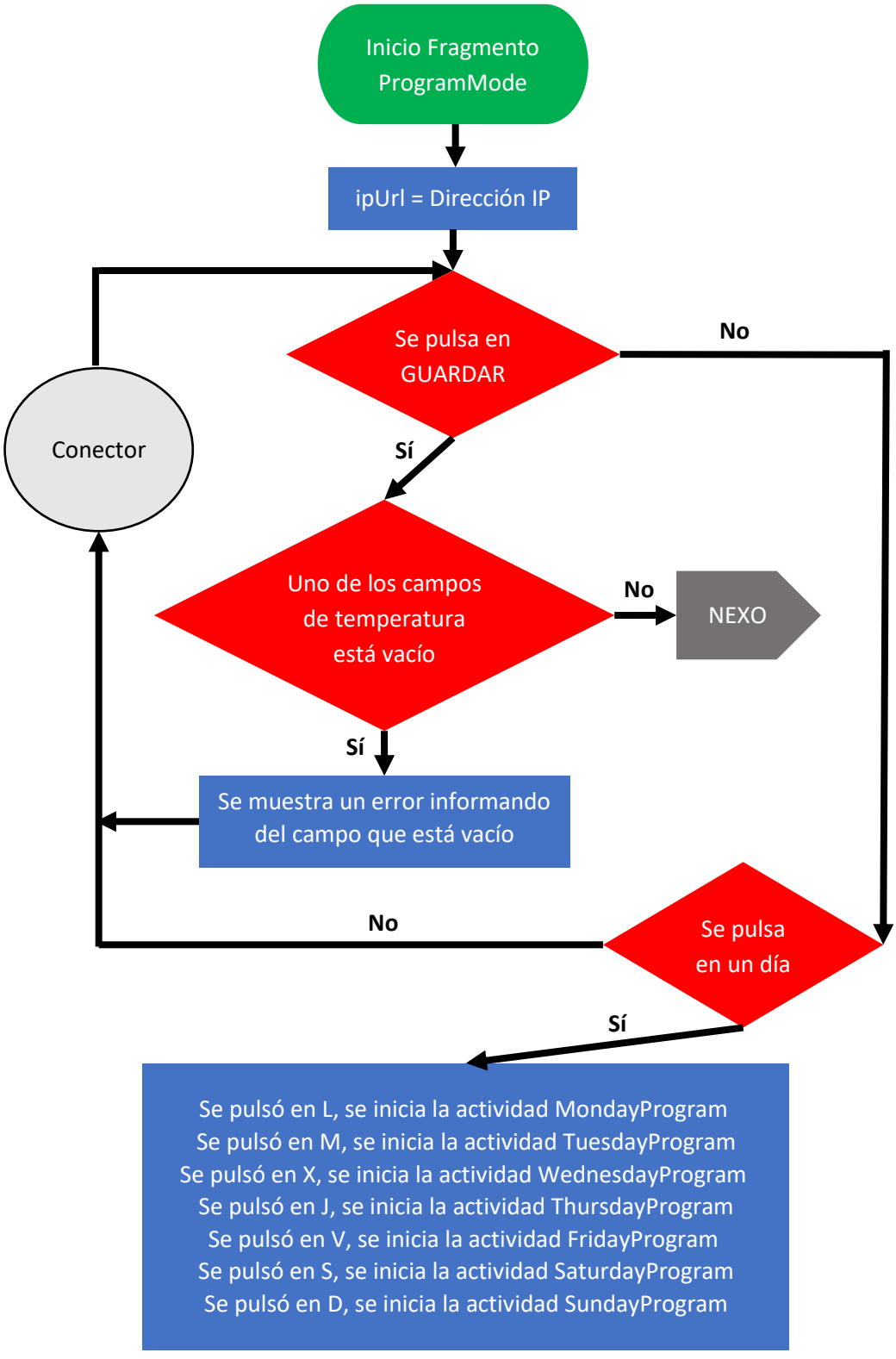

Diagrama 32: Diagrama de flujo del fragmento ProgramMode.

