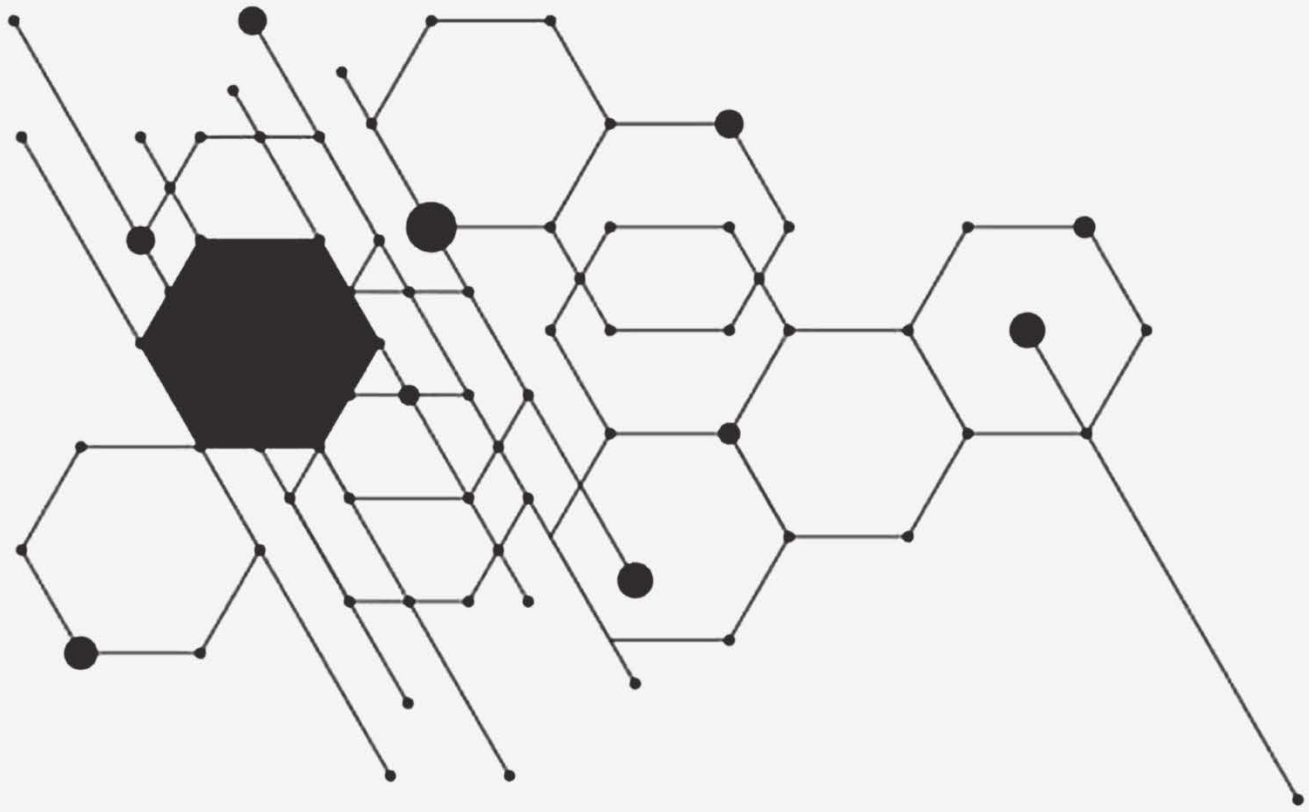
Trabajo de Fin de Grado

Xana: Prototipo de asistente domótico controlado por voz

Autor: Jorge Álvarez Pedrón

Tutor: Leopoldo Armesto Ángel

Curso 2019 ~ 2020



Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A mi hermano Víctor, por todo el amor y alegría que siempre nos ha dado, sobre todo en los momentos más difíciles. Estés donde estés, nunca te olvidaremos.

Agradecimientos

A mi madre Encarna, por su inmensurable amor y apoyo incondicionales y por su dedicación a sus seres queridos.

A mis amigos: Carlos, Clara, María, Sergio y los demás, por su apoyo moral siempre que lo he necesitado, por su ayuda en este proyecto, por hacerme mejorar como persona y, sobre todo, por su enorme paciencia conmigo. Y a mis amigos y compañeros de la carrera, sobre todo a Edu, Eva y los demás, por ayudarme y soportarme durante cuatro largos años.

A mi hermano Vicente y mi cuñada Rosa, por su amor y sus tartas de tres chocolates. A José Soler y a Jordi Enguídanos, por su interés y recomendaciones sobre aspectos técnicos que me han sacado de más de un atasco.

Y a todas las personas que me han apoyado durante estos cuatro años de carrera. Gracias por tanto.

Índice general de documentos

Anotación

Puesto que los distintos productos diseñados en este proyecto están pensados para actuar como prototipo y no como un modelo comercial, no se diseñarán sus placas de circuito impreso ni su carcasa exterior y se ha considerado innecesario incluir un pliego de condiciones en el proyecto. Toda la información necesaria sobre el montaje está disponible en los planos.

Resumen

La domótica se define como un conjunto de sistemas ubicuos en una misma vivienda y conectados entre sí capaces de aportar eficiencia energética, confort, accesibilidad, seguridad y comunicación mediante la automatización de los elementos del hogar. Este proyecto aborda el diseño de un prototipo de asistente domótico controlado por voz llamado Xana. El controlador podrá recibir instrucciones de voz precisas para navegar por distintos menús de voz y ejecutar 5 de las funcionalidades básicas para un hogar automatizado: control de luces, de persianas, de temperatura y de música y activación y desactivación de enchufes. Cada una de estas 6 funcionalidades (control por voz y 5 actuadores) constituirá un módulo independiente conectado a los demás vía wifi gracias al microcontrolador ESP32 de Espressif Systems, excepto el control de música, que estará integrado en el controlador principal.

Este prototipo es ampliable a funciones como reproducción de música vía Bluetooth y programación de eventos o alarmas que controlen cualquiera de las anteriores características, entre otras. La finalidad de Xana es crear un punto de partida funcional para un posible futuro proyecto comercial real.

La parte de software de Xana se desarrollará en lenguaje C++ utilizando el entorno de desarrollo Visual Studio Code de Microsoft con las librerías de Arduino. El hardware principal se testeará en una placa de prototipos o “protoboard” y más adelante se diseñará su circuito para PCB en el software Proteus Design Suite.

Este proyecto supone un avance en el campo de la domótica debido a la accesibilidad que proporciona su arquitectura de bajo coste y su compatibilidad con elementos comunes del hogar no necesariamente compatibles con el IoT.

Palabras clave: Domótica, casa domótica, casa inteligente, IoT, internet de las cosas, automatización, reconocimiento de voz, ESP32, sistema domótico, controlador domótico, asistente de voz, Xana, wifi, iluminación RGB, LED RGB, mando IR, Arduino, motor de persiana.

Resum

La domòtica es defineix como a un conjunt de sistemes omnipresents a un mateix habitatge i connectats entre sí capaços d'aportar eficiència energètica, confort, accessibilitat, seguretat i comunicació mitjançant la automatització dels elements del llar. Aquest projecte aborda el disseny d'un prototip d'assistent domòtic controlat per veu, anomenat Xana. El controlador serà capaç de rebre instruccions de veu precises per a navegar a través de diferents menús de veu i executar 5 de les característiques bàsiques d'un habitatge automatitzat: control de llum, persianes, temperatura i de música i activació i desactivació d'endolls. Cadascuna d'aquestes 6 funcionalitats (control de veu i 5 actuadors) constituirà un mòdul separat connectat als altres via wifi gràcies al microcontrolador ESP32 d'Espressif Systems, a excepció del control de música, que formarà part del controlador principal.

Aquest prototip és ampliable a funcionalitats com la reproducció de música per Bluetooth i la programació de rutines o alarmes que controlen qualsevol de les característiques anteriors, entre d'altres. El propòsit de Xana és crear un punt de partida funcional per a un possible futur projecte comercial real.

La part de software de Xana es desenvoluparà en llenguatge de C++ utilitzant l'entorn de desenvolupament Visual Studio Code de Microsoft amb llibreries d'Arduino. El hardware principal es provarà en una placa de prototipus o "protoboard" i més tard es dissenyarà el seu circuit per a PCB al programa Proteus Design Suite.

Aquest projecte representa un avenç en el camp de la domòtica per raó de l'accessibilitat que proporciona la seua arquitectura de baix cost i la seva compatibilitat amb elements comuns de la llar no necessàriament compatibles amb IoT.

Paraules clau: Domòtica, llar domòtic, llar intel·ligent, automatització, IoT, internet de les coses, reconeixement de veu, ESP32, sistema domòtic, controlador domòtic, assistent de veu, Xana, wifi, il·luminació RGB, LED RGB, comandament IR, Arduino, motor de cecs.

Abstract

Home automation can be defined as a group of ubiquitous systems in the same house, interconnected to each other and able to provide energy efficiency, comfort, accessibility, security and communication by means of home elements automation. This project addresses the design of a prototype for a voice-controlled home automation assistant called Xana. The controller will be able to receive precise voice instructions to navigate through different voice menus and execute 5 of the basic features for an automated home: light, blinds, temperature and music control and plug switching. Each of these 6 features (voice control and the 5 actuators) will constitute an independent module connected to the others via Wi-Fi using the Espressif Systems' ESP32 microcontroller, except the music player, wich will be integrated into the main controller.

This prototype could be expanded to serve for functions such as music playing via Bluetooth or an SD card and scheduling of events or alarms that control any of the above features, among others. The purpose of Xana is to create a functional starting point for a possible future real business project.

Xana’s software will be developed in C++ language with Arduino libraries using Microsoft's Visual Studio Code IDE. The main hardware will be tested on a protoboard, and afterwards the PCB will be designed in the Proteus Design Suite software.

This project is a breakthrough in the field of home automation due to the accessibility provided by its low-cost architecture and its compatibility with common household elements not necessarily compatible with the IoT.

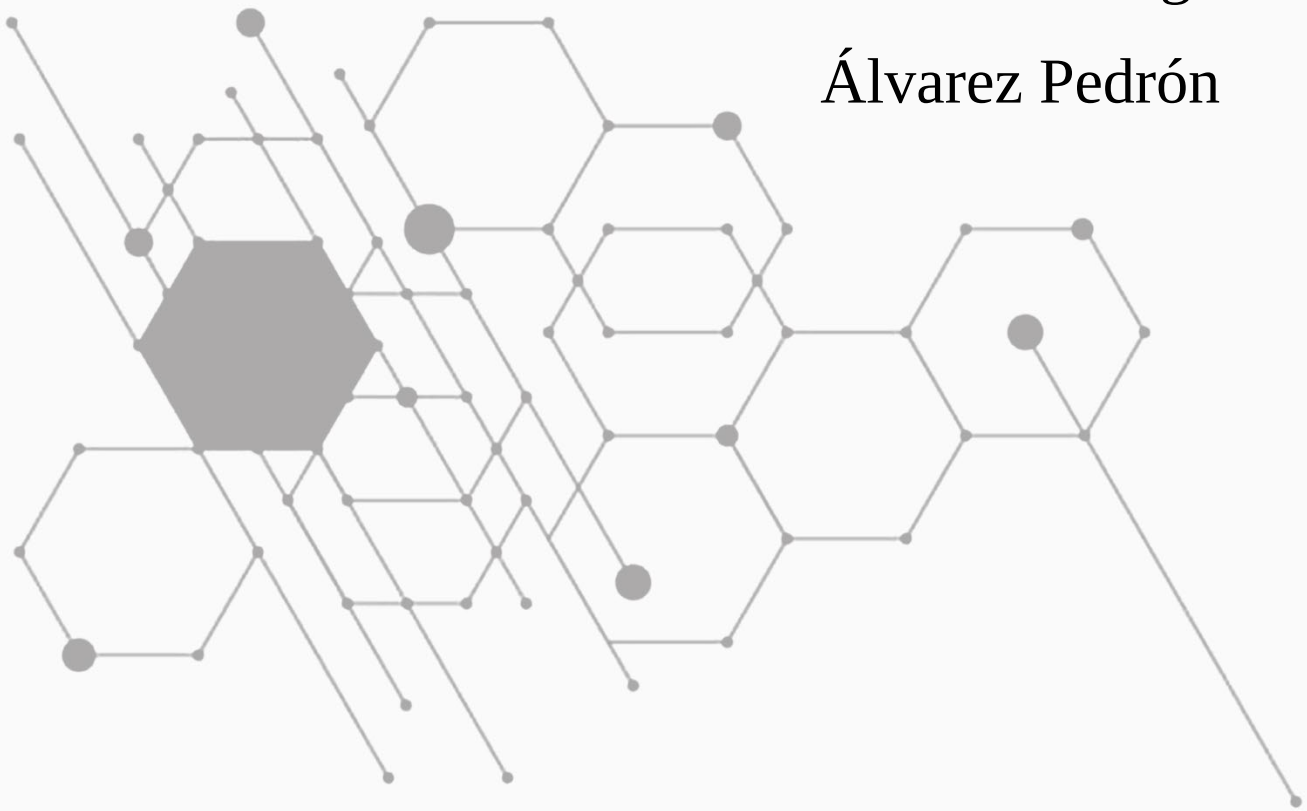
Key words: Domotics, domotic house, smart home, IoT, Internet of Things, home automation, voice recognition, ESP32, home automation controller, voice assistant, Xana, Wi-Fi, RGB lighting, RGB LED, IR remote, Arduino, blind motor.

Documento 1: Memoria

Xana: Prototipo de asistente domótico controlado por voz

Trabajo de Fin de Grado

Autor: Jorge Álvarez Pedrón



Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice de la memoria

Índice de imágenes

Índice de tablas

1. Justificación y contexto

1.1. Introducción a la domótica

La domótica se define como un conjunto de sistemas ubicuos en una misma vivienda y conectados entre sí capaces de aportar eficiencia energética, confort, accesibilidad, seguridad y comunicación mediante la automatización de los elementos del hogar. El término está formado por “domo-” (que deriva de “domus”, hogar en latín) y “-tica” (del griego, “que funciona por sí solo”). Así pues, encontramos variantes de esta tecnología como la inmótica (domótica aplicada a inmuebles de mayor envergadura no necesariamente residenciales, como hoteles, escuelas, hospitales, etc.) o urbótica (domótica aplicada espacios urbanos, como barrios y ciudades), pero todas ofrecen las mismas características en espacios diferentes.

Las primeras instalaciones inmóticas aparecen en Escocia a finales de los años 70 con el objetivo de controlar los sistemas de calefacción de una manera eficiente y automática. Utilizaban el protocolo de comunicaciones X-10, todavía en uso en la actualidad, consistente en acoplar señales de control a la red eléctrica de 50 ó 60 Hz. Con la popularización de los ordenadores en las oficinas, en la década de 1990 se comenzaron a instalar Sistemas de Cableado Estructurado (SCE), que creaban una red de área local (LAN, por sus siglas en inglés) con conectores estándar mediante la cual era posible la comunicación entre ordenadores y otros dispositivos compatibles, como sensores, luces, altavoces, sistemas de calefacción, cámaras, etc. Estos eran los primeros edificios inteligentes.

Con la evolución de las tecnologías y la popularización de las comunicaciones inalámbricas, sobre todo las de radiofrecuencia como el wifi y el Bluetooth, los dispositivos electrónicos se han ido volviendo cada vez más inteligentes, completos y complejos y capaces de comunicarse entre ellos. Esto ha dado lugar a que, en la actualidad, toda la información esté gobernada por el “internet de las cosas”, o IoT por sus siglas en inglés. Este término hace referencia a la interconexión de cada vez más objetos cotidianos (teléfonos móviles, relojes, etc.) mediante internet con el fin de intercambiar información y aportar un grado de automatización a la vida del usuario. Solo en los últimos cinco años, se ha duplicado la cantidad de dispositivos conectados a internet, de quince a treinta billones, y se espera que antes de 2025 se vuelva a duplicar.

Esta integración de la tecnología y la automatización en nuestra vida cotidiana abre las puertas a la domótica facilitando en gran medida procesos como la transmisión de datos, la sensorización de variables físicas, el entrenamiento de las inteligencias artificiales o el accionamiento de todo tipo de actuadores y eventos. Todo esto augura una evolución e integración tecnológica y social que abre de par en par las puertas a la experimentación y desarrollo de nuevos sistemas domóticos más cercanos y accesibles para el usuario medio.

1.2. Características de la domótica

Como indica su definición, la domótica contribuye a mejorar la calidad de vida dentro del hogar aportando sus cinco características fundamentales: confort, eficiencia energética, accesibilidad, seguridad y comunicaciones. Todo esto lo hace a través del control de los electrodomésticos inteligentes, como bombillas, televisores, neveras o sistemas de calefacción y de elementos de control, como pueden ser rutinas preprogramadas o sensores de luz, temperatura, etc.

Cuando hablamos de **confort**, hacemos referencia a la automatización de todas aquellas acciones que suponen un trabajo evitable para el usuario: creación de rutinas automáticas como subir las persianas o encender la cafetera a la misma hora que suena el despertador, auto calibración de la temperatura, control de elementos con la voz, etc.

La **eficiencia energética** hace referencia a la gestión inteligente de los recursos energéticos básicos: agua y electricidad. Esto se lleva a cabo mediante el control de la intensidad de la luz con sensores de iluminación o de presencia, riego automático, aprovechamiento del uso eléctrico en las franjas horarias de menor coste o incluso con la monitorización de los hábitos de consumo y creación de rutinas más eficientes.

También aporta al usuario una **accesibilidad** extra al control de todos los elementos del hogar, lo que, además de contribuir al confort, es especialmente beneficioso para personas con problemas de movilidad reducida, visión, etc.

La monitorización de diversos factores mediante sensores contribuye a la **seguridad** del hogar de tres maneras distintas. La primera es la prevención mediante un control automatizado seguro de sus actuadores y sistemas como cerraduras inteligentes. La segunda es la alarma, pudiendo detectar incluso con antelación factores que puedan poner en riesgo la integridad de los elementos del hogar o de sus habitantes. La tercera forma de protección son los protocolos de actuación, como avisos automáticos a servicios de emergencia (policía, bomberos, etc.), detención de la alimentación eléctrica controlada, etc.

Por último, un hogar inteligente debe facilitar la **comunicación** entre todos sus elementos inteligentes y entre el usuario y estos elementos, haciendo posible un control cómodo y sencillo del hogar por parte del usuario.

Como contrapartida, ha de tenerse en cuenta que, para disponer de estas características, debe realizarse una inversión económica inicial, así como la correcta instalación del sistema domótico, y un mantenimiento constante (alimentación, conexión a internet si la hubiera, actualizaciones de hardware y software, reparaciones, recalibraciones o reemplazos de elementos, etc.).

1.3. Arquitectura de un sistema domótico

Un sistema domótico consta de cinco partes fundamentales:

- **Sensores**, mediante los cuales se monitorizan las variables físicas (luz, humedad, temperatura...) o instrucciones del usuario (pulsadores, micrófonos...).
- **Actuadores**, que ejecutan la acción final tras recibir la orden controlando así los elementos del hogar para proporcionar confort, eficiencia energética, etc. (son los motores, relés, sistemas de calefacción...).
- Una o varias **unidades de control**, que procesan los datos recibidos de los sensores y crean y almacenan rutinas para emitir órdenes a los actuadores. Pueden ser microcontroladores, ordenadores, PLC, etc.
- **Buses** de datos, por los que circula toda la información entre las partes anteriores y gracias a la cual funcionan de forma sincronizada. Pueden ser alámbricos (cables ethernet, par trenzado...) o inalámbricos (redes de radiofrecuencia, señales de infrarrojos...).
- Una o varias **interfaces** de comunicación. Este término hace referencia tanto a los protocolos que utilizan los medios para codificar la información y comunicarse entre sí (SPI, UART, X-10...) como a los elementos audiovisuales que sirven de medio de interacción entre éstos y el usuario (pantallas, altavoces, etc.).

En función de cómo sea la distribución de las unidades de control, o, dicho de otro modo, de "dónde reside la inteligencia del sistema", los sistemas domóticos se clasifican en varias arquitecturas. Las principales arquitecturas que se suelen encontrar son:

- **Arquitectura centralizada:** un único procesador central recibe la información de todos los sensores e interfaces y, una vez procesada, envía las órdenes directamente a todos los actuadores. Es un sistema de bajo coste, pero requiere un procesador que pueda soportar toda la carga de trabajo de forma eficiente, y si éste falla, falla todo el sistema.
- **Arquitectura distribuida:** es el caso opuesto a la arquitectura centralizada, en el que la inteligencia se reparte entre varios módulos, cada uno asociado a unos sensores y actuadores específicos y que se comunican entre sí. Es un sistema más seguro, ya que los módulos pueden gestionar sus entradas y salidas de manera independiente, pero es más propenso a los fallos, ya que, al no estar jerarquizado, en el caso de un conflicto de intereses entre varios módulos, los resultados pueden ser inesperados o indeseados.
- **Arquitectura híbrida:** también llamada arquitectura mixta, es aquella que combina uno o varios controladores centrales de manera jerarquizada y módulos de control distribuidos. Es el caso más complejo, pero a su vez, el más seguro y eficiente, ya que maximiza la velocidad y calidad de cómputo mientras que minimiza la probabilidad de un fallo general y errores inesperados.

Otra manera de clasificar los sistemas domóticos es en función de cómo comparten la información según el tipo de red de datos que forman, es decir, de cómo están conectados de forma física (o inalámbrica) sus buses. Además de los cinco tipos ilustrados en la *imagen 1*, existen muchas redes que usan formatos de comunicación híbridos que combinan dos o más de estos tipos.

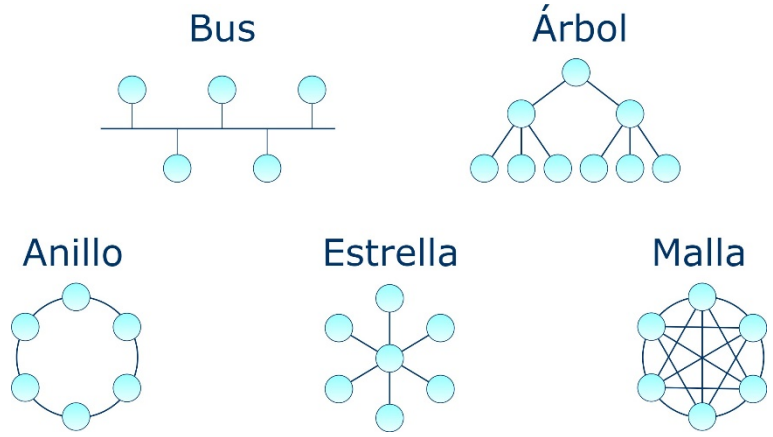


***Imagen 1 - Tipos de redes de comunicación**. Los círculos representan elementos reales como sensores o controladores y las líneas representan el flujo de información, que puede ser unidireccional o bidireccional. La diferencia entre bus y malla reside en que un dispositivo en una red bus, manda y recibe información a todos los dispositivos de la red, mientras que, en una red en malla, el dispositivo emisor elige con quien comunicarse.*

Una última forma de clasificación de los sistemas domóticos es según si sus estándares de comunicación (protocolos de transmisión de datos) son abiertos o cerrados.

- Llamamos **estándares abiertos** a aquellos que define una empresa, organización o conjunto de éstas y que pretenden ser un estándar para facilitar la interconexión de elementos de distintas marcas y diseñadores y favorecer así la integración tecnológica. Un ejemplo de estos estándares puede ser el lenguaje de marcado HTML o protocolos de comunicación como el I2C o el I2S para audio.
- Por el contrario, los **estándares cerrados** son aquellos desarrollados por empresas de manera privada y sujetos a algún tipo de patente para beneficiarse de cierta exclusividad. Esta exclusividad, aporta a la empresa beneficios económicos a través de patentes, beneficios de mercado debidos a la no compatibilidad con fabricantes externos y seguridad ya que se previene así la ingeniería inversa a sus productos por parte de sus competidores de mercado. Ejemplos de este tipo de estándares pueden ser el conector *lighting* de Apple o el formato de ficheros de texto ".docx" de Microsoft.

Cabe mencionar al *Institute of Electrical and Electronics Engineers* (IEEE, leído en español como *i-e-cubo* o *i-triple-e*), una asociación mundial sin ánimo de lucro dedicada al desarrollo, difusión e integración de este tipo de estándares abiertos, pudiendo destacar los estándares de conexión *Ethernet* (*IEEE 802.3*) y de la comunicación wifi (*IEEE 802.11*).

1.4. Mercado actual de la domótica

Existen muchas maneras posibles de domotizar un hogar, pero en el mercado actual destacan mayoritariamente dos tipos de ecosistemas: los controladores domóticos locales y los asistentes de voz comerciales que controlan elementos inteligentes compatibles a través de internet. Ambas posibilidades utilizan arquitecturas híbridas, aunque la primera es mucho más centralizada mientras la segunda es casi distribuida.

1.4.1. Controladores domóticos

También llamados centrales domóticas, son dispositivos del tamaño de un smartphone que pueden programarse y recibir actualizaciones a través de un ordenador o de una aplicación vía wifi. Están diseñados para automatizar procesos y rutinas en función de los sensores y actuadores compatibles, con los que se conectan generalmente mediante protocolos de radiofrecuencia de baja potencia similares al wifi.

Actualmente, los **protocolos** de comunicación más utilizados por controladores son el X-10, nombrado en el punto 1.1 de este documento, y KNX, ambos siendo redes tipo bus y con estándares abiertos. Dada la sensibilidad a interferencias eléctricas que sufre el X-10, cada vez se utiliza más KNX en las instalaciones domóticas e inmóticas, que, aunque generalmente utilice una conexión de par trenzado, también puede transmitirse por ethernet o radiofrecuencia además de mediante la red eléctrica. Otra ventaja es que, al tratarse de redes de tipo bus, pueden crearse sistemas de arquitecturas distribuidas sin necesidad de instalar un controlador central.

La principal **ventaja** de un controlador es su polivalencia, ya que puede configurarse y programarse como si fuera un ordenador, sin límite aparente de acciones de control. Además, es un sistema robusto en cuanto a seguridad e independiente, ya que no depende ni comparte información con empresas privadas ni requiere conexión a internet para realizar sus labores. No al mismo nivel en este tipo de instalaciones, pero también están muy extendidos protocolos de radiofrecuencia como Z-wave de los que se hablará en el siguiente punto. También cabe destacar el protocolo abierto de tipo de red híbrida LonWorks para las instalaciones inmóticas, urbóticas e industriales.

Los principales **fabricantes** del mercado son, desde un punto de vista más conservador, *TYDOM*, *Eedomus* y *Vera*, aunque empiezan a aparecer de manera muy notable ecosistemas más personalizables y de bajo coste similares a los asistentes de voz del apartado siguiente. Por lo general, no poseen asistentes de voz, pero pueden controlarse y personalizarse mediante aplicaciones para teléfonos móviles. Además, los productos compatibles suelen ser de la misma marca que la desarrolladora. Estos ecosistemas son productos de las grandes empresas de tecnología y ocio como Samsung, Xiaomi o LG.

1.4.2. Asistentes de voz

Un asistente de voz es una inteligencia artificial o conjunto de algoritmos capaz de sintetizar la voz natural de un ser humano a instrucciones precisas en tiempo real, generalmente a través de internet. Su intención principal es permitir al usuario interactuar con el asistente como si lo hiciera con otro ser humano. Cuentan así, además de con un sintetizador de voz a texto (STT, por sus siglas en inglés) para analizar las instrucciones, con un sintetizador de texto a voz (TTS, por sus siglas en inglés) para que la comunicación pueda ser bidireccional y responder a las peticiones del usuario.

Además de sus **funciones** meramente domóticas, al ser asistentes de voz y tener conexión a internet, también realizan funciones de organización, como gestión de calendarios, alarmas y recordatorios; administración de llamadas telefónicas, búsqueda de consultas a través de internet, funciones musicales, etc. Suponen así una mejora a la accesibilidad sobre todo para personas con movilidad reducida y mayor confort para el usuario medio.

El **origen** de estos asistentes, incluso previo al de la domótica, consta de 1946, cuando, con fines militares, se creó una inteligencia artificial llamada CALO (*Cognitive Assistant that Learns and Organizes*), pero no desarrolló capacidad de control por voz hasta 2003, convirtiéndose en la precursora de Siri, el asistente de Apple. Fue en 1952 cuando la compañía Bell Labs de Nokia creó un marcador por voz, que reconocía los números del 0 al 9 comparándolos con ficheros de voz previamente grabados en su memoria. Esta tecnología fue evolucionando, sobre todo bien entrado el Siglo XXI, hasta convertirse en asistentes funcionales para teléfonos móviles y ordenadores capaces de realizar las funciones citadas en el anterior párrafo.

En 2014, Amazon creó el asistente virtual **Alexa**, el primero diseñado para controlar un hogar mediante la voz. A éste, se fueron sumando sus principales competidores del mercado actual: **Google Home** y Apple Homekit, este último con una menor incisión en el panorama domótico actual.

El reconocimiento de voz, así como la búsqueda de información o la instalación de actualizaciones, se realiza a través de internet en sus altavoces inteligentes. La comunicación con los actuadores domóticos (bombillas, enchufes, etc.) puede realizarse mediante varios **protocolos** inalámbricos distintos, todos basados en la radiofrecuencia:

- **Wifi**: Abreviatura del término inglés “Wireless Fidelity”, esta tecnología se basa en el protocolo IEEE 802.11. Funciona a 2,4 GHz en España y se basa en un router que actúa como punto de acceso entre internet y otros dispositivos, o entre los propios dispositivos. Utiliza por lo tanto una red en forma de estrella, en la que los dispositivos se conectan al router a través de una “Gateway” o puerta de enlace que éste expone. Ofrece un rango de conexión de un máximo de 20 metros y está pensado para la transmisión de datos de gran tamaño. Soporta una cantidad limitada de aproximadamente 15 dispositivos conectados, a partir de la cual comienza a dar fallos de conexión.

- **Zigbee y Z-Wave**: Son dos protocolos de bajo consumo pensados para la transmisión de instrucciones y no de datos, lo que permite mayores rangos y velocidades con mayor eficiencia energética. Zigbee está basado en el estándar abierto IEEE 802.15.4 de Zigbee Alliance, por lo que está más extendido a nivel de mercado, mientras que Z-Wave es un protocolo cerrado propiedad de Silicon Labs. Funcionan mediante una red en forma de malla, donde la información va saltando entre dispositivos hasta llegar un concentrador o "hub". Este concentrador puede conectarse a un router para obtener conexión a internet ocupando una sola puerta de enlace, lo que soluciona los problemas de conexión con un número elevado de dispositivos conectados. La velocidad de transmisión es de 868 MHz en Europa y Asia y 915MHz en América para ambos protocolos, entre otras, aunque Zigbee puede utilizar 2.4 GHz a nivel global, lo que lo hace más compatible geográficamente y con otras tecnologías como el wifi.

La siguiente tabla, *tabla 1*, muestra la comparativa entre las características principales de estos tres protocolos, que supondrán unas ventajas y desventajas respecto a las demás en cada ámbito de aplicación:

	Wi Fi	ZigBee	Z-WAVE
Tipo de estándar	Abierto	Abierto	Cerrado
Tipo de red	Estrella	Malla	Malla
Internet	Sí	Requiere un router	Requiere un router
Velocidad de red	2,4 / 5 / 24 GHz	868 / 915 MHz / 2,4 GHz	868 / 915 MHz
Dispositivos conectados	Hasta 15	Hasta 65.535	Hasta 232
Velocidad de transmisión	Hasta 60 Mbps	40 - 250 kbps	9,6 - 100 kbps
Saltos máximos entre nodos	1	Sin límites	4
Distancia máxima entre nodos	20 m	10 m	30 m
Vulnerabilidad a interferencias	Media	Media	Baja
Consumo de energía	Bajo	Muy bajo	Muy bajo

***Tabla 1 - Comparativa entre los distintos protocolos de comunicación** utilizados por asistentes de voz en domótica.*

Cabe destacar en este apartado el protocolo abierto **LoRaWAN** (*Long Range Wide Area Network*), desarrollado por la asociación sin ánimo de lucro LoRa Alliance. Su finalidad es conectar todos los sensores y actuadores del mundo de manera segura a través del internet de las cosas mediante la red de estrella LoRa, ilustrada en la *imagen 2*.

Este protocolo de bajo coste, pretende ser el estándar de la urbótica a nivel global, así como de la inmótica y la domótica, ofreciendo una red de nodos de bajo coste, bajo consumo (hasta 10 años de uso con una sola batería), largo alcance (hasta 20 km entre nodos), seguro y con alta tolerancia a las interferencias.

Su principal competidor de pago es **Sigfox**, una empresa con las mismas aspiraciones y un protocolo cerrado, el cual requiere de una suscripción anual para conectarse a la red.

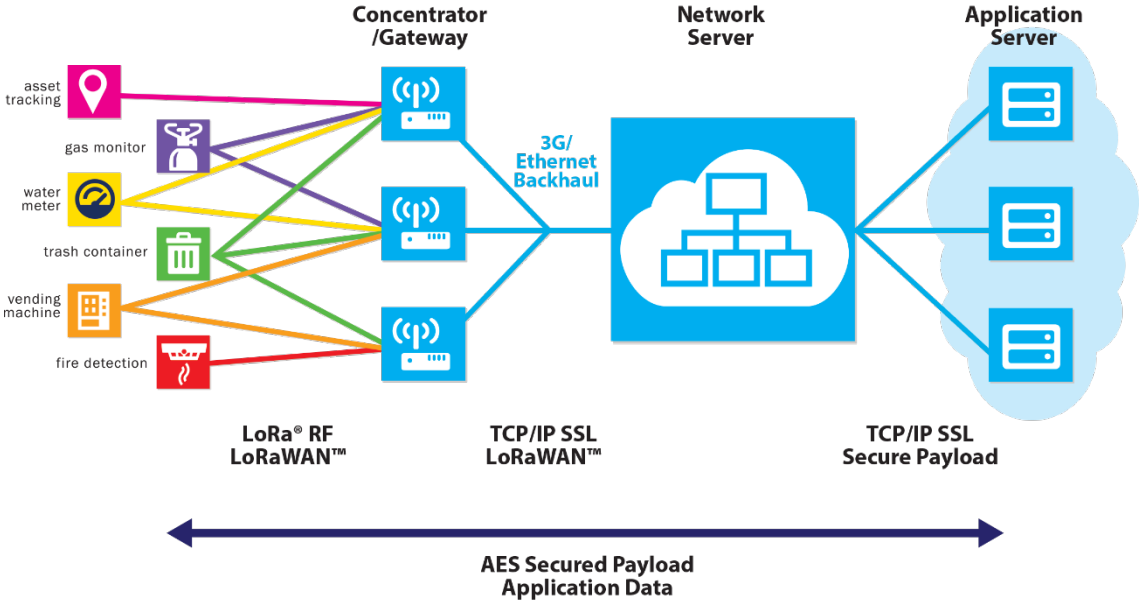


Imagen 2 - Red de comunicaciones LoRa *para la conexión global de todos los dispositivos a través del IoT.*

1.5. Seguridad

Dada la accesibilidad que aportan gracias a su gran compatibilidad con fabricantes de terceros, la posibilidad del control por voz y sus aplicaciones para móvil, solo los altavoces inteligentes de Google y Amazon suman casi el 90% de la c**uota de mercado** de controladores y no está previsto que otras marcas como Xiaomi y Apple ganen terreno en los próximos años, según un estudio de mercado de Voicebot.

El **funcionamiento** de estos altavoces inteligentes se basa en la escucha continua, esto es, sintetizan todo el rato el audio que captan para detectar voces humanas y las transforman en texto, que a su vez interpretan las inteligencias artificiales para beneficio de su aprendizaje automático (más conocido con el término inglés "*machine learning*").

Estos datos, que son tratados de manera anónima, constituyen una información sobre el perfil de los usuarios muy importante para algunos mercados. Tal es esto, que ambas marcas ya han recibido multas y sido objeto de investigaciones por la venta de estos datos a empresas de terceros. Por este motivo, casi un tercio de los usuarios de asistentes de voz consideran un problema su falta de **privacidad**, según un estudio realizado por la consultoría SEIM a finales de 2019.

Al margen de la gestión de datos por parte de las compañías de asistentes de voz, existen otros peligros para el usuario, como la falta de un identificador de usuario. Debido a esto, cualquiera puede hablarle a un altavoz inteligente y pedirle que realice compras o desbloquee cerraduras inteligentes sin consentimiento del propietario. Por la construcción de este tipo de aparatos, esto puede realizarse incluso de manera imperceptible a través de ultrasonidos, haces de luz ultravioleta que emulen la voz humana (como demostraron unos experimentos de la Universidad de Michigan y la Universidad de Electro-Comunicaciones de Japón) o instrucciones ocultas entre las frecuencias de mensajes de audio aparentemente inofensivos (como demostró un experimento de la Universidad de California en Berkeley).

Paralelamente a las vulnerabilidades que presenta el hardware, el **software** de estos sistemas también es sensible a ataques a través de la red, con los que pueden obtenerse datos personales y bancarios del usuario o controlar sus dispositivos de manera remota e indeseada. Hay que tener en cuenta pues que, a la par que la tecnología evoluciona para hacer más cómoda la vida de los usuarios, también aumenta la cantidad de datos que se exponen a ataques informáticos y los métodos utilizados para engañar a los protocolos de seguridad.

2. Objeto del proyecto

El presente proyecto pretende diseñar el prototipo de un controlador domótico funcional con control por voz llamado Xana y cuatro de sus actuadores. Se pretende que sea lo suficientemente funcional, de bajo coste y energéticamente eficiente como para que una versión mejorada pudiera resultar competitiva a nivel de mercado.

Dadas las ventajas de los asistentes virtuales (control domótico por voz, accesibilidad, etc.) y sus problemas de privacidad y seguridad, se pretende diseñar un controlador de bajo coste con las características principales de un asistente virtual. Éste será capaz de reconocer instrucciones de voz concretas relacionadas con el control de la luz, las persianas la temperatura y la música y con la activación y desactivación de enchufes.

Al tratarse de un controlador y no un asistente virtual, al menos en esta versión de prototipo, Xana no dispondrá de acceso a internet, por lo que carecerá de facilidades como búsquedas web, compatibilidad con asistentes como Google Home o Alexa o reproducción de música en streaming. Sí contará con un altavoz para guiar al usuario a través de sus opciones y confirmar las acciones de control, así como para reproducir música guardada de manera local a través de un módulo para tarjetas SD.

Pese a que no se implementará debido a la complejidad del proyecto, se añadirá a su menú de voz una opción para configurar una alarma o rutina. Esta opción podrá utilizarse más adelante implementando un reloj externo en tiempo real (RTC, por sus siglas en inglés) como el circuito integrado DS3231.

2.1. Controlador principal

El controlador principal, Xana, activará el reconocimiento de instrucciones tras detectar su nombre, que será la palabra de activación. Este método se usa en los asistentes virtuales para evitar detectar órdenes por error. Deberá reconocer instrucciones de voz concretas desde cualquier parte de la habitación en la que se encuentre (considerando una habitación de no más de 25 m^2 m de superficie).

Tendrá que procesar estas instrucciones y controlar el actuador correspondiente de manera remota en cualquier parte de la vivienda (considerando una vivienda de no más de 200 m^2 de superficie), así como permitir cierto grado de automatización de éstos. Estará dotado de una conexión externa que permita tanto visualizar el seguimiento de las instrucciones de voz como actualizar el software del dispositivo. Se asegurará la compatibilidad con futuras actualizaciones mediante un código modular y optimizado y memoria de programa suficiente.

2.2. Control de luces

Se diseñará un actuador capaz de controlar un sistema de iluminación como bombillas LED o tiras de LED de manera remota o manual. Este tipo de actuadores funcionan, por lo general a 12V, por lo que esta será la tensión de salida máxima del sistema.

Las variables a controlar serán el color (en formato RGB) y la intensidad (en porcentaje). Los LED de iluminación funcionan, por lo general, a 3 W con iluminación blanca al 100% de intensidad, por lo que habrá de asegurar 1W por cada canal de color.

2.3. Interruptor de enchufes

Se diseñará un sistema de 4 enchufes capaces de activarse y desactivarse de manera remota o manual. Cada uno de los enchufes deberá soportar una carga máxima de 2.000 W, suficiente para alimentar a la gran mayoría de electrodomésticos comunes.

2.4. Control de motor de persiana

Se diseñará un actuador capaz de controlar un motor eléctrico de corriente alterna como los utilizados para subir y bajar las persianas. Éste podrá subir y bajar las persianas de manera manual o remota proporcionando al motor la conmutación de dos líneas de fase y dos de neutro: una fija para el bobinado de trabajo y una que conmutará el controlador para controlar la dirección del bobinado de arranque.

El controlador deberá saber en qué posición se encuentra la persiana en cada momento para poder moverla hasta sus extremos o dejarla en alguna posición intermedia.

2.5. Control de temperatura

Se diseñará un actuador capaz de controlar un sistema de aire acondicionado copiando y emulando los comandos del mando de control original y accionándose de manera remota. Podrá realizar las funciones de encender y apagar, aumentar y disminuir la temperatura del sistema del sistema y activar y desactivar el modo "*swing*" de las paletas de difusión del aire o cualquier otra función que se le desee configurar.

2.6. Sistema de sonido

El controlador principal constará de un altavoz con una potencia suficiente para proyectar sonido de calidad en una habitación mediana y será capaz de reproducir música desde un lector de tarjetas microSD.

3. Planteamiento de soluciones alternativas y descripción de la solución adoptada

3.1. Tecnología de control

Se plantean tres modos distintos de automatizar un proceso, en este caso domótico: usar un PLC (controlador lógico programable, por sus siglas en inglés), un miniordenador o un microcontrolador. A continuación, se hablará de las ventajas y desventajas que suponen cada alternativa y se profundizará en la solución adoptada.

3.1.1. PLC

Un PLC o autómata programable es un tipo de computadora utilizada en la automatización de procesos industriales. Es capaz de controlar un elevado número de entradas y salidas y realizar procesos secuenciales con gran precisión y velocidad, lo que proporciona gran seguridad al sistema. Además, es un sistema muy expandible y permite acoplar gran cantidad de módulos con distintas funcionalidades.

Imagen 3 - Ejemplo de autómata programable *Modicon M241 del fabricante Schneider, que ronda un precio de 400€.*

Por otro lado, es un sistema caro, pudiendo costar de pocos cientos a miles de euros los de mayores prestaciones. Al estar enfocados a procesos cíclicos e industriales, son complejos de programar al nivel de realizar un asistente de voz y requieren de un software específico del fabricante con sus propios métodos de programación. Además, están pensados para actuadores de alta potencia, como sistemas neumáticos o motores de pares elevados.

Sería usual el uso de un PLC en un sistema centralizado o en uno híbrido con muchos nodos, cada uno controlando una elevada cantidad de entradas y salidas, y dada su arquitectura, requiere cables especiales y resultaría complejo y caro realizar un sistema de control inalámbrico. Por lo tanto, esta alternativa quedaría descartada.

3.1.2. Miniordenador

Un miniordenador u ordenador de placa reducida es una computadora de bajo coste y tamaño reducido enfocada a la creación de proyectos electrónicos complejos, como centros multimedia, ordenadores portátiles de bajas prestaciones, robots, etc.

Imagen 4 - Ejemplo de miniordenador *Raspberry Pi 3B, uno de los modelos más utilizados actualmente.*

Aunque los hay de bajo coste, como el Raspberry Pi Zero, la mayoría cuestan alrededor de 35€ y pueden ofrecer distintas prestaciones, como varios gigabytes de memoria RAM, procesadores de varios núcleos y gran capacidad, conectividad wifi y Bluetooth, etc., lo que los convierte en sistemas muy versátiles con gran potencia de cálculo. Todo esto, al igual que el caso del PLC, son características sobredimensionadas para el proyecto en el que nos encontramos.

Por otra parte, los miniordenadores funcionan con un sistema operativo generalmente basado en Linux, lo que limita las funcionalidades que puede ofrecer, complica ligeramente la programación y requiere un tiempo extra cada vez que se reinicia el dispositivo para que el sistema operativo se encienda.

3.1.3. Microcontrolador y entorno de desarrollo

Un microcontrolador es un circuito integrado programable capaz de realizar cálculos matemáticos simples, comunicarse de manera lógica, interpretar la información de sus pines de entrada y actuar en consecuencia de todo esto sobre sus pines de salida de manera lógica.

Por norma general, vienen integrados en un SoC (*System on Chip*) que los permiten funcionar de manera autónoma, casi como un ordenador, gracias a otros componentes como memorias RAM o Flash, módulos de comunicaciones, reloj, etc. Un solo µC puede ser la base de varios SoC completamente diferentes. Estos chips se utilizan en módulos que les añaden funcionalidades como sensores, controladores de frecuencia, antenas, DAC y ADC, etc. También pueden utilizarse en placas de desarrollo enfocadas a realizar pruebas y prototipos como las famosas placas de Arduino. Estas placas añaden facilidades como un controlador USB, reguladores de voltaje, más memoria de programa y pines compatibles con placas de prototipos, como se muestra en la *imagen 5*.

Imagen 5 - Ejemplo de SoC, módulo y placa de desarrollo *basados cada uno en el elemento anterior. El microcontrolador utilizado en este caso es un Tensilica L106.*

Pese a su limitada potencia de cálculo, son fáciles de programar con lenguajes de programación de alto nivel como C++ gracias a los ensambladores, que traducen estos lenguajes en código ejecutable por el microcontrolador. Esto los convierte en elementos muy versátiles, ya que las únicas limitaciones residen en el hardware, que puede ampliarse fácilmente mediante elementos externos. No disponen de sistema operativo, sino de un *bootloader*, un programa fijo en el microcontrolador que carga directamente el programa principal programado por el usuario. Además, debido a su arquitectura, son elementos muy baratos y energéticamente muy eficientes, lo que los convierte en los controladores ideales para aplicaciones electrónicas que no requieran de gran capacidad de computación, como en este caso es la domótica.

3.1.4. Plataforma del microcontrolador

Existe una gran diversidad de opciones para realizar un proyecto electrónico con un microcontrolador. A continuación, en la *tabla 2*, se realizará una comparativa entre los entornos y ecosistemas más utilizadas y se elegirá la plataforma que mejor se adopte al proyecto.