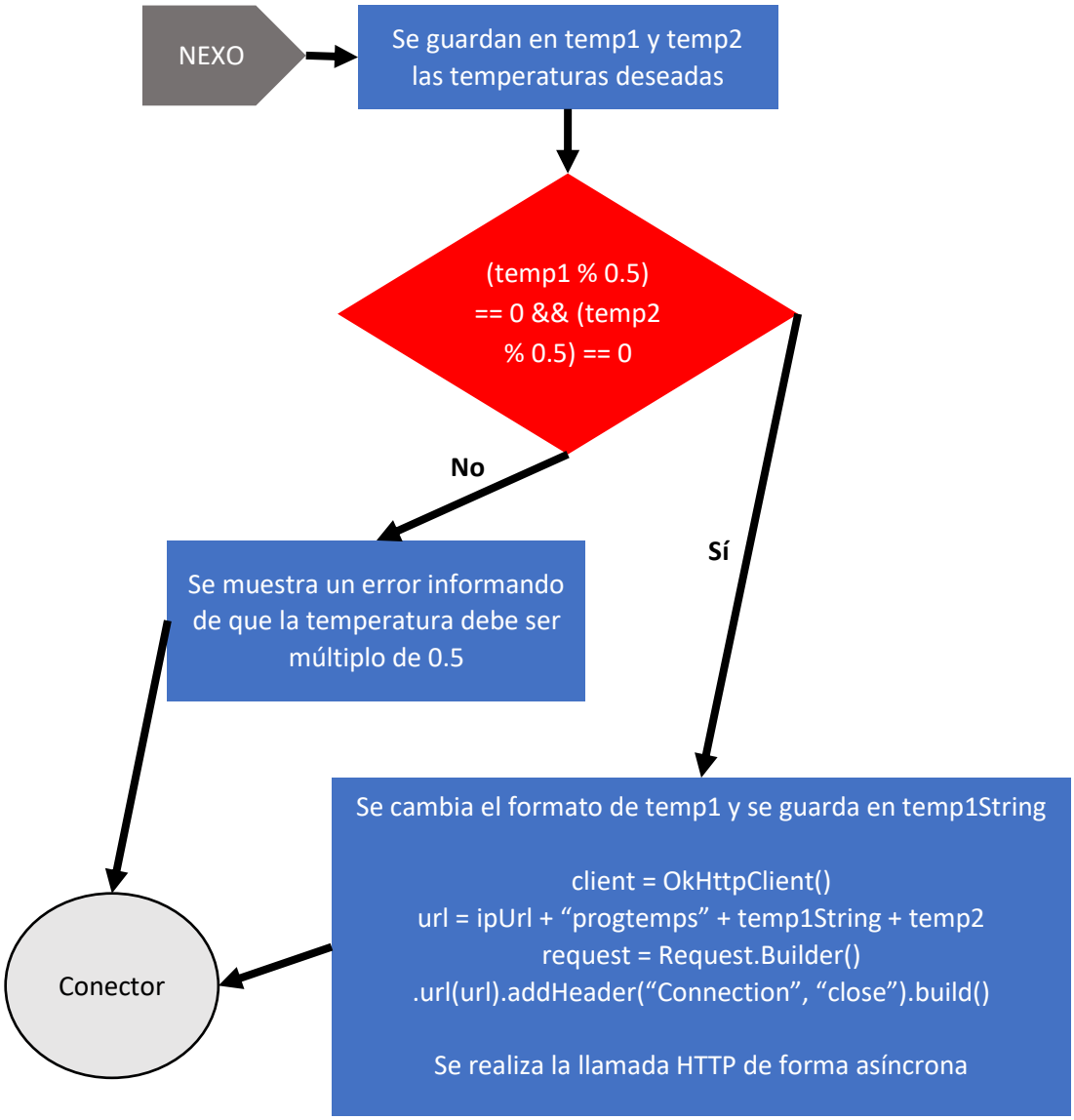

Diagrama 33: Diagrama de flujo del fragmento ProgramMode.

2.3.5.5 Actividades de los programas de los días

Las actividades para la configuración de los programas de los días de la semana se corresponden con la pantalla de Programa, menos la parte del menú, de la **Figura 38**. Estas actividades se inician desde el fragmento ProgramMode, nos permiten realizar la configuración de los programas de los días. Además, podemos ver los programas, que actualmente están en uso, de cada día. Para enviar los programas deseados o ver los actuales, debemos comunicarnos con el microcontrolador a través del ESP. Esta comunicación se realiza empleando el protocolo HTTP, por ello hacemos uso de la biblioteca OkHttp en Android Studio.

Hay que indicar que los grupos de botones presentes en las pantallas de estas actividades van a servir tanto para realizar la configuración del programa como para ver el programa actual que se recibe.