Antes de realizar la comparativa, se ha elegido el **lenguaje** Arduino para la programación del microcontrolador. Arduino es un lenguaje de programación basado en C++, homónimo al su software de desarrollo y a su hardware principal de aplicación. Esto se debe a la gran cantidad de información que hay en internet sobre su programación, proyectos, librerías, etc. y su compatibilidad con librerías de fabricantes de componentes. Todo esto hace que el control de GPIO sea mucho más sencillo que en C++ común.

Como **entorno de desarrollo** del software (IDE, por sus siglas en inglés), se utilizará el software gratuito Visual Studio Code y su gestor de proyectos Platformio, capaz de programar una inmensa variedad de microcontroladores en lenguaje Arduino y de gran ayuda a la hora de organizar proyectos con un gran número de módulos y librerías.

Familias:	Arduino	Espressif	STM32	Teensy	Microchip
Compatibilidad con las librerías de Arduino	Total	Casi total	Casi total	Casi total	Media
Aplicación	Está pensado para prototipos, aunque podría utilizarse en aplicaciones reales	Proyectos con wifi y Bluetooth de muy bajo consumo y altas prestaciones	Proyectos de altas prestaciones y fiabilidad y muy alta eficiencia energética	Prototipos de mayor precisión y prestaciones que Arduino. También podría utilizarse en aplicaciones reales	Proyectos de bajo coste y prestaciones y alta fiabilidad y eficiencia energética
Información en internet (foros, fabricantes, etc.)	Gran variedad	Gran variedad	Bastante	Bastante	Alguna
Compatibilidad con VS Code	Total	Total	Total	Parcial	Muy reducida
Conexiones externas	Bastantes puertos disponibles	Muchos puertos disponibles	Muchos puertos disponibles	Muchos puertos disponibles	Pocos puertos disponibles
Comunicaciones alámbricas	I2C, SPI, UART	I2C, I2S, SPI, UART, SDIO; varios canales de cada	I2C, SPI, UART	I2C, SPI, UART	I2C, SPI, UART (varios canales)
Comunicaciones inalámbricas	Wifi (algunos modelos)	Wifi, Bluetooth LE, Zigbee (algunos modelos)	Wifi, Bluetooth LE, Zigbee, Lora, Sigfox	Wifi & Bluetooth (algunos modelos)	Wifi & Bluetooth (algunos modelos)
Tipo de plataforma	Abierta	Abierta	Cerrada	Cerrada	Cerrada
Elementos requeridos para programarlo	Cable USB	Cable USB	ST-Link CubeMX	Cable USB	PICkit3 o cable USB (según el modelo)
Consumo	Medio	Muy bajo	Ultra bajo	Bajo	Bajo
Precio	< 10 €	< 10 €	~ 10€ - 20 €	~ 30 €	< 10 €
Tamaño	Depende del modelo	Pequeño	Pequeño	Pequeño	Mediano

Tabla 2 - Comparativa entre plataformas de microcontroladores.

No se han incluido en la comparativa los módulos **EBYTE** E18-MS1 dada su similitud con los WROOM de la familia Espressif. Se han descartado debido a la complejidad de su programación y nula compatibilidad con el entorno de desarrollo seleccionado. Pese a su principal ventaja competitiva: están enfocados a comunicación ZigBee con un consumo de energía ultra bajo.

Estudiando las características particulares de cada modelo dentro de cada familia, y basándose en diseños de referencia comerciales, se considera que, por su bajo coste y consumo, sus altas prestaciones y su compatibilidad con el lenguaje y plataforma de desarrollo escogidos, el controlador ideal para el presente proyecto se trata del módulo **ESP32-WROOM-32**, que aparece en la *imagen 5* de este documento.

Este módulo en particular tiene un coste de entre 2€ y 3€, conexión wifi y Bluetooth LE (*Low Energy*) 5.0 y sus principales características son:

- Microcontrolador de doble núcleo a 240 MHz.
- 4 MB de memoria Flash.
- 520 kB de memoria SRAM.
- 38 pines analógico-digitales (solo 16 pueden configurarse para salida PWM).
- 2 puertos UART, 3 SPI, 2 I2C, 2I2S y SDIO.
- 2 ADC de 12 bits y 2 DAC de 8 bits.
- Tamaño: 18±0.2 mm x 25.5±0.2 mm x 3.1±0.15 mm

3.2. Reconocimiento de voz

Se han estudiado diversos métodos de reconocimiento de voz. El más simple y eficiente, debido a su entrenamiento continuo, es la **inteligencia artificial** semi gratuita que ofrece el servicio Google Cloud Plataform. Puesto que la plataforma empieza a solicitar cobros a partir de cierto número de horas y que por motivos de seguridad y privacidad se ha decidido prescindir de conexión a internet para el reconocimiento de voz, se ha descartado esta opción.

El segundo tipo de sistemas de reconocimiento de voz que se han estudiado son los **softwares** compatibles con Arduino y con capacidad de reconocimiento de palabras en español. Destacan entre éstos BitVoicer, Julius y Microsoft Speech, que puede utilizarse en un código sencillo en lenguaje C#. Estos últimos son gratuitos y cuentan con un algoritmo de alta precisión, pero al igual que el resto, necesitan estar conectados a un ordenador con sistema operativo Windows (o en algunos casos Linux o Mac), por lo que también han sido descartados.

La última opción es el uso de **hardware** de reconocimiento de voz. En primer lugar, se consideraron los "*shields*" de Arduino, placas modulares que se apilan sobre los Arduino para proporcionarles capacidades extra como muestra la *imagen 6*. Los modelos estudiados han sido los basados en los microcontroladores RSC-4128 y XFS-5051-CE, con precios de alrededor de 50€ y 35€ respectivamente y solo con capacidad de reconocimiento de la lengua inglesa, y el Arduino MOVI Shield. Este último, capaz de reconocer hasta 1000 palabras en español y con un precio de alrededor de 75€, también aporta prestaciones de síntesis de texto a voz (TTS), lo que lo convertiría en una alternativa ideal de no ser por su gran tamaño y elevado precio.

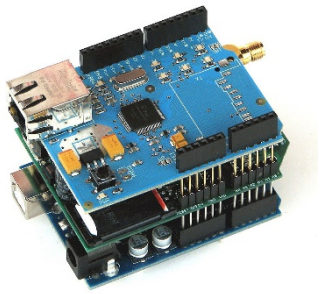


Imagen 6 - Ejemplo de "shields" *sobre un Arduino UNO*

Finalmente, el hardware que utilizado será el módulo "**Voice Recognition Module V3.1**" (o VRM, para abreviar) del fabricante Elechouse. Este módulo es capaz de reconocer de manera simultánea, mediante comparación, 7 de las hasta 80 instrucciones de voz previamente grabadas en su memoria. Tiene un consume de energía medio, con picos de hasta 40 mA, funciona a 5V y se comunica con el microcontrolador a través de dos entradas digitales que simularán una comunicación serie (UART) gracias a las librerías de su fabricante. Su precio varía entre los 14€ y los 20€ según el distribuidor y su precisión es del 99% de comandos reconocidos en un entorno ideal.

Otra alternativa barajada es el módulo LD3320, con características muy similares, pero más complejo de programar.

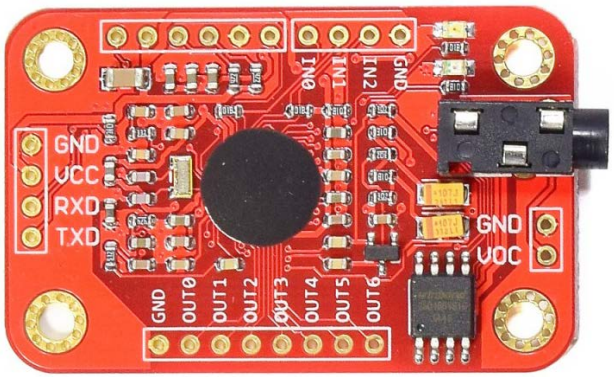


Imagen 7 - Voice Recognition Module V3.1 *de Elechouse*

3.3. Actuadores

Para los actuadores, se necesita un microcontrolador capaz de recibir instrucciones del controlador principal vía wifi y con las entradas (E) y salidas (S) suficientes para cumplir los objetivos de los puntos 2.2 al 2.6. Volviendo a utilizar la *tabla 2* y basándose en diseños de referencia comerciales, se concluye que la mejor opción es continuar utilizando el módulo **ESP32-WROOM-32**.

La elección de los componentes específicos de cada parte se realizará en el *apartado 4* tras ser dimensionados.

3.4. Comunicaciones

Dadas las características del microcontrolador utilizado, la comunicación entre estos se realizará vía wifi, ya que tiene unas características de velocidad de transmisión, seguridad y alcance mucho mayores que el Bluetooth. Por el contrario, el consumo de energía también es mayor, pasando de aproximadamente 100 mA a 200 mA cuando transmite datos.

Los módulos se comunicarán entre sí mediante una **red de árbol**, en la que el controlador principal se comunicará con los controladores de cada función (luces, persianas, etc.) y éstos lo harán con todos los sensores y actuadores que tengan asociados, como muestra el esquema de la *imagen 8*. Por lo tanto, la **arquitectura** del sistema será híbrida, ya que existe un controlador principal que lo coordina todo y uno secundario que funciona de forma autónoma para cada tarea de control. Las ventajas de esto son una mayor flexibilidad y eficiencia y asegurar el correcto funcionamiento del resto del sistema si una de sus partes fallara.

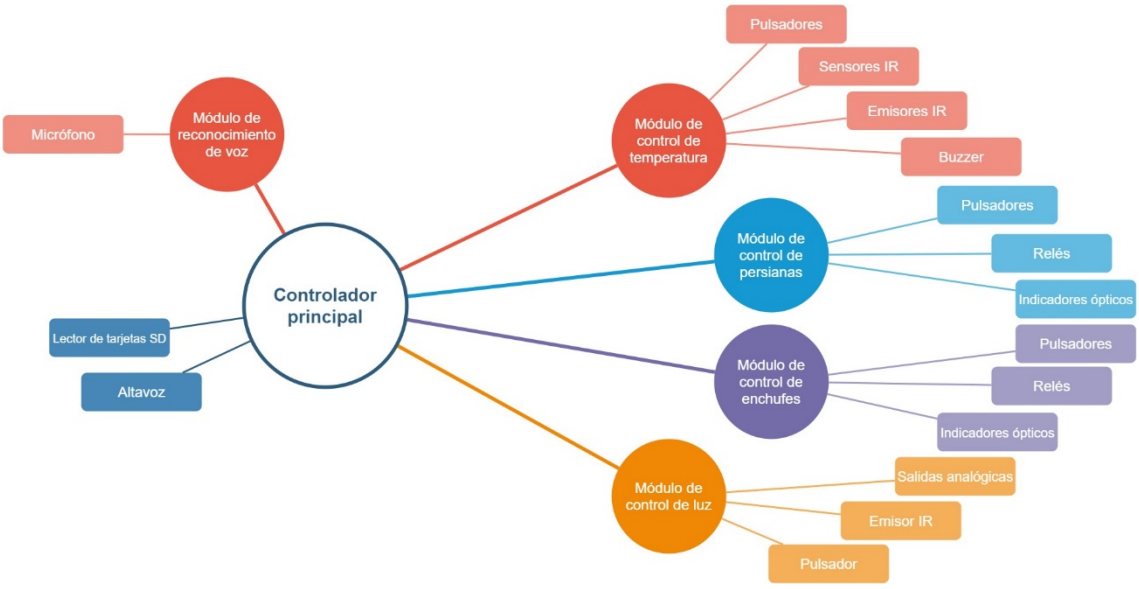


***Imagen 8 - diagrama de comunicaciones de Xana**. Las conexiones entre los elementos serán vía pines GPIO excepto entre el controlador principal y los cuatro módulos de su derecha, que se comunicarán por wifi local.*

La *imagen 9* muestra un resumen del funcionamiento general de Xana, en el que se incluyen sus 5 módulos.

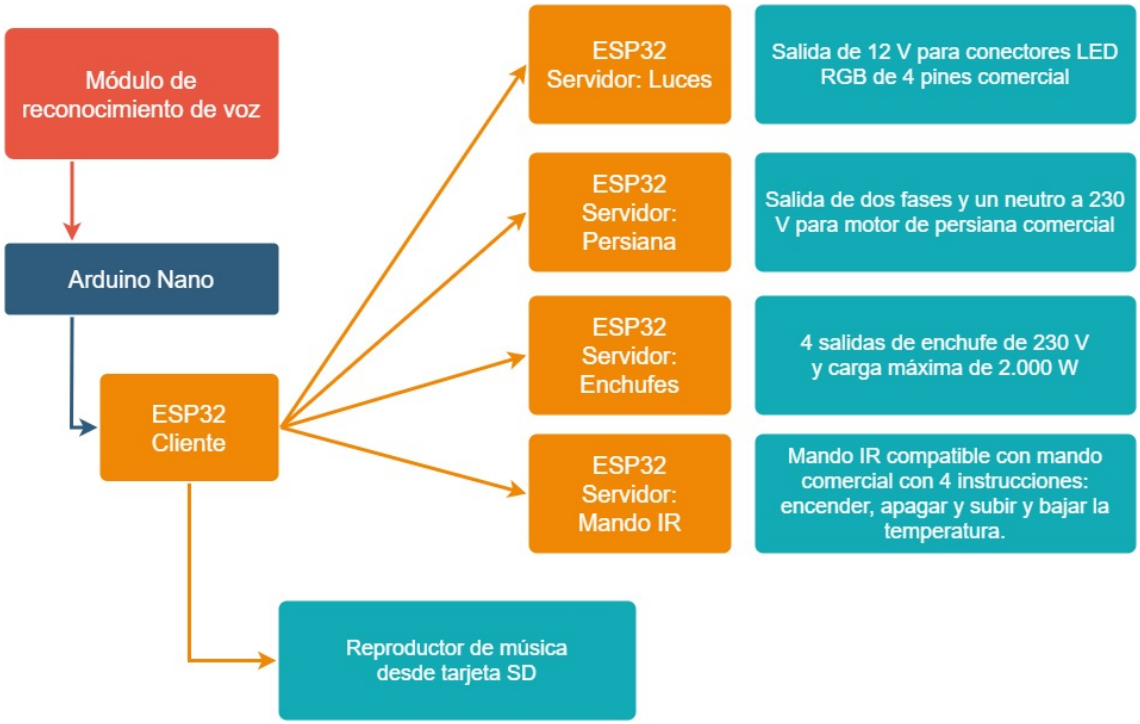


Imagen 9 - Esquema general de Xana.

4. Descripción de la solución adoptada

4.1. Modo de funcionamiento

El modo de funcionamiento de Xana será muy similar a los asistentes de voz, pero sin la capacidad de sintetizar oraciones en instrucciones concretas. Las **instrucciones de voz** que se le proporcionen deberán ser ya instrucciones concretas en el formato *{palabra de activación, actuador, acción o acciones, (matiz)}*, donde matiz es una instrucción opcional que tienen algunos actuadores como las luces o las persianas. Así, en lugar de decirle "*Xana, enciende las luces*", el comando será "*Xana, luces, encender*".

Puesto que el VRM (el módulo de reconocimiento de voz) puede almacenar hasta 80 comandos, pero solamente cargar hasta 7 a la vez en el reconocedor, han de diseñarse una serie de funciones que puedan organizarse en un árbol de comandos con las opciones presentadas de siete en siete. La séptima palabra será "Xana" en el conjunto de instrucciones principal y "volver" en los demás grupos, para poder cancelar la instrucción a mitad en cualquier momento.

Los **actuadores** elegidos, que serán los seis presentes en el menú principal, son los cinco acotados en los objetivos del proyecto y una sexta opción para configurar una alarma o rutina. Como se menciona en el cuarto párrafo del *apartado 2* de este documento, esta rutina no será funcional, pero sí configurable. Es decir, se podrán configurar sus tres funcionalidades (hora, frecuencia de repetición y actuador que activará) para que pueda funcionar en una versión futura en la que se implemente un reloj de tiempo real como el DS3231.

El **diagrama de flujo** del *anexo 1* del proyecto muestra el ciclo completo del sistema de control por voz de Xana, desde que se conecta el sistema y se reconoce la palabra de activación, hasta que actúa sobre uno de los elementos de control. Las imágenes 9 y 10 muestra una versión reducida de este diagrama a modo de ejemplo visual, donde el nivel 1 corresponde a la selección del actuador, el 2 a la acción o acciones de control y el 3 al matiz o matices de la acción.

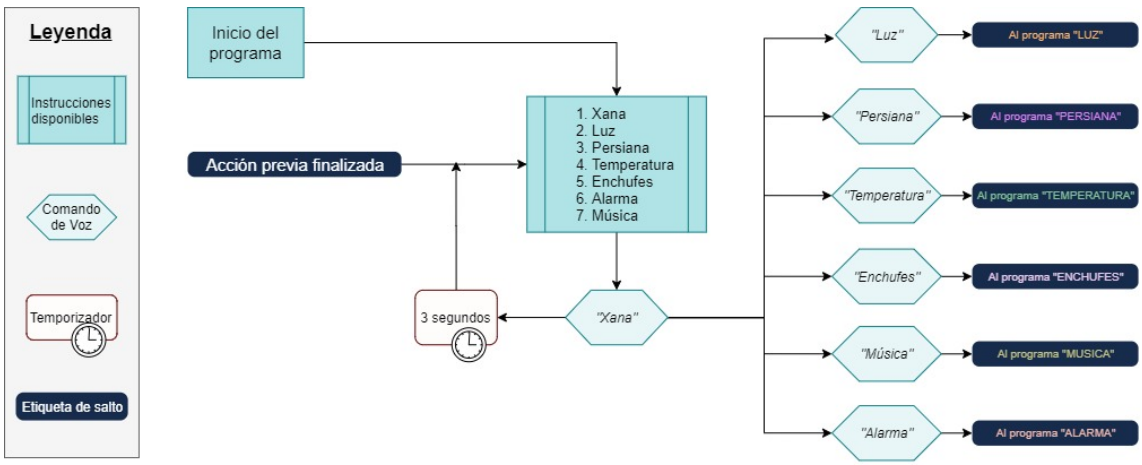


Imagen 10 - Diagrama de flujo de Xana reducido (nivel 1).

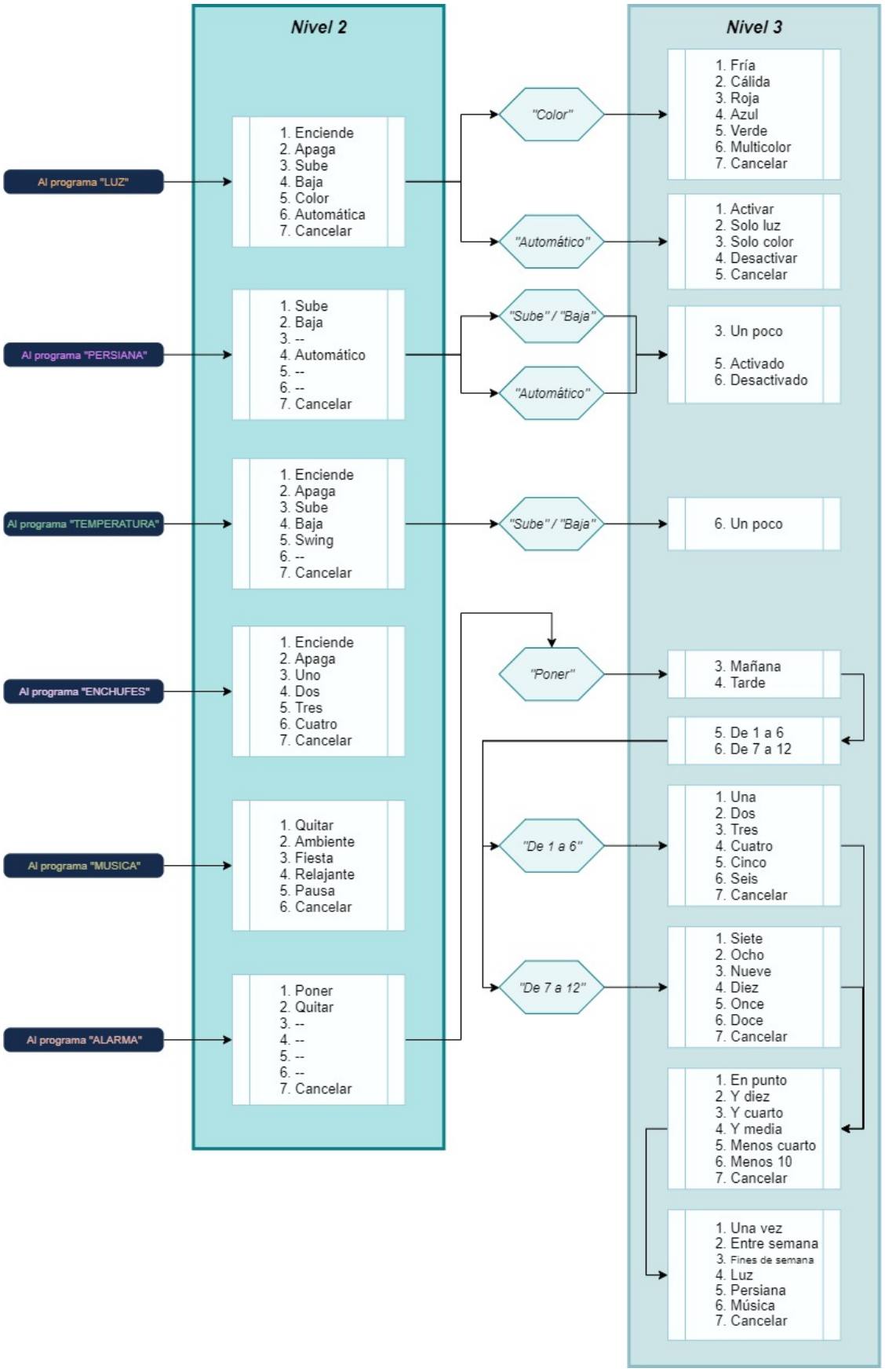


Imagen 11 - Diagrama de flujo de Xana reducido (niveles 2 y 3). *Tras cada cuadro de "instrucciones disponibles", los comandos de voz que no están seguidos de un hexágono con su nombre son los que actúan sobre los elementos de control y, a continuación, devuelven el sistema al nivel 1 del diagrama. Las instrucciones sustituidas por dos guiones (--) están desactivadas para utilizarlas en el siguiente nivel si procediera.*

4.2. Reconocimiento de voz

Las librerías que proporciona el fabricante del VRM para su configuración y uso son librerías enfocadas a Arduino. Se han modificado para que funcionen en la arquitectura del ESP32, y aunque se ha logrado que el código compile, no se ha conseguido que la comunicación entre ambos módulos sea efectiva. Dada esta circunstancia, se ha optado por utilizar una placa de Arduino como mediadora entre ambos módulos. Por su pequeño tamaño y bajo coste, la placa que se ha decidido utilizar ha sido el **Arduino nano**.

4.2.1. Configuración del módulo

El módulo, mostrado en la *imagen 7* del documento, se configura a través de una interfaz del puerto serie del Arduino mediante comandos específicos, como muestra la *imagen 12*.

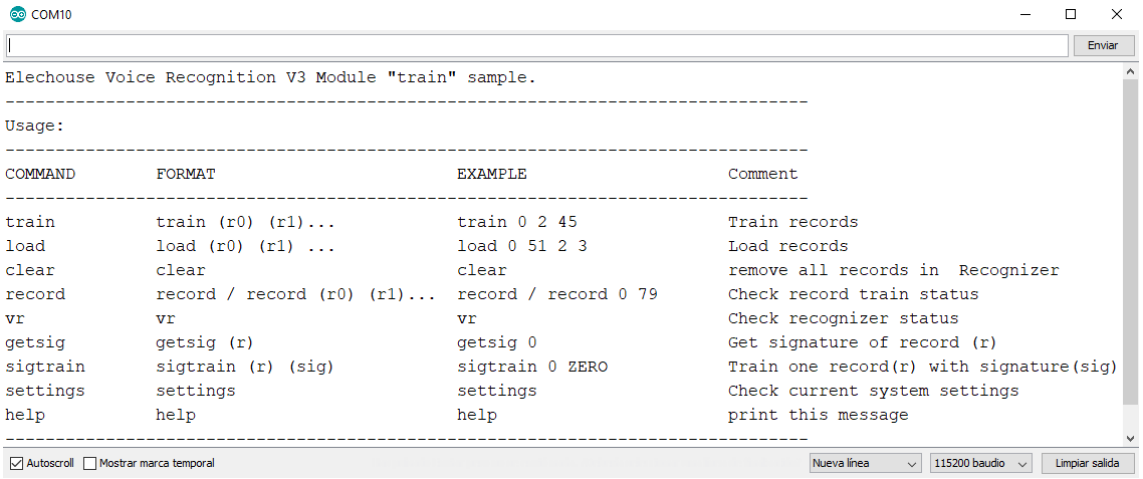


***Imagen 12 - Interfaz de configuración del Voice Recognition Module V3.1** a través del puerto serie de Arduino.*

Una vez grabadas y configuradas las 51 instrucciones de voz que aparecen en el diagrama de flujo del sistema, se procede a crear las dos librerías que facilitan en acceso a los comandos de la *imagen 12* en cualquier punto del programa sin la necesidad de usar el canal serie. Éstas son "*Instrucciones.h*", que asigna a la dirección de cada comando de voz una etiqueta con su nombre y la librería "*XanaVRM.h*", basada en la librería que utiliza fabricante. Estas librerías conformarán los *anexos 2 y 3* del proyecto.

4.2.2. Conexiones

El VRM tiene dos entradas de audio. La primera es un puerto Jack TRS de 3.5mm para micrófonos pasivos de condensador, los cuales tienen un rango demasiado corto para cumplir las especificaciones del proyecto. La segunda entrada es una entrada también analógica formada por dos pines, a los que conectaremos el circuito que la *imagen 13*, que se explicará a continuación.

El circuito consta de un micrófono analógico de bajo coste y alto rango y precisión, el ICS40180, que se alimenta a 3,3V. Para asegurar su correcto funcionamiento en el VRM, se amplificará mediante el amplificador operacional TL081, usado principalmente en aplicaciones de audio, en configuración no inversora que aconseja su fabricante. El condensador *C1* es un condensador de acoplo, cuya función es filtrar la componente continua de la señal de audio. Los condensadores *C2* y *C3* son condensadores de desacoplo, que estabilizarán la alimentación de ambos componentes para minimizar la distorsión de la señal de audio.

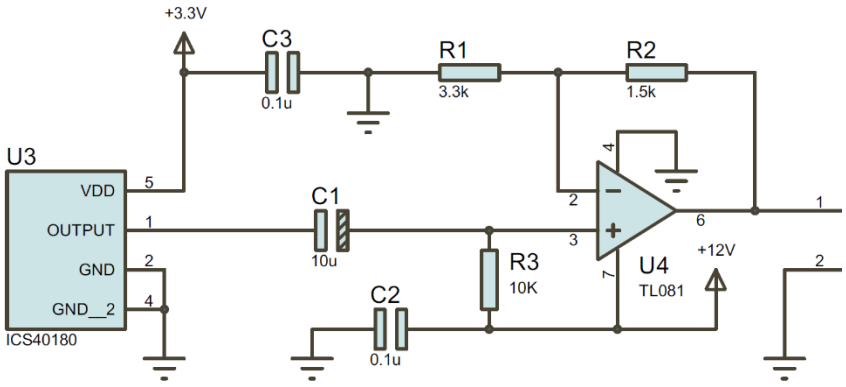


Imagen 13 - Circuito de acondicionamiento de un micrófono externo para el VRM

La ganancia señal que se pretende obtener es de 5 V a la salida del TL081 cuando la del micrófono sea 3,3 V, es decir:

$$G_{máx} = \frac{V_{out}}{V_{in}} = \frac{5}{3{,}3} \cong 1{,}52\ {}^{V}/_{V}$$

Dada la tabla de valores de resistencia normalizadas de la serie E12 del *anexo 28* de este proyecto, se ha encontrado que la relación de resistencias *R1* y *R2* que más se acercan a la ganancia máxima calculada son 3,3 kΩ y 1,5 kΩ, respectivamente. La amplificación sería la siguiente:

$$G = \frac{R_1 + R_2}{R_1} = \frac{3{,}3 + 1{,}5}{3{,}3} \cong 1{,}45\ {}^{V}/_{V}$$

$$V_{out_{máx}} = V_{in_{máx}} \cdot G \cong 3{,}3 \cdot 1{,}45 = 4{,}8\,V$$

En la *imagen 14*, se muestra la función de transferencia de esta configuración del amplificador operacional. Puede observarse en la gráfica que cuando el micrófono proporciona una salida (V_{in+}) de 3,3 V, la señal a la entrada del VRM (V_{out}) es de prácticamente 5 V.

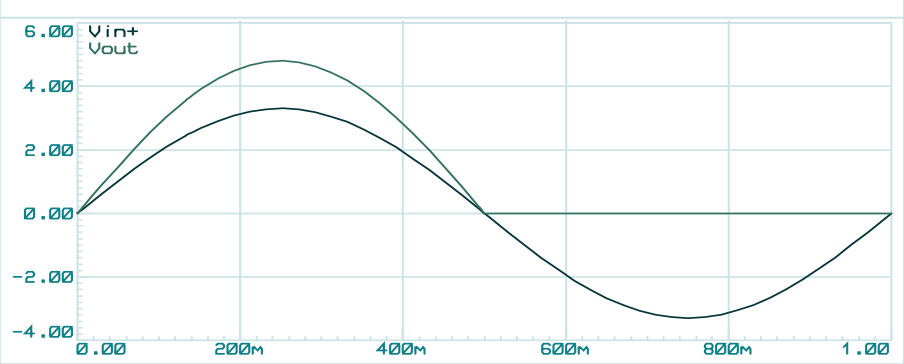


Imagen 14 - función de transferencia de un amplificador operacional *en configuración no inversora, alimentado entre 0 V y 12 V y con una ganancia de 1,45 V/V.*

4.2.3. Algoritmo de reconocimiento

Aprovechando que coexistirán dos microcontroladores en el controlador principal, se utilizará el Arduino nano para la interpretación contina de instrucciones de voz y el ESP32 para el resto de las funcionalidades de éste.

El código del Arduino está construido de manera modular basándose en las dos librerías de reconocimiento de voz, dos librerías para cada actuador; una para navegar por los menús de voz de nivel 2 y nivel 3 y otra que gestiona las instrucciones y las comunica al ESP32, y un ciclo principal que lo engloba todo correspondiente al nivel 1 del diagrama de flujo. Su funcionamiento es el que muestra la *imagen 15*:

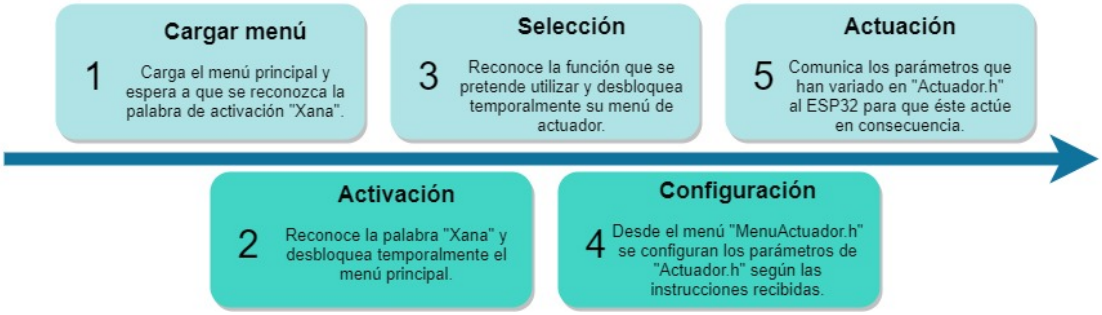


Imagen 15 - Proceso de interpretación de instrucciones *basado en un sistema de librerías modular. En cualquier momento de los pasos 3 y 4, si se detecta la instrucción "Cancelar", se volverá al paso 1.*

Las librerías que intervienen en este proceso ("*XanaVRM.cpp*", "*menuAlarma.h*", "*alarma.h*", "*menuEnchufe.h*", "*enchufes.h*", "*menuLuz.h*", "*luz.h*", "*menuMusica.h*", "*musica.h*", "*menuPersiana.h*", "*persiana.h*", "*menuTemperatura.h*" y "*temperatura.h*") conformarán los *anexos 4 al 16* de este proyecto.

4.3. Comunicaciones del controlador principal

En este apartado se distinguen dos tipos de comunicaciones: la comunicación alámbrica entre los propios elementos del controlador principal y la comunicación inalámbrica entre éste y los controladores de cada uno de los actuadores remotos (luz, persiana, mando de temperatura y enchufes).

4.3.1. Comunicación entre módulos del controlador principal

Por un lado, la **comunicación entre el VRM y el Arduino** puede realizarse de dos maneras. Una es través de sus 7 salidas digitales, una por instrucción cargada en el reconocedor, y sus 3 entradas digitales con las que cargar un conjunto de instrucciones predefinido en el reconocedor. La opción elegida es usando el canal serie mediante el que se ha configurado en el *apartado 4.2.1*. y las librerías creadas en el mismo, ya que permiten una programación más intuitiva y la posibilidad de mostrar mensajes a través del canal serie.

Esta comunicación se realiza a través de un canal serie simulado por software gracias a la librería "*SoftwareSerial.h*", ya que el Arduino nano solamente dispone de un canal serie por hardware, el cual se utiliza para programar su microcontrolador.

Por otro lado, la **comunicación entre el Arduino Nano y el Esp32** puede realizarse mediante los protocolos UART, SPI o I2C. Ya que solamente se van a conectar dos dispositivos, la velocidad de transmisión de datos no será un factor determinante y no hay necesidad de que la conexión sea síncrona, se volverá a escoger el protocolo UART mediante un segundo canal serie simulado por software, ya que es la opción más sencilla de implementar. Se ha tenido en cuenta a la hora de diseñar el código que no pueden estar activos los dos canales serie simulados al mismo tiempo. La solución ha sido que la comunicación con el VRM esté activa en todo momento excepto en los instantes en los que el Arduino Nano necesite usar el otro canal para comunicarse con el ESP32. Por su parte, El ESP32 dispone de tres canales serie implementados por hardware así que no ha sido necesario simular uno por software.

El siguiente factor a tener en cuenta es que el Arduino funciona a 5 V, mientras que el ESP23 funciona con 3,3 V, por lo que éste último no puede recibir en sus entradas una señal directamente de la salida del primero. Una posible solución habría sido utilizar un *Level Shifter* o adaptador de nivel, como se muestra en la i*magen 16*. Este es un circuito basado en un MOSFET de canal N y dos resistencias de pull-up que sigue la señal de su entrada a nivel bajo (LV1) en la otra con un voltaje mayor a su salida (HV1) y viceversa.

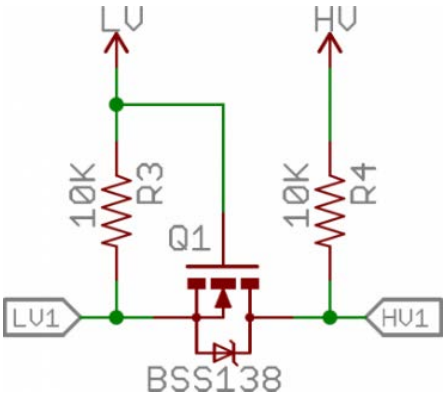


***Imagen 16 - Level shifter** o adaptador de nivel basado en el MOSFET BSS138*

Puesto que la comunicación va a ser unidireccional del Arduino al ESP32 y solo será necesario reducir el voltaje de 5 V a 3,3 V, la opción más adecuada en este caso es utilizar un divisor resistivo con dos resistencias de 1,8 kΩ y 3,3 kΩ. Esto dejaría la señal del Arduino con una tensión de, aproximadamente, de 3,24 V, por lo que la comunicación puede ejecutarse sin problemas.

4.3.2. Comunicación entre el controlador principal y los actuadores

Para los controladores remotos, la comunicación se realizará mediante una arquitectura **cliente-servidor vía wifi**, en la que el controlador principal será el cliente. Con esto se consigue mayor simplicidad en el código y que los actuadores puedan ser controlados por terceros, como smartphones si se quisiera, que tengan la contraseña. Para el control de la música se utilizará un módulo lector de tarjetas SD conectado por SPI, ya que ofrece mejores prestaciones de audio que las entradas SDIO del ESP32. Se profundizará en esta conexión en el *apartado 4.8.* de este documento.

El controlador principal, que actuará de **cliente**, cuando reciba por su canal serie principal un mensaje de Arduino, comprobará si tiene el formato "`Orden enviada:`" seguido de un salto de línea. En caso afirmativo, leerá los primeros tres caracteres de la siguiente línea, la instrucción, para saber con qué actuador comunicarse. Cada instrucción tiene un formato diferente dependiendo del elemento a controlar:

- Para la luz: "*LUZXXXRRRGGGBBB*", donde:
  - *XXX* corresponde a la intensidad (de 0 a 100).
  - *RRR*, *GGG* y *BBB* son los tres parámetros del color (entre 0 y 255).
- Para la persiana: "*PERxxxxY*", donde:
  - *xxxx* deberá corresponder con las palabras "`sube`" o "`baja`".
  - *Y* es el bit que indica si la persiana debe deslizarse hasta su extremo (si *Y* es 1) o solo modificar su posición un 20% (si es 0).
- Para los enchufes: "*ENCXY*", donde:
  - *X* será el identificador del enchufe a controlar (del 1 al 4).
  - *Y* es el bit que indica si se pretende encender (si *Y* es 1) o apagar (si es 0).
- Para el control de temperatura: "*TEMxxx*", donde:
  - *xxx* deberá corresponder con las palabras "`onn`", "`off`", "`mas`", "`men`" o "`swg`", que se corresponden con las instrucciones de encender, apagar, subir la temperatura, bajarla y activar el modo *swing* del aire acondicionado, respectivamente.
- Para la música: "*MUSxxxx*", donde:
  - *xxxx* deberá corresponder con las palabras "`abte`", "`prty`", "`relx`", "`para`" o "`quit`", que se corresponden con las instrucciones de reproducir música de la carpeta "ambiente", "fiesta" o "relax"; pausar o reanudar la reproducción actual o parar toda la música y conexiones, respectivamente.

Cuando el actuador sea este último, se procederá como se describe en el *apartado 4.8.*, correspondiente al control de sonido. En cualquier otro caso, el cliente buscará el punto de acceso que debería haber creado el servidor correspondiente y se intentará conectar a él con una dirección IP dinámica. Si lo consigue, realizará una petición HTTP con la forma `HTTP://[IP del servidor]/[request]`, donde la IP del servidor será siempre una IP estática con la dirección 192.168.4.1 y "request" será la misma instrucción que ha recibido del Arduino por el canal serie. Si no es posible conectar con el servidor, desistirá a los 10 segundos y descartará la instrucción.

El código principal de este cliente, "*XanaClient.cpp*" podrá encontrarse como el *anexo 17* de este proyecto.

En el caso de los **servidores**, crearán un punto de acceso o AP (*Access Point*) y esperarán a que se conecte un cliente (o a recibir una instrucción por hardware). Descompondrán la *request* recibida en instrucciones simples como se ha indicado en el esquema de este apartado y actuarán cada cual en función de lo descrito en su correspondiente apartado (del *4.4.* al *4.7.*). Después de responder a la petición del cliente, volverán a liberar el punto de acceso para otra futura conexión.

Se ha creado una librería general, "*XanaWifi.h*", para acceder a los métodos wifi de manera sencilla y cinco **librerías** particulares, "*Xana_ClientLib.h*", "*Xana_ ServerLuzLib.h*", "*Xana_ ServerEnchufeLib.h*", "*Xana_ ServerPersianaLib.h*" y "*Xana_ ServerTemperaturaLib.h*", para gestionar la comunicación entre el cliente y los servidores y el consiguiente control de los actuadores. Conformarán los *anexos 18 al 23* del proyecto.

4.4. Control de luces

En primer lugar, hay que diferenciar entre los dos actuadores sobre los que podrá gobernar este controlador: tiras de LED y bombillas RGB. Cada vez que el ESP32 reciba una orden del cliente, escribirá el color en ambas salidas.

4.4.1. Tiras LED

Para las tiras LED, la conexión más estandarizada es un conector de 4 pines en formato [*+12V, G, R, B*], donde RGB son tensiones entre 0 V y 12 V que se corresponden a valores entre 0 y 255 respectivamente. En la *imagen 17*, puede observarse el funcionamiento de tres LED RGB dentro de una tira. Al funcionar con ánodo común, el color que escribirá el microcontrolador será el inverso del que se desea mostrar. Por ejemplo, si se pretende escribir el color rojo, (255, 0, 0) en formato RGB, deberá escribirse el color (0, 255, 255); así, el canal R recibirá una tensión de 0 V y la diferencia de potencial de los tres LED rojos será de 12 V, mientras que en los otros dos canales se estarán recibiendo 12 V y no circulará la corriente.

Dependiendo del tipo de LED utilizados y de su distribución, el **consumo** máximo de cada metro de tira puede variar entre unos 5W y unos 15 W. La alimentación que se utilizará (ver el *apartado 4.9.* de la memoria) puede proporcionar una potencia máxima de casi 24 W, lo que se considera suficiente para iluminar una habitación.

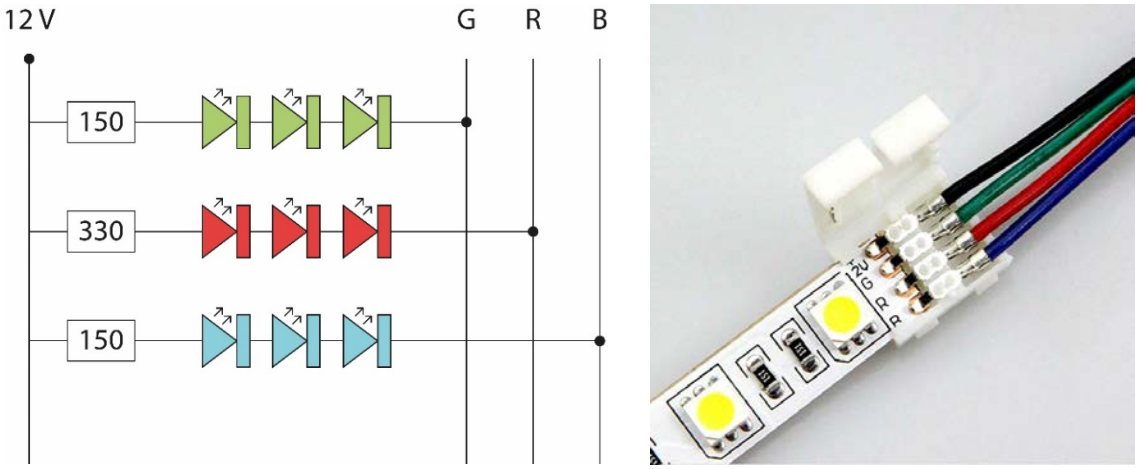


Imagen 17 - Anatomía de un fragmento de tira LED *con tres LED RGB y ejemplo de conexión.*

Para calcular los valores de cada canal, de acuerdo con la teoría del color, el microcontrolador multiplica el valor RGB que desea configurarse por la **intensidad** lumínica en porcentaje. Esto es, si se desea proyectar una luz blanca (255, 255, 255) con una intensidad del 10%, el código RGB que se escribirá será (26, 26, 26).

Para amplificar las señales de control del ESP32, donde la intensidad de color 255 equivale a 3,3 V, hasta los 12 V, se ha utilizado un amplificador operacional en configuración no inversora igual que en el a*partado 4.2.2.*, como muestra la i*magen 18*. Esta vez el AO utilizado ha sido el TLV170, con características *rail-to-rail*, para poder proporcionar una salida máxima lo más cercana posible a los 12 V por cada canal.

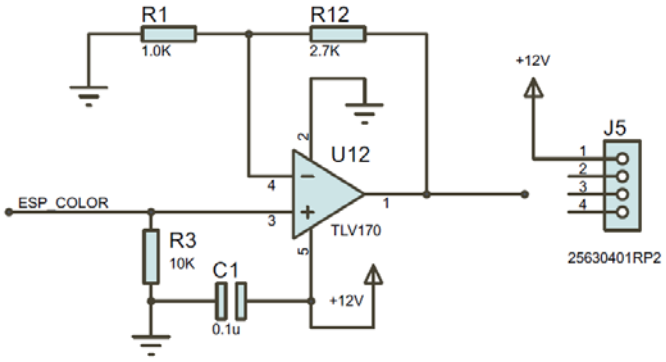


Imagen 18 - Configuración no inversora del AO *para las señales de control del color, similar al de la imagen 13. Este circuito se repetirá 3 veces, una por cada señal de color.*

La ganancia necesaria, en este caso es la siguiente:

$$G_{mín} = \frac{V_{out}}{V_{in}} = \frac{12}{3{,}3} \cong 3{,}64\ {}^{V}/_{V}$$

A la hora de aproximar, puesto que la intensidad de iluminación se regula en saltos del 20%, se ha preferido obtener una ganancia mayor que la teórica para que el 100% de intensidad suponga siempre la salida máxima que el AO puede suministrar. Las resistencias normalizadas que aportan una relación más cercana a este valor son 1 kΩ para *R1* y 2,7 kΩ para *R2*. La ganancia por lo tanto será:

$$G = \frac{R_1 + R_2}{R_1} = \frac{1 + 2{,}7}{1} = 3{,}7\ {}^{V}/_{V}$$

Y el valor de intensidad al que se proporcionará la salida máxima, es decir, la tensión de saturación del AO será:

$$V_{out_{máx}} = \frac{V_{AO_{sat}}}{G} \cong \frac{12 - 0{,}2}{3{,}7} = 3{,}2\ V$$

Esto supone que a partir de una intensidad de color del 97%, la intensidad de iluminación será máxima, lo que supone un error asumible.

4.4.2. Bombillas LED

Las bombillas LED comerciales pueden controlarse a través de aplicaciones propias por Bluetooth o por wifi conectándose a una red local o pueden controlarse mediante luz infrarroja con unos mandos genéricos de 24 ó 44 botones. Los botones de estos mandos son todos los mismos y sus códigos IR están normalizados. Puesto que Xana solo dispone de 10 funciones para controlar una bombilla RGB, se simularán los códigos IR de un mando de 24 botones genérico, como muestra la *imagen 19*.