A la hora de realizar la comunicación HTTP en este fragmento, seguimos el siguiente proceso:

1. Usando la dirección IP del ESP, creamos una URL, http://Direccion IP/, para poder comunicarnos con el ESP usando HTTP.
2. Se crea un cliente de OkHttp, usando la función OkHttpClient().
3. Se añade un string identificativo y la inicial del día en la URL creada. Con esta cadena indicamos al STM32F429I-DISC1 la acción o acciones que tiene que realizar. Además, en caso de enviar el programa, se añade a la URL el programa del día.
4. Se define el objeto con el que vamos a realizar la llamada HTTP. Para ello usamos la función Request.Builder(). Con esta función pasamos la URL y añadimos una cabecera que indica que la conexión debe cerrarse al finalizar el proceso de comunicación.
5. Se realiza la llamada HTTP de forma asíncrona. En el caso de enviar el programa, como no tenemos ningún dato que recibir, no hacemos nada en caso de tener respuesta. Pero, en el caso de que queramos ver el programa actual, al recibir una respuesta correcta, obtenemos el cuerpo de esta y filtramos la información para obtener, en este caso, el programa enviado desde el STM32F429I-DISC1. Para actualizar la pantalla del móvil con la información recibida, tenemos que hacerlo en el hilo donde se está ejecutando la interfaz de usuario. Para conseguir esto usamos la función runOnUiThread.

Los **Diagramas 34 y 35** representan, de forma genérica, el diagrama de flujo de las actividades presentadas en este apartado.

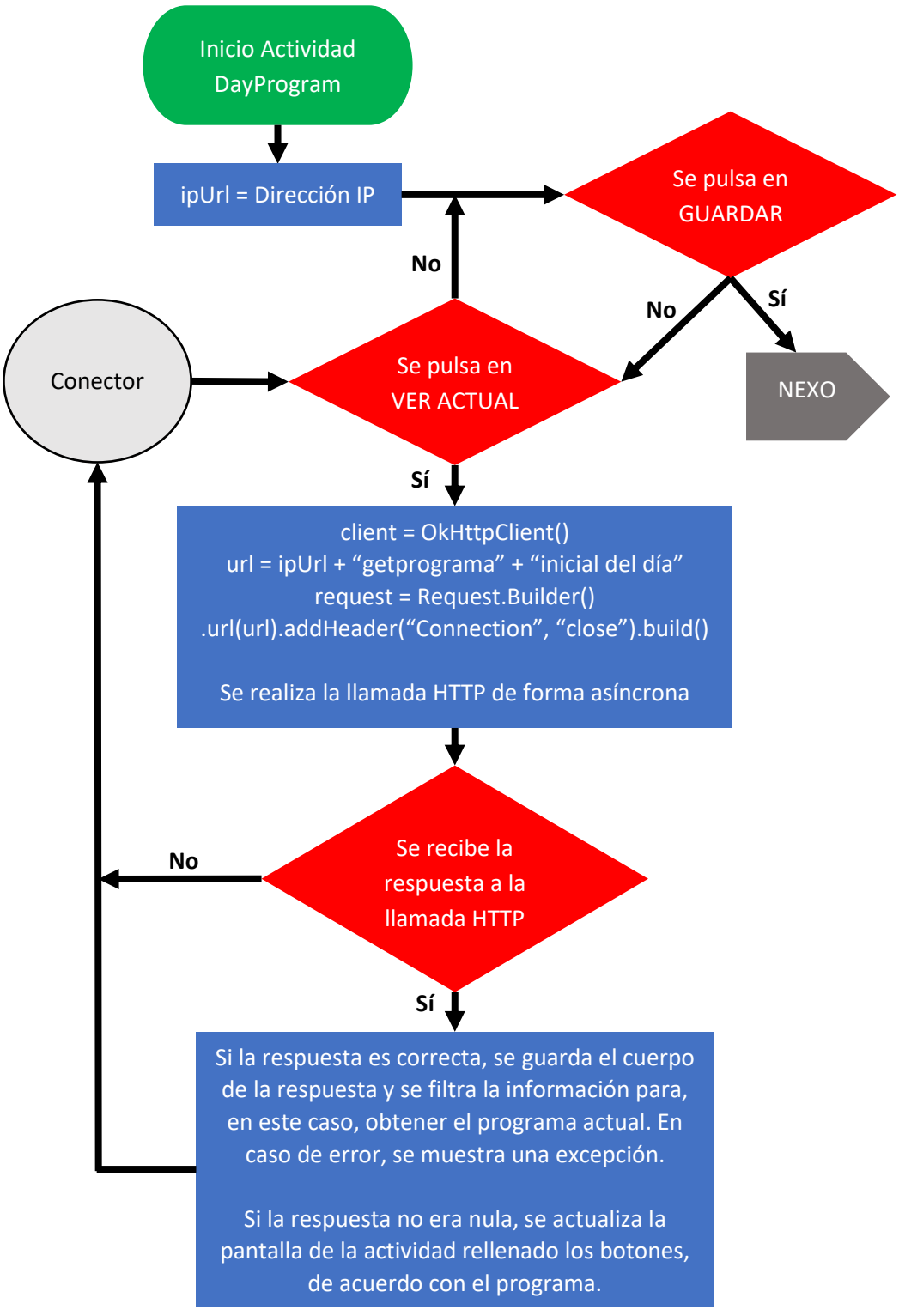

Diagrama 34: Diagrama de flujo de una actividad del programa de un día.