Imagen 19 - Mandos de bombillas LED RGB *mediante infrarrojos de 44 y 24 botones.*

Los códigos que se utilizarán serán los siguientes:

Comando	Botón	Código
Encender	“ON”	0xF7C03F
Apagar	“OFF”	0xF7408F
Subir brillo		0xF700FF
Bajar brillo		0xF7807F
Rojo	R	0xF720DF

Comando	Botón	Código
Azul	B	0xF7609F
Verde	G	0xF7A05F
Luz fría	W	0xF7E01F
Luz cálida	(Amarillo)	0xF728D7
Multicolor	Fade	0xF7C837

Tabla 3 - Códigos IR en hexadecimal *que Xana utilizará para comunicarse con las bombillas RGB.*

4.4.3. Control automático

La base del controlador principal dispondrá de un **sensor de luminosidad** del tipo LDR de sensibilidad regulable, tal y como muestra la *imagen 20*. Las entradas analógicas del ESP32 detectan variaciones de 12 bits de precisión, o lo que es lo mismo, 3,3 V entre 4.096 escalones es una precisión de aproximadamente 0,8 mV. Al no necesitar tanta resolución, se reducirá a la mínima permitida, 9 bits (6,4 mV), mediante el comando `analogSetWidth(width)`. Los LDR disminuyen su resistividad con la luz de manera logarítmica, según la gráfica de la i*magen 21*.

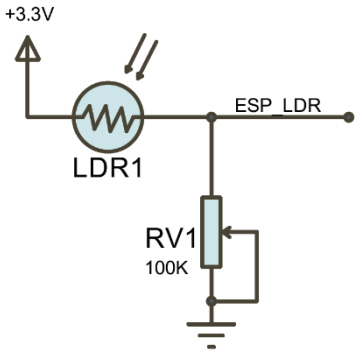


Imagen 20 - Sensor de luminosidad *con un LDR de sensibilidad regulable y salida proporcional al nivel de luz.*

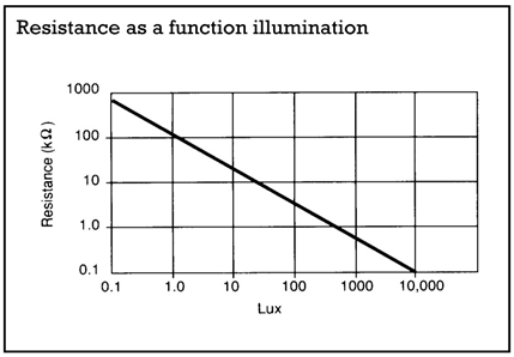


Imagen 21 - Relación entre la resistividad que presenta un LDR y la iluminación que recibe*. Los niveles extremos de luminosidad corresponden a la luz del día (10.000 lux) y a la oscuridad casi total (0,1 lux).*

En el **modo automático de luz** del Xana, el ESP32 del cliente leerá la salida del sensor de la *imagen 20* y actuará en consecuencia. Solicitará al servidor aumentar un 5% la intensidad de su salida cuando este valor baje de 2,5 V y realizará la petición contraria cuando suba de 3 V. Con estos márgenes, se asegura una iluminación constante, automática y energéticamente eficiente hasta que el usuario decida apagarla.

El **control de color** está pensado para cambiar los tonos de luz de cálida a fría conforme avanza el día para sincronizarse con los tonos de luz natural y ofrecer una experiencia de iluminación más saludable para la experiencia humana, de acuerdo con un reciente estudio de la Universidad de Manchester. Este control no se implementará en esta versión prototipo debido a la falta de un reloj como el DS3231 nombrado en varias ocasiones.

El código de "*Xana_ ServerLuz.cpp*" constará en el *anexo 24* de este proyecto. La parte de control automático reside en el código del controlador principal del *anexo 17*.

4.5. Control de enchufes

Cada uno de los 4 enchufes se activará mediante un relé mecánico, que protegerá el circuito frente a las altas potencias del circuito de la carga. El controlador conmutará un el relé bien si recibe esa petición del cliente bien si se acciona su pulsador manual correspondiente. Si el cliente solicita encender un relé ya activo, no ocurrirá ningún cambio y viceversa.

Los relés elegidos son el modelo G5LE-1A-DC5, que soportan una carga máxima de 10 A con una tensión de 250 VAC, es decir, 2.500 W. Se ha elegido este tipo de relés por su capacidad de aislamiento y por su bajo precio. Como se observa en la *imagen 22* cada relé cuenta con un diodo de protección en antiparalelo (*D1*) para evitar que la corriente que almacena dañe los demás elementos del circuito al desactivarse la salida del microcontrolador. También cuentan con un LED (*D2*) que se encenderá para indicar al usuario que el enchufe está en marcha. El conector *J1* corresponde con las clavijas de salida del enchufe.

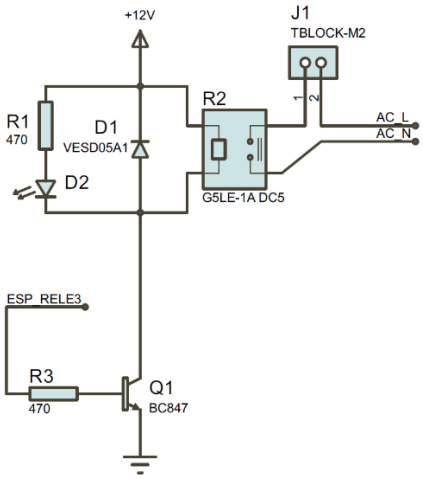


Imagen 22 - Circuito de control del relé. *Este circuito se repetirá cuatro veces.*

El código de "*Xana_ ServerEnchufe.cpp*" constará en el *anexo 25* del proyecto.

4.6. Control de motor de persiana

Los motores que se utilizan para mover persianas son **motores** monofásicos de fase partida, es decir, de corriente alterna. Éstos controlan su dirección mediante un bobinado de arranque y mantienen el motor en marcha mediante el bobinado principal o bobinado de trabajo. En la *imagen 23*, se muestra un esquema de conexiones de uno de estos motores que describe este funcionamiento.

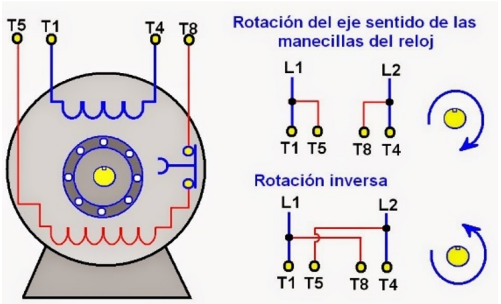


Imagen 23 - Conexiones de un motor monofásico de fase partida. *El bobinado azul se corresponde con el bobinado de trabajo y el rojo, con el de arranque.*

El controlador realiza esta **inversión de la alimentación** de arranque mediante dos relés, tal y como muestran las *Imágenes 24* y *25*. El relé que se utilizará será un JW2HN-DC9V, un relé de una entrada y dos salidas de forma C capaz de soportar hasta 5 A a 230 VAC, lo que alimenta con creces a la gran mayoría de motores de persianas del mercado. Cuando el motor alcanza la velocidad suficiente, él solo desconecta el bobinado de arranque mediante una llave centrífuga. El circuito utilizado para activar y desactivar la corriente de fase y neutro que alimenta al motor será el mismo que el mostrado en la *imagen 22* del apartado anterior.

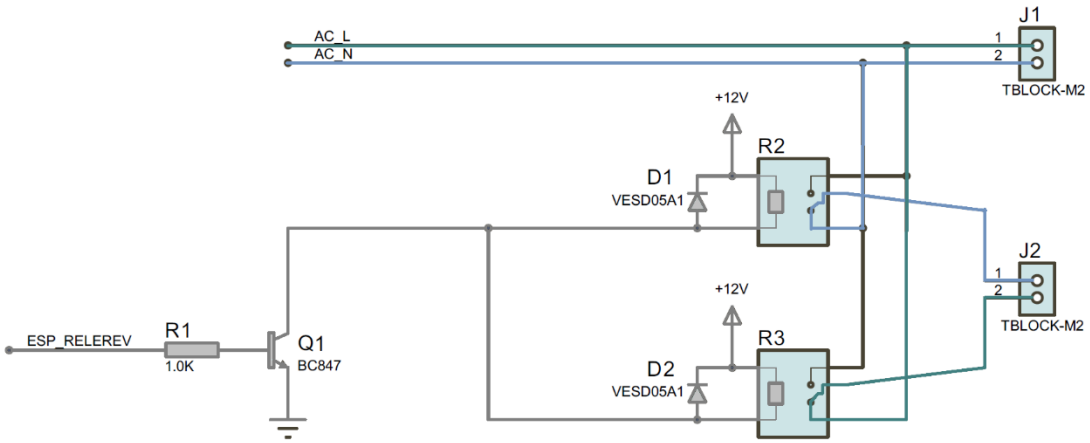


***Imagen 24 - Circuito de cambio de sentido del motor con los relés de inversión desactivados**. Los conectores J1 y J2 alimentan a las bobinas de trabajo y arranque, respectivamente.*

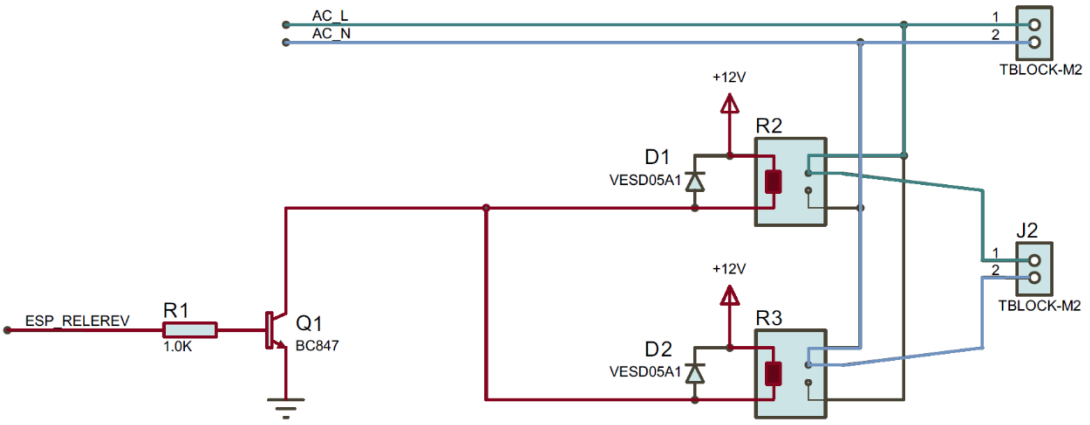


***Imagen 25 - Circuito de cambio de sentido del motor con los relés de inversión activados**. Se observa que las líneas del conector J2 se han invertido respecto al circuito anterior.*

Tanto si se recibe la petición del cliente como si se detecta un estado alto en la entrada del pulsador correspondiente, el microcontrolador activará los relés de alimentación y de cambio de sentido para subir la persiana y solamente el de alimentación para bajarla. El accionamiento manual de la persiana se llevará a cabo a través de tres pulsadores: uno para subir la persiana, otro para bajarla y un tercero para calibrar su altura debido al método utilizado para conocer su posición, que se explicará en el siguiente apartado.

4.6.1. Control de posición

Ante la dificultad y coste de colocar sensores de presencia en el carril de la persiana o encoders en el motor de terceros, el control de posición se realizará completamente por software. El código de "*Xana_ ServerPersiana.cpp*" constará en el *anexo 26* de este proyecto.

Para trabajar con distancias sin utilizar sensores, se guardará el tiempo que tarda la persiana en pasar de completamente subida a completamente bajada, t_{100}, como el 100% de su posición, y se trabajará con distancias relativas. Así, si se pretende subir la persiana un 30% respecto a su altura total, solamente habrá que activar el motor durante $0{,}3 \cdot t_{100}$ segundos. Además, se guardará en todo momento el valor de la posición de la persiana como el tiempo que debería estar en marcha el motor para bajarla completamente. Por ejemplo, si t_{100} fueran 10 segundos y la persiana estuviese subida al 40% de su altura total, la posición de la persiana se guardaría como $t_h = 4$ segundos. Para que esto sea posible, el proceso seguido será el siguiente:

- **Al iniciarse** el dispositivo, el ESP32 leerá de su memoria de programa (no volátil) el último tiempo de altura total, t_{100}, y el último tiempo de altura actual, t_h, guardados en la EEPROM.
- Si no se han configurado nunca estos parámetros o si la persiana ha cambiado de altura con el controlador desconectado, el usuario deberá realizar una **calibración** de la siguiente manera:
    - Se subirá la persiana hasta su altura máxima manteniendo pulsado el botón de subir.
    - Se pulsará el botón de configuración y, sin soltarlo, se bajará la persiana hasta su altura mínima manteniendo pulsado el botón de bajar.
    - Cuando la persiana esté completamente bajada, se soltarán ambos botones, con lo que el microcontrolador guardará en su programa el tiempo que ha tardado en pasar del 100% al 0% como t_{100} y la posición actual, t_h, como 0 segundos.
- **Cuando se suba** la persiana, se sumará a t_h el tiempo que el motor ha estado en marcha.
- **Cuando se baje** la persiana, se restará a t_h el tiempo que el motor ha estado en marcha.
- **Si t_h es mayor que t_{100} o menor que 0** porque se ha pulsado el botón durante más tiempo que el que la persiana ha tardado en llegar a cualquiera de sus extremos, t_h se igualará a t_{100} o a 0, en función de la operación que estuviera realizando. Esto puede ocurrir debido a que los motores de persiana tienen un controlador que detendrá el giro en la altura máxima o mínima configuradas en su instalación, aunque sigan recibiendo corriente.
- **Al finalizar cualquier operación**, se guardará la altura t_h en la EEPROM para que cuando el dispositivo se reinicie no haga falta volver a configurarlo.

4.6.2. Control automático

Se utilizará el mismo LDR que en el *apartado 4.4.3.* sobre las luces para bajar completamente la persiana cuando se detecte un nivel de luz considerado como oscuro. Esto equivaldría a una entrada analógica menor de 1 V.

4.7. Control de temperatura

Este actuador tiene como finalidad controlar un aire acondicionado común emulando los controles de su mando a distancia.

Para **copiar los códigos** IR del mando original, se pulsará el botón de configuración y un *buzzer* emitirá dos tonos de frecuencia ascendente. El ESP32 leerá y guardará los códigos del mando gracias a un receptor IR en el siguiente orden: encender/apagar, subir temperatura, bajar temperatura y modo swing. El zumbador emitirá un pitido cada vez que se registre un código y tres cuando se complete la configuración. Si pasados unos segundos no se ha recibido nada, se emitirán dos tonos de frecuencia descendente.

Los códigos se almacenarán en la **memoria EEPROM** tras la configuración y se leerán al inicio del programa, igual que en el controlador del apartado anterior. El código de "*Xana_ ServerTemperatura.cpp*" constará en el *anexo 27* del proyecto.

4.8. Sistema de sonido

Este sistema, que formará parte del controlador principal, podrá reproducir archivos de audio almacenados de manera local en una tarjeta microSD y guiar al usuario por los menús de voz mediante respuestas de voz almacenadas también en la tarjeta microSD. Se han diseñado dos variantes para el hardware del controlador, una más simple y de bajo coste y otra que ofrece mayores prestaciones de calidad de audio y ganancia. El volumen se controlará por software.

4.8.1. Estudio de potencia

Como se explicará más adelante en el *apartado 4.9.* de este documento, la fuente de alimentación escogida es una fuente de 12 V con una corriente máxima de 2 A. De manera teórica, se ha calculado el consumo máximo del circuito sin el altavoz sumando el consumo individual de todos sus componentes, basándonos en sus *datasheets* o, en el caso del VRM, en mediciones experimentales. Este consumo de microcontroladores, amplificadores, reguladores de tensión, resistencias, etc. se estima en aproximadamente 250 mA. Esto significa que con todos los componentes del controlador consumiendo su máxima potencia, la fuente podría ofrecer al altavoz hasta 1,75 A de corriente, lo que a 12 V supondría una potencia máxima de 21 W.

Los datos de consumo máximo utilizados han sido los siguientes:

- ESP32-WROOM-32: 138 mA.
- Arduino nano: 35 mA.
- VRM: 16 mA.
- ICS40180: 0,2 mA.
- TL081: 1,4 mA.
- Circuito de amplificación de la salida de audio: máximo 50 mA.
- LM11173V3: 5 mA.
- Adaptador de microSD: 1 mA.
- Resto de componentes: aproximadamente 10 mA.

Sin embargo, realizando medidas experimentales, como se ilustra en la i*magen 26*, el consumo medido no salía del rango entre los 66 mA y los 70 mA.

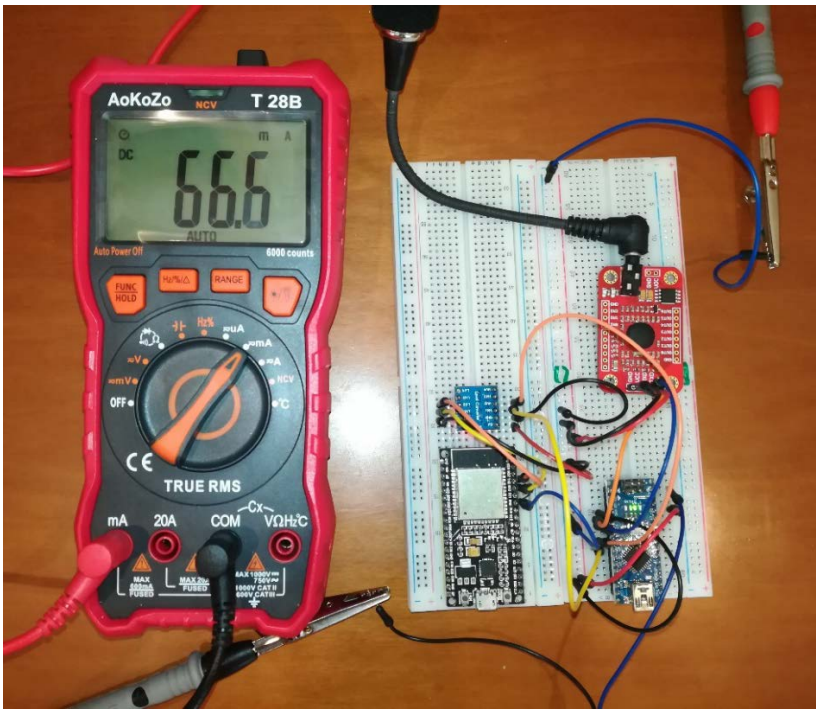


Imagen 26 - Medición experimental del consumo del controlador principal. *El multímetro digital, que actúa en este caso de amperímetro, está conectado en serie entre el terminal negativo de la alimentación de 5V y la línea de masa del circuito.*

4.8.2. Circuito de amplificación de audio (variante 1)

Para esta variante, el circuito de amplificación desde el ESP32 hasta el altavoz es muy similar al utilizado en la amplificación del micrófono mostrado en la *imagen 13* del *apartado 4.2.2*, utilizando los valores de condensadores y resistencias sugeridos por el fabricante, tal y como se muestra en la i*magen 27*. En este caso, el amplificador operacional utilizado ha sido el LM1875, con características de baja distorsión, bajo ruido y salida mono con un offset de 1 V. El control de volumen se realizará por software mediante la entrada digital de dos botones (uno para aumentarlo y otro para disminuirlo).

En este caso, la salida de audio desea amplificarse desde los 3,3 V hasta los 11 V que ofrece como salida máxima el LM1875. Por lo tanto, la ganancia máxima será:

$$G_{máx} = \frac{V_{out}}{V_{in}} = \frac{11}{3{,}3} \cong 3{,}33\ {}^{V}/_{V}$$

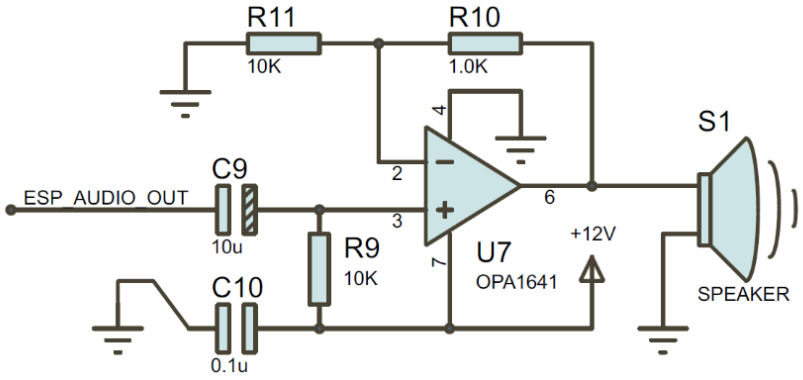


***Imagen 27 - circuito de amplificación de audio** de ganancia regulable.*

Las resistencias normalizadas que más se acercan a este valor son 51 kΩ para *R11* y 22 kΩ para *R10*. Por lo tanto, la potencia máxima que se podrá ofrecer al altavoz será:

$$G = \frac{R_{11} + R_{10}}{R_{11}} = \frac{22 + 51}{22} \cong 3{,}32\ {}^{V}/_{V}$$

$$V_{out_{máx}} = V_{in_{máx}} \cdot G \cong 3{,}3 \cdot 3{,}32 \cong 10{,}96\ V$$

$$P_{máx} = V_{out_{máx}} \cdot I_{in_{máx}} \cong 10{,}96 \cdot 1{,}75 \cong 19{,}17\ W$$

Por lo que el altavoz elegido será de una resistencia mínima de:

$$P_{máx} = \frac{{V_{out_{máx}}}^2}{R_{mín}} \rightarrow R_{mín} = \frac{{V_{out_{máx}}}^2}{P_{máx}} \cong \frac{11^2}{19{,}17} \cong 6{,}31\ \Omega$$

Los altavoces comerciales más comunes son de 4 Ω y 8 Ω, aunque también los hay de 2 Ω, 6 Ω, 16 Ω y hasta 32 Ω para aplicaciones muy específicas. El altavoz escogido, por lo tanto, será un **altavoz de 8 Ω** de rango completo, cuyo consumo máximo será de:

$$P_{máx} = \frac{{V_{out_{máx}}}^2}{R} = \frac{11^2}{8} \cong 15{,}125\ W$$

$$I_{máx} = \frac{V_{out_{máx}}}{R} = \frac{11}{8} \cong 1{,}375\ A$$

4.8.3. Circuito de amplificación de audio (variante 2)

Para la amplificación en esta opción, se ha utilizado el amplificador de audio de clase D de alto rendimiento TAS2770, que ofrece una salida diferencial flotante de hasta 15 W alimentado a 12 V. Este amplificador profesional es el que utilizan dispositivos como los Echo Dot de Amazon debido a su alta calidad de sonido y a su configurabilidad mediante comandos hexadecimales vía I^2C. Las señales de audio se envían en formato PDM (*Pulse Density Modulation*) o mediante su interfaz I^2S.

Entre las posibilidades que ofrece están la entrada de múltiples fuentes de audio, la configuración de ganancia por software, la amplificación de micrófono y distintos modos de funcionamiento para que sea más energéticamente eficiente. En la *imagen 28* se observa la configuración escogida para este dispositivo. En ésta, las entradas se han reducido de 3,3 V a 1,81 V, su voltaje de funcionamiento, mediante un divisor resistivo, y en el canal SDA de la interfaz I^2C se ha implementado un adaptador de nivel similar al mostrado en la *imagen 16* del apartado 4.3.1., mostrado en la *imagen 29*.

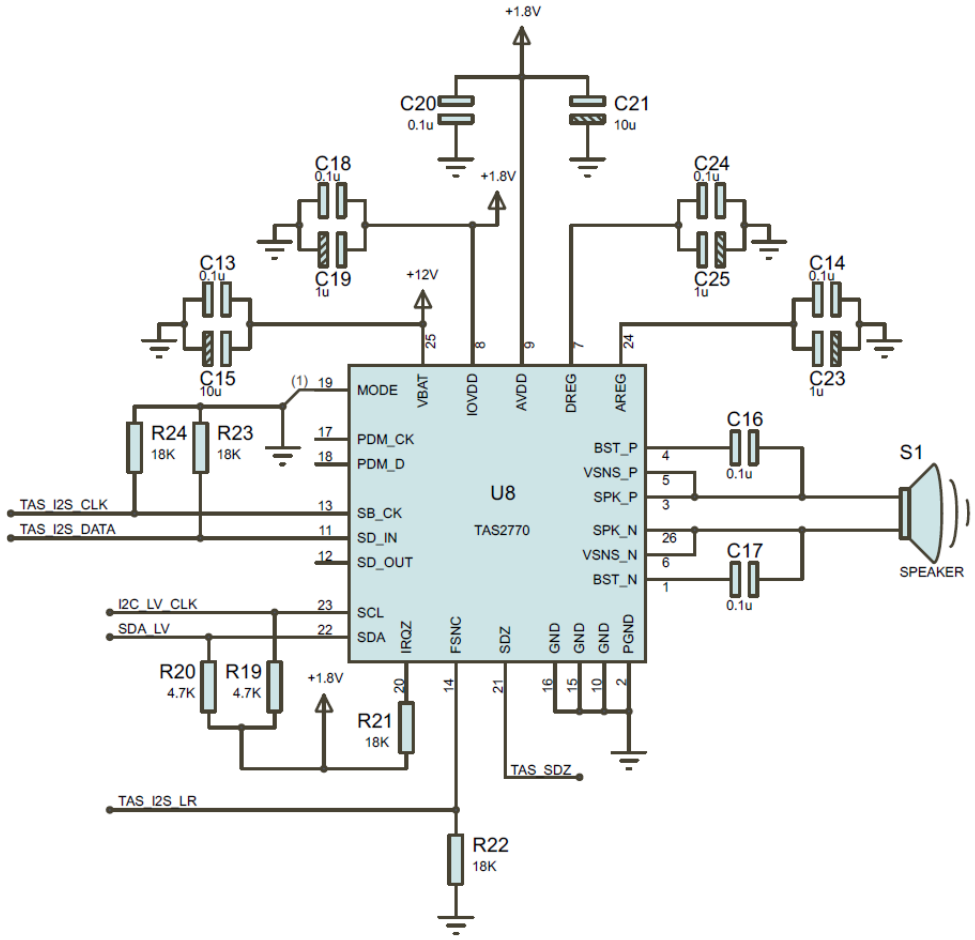


Imagen 28 - Amplificador de audio de clase D de alto rendimiento TAS2770 *configurado para entrada de audio por I^2S y salida diferencial flotante de ganancia, volumen y modo de funcionamiento configurables por I^2C.*

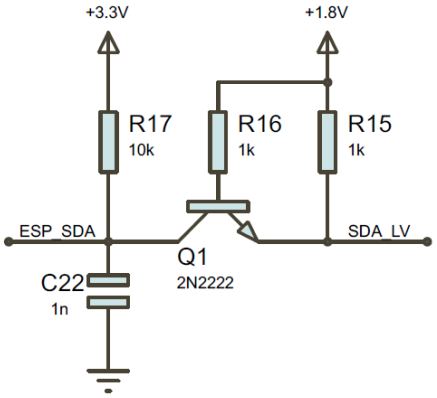


***Imagen 29 - Adaptador de nivel bidireccional con un transistor NPN**. La función del condensador C22 es suavizar los picos de subida de la señal de menor voltaje para minimizar el riesgo de falsas lecturas.*

Puesto que el componente principal de este circuito no puede conectarse a una protoboard, debido a su pequeño tamaño y a su encapsulado sin pines de 4 mm x 3,5 mm, a la hora de montar el prototipo, crear el código y realizar el presupuesto, se tendrá en cuenta únicamente la variante 1 del circuito de amplificación de audio. Esta variante, sin embargo, sería más interesante en futuras versiones del producto.

4.8.4. Entrada de audio

Las tarjetas **microSD** funcionan a un voltaje de 3,3 V, por lo que no se requiere ningún adaptador especial para comunicarse con ellas, y la transmisión de datos se realiza mediante el canal SPI. La librería "SD.*h*" de Arduino simplifica la comunicación SPI mediante comandos para navegar entre sus ficheros. La librería "*TMRpcm-master.h*" interpreta los ficheros en formatos digitales compatibles con la modulación de audio PCM (*Pulse Code Modulation*), como son el MP3 o el WAV, en impulsos analógicos para enviarlos al amplificador de la variante 1 del circuito de audio. No se ha utilizado la salida DAC del ESP32, ya que es de 8 bits y una buena calidad de audio no se consigue con resoluciones menores a los 16 bits. En caso de usar la variante 2, la librería "*Audio.h*", se encargaría de transformar estos formatos en señales sincronizadas con un reloj para mandarlo al TAS2770 mediante su interfaz I^2S.

En la tarjeta microSD se encuentran 4 **carpetas** a las que accederá el ESP32: una carpeta oculta "*.respuestas*" con las frases de voz grabadas para que Xana responda al usuario al realizar cada acción y sonidos que lo guíen por los menús de voz, y tres carpetas con música, ambiente, fiesta y relax, en las que el usuario guardará las canciones en formato WAV. Cada carpeta podrá contener hasta 65.535 archivos de audio, numerados de manera ascendente a partir del 1. Al entrar en una de las carpetas de música, Xana reproducirá de manera aleatoria un archivo entre el "00001.wav" y el "65535.wav", por ejemplo, el "00021.wav". Si este archivo no existe, se supondrá que tampoco existen los nombrados con un número mayor y se volverá a buscar un archivo entre el 00001 y el 00020. Si no se encuentra ningún archivo de audio, se reproducirá un mensaje de error de la carpeta de respuestas.

La *imagen 30* muestra el proceso de transmisión de audio desde la tarjeta SD hasta el altavoz, incluyendo ambas variantes del circuito de amplificación.

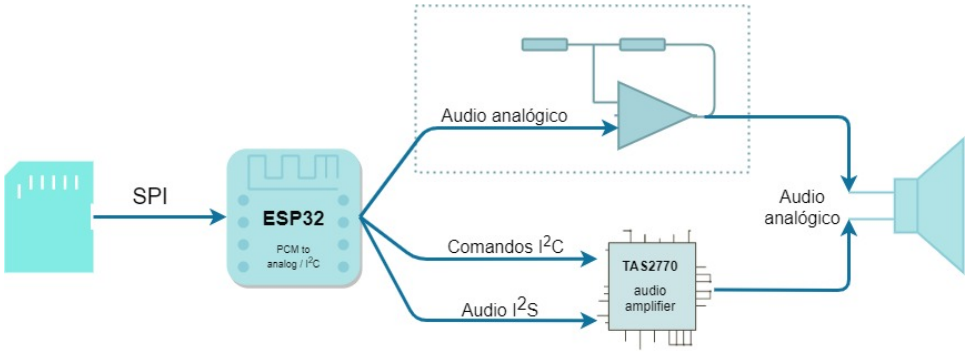


***Imagen 30 - proceso de transmisión de audio** desde la tarjeta SD hasta el altavoz*

4.9. Alimentación

Se han estudiado las necesidades de alimentación de los diferentes módulos de Xana, tomando como genéricos algunos datos del estudio de potencia realizado en el *apartado4.8.1.*, y se han agrupado en tres categorías distintas: los módulos de alimentación de potencia, los módulos con salidas de corriente alterna y el módulo de requerimientos moderados.

- El controlador principal requiere una tensión de **12 V** para el sistema de sonido y consumirá una potencia máxima de casi **1,5 A**.
- El controlador de las luces requiere **12 V** para cada una de sus tres salidas (R, G y B) y una potencia de, como se indica en el *apartado 4.4.1.*, hasta 15 W por metro de tira LED. Esto supone 1,25 A más los, redondeando, 150 mA del resto de componentes, **1,4 A**.
- El controlador de enchufes consume los 150 mA de los componentes generales más 320 mA de los cuatro relés, un máximo de **470 mA**. Requiere conexión directa con la alimentación alterna de **230 V** para sus salidas.
- El controlador de motor de persianas consume los 150 mA de los componentes generales más 140 mA de los dos relés, un máximo de casi **300 mA**. Requiere conexión directa con la alimentación alterna de **230 V** para sus salidas.
- El mando de temperatura no tiene componentes propios que requieran más potencia que la media, por lo que su consumo no superará los **0,15 A**. Ninguno de sus componentes necesita una tensión de alimentación mayor a **3,3V**.

4.9.1. Módulos de alimentación de potencia

Se incluyen los módulos del controlador principal y el controlador de luces. Para éstos, se utilizará un adaptador de tensión AC/DC con una entrada de corriente alterna de 230 V (el voltaje típico de la red eléctrica en la mayoría de países de Europa y Asia) y

una salida de corriente continua de 12 y hasta 2 A de corriente. La conexión de salida del adaptador será una clavija DC Jack de 5,5 mm x 2,5 mm, una de las más comunes para este tipo de fuentes de alimentación. La *imagen 31* muestra un adaptador de estas características.

***Imagen 31 - Adaptador AC/DC** típico de 2 A con clavija Jack de 5,5 mm x 2,5 mm.*

4.9.2. Módulos con salidas de corriente alterna

Este grupo lo conforman el controlador de enchufes y el del motor de persianas. Los requerimientos de entrada y salida de una línea de corriente alterna imposibilitan el uso de un adaptador de pared como en el caso anterior. La solución de diseño ha sido la implementación de un transformador que, al igual que el resto del circuito, deberá estar correctamente aislado para proteger al usuario de descargas eléctricas. En las *imágenes 32 y 33* se muestra el diseño electrónico de un transformador comercial como el utilizado para el grupo anterior.

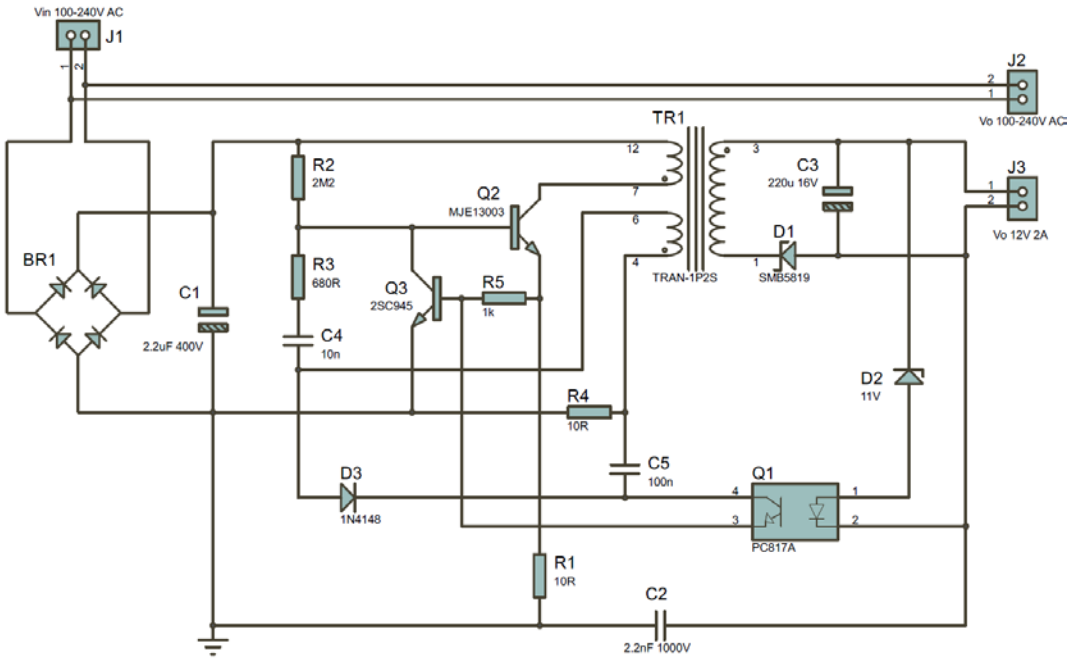


***Imagen 32 - Diseño de la electrónica de un transformador de pared comercial** con salida de 12 V y 2 A. Este modelo protege el circuito de salida de la alta potencia de la red eléctrica mediante un optoacoplador.*

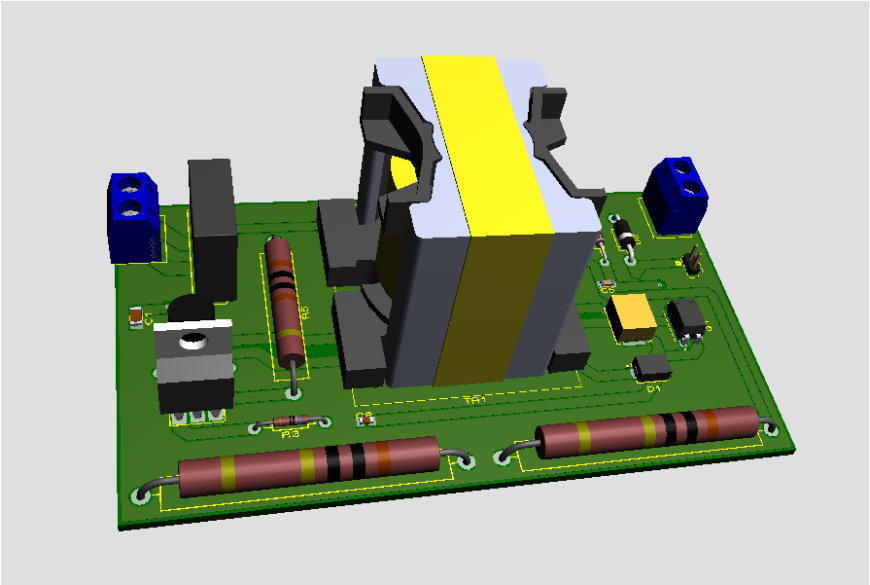


***Imagen 33 - Modelo 3D del transformador comercial** anterior en una PCB de 100 mm x 58 mm.*

El transformador diseñado para los dos módulos de este grupo contará con una salida de corriente alterna, conectada directamente a la entrada, para las salidas de los relés y una salida de corriente continua para alimentar los circuitos de control. La tensión de salida DC se ha elegido de 12 V para simplificar el diseño del conjunto de módulos que componen Xana, aunque podría disminuirse a 9 V o a 7,5 V redimensionando el relé JW2HN-DC9V a un JW2HN-DC6V. La salida DC será de 0,5 A para el módulo de la persiana y de 1 A para el módulo de los enchufes. Se elige el siguiente valor normalizado a 0,5 A pese a que el consumo máximo es de 0,47 A ya que, según el Datasheet del ESP32, el microcontrolador puede tener picos momentáneos de consumo y reiniciarse si la fuente no es capaz de ofrecerlos.

4.9.3. Módulo de requerimientos moderados

Puesto que este módulo no necesita alimentarse a más de 3,3 V, se utilizará un conector micro USB como entrada de corriente. Esto favorecerá la integración del módulo con el entorno haciendo posible conectarlo a cualquier dispositivo con salida USB. El consumo no constituye un criterio de diseño en este caso, pues la mayoría de dispositivos con salida USB hoy en día tienen una intensidad de salida máxima que, como mínimo, ronda los 1.000 mA.

4.10. Aspectos adicionales de la solución adoptada.

4.10.1. Diseño del hardware

En este apartado se pretende explicar o matizar algunos aspectos sobre el conjunto de la solución adoptada que no entraban del todo en el planteamiento de los otros puntos.

A medida que se implementen nuevas funcionalidades en el controlador principal, el tamaño del firmware irá aumentando. Para evitar que la falta de memoria de programa sea un problema en futuras versiones de Xana, se ha añadido una **memoria Flash externa** del tipo Quad-SPI al controlador principal, que no será necesaria en esta versión del producto. Este componente corresponde al circuito integrado U7 del documento número 2: planos.

La configuración utilizada en los planos para los cinco módulos **ESP32-WROOM-32** es la configuración típica del módulo, a la que se ha añadido un puerto físico conectado a su interfaz UART0 para poder configurarlo de forma externa mediante un driver USB como el CP2102. También se ha añadido un condensador entre el pin 3 y masa para que a la hora de programarlo entre automáticamente en el modo “boot” sin necesidad de pulsar ningún botón.

A la hora del montaje, los **condensadores de desacoplo** deberán situarse lo más cerca posible del pin al que están asignados en los planos.

Todo el **dimensionado de componentes**, así como su elección y configuración, se ha realizado en base a recomendaciones del fabricante o diseños de referencia de otros fabricantes comerciales. La elección de cada componente en particular se ha realizado con criterios económicos y de eficiencia energética entre grupos de componentes en stock y características similares en las páginas web de Mouser, Farnell y DigiKey, entre otras.

Los **pulsadores** utilizados en los planos cuentan con un condensador conectado en paralelo para evitar falsas lecturas debidas a los rebotes eléctricos. Como medida extra, se ha implementado un pequeño delay (80 ms) por software cada vez que se registra un nivel alto a la entrada de un pulsador.

El **Arduino Nano** funciona a una tensión nominal de 5 V, pero puede alimentarse a voltajes de 7 V a 12 V mediante el regulador de voltaje que lleva incorporado. Al ser un módulo comercial completo, no requiere la conexión de condensadores de desacoplo, pines o reguladores de tensión externos.

4.10.2. Otros aspectos

En el documento 3: presupuesto, los **transformadores AC/DC** mostrados en el *apartado 4.9.2.* de este documento se contabilizarán como transformadores comerciales comprados a terceros como los del *apartado 4.9.1.* en lugar de presupuestar cada componente por separado.

Las **librerías externas** utilizadas en el software no se añadirán como anexos al no haberse diseñado en este proyecto

El *anexo 29* del proyecto muestra la **organización** por carpetas de todos los archivos que conforman **el firmware** de cada módulo.

4.10.3. Modelos 3D

Las *imágenes 34 y 35* muestran el **modelo 3D** de las PCB de cada uno de los módulos del proyecto. Estos modelos, al igual que el mostrado en la *imagen 33* no son definitivos, ya que se basan en una versión anterior de los circuitos de Xana y el trazado de pistas lo ha realizado de manera automática el motor ARES de Proteus, sin tener en cuenta criterios como: acortar al mínimo posible las distancias de las pistas de reloj, dimensionar las pistas de mayor potencia, separar las pistas de señal de las de alimentación para evitar capacitancias no deseadas, etc.

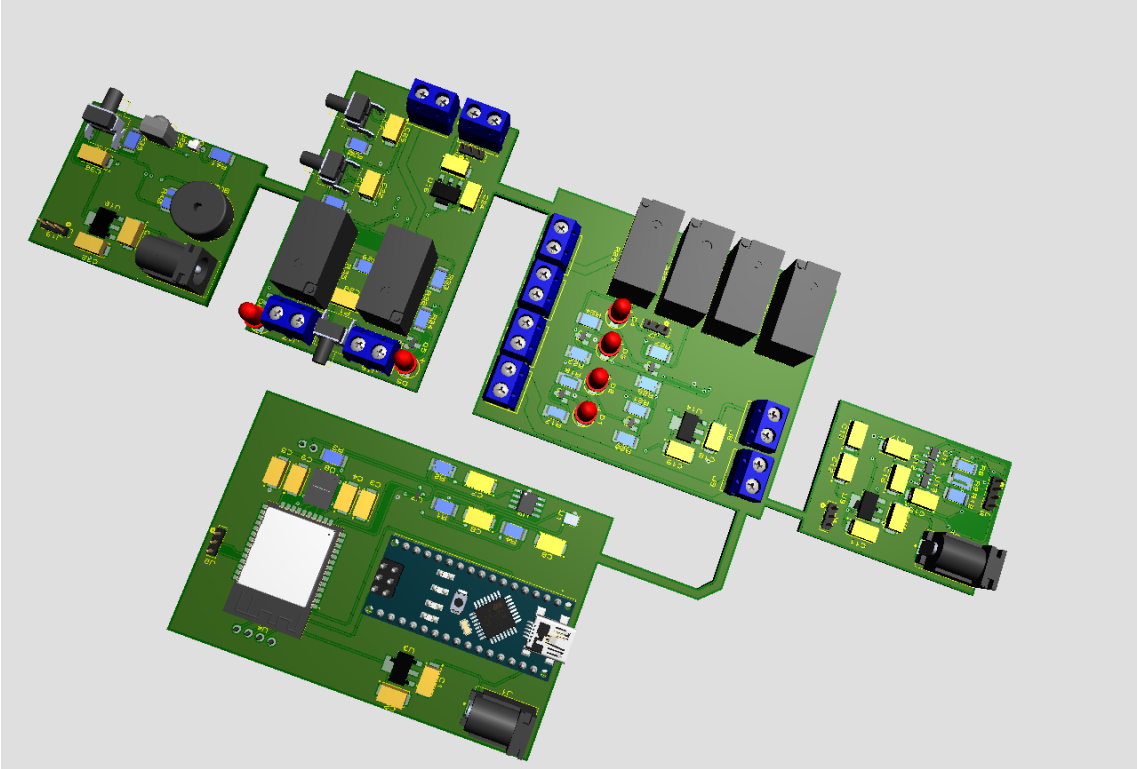


***Imagen 34 - Modelo 3D de una versión previa de los 5 módulos de Xana**, vista superior.*

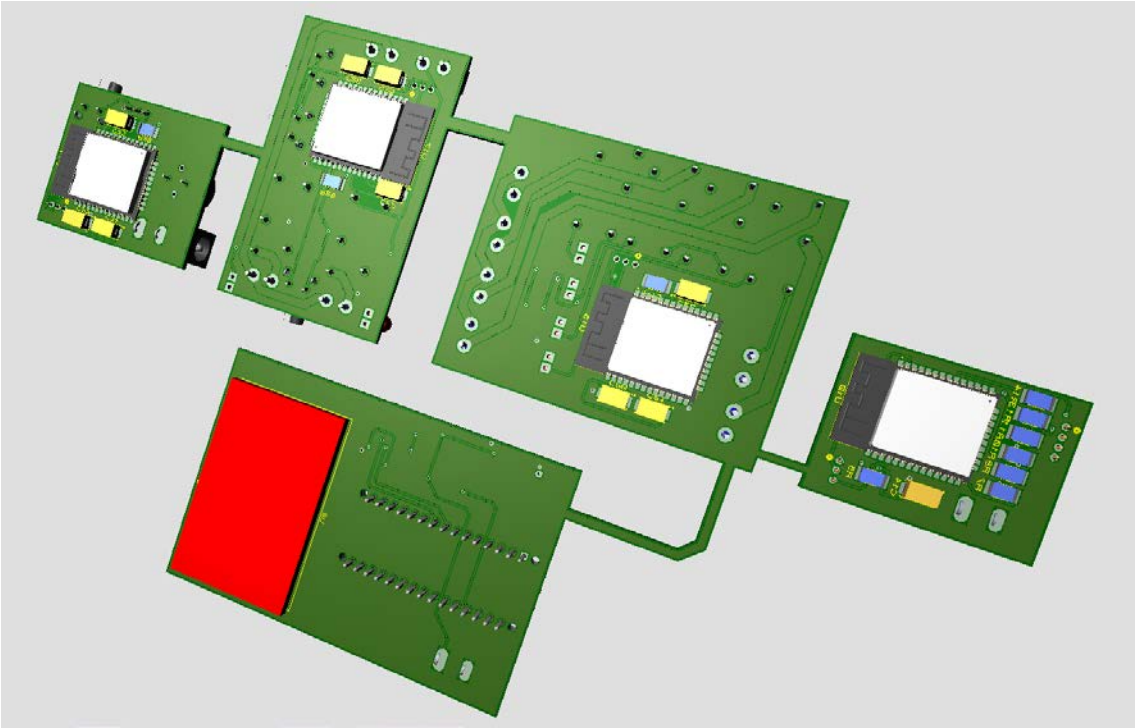


***Imagen 35 - Modelo 3D de una versión previa de los 5 módulos de Xana**, vista inferior.*

5. Conclusiones

Se ha desarrollado un **controlador domótico de bajo coste controlado por voz** y se ha comprobado el correcto funcionamiento de todos sus comandos. El controlador es capaz de conectarse vía wifi y enviar peticiones http a sus servidores remotos. Estos servidores son capaces de recibir peticiones http de cualquier cliente y actuar en función de éstas. Se han logrado, además, todos los objetivos técnicos y específicos del proyecto definidos en los *apartados 2.1. al 2.6.*

El controlador principal puede reproducir archivos de **audio** con una calidad de sonido y volumen suficiente para utilizarse como reproductor de música en una habitación de tamaño estándar. La utilización de un micrófono MEMS de alto rendimiento en vez del micrófono de condensador pasivo aportado por el fabricante logra un rango de detección de instrucciones efectivo suficiente para no tener que desplazarse hasta la posición de Xana para darle órdenes.