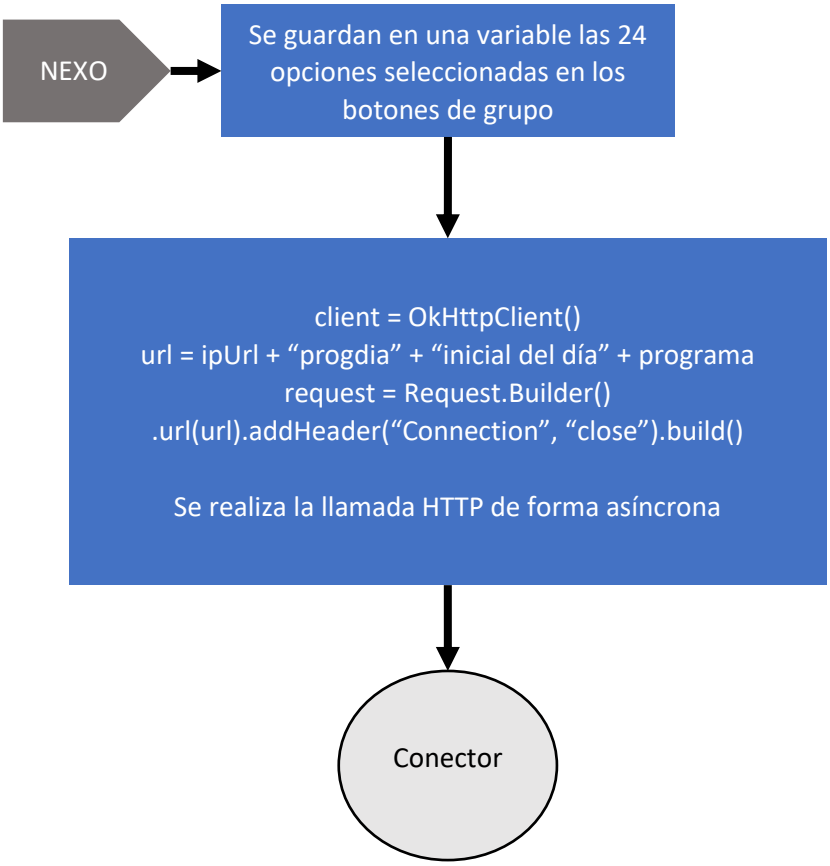

Diagrama 35: Diagrama de flujo de una actividad del programa de un día.

Capítulo 3: Comunicación entre HomeT y el STM32F429I-DISC1

3.1 Protocolo HTTP

Durante el proyecto se ha hablado del uso del protocolo HTTP para la comunicación entre la aplicación móvil y el microcontrolador STM, pero no se ha profundizado en sus características o en cómo se iba a implementar. A continuación, se van a ver los conceptos básicos y las partes relevantes de este que se han usado en la comunicación.

HTTP es un protocolo de comunicación que permite la transferencia de información a través de archivos de lenguaje de marcado, como XML y HTML, en internet.

> Sigue el clásico modelo cliente-servidor, en el que un cliente establece una conexión con el servidor, realiza una petición y espera hasta que recibe una respuesta de este. Se trata de un protocolo sin estado, es decir, no se guarda ninguna información sobre las conexiones anteriores. (Mozilla, 2022a)

Hay distintas peticiones dentro del protocolo HTTP, algunas de ellas son GET, POST, PUT o HEAD. Estas peticiones se realizan mediante el uso de las URL.

> Una URL no es más que una dirección dada a un recurso único y específico en la Web. En teoría, cada URL apunta a un único recurso. Dichos recursos pueden ser páginas HTML, documentos CSS, imágenes, etc. En la práctica, hay algunas excepciones, siendo la más común una URL apuntando a un recurso que ya no existe o que ha sido movido. (Mozilla, 2022b)

Una URL está compuesta de diferentes partes, en la **Tabla 8** se muestran las partes principales de la URL de ejemplo: http://www.example.com/path/to/myfile.html.

Parte	Significado
http	Es el protocolo. La primera parte de una URL indica qué protocolo debe usar el navegador.
www.example.com	Es el nombre de dominio. Indica qué servidor web se solicita. Alternativamente, es posible usar directamente una dirección IP, cosa que hemos usado en este trabajo.
/path/to/myfile.html	Es la ruta al recurso en el servidor web. Representa la ubicación o acceso a un elemento en concreto del servidor.

Tabla 8: Significado de las partes de una URL de ejemplo.

Fuente: (Mozilla, 2022b).

3.2 Comunicación desarrollada

3.2.1 Servidor

Hay que aclarar que, en este proyecto, se ha usado, únicamente, la petición GET en la comunicación entre la aplicación móvil y el STM32F429I-DISC1. Esta petición tiene como único objetivo recabar información del servidor.

Como se vio en la **Tabla 6**, el ESP8266 puede utilizar sin ningún problema el protocolo de comunicación HTTP. Para completar la parte del servidor, el cual se corresponde con el conjunto formado por el ESP y el STM32F429I-DISC1, faltaría añadir, en la programación del microcontrolador STM, que cada repuesta que se envía al cliente desde el servidor debe seguir un formato parecido al que se ve en la **Figura 40**.

```
HTTP/1.1 200 OK
**Content-Encoding**: gzip
**Content-language**: en
**Content-Length**: 5107
**Content-Type**: text/html; charset=utf-8
**Date**: Mon, 09 Nov 2015 12:08:48 GMT
**Last-Modified**: Mon, 09 Nov 2015 11:50:11 GMT
**Link**: <http://php.net/index>; rel=shorturl
**Server**: nginx/1.6.2
**Vary**: Accept-Encoding
**X-Frame-Options**: SAMEORIGIN
**X-Powered-By**: PHP/5.6.13-0+deb8u1

<!DOCTYPE html...
...
```

Figura 38: Ejemplo de una respuesta HTTP.

Fuente: (Lázaro, 2018).

En la figura anterior podemos ver, en primer lugar y en texto de color blanco, una línea de texto que indica el protocolo de comunicación, la versión HTTP usada (1.1) y una indicación de que la petición realizada se ha procesado correctamente (200 OK). A continuación, podemos ver distintas cabeceras que dan información sobre el lenguaje usado, el tamaño del contenido, la fecha de envió o el tipo de contenido de la respuesta. Más adelante, en color rojo, aparece el cuerpo de la respuesta. El cuerpo contiene toda la información que se quiere enviar al cliente y que no aparece en las cabeceras.

Aunque lo visto anteriormente se trata de la respuesta tipo a una petición GET, en el presente proyecto se ha reducido el formato al mínimo posible, como se vio en la **Figura 36**.

Este formato mínimo utilizado consta, únicamente, de la primera línea de texto, de la cabecera, que indica el tipo de contenido en el cuerpo de la respuesta, en este caso lenguaje HTML, y del propio cuerpo, el cual contiene la información de la respuesta.

3.2.2 Cliente

La parte que falta de preparar para tener una correcta comunicación, usando el protocolo HTTP, es la del cliente, es decir, la aplicación móvil HomeT.

Para que la comunicación pueda llevarse a cabo, primero, debemos añadir en el archivo XML AndroidManifest las dos líneas de código que aparece abajo.

La siguiente línea permite el uso de Internet en la aplicación Android.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

La siguiente línea permite el uso de una URL de tipo http. De serie solo permite el uso de https.

```
android:usesCleartextTraffic="true"
```

Por último, debemos incorporar la librería OkHttp al proyecto realizado en Android Studio. Además, a la hora de programar debemos seguir los pasos vistos en los procesos de los apartados **2.3.5.2 Fragmento Home**, **2.3.5.3 Fragmento ManualMode** y **2.3.5.4 Fragmento ProgramMode**.

3.2.3 Protocolo de peticiones

Como se ha dicho antes, las peticiones GET que se realizan desde la aplicación HomeT al STM32F429I-DISC1 van ligadas al uso de una URL. A continuación, en la **Tabla 9**, vamos a ver las URL de las distintas peticiones existentes y qué información y acciones a ejecutar solicitan. Es importante comentar que todas las URL siguen el formato siguiente: http://Dirección IP del ESP/ruta particular de la petición. En la Ruta URL se ha indicado las variables en negrita.

Petición	**Ruta URL de la petición**	**Cambios en el servidor**	**Información recibida por el cliente**
Home	home	Ninguno	Temperatura y modo actuales
Manual	manual**manualTemp**	Se pasa a modo manual y se actualiza la temperatura manual con el valor de manualTemp	Ninguna
Progtemps	progtemps**temp1Stringtemp2**	Se pasa a modo programa y se actualizan las temperaturas de programa con los valores de temp1String y temp2	Ninguna
Getprograma	getprogramaincialdia	Ninguno	El programa del día indicado por su inicial
Progdia	progdiainicialdia**program**	Se actualiza el programa del día, indicado por su inicial, con el valor de program	Ninguna

Tabla 9: Protocolo de peticiones realizado.