El uso de microcontroladores y otros elementos de alto **rendimiento energético** favorece un consumo mínimo de energía y cimenta una domótica sostenible y ecológica, así como un gasto económico de mantenimiento mínimo.

Dada la actual complejidad del proyecto y su tiempo de desarrollo, no se han explotado ciertas características, como la implementación de un reloj en tiempo real para la configuración de alarmas o la capacidad de reproducción de música a través de Bluetooth, aunque sí que se han establecido las bases de software suficientes para su desarrollo en **futuras versiones**. En vista de el tiempo de aprendizaje que conlleva cada aspecto del diseño de un proyecto, para una versión ampliada de Xana sería necesaria la colaboración de un equipo multidisciplinar en el que cada persona pudiese aportar conocimientos técnicos, ideas, etc. al proyecto.

Por último, para el desarrollo y edición de las aproximadamente 5.000 líneas de **código**, ha sido esencial la organización modular de este código en los 26 ficheros adjuntados como anexos y su correspondiente distribución en carpetas. Esto ha dado pie a un código mucho más flexible y eficiente, abierto a modificaciones y fácil de entender y editar. También ha sido muy importante para esto último la correcta documentación del código mediante comentarios en formato Doxygen y el uso de una guía de estilos a la hora de escribir y dar un formato común a todo el código.

6. Webgrafía

- Manual del VRM:
  - https://www.elechouse.com/elechouse/images/product/VR3/VR3_manual.pdf

- Sobre los módulos de Espressif:
  - https://makeradvisor.com/esp32-vs-esp8266/#:~:text=The%20ESP32%20and%20ESP8266%20are,processor%20that%20runs%20at%2080MHz.
  - https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/

- Sobre la comunicación cliente-servidor entre dos ESP32:
  - https://randomnerdtutorials.com/esp32-client-server-wi-fi/
  - https://randomnerdtutorials.com/esp32-access-point-ap-web-server/
  - http://easycoding.tn/index.php/esp32/esp32-local-network/

- Sobre los métodos HTTP en lenguaje Arduino:
  - https://randomnerdtutorials.com/esp32-http-get-post-arduino/

- Sobre la domótica y los asistentes virtuales, características y seguridad:
  - https://openwebinars.net/blog/domotica-estado-actual-del-sector/
  - https://web.archive.org/web/20110626122220/http:/www.ingenieria.org.ar/archivos/Domotica-CIEC.pdf
  - http://www.cedom.es/es
  - https://voicebot.ai/google-home-google-assistant-stats/
  - http://domotica1003.weebly.com/
  - https://www.belden.com/blog/smart-building/a-prediction-for-cabling-standards-in-2019
  - https://revistascientificas.cuc.edu.co/moduloarquitecturacuc/article/view/141/pdf_72
  - https://biblioteca.fundaciononce.es/publicaciones/colecciones-propias/coleccion-accesibilidad/accesibilidad-de-los-asistentes-virtuales
  - https://www.prodigiosovolcan.com/sismogramas/informe-voz/informe_voz_audio_espana.pdf
  - https://medium.com/dish/75-years-of-innovation-calo-cognitive-assistant-that-learns-and-organizes-9f176f2de05d
  - https://lightcommands.com/
  - https://www.kaspersky.es/blog/ultrasound-attacks/17764/
  - https://arxiv.org/abs/1801.01944

- Sobre las conexiones inalámbricas y los estándares:
    - ieee.org
    - https://z-wavealliance.org/
    - https://www.domodesk.com/162-a-fondo-z-wave-sin-cables.html
    - https://www.xataka.com/seleccion/zigbee-z-wave-que-que-se-diferencian-que-marcas-domotica-compatibles
    - https://www.seas.es/blog/automatizacion/que-es-knx/#:~:text=Es%20un%20protocolo%20est%C3%A1ndar%20porque,TP1%20(Par%20trenzado)

- Sobre los protocolos de comunicación alámbricos:
    - https://www.drouiz.com/blog/2018/06/25/uart-vs-spi-vs-i2c-diferencias-entre-protocolos/

- Sobre los LED y la luz:
    - https://www.manchester.ac.uk/discover/news/researchers-discover-when-its-good-to-get-the-blues/
    - https://www.inventable.eu/2011/04/27/anatomia-de-una-leds-string-rgb/

- Sobre los Level Shifters:
    - https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide/all

Documento 2: Planos

Índice de circuitos:

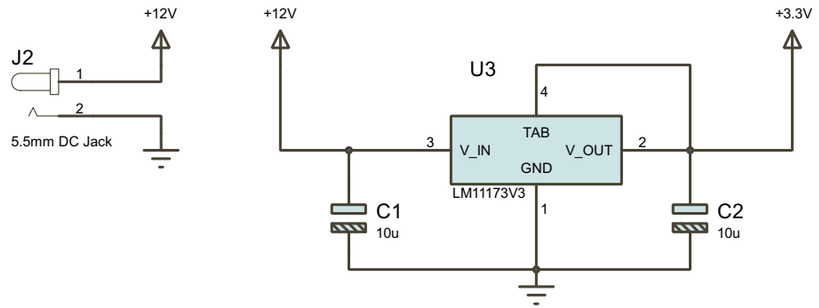
Los siguientes circuitos están pensados para montarse en una placa de prototipos, por lo que no se ha diseñado su PCB. Cada circuito es completamente autónomo e independiente del resto.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

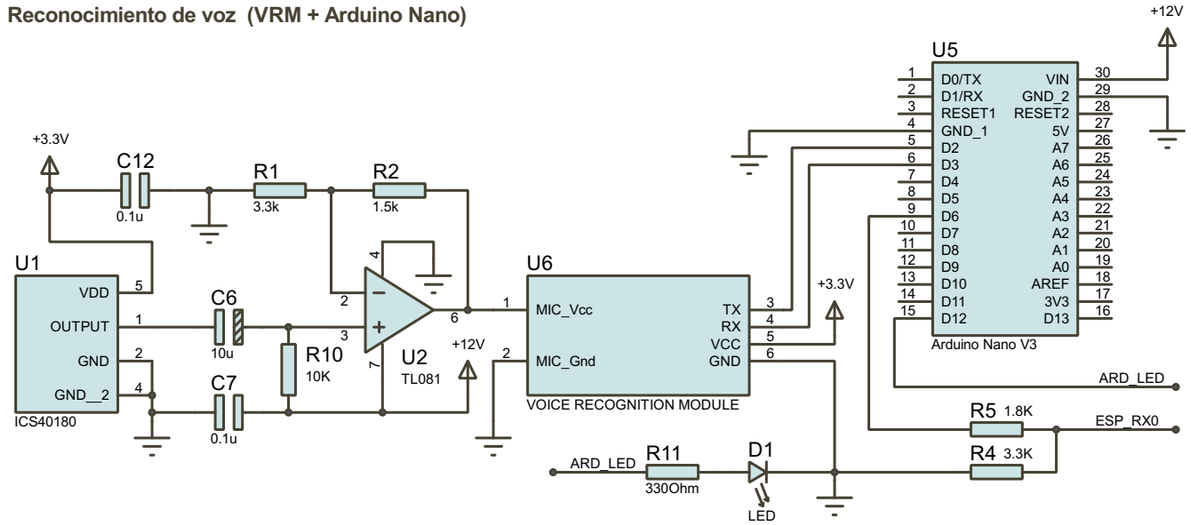
Autor: Jorge Álvarez Pedrón

Proyecto	Fecha
Xana: prototipo de controlador domótico con control por voz (TFG)	31/08/2020
Hoja 00. Índice	**Versión** V1.0

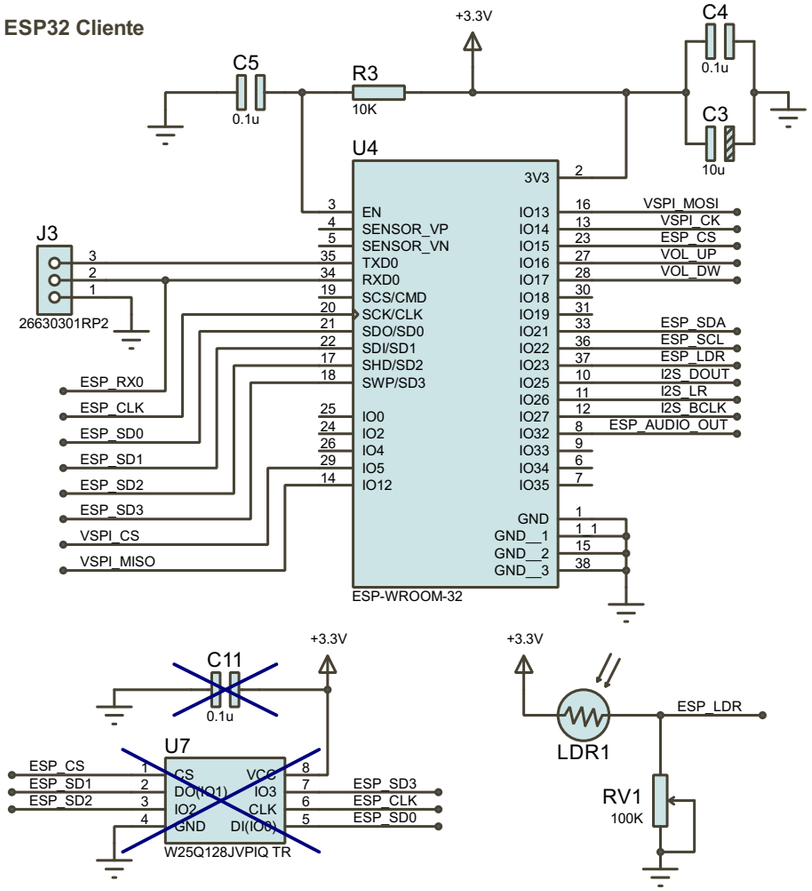
Alimentación



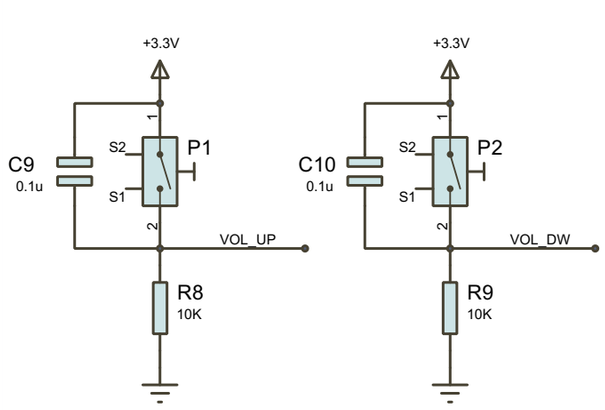
Reconocimiento de voz (VRM + Arduino Nano)

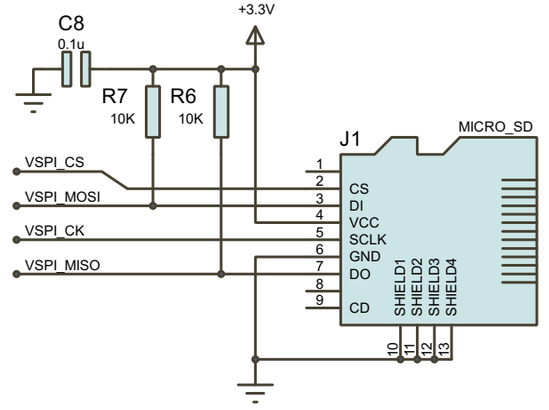


ESP32 Cliente



Pulsadores + Módulo SD

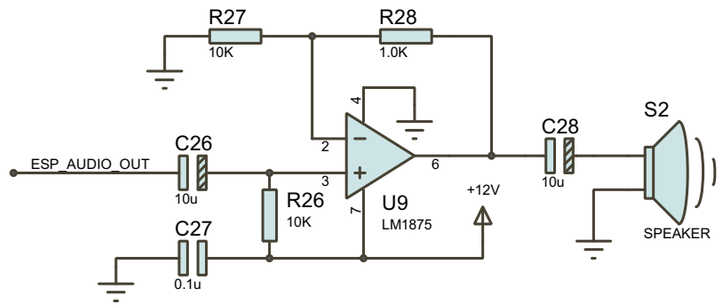




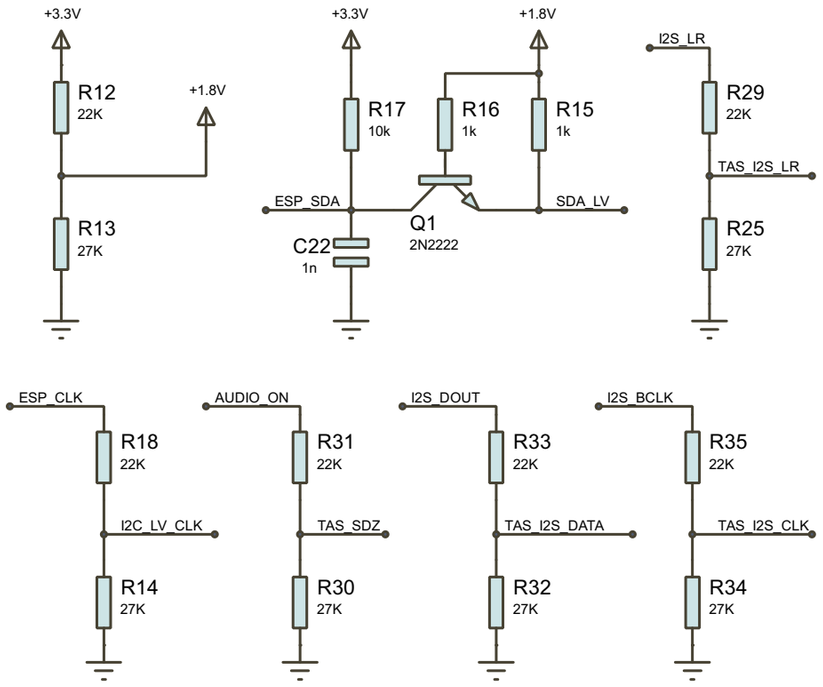
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Autor	Proyecto	Fecha
Jorge Álvarez Pedrón	Xana: prototipo de controlador domótico con control por voz (TFG)	31/08/2020
	Hoja: 01. Controlador principal (1/2)	Versión: V1.0

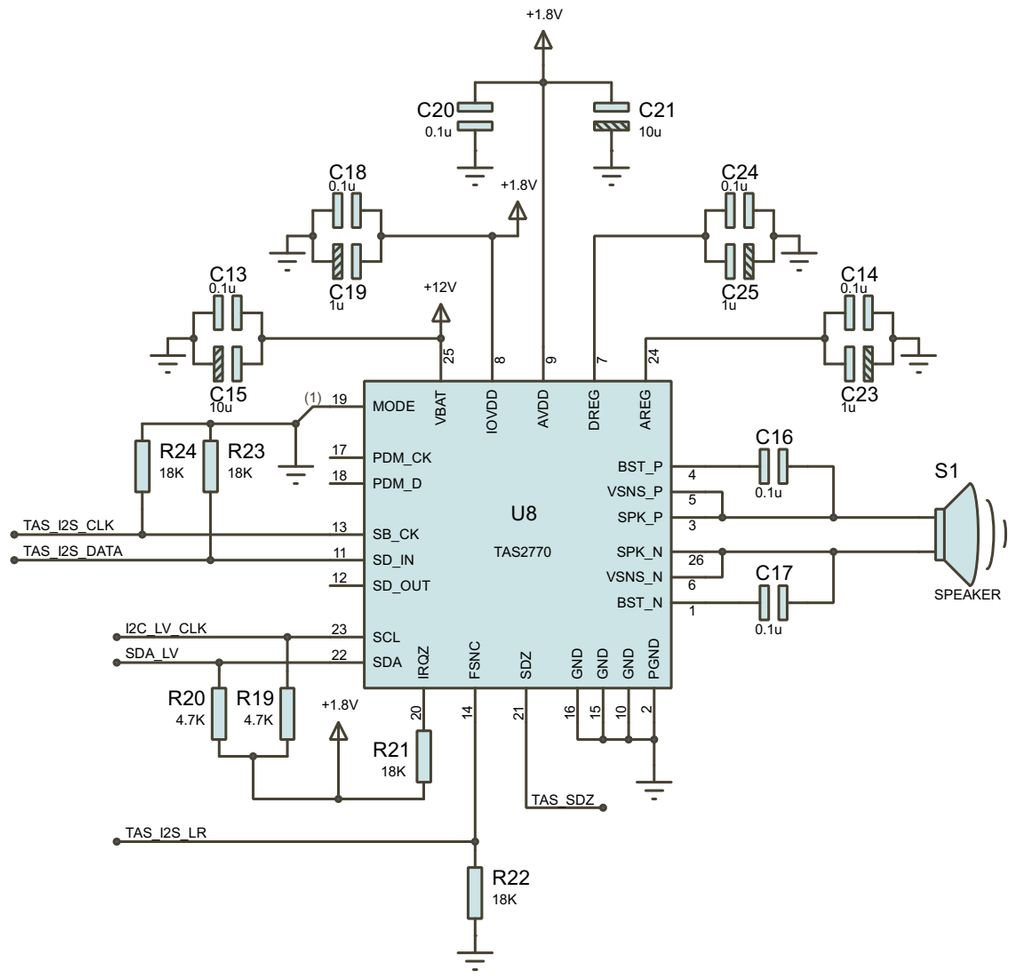
Amplificador de audio (Variante 1)



Adaptadores de tensión (solo para la variante 2)



Amplificador de audio (Variante 2)



(1) I2C Slave adress: 0x82

Nota de diseño: esta variante no se montará en esta versión.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Autor
Jorge Álvarez Pedrón

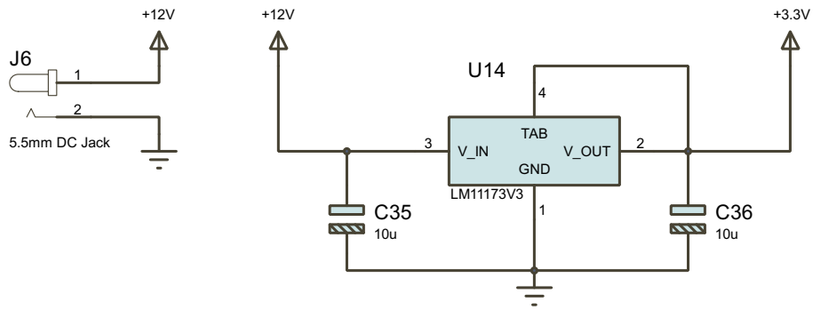
Proyecto
Xana: prototipo de controlador domótico con control por voz (TFG)

Fecha
31/08/2020

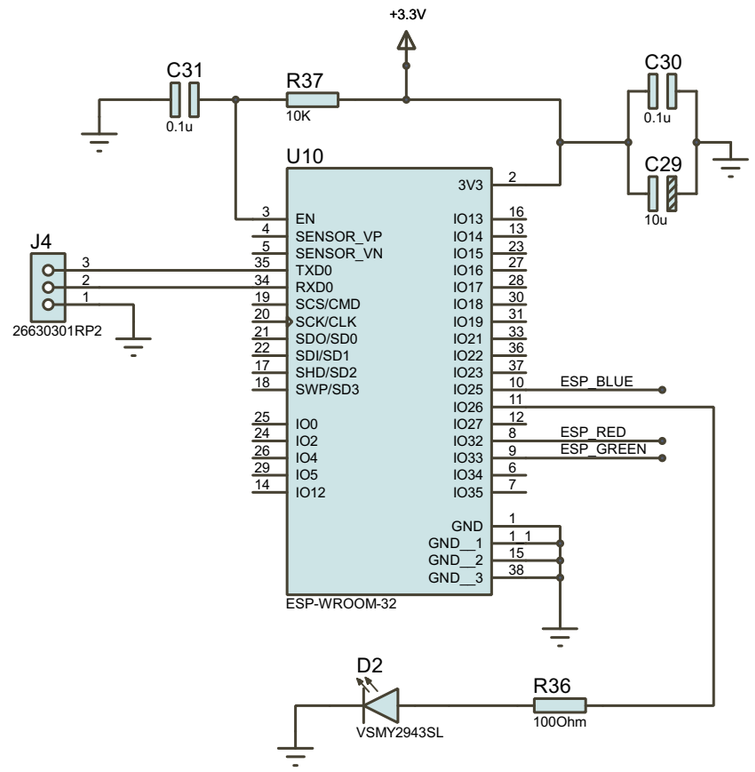
Hoja
01. Controlador principal (2/2)