Fuente: Elaboración propia.

3.2.4 Comunicación fuera del domicilio

La comunicación desarrollada, hasta este punto, funciona, pero solo cuando nuestro dispositivo móvil se encuentra conectado a la red wifi doméstica, es decir, cuando el teléfono móvil se encuentra en la misma red de comunicación que el ESP12-S.

En el caso de que, por ejemplo, estemos fuera de casa y queramos modificar algún parámetro, la comunicación será inviable. Por lo tanto, urge buscar una solución para permitir la comunicación con el termostato cuando estemos fuera del rango de la red wifi doméstica.

La solución encontrada para este problema se basa en redirigir las peticiones HTTP, que llegan al puerto 80 de nuestro router, al dispositivo ESP-12S. Esto se puede realizar fácilmente siguiendo el siguiente proceso.

Primero, tenemos que acceder al menú de configuración del router, para ello escribimos, en la barra de direcciones de un navegador web, la dirección IP de nuestro router y, a continuación, introducimos la contraseña pertinente. Una vez estamos en el menú de configuración, pulsamos en Configuración Avanzada, luego en Advanced Setup, luego en NAT y, por último, en Virtual Servers. Ahora, tenemos que completar, tal y como aparece en la **Figura 41,** la configuración de la redirección al ESP-12S.

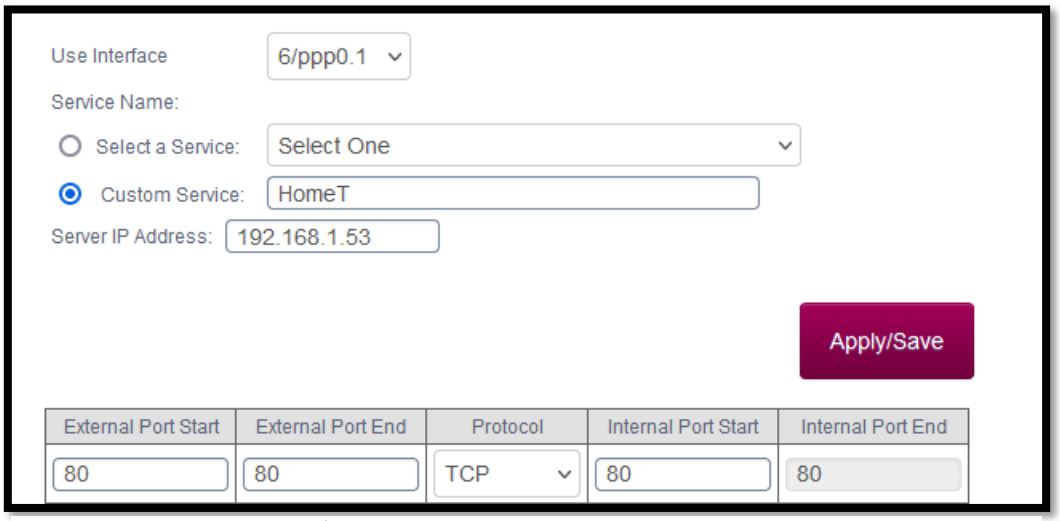

Figura 39: Configuración del servidor virtual para redireccionar peticiones al ESP-12S.

Fuente: Elaboración propia.

Primero, debemos elegir la interfaz ppp0.1, ya que esta es la interfaz utilizada cuando en el puerto se va a usar internet, es decir, cuando el puerto se comunica con el exterior. A continuación, le damos un nombre al servicio que vamos a configurar, en este caso le damos el nombre de la aplicación, y, también, ponemos la dirección IP del ESP-12S, receptor de las peticiones redireccionadas.

Finalmente, ponemos el rango de puertos que van a ser configurados, tanto los externos como los internos. En nuestro caso, es, únicamente, el puerto 80. Además, debemos utilizar el protocolo TCP, ya que es el protocolo manejado por el puerto 80. Después de realizar la configuración del router, debemos tener en cuenta que para poder acceder desde fuera de casa al termostato tenemos que usar la IP pública o externa de nuestro router, en vez de la dirección IP del ESP-12S, por lo que habría que modificar la programación del proyecto de Android Studio. También podemos realizar la comunicación con el termostato usando un navegador web e introduciendo las URL vistas en el apartado **3.2.3 Protocolo de peticiones**, pero, en este caso, usando la dirección IP pública del router.

Conclusiones y mejoras

Conclusiones

En primer lugar, hay que indicar que se han cumplido los principales objetivos de este trabajo fin de grado, indicados en el apartado **Objetivos y justificación del proyecto**. Se ha desarrollado todo el *software* necesario, usando un procesador ARM Cortex M4, para el correcto control de una caldera. Este control se puede realizar directamente desde el dispositivo físico o usando la aplicación móvil desarrollada.

A continuación, voy a plasmar algunas de las conclusiones que, personalmente, he obtenido al realizar el proyecto:

- Dentro de la ingeniería, la domótica es un campo con un gran futuro y que facilita enormemente las vidas de las personas, pero necesita expandirse aún más en nuestro país.
- Las características esenciales que deben tener los sistemas domóticos para tener éxito en el mercado son las siguientes: deben tener un coste moderado, ser fáciles de utilizar y deben mantener un cierto parecido en el funcionamiento con los sistemas a los que sustituyen.
- El grado cursado otorga competencias básicas en un amplio rango de campos presentes en este proyecto. Los campos que toca este proyecto son, en mayor o menor medida, la programación, el control de sistemas, la electrónica, la comunicación y la automatización.
- Realizar una comprobación en los componentes usados para verificar que tienen el *firmware* que se presupone que tienen.
- La importancia de aplicar correctamente la ingeniería inversa. Es decir, buscar, de forma adecuada, información relevante para tu proyecto y, también, adaptar correctamente el trabajo ya realizado por otras personas a las necesidades requeridas.
- Los depuradores de código presentes en los IDEs son una herramienta excelente para localización de errores difíciles de situar en el código.
- La importancia de acotar los problemas que van surgiendo durante la realización de un proyecto. En otras palabras, ir descartando posibilidades para localizar dónde está realmente el problema. En este proyecto, por ejemplo, los problemas surgidos podrían estar relacionados con una mala programación, con una conexión defectuosa o con un error en la elección de los parámetros de configuración.

- El hecho de realizar un trabajo fin de grado a la orden del día dentro del mundo de la ingeniería, que sea interesante y con cierta relación con lo que busca una empresa, puede ser de gran utilidad a la hora de abrirte paso en el mundo laboral.

Mejoras

A continuación, se muestran algunas de las mejoras y futuras líneas de desarrollo del presente proyecto:

- El principal progreso, partiendo de lo realizado hasta ahora, es seguir con el desarrollo del termostato inteligente como si se tratara del producto a comercializar por parte de una empresa. En este proyecto se ha conseguido obtener un *software* y una aplicación móvil totalmente funcionales, por lo que la parte *software* del producto estaría hecha, en su mayor parte. El producto quedaría completo con el desarrollo de una placa de circuito impreso que albergara las conexiones y componentes (pantalla LCD, pines, circuitos integrados, sensores…) necesarios para el funcionamiento del termostato inteligente. Además, habría que crear una envolvente o carcasa para albergar la placa. También habría que analizar los costes asociados a la fabricación del producto y establecer, claramente, los consumos relacionados con este.
- Se puede realizar una mejora y simplificación del código desarrollado en C, en el STM32F429I-DISC1, y en Kotlin, en la aplicación móvil.
- Gracias al uso de Firebase en la aplicación móvil podemos llegar a manejar una gran cantidad de datos generados por los usuarios de la aplicación HomeT. Estos datos son muy codiciados por las empresas, ya que dan información muy valiosa para la mejora de la propia aplicación y para conocer a los clientes de esta. Algunos datos que se pueden manejar son el país del usuario, las horas de activación de la caldera, los programas diarios y temperaturas más solicitadas, el número de usuarios activos en cada hora, etc. Por lo que continuar este proyecto desarrollando los aspectos relacionados con los campos del almacenamiento y gestión de datos puede ser interesante.

- Para completar la seguridad del termostato inteligente habría que desarrollar un proceso de "enlace" entre la aplicación móvil y el termostato. Este tipo de procesos se basan en seguir los siguientes pasos:
    1. Descargar la aplicación en nuestro teléfono móvil, registrar una cuenta y acceder, con ella, a la aplicación.
    2. Una vez dentro, usando la opción de Añadir un dispositivo, buscamos nuestro termostato y lo seleccionamos.
    3. Acudimos al termostato y presionamos un botón en particular, normalmente indicado por la aplicación. Esto crea una red wifi temporal en la que solo está el termostato y, también, alerta a la aplicación de la presencia del dispositivo.
    4. En la aplicación móvil confirmamos el termostato seleccionado. A continuación, seleccionamos la red wifi del domicilio e introducimos la contraseña. El termostato sale de la red wifi temporal y se conecta a la red wifi que hemos seleccionada. Con esto, la aplicación y el termostato quedan conectados.
    5. Además, para tener una mayor seguridad, se puede añadir una clave identificativa en las cabeceras de las peticiones HTTP que llegan desde la aplicación móvil.
- Ahora, que se tiene un dispositivo inteligente completamente funcional y, por tanto, se conoce el proceso de desarrollo de un sistema domótico, podemos expandir el proyecto a otras partes del hogar. En otras palabras, se puede usar el microcontrolador del proyecto como una unidad central para el control de la domótica de la casa. Añadiendo los sensores pertinentes y el código de programación necesario podemos llegar a controlar la subida y bajada de las persianas de la vivienda, el riego del jardín exterior o el aire acondicionado de la casa.
- Añadir, mediante programación, la posibilidad de modificar la histéresis del termostato y del reloj de tiempo real, tanto en el dispositivo físico como en la aplicación.

Bibliografía

ada, l. (29 de Julio de 2012). *adafruit*. Recuperado el 8 de Julio de 2022, de adafruit Web site: https://learn.adafruit.com/tmp36-temperature-sensor/

Advanced Monolithic Systems. (s.f.). *Datasheet AMS1117.* Dublin.

AliExpress. (s.f.). *AliExpress*. Recuperado el 26 de Junio de 2022, de AliExpress Web site: https://es.aliexpress.com/item/32754485004.html?priceBeautifyAB=0

Amazon. (s.f.). *Amazon*. Recuperado el 24 de Junio de 2022, de Amazon Web site: https://www.amazon.com/Amazon-Smart-Thermostat/dp/B08J4C8871

ANALOG DEVICES. (2010). *Datasheet Low Voltage Temperature Sensors TMP35/TMP36/TMP37.*

CASADOMO. (1 de Febrero de 2018). *CASADOMO*. Recuperado el 20 de Junio de 2022, de CASADOMO Web site: https://www.casadomo.com/2018/02/01/mercado-domotica-inmotica-continua-creciendo-ultimo-estudio-cedom

Emerson. (s.f.). *Emerson*. Recuperado el 24 de Junio de 2022, de Emerson Web site: https://sensi.emerson.com/en-us/shop/sensi/products/sensi-touch-smart-thermostat

Espressif Systems. (2021). *ESP8266 Non_OS AT Instruction Set.*

Espressif Systems. (2022). *Datasheet ESP8266EX.*

Google. (s.f.). *Store.Google*. Recuperado el 24 de Junio de 2022, de Store.Google Web site: https://store.google.com/es/product/nest_learning_thermostat_3rd_gen?hl=es

Gúzman Navarro, F. (2015). *Domótica.* Madrid: RA-MA Editorial.

Heschen. (s.f.). *Heschen*. Recuperado el 12 de Julio de 2022, de Heschen Site web: https://heschen.com/es/products/heschen-srd-03vdc-sl-c

Hunter Adams, V. (s.f.). *vanhunteradams*. Recuperado el 16 de Julio de 2022, de https://vanhunteradams.com/Protocols/UART/UART.html

Lázaro, D. (2018). *diego*. Recuperado el 13 de Noviembre de 2022, de diego Web site: https://diego.com.es/respuesta-http

Mozilla. (13 de Octubre de 2022a). *MDN Web Docs*. Recuperado el 12 de Noviembre de 2022, de MDN Web Docs Web site: https://developer.mozilla.org/es/docs/Web/HTTP

Mozilla. (7 de Noviembre de 2022b). *MDN Web Docs*. Recuperado el 12 de Noviembre de 2022, de MDN Web Docs Web site:

https://developer.mozilla.org/es/docs/Learn/Common_questions/What_is_a_URL

ON Semiconductor. (2013). *Datasheet 2N2222A.*

Pérez Turiel, J. (2021). *Apuntes de Control y Comunicaciones Industriales.* Valladolid.

PHILIPS. (1997). *Datasheet 2N2222; 2N2222A.*

Portaltic/EP. (19 de Diciembre de 2020). *europapress*. Recuperado el 20 de Junio de 2022, de europapress Web site: https://www.europapress.es/portaltic/sector/noticia-domotica-crecera-300-espana-2024-expertos-dan-claves-crear-hogar-inteligente-20201219103037.html

SONGLE RELAY. (s.f.). *Datasheet SRD-03VDC-SL-C.*

ST. (2018). *Datasheet STM32F427xx STM32F429xx.*

ST. (2020). *User manual Discovery kit with STM32F429ZI MCU.*

WAVESHARE. (s.f.). *WAVESHARE*. Recuperado el 14 de Julio de 2022, de WAVESHARE Web site: https://www.waveshare.com/esp-12s.htm

Wikipedia. (s.f.). *Wikipedia*. Recuperado el 14 de Julio de 2022, de Wikipedia Web site: https://en.wikipedia.org/wiki/ESP8266