Versión
V1.0

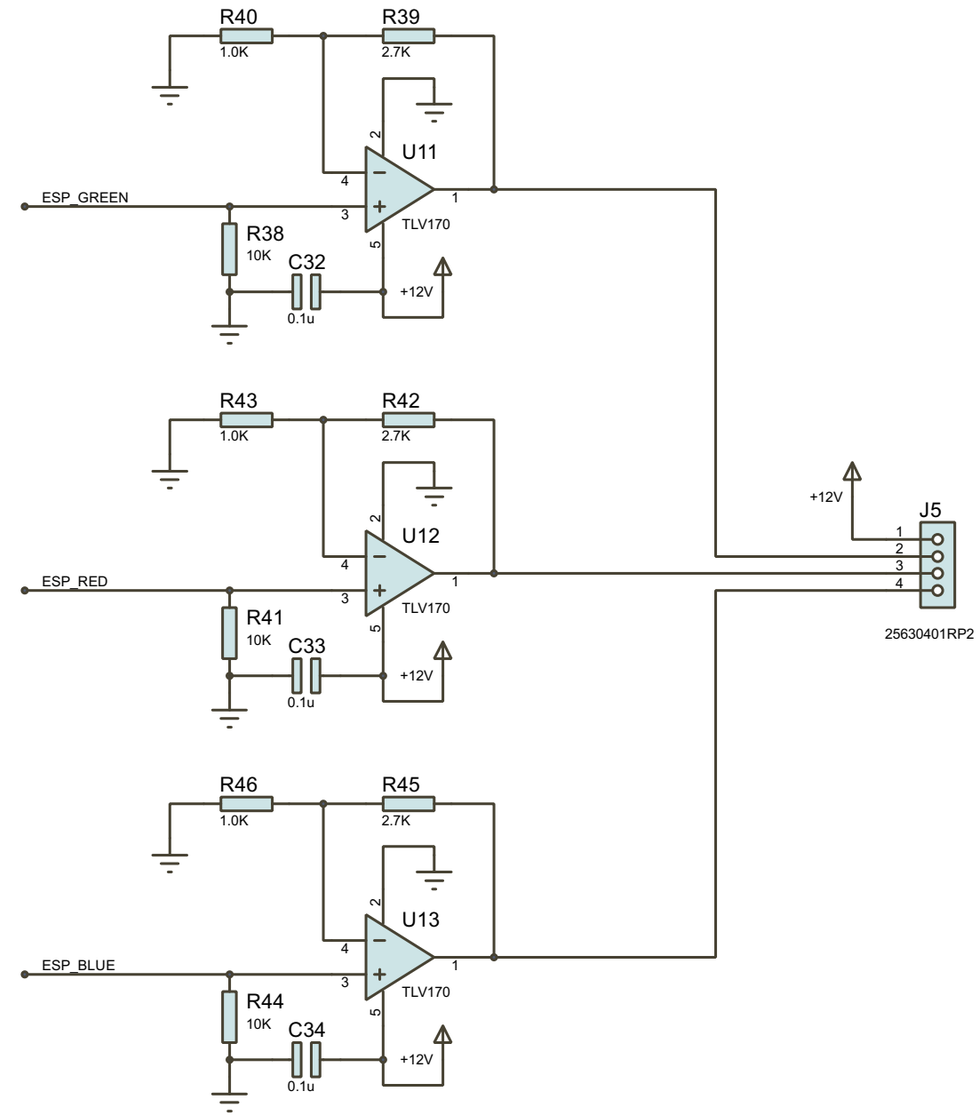
Alimentación



ESP32 Servidor: Luces

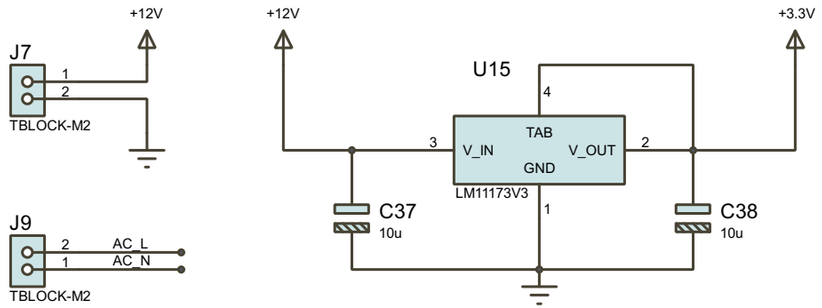


Salida RGB



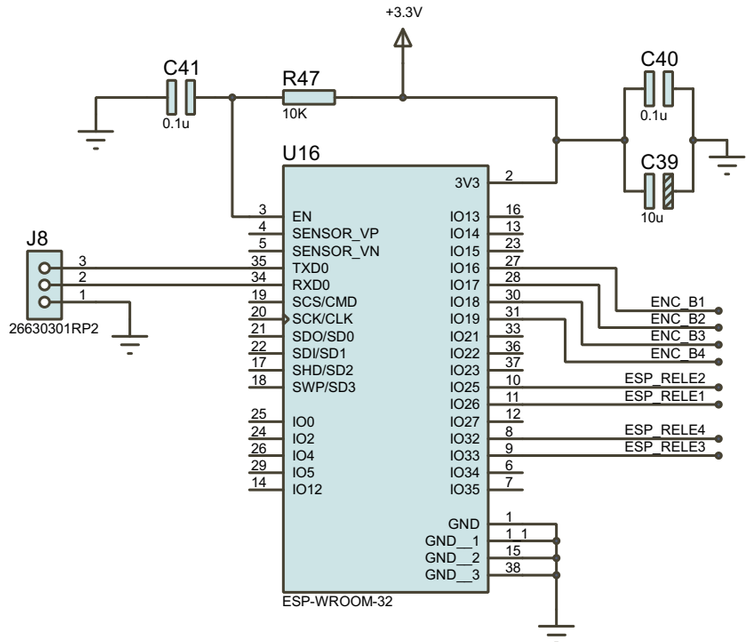
UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Autor: Jorge Álvarez Pedrón	Proyecto: Xana: prototipo de controlador domótico con control por voz (TFG)	Fecha: 31/08/2020
		Hoja: 02. Actuador: Luces	Versión: V1.0

Alimentación

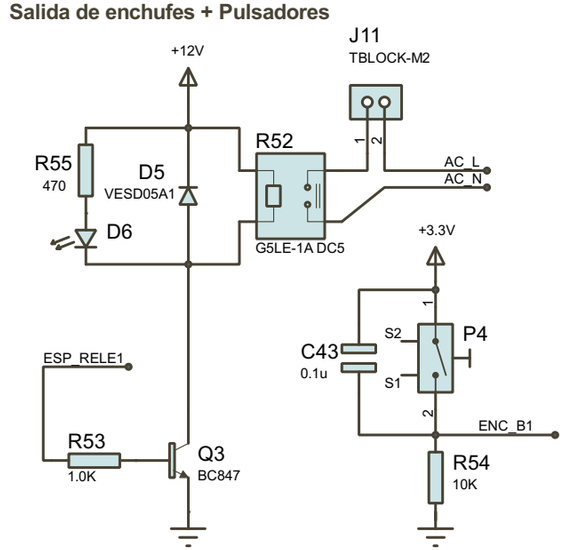


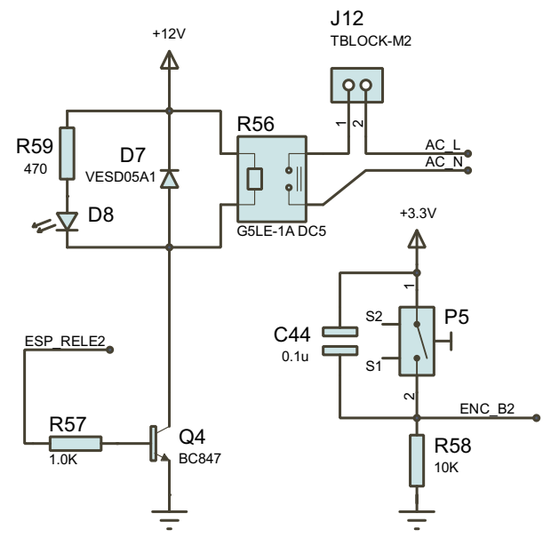
Nota: el transformador de 12V de este diseño está integrado en la carcasa y recibe la alimentación alterna de 230V del conector J9 para proveer a su salida los 12V del conector J7

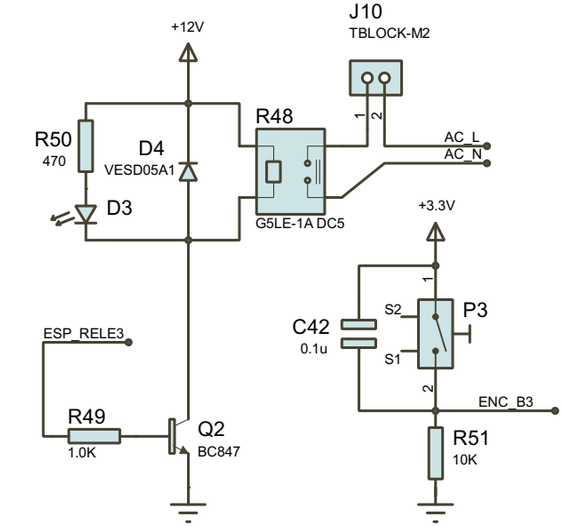
ESP32 Servidor: Enchufes

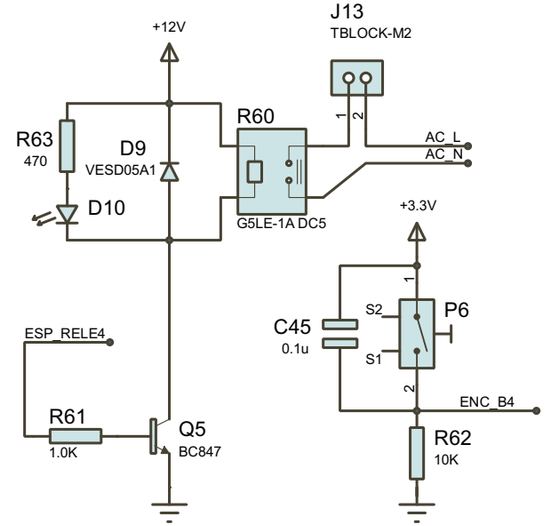


Salida de enchufes + Pulsadores



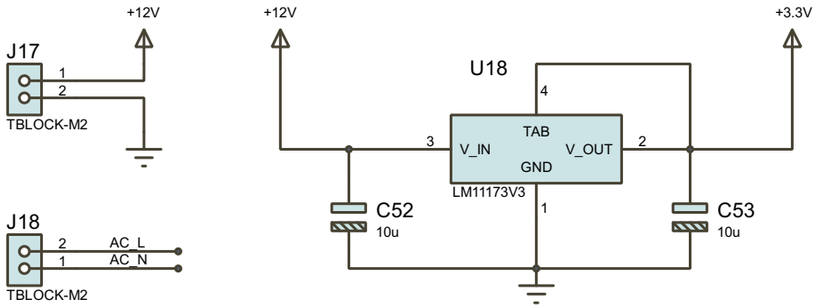






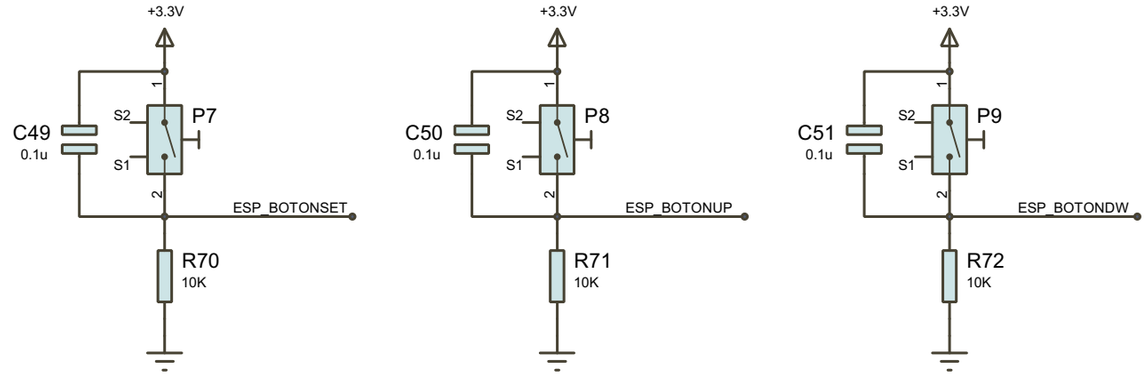
UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Autor: Jorge Álvarez Pedrón	Proyecto: Xana: prototipo de controlador domótico con control por voz (TFG)	Fecha: 31/08/2020
		Hoja: 03. Actuador: Enchufes	Versión: V1.0

Alimentación

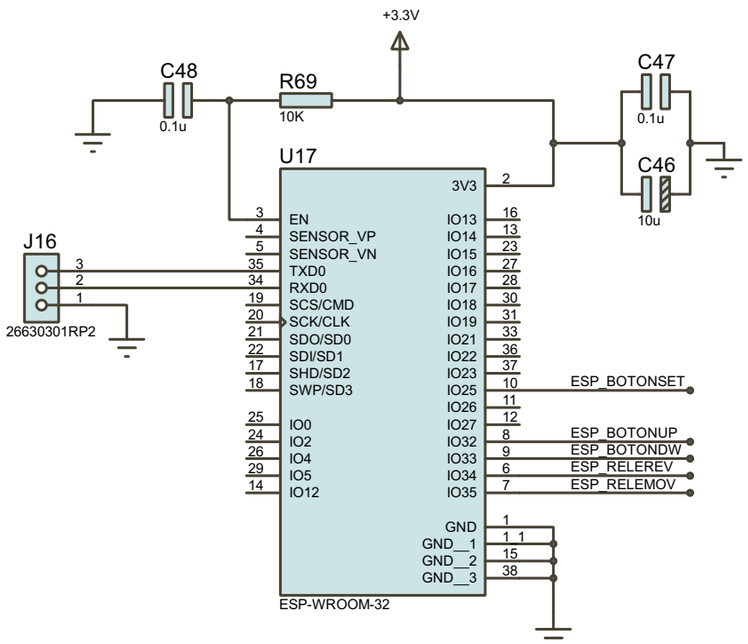


Nota: el transformador de 12V de este diseño está integrado en la carcasa y recibe la alimentación alterna de 230V del conector J18 para proveer a su salida los 12V del conector J17

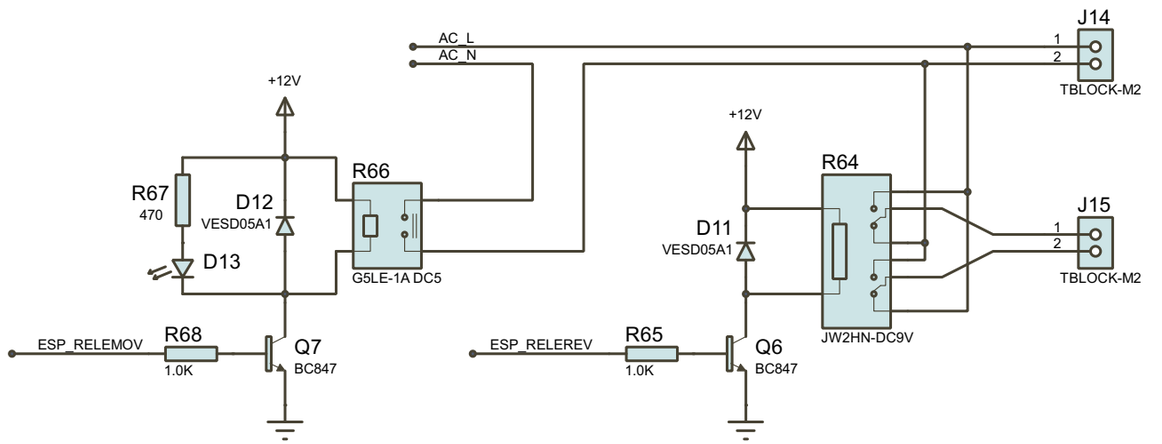
Pulsadores



ESP32 Servidor: Enchufes

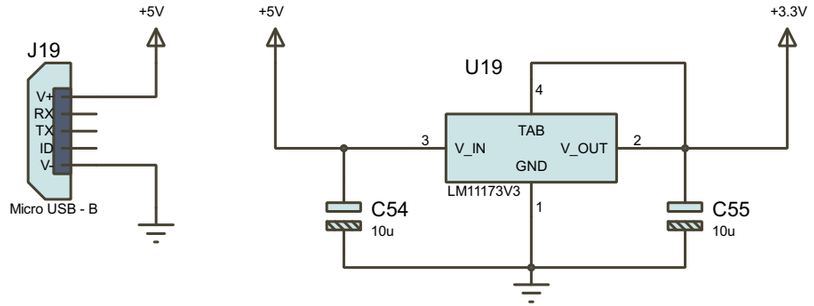


Salida al motor

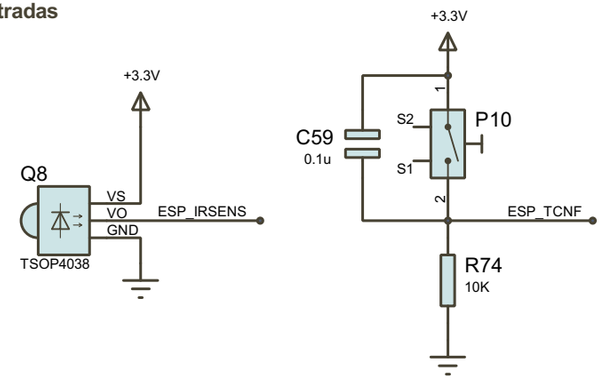


UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Autor: Jorge Álvarez Pedrón	Proyecto: Xana: prototipo de controlador domótico con control por voz (TFG)	Fecha: 31/08/2020
		Hoja: 04. Actuador: Motor de persiana	Versión: V1.0

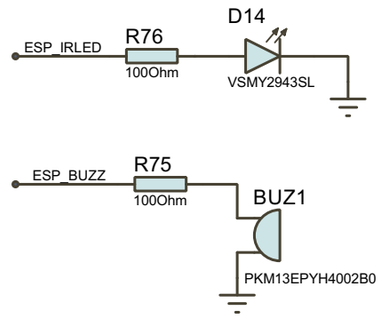
Alimentación



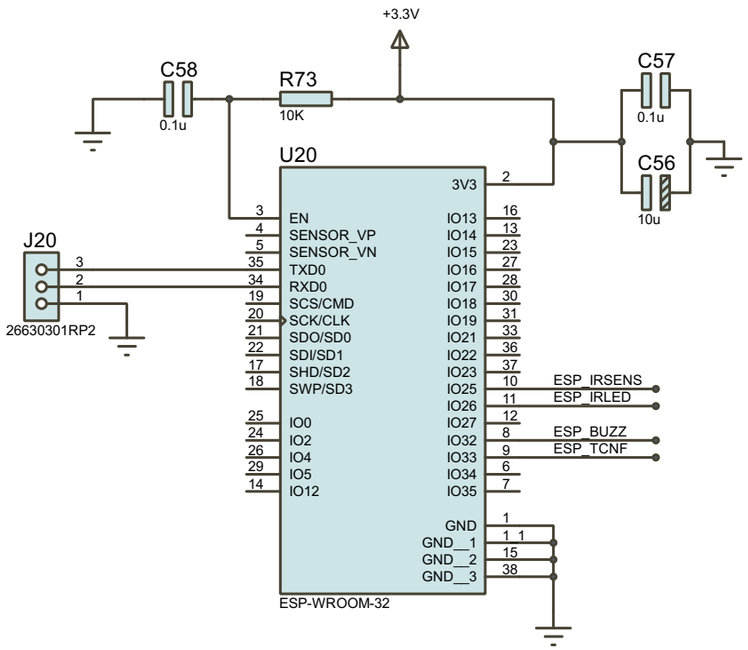
Entradas



Salidas



ESP32 Servidor: Temperatura



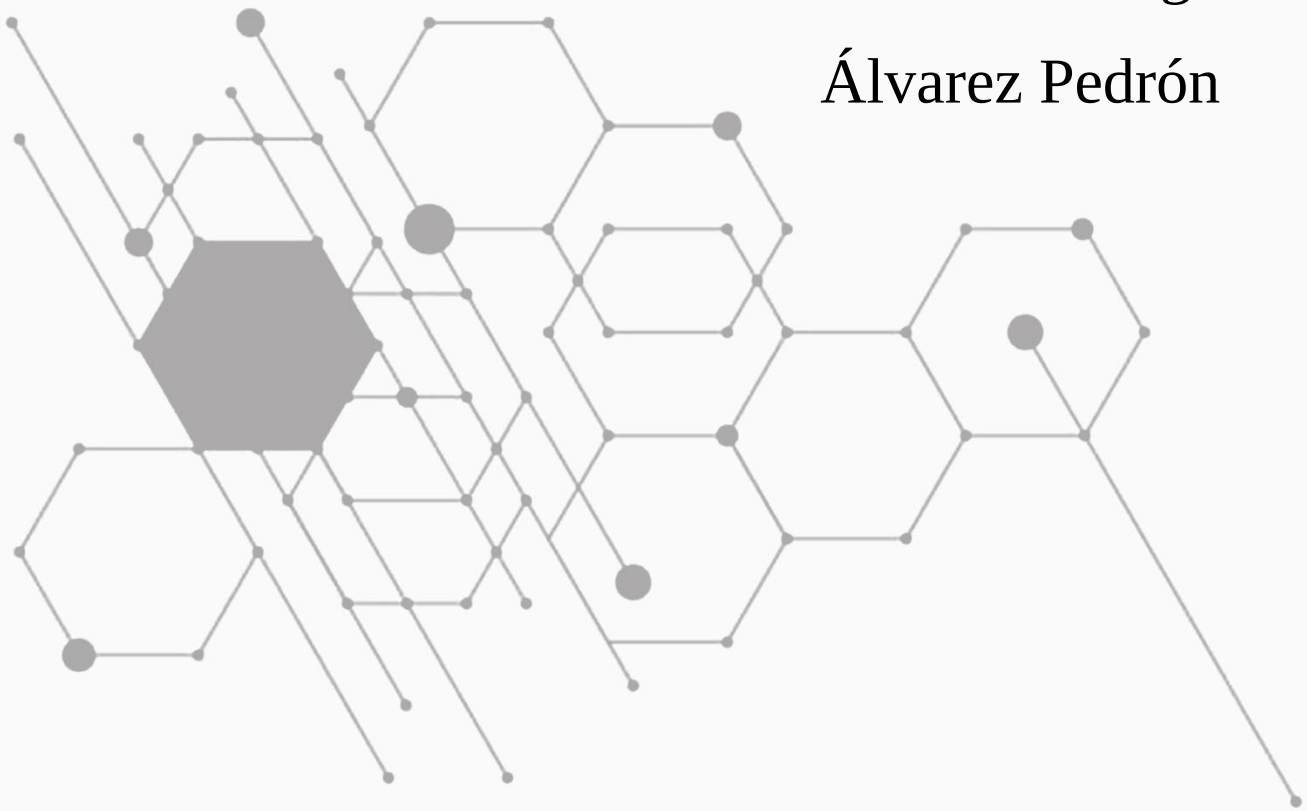
UNIVERSITAT POLITÈCNICA DE VALÈNCIA	Autor: Jorge Álvarez Pedrón	Proyecto: Xana: prototipo de controlador domótico con control por voz (TFG)	Fecha: 31/08/2020
		Hoja: 05. Actuador: Mando de temperatura	Versión: V1.0

Documento 3: Presupuesto

Xana: Prototipo de asistente domótico controlado por voz

Trabajo de Fin de Grado

Autor: Jorge Álvarez Pedrón



Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice del presupuesto

1. Coste de materiales

1.1. Módulos

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
5	U3, U14, U15, U18, U19	Regulador de voltaje LM11173V3	0,538 €	2,69 €
5	U4, U10, U16, U17, U20	Módulo ESP-WROOM-32	2,200 €	11,00 €
1	U5	Placa de desarrollo de prototipos Arduino Nano	1,480 €	1,48 €
1	U6	Módulo de reconocimiento de voz V3,1 de elechouse	13,800 €	13,80 €
			Total módulos:	28,97 €

1.2. Circuitos integrados

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	U2	Amplificador operacional de audio TL081	0,206 €	0,21 €
1	U9	Amplificador operacional de audio de potencia LM1875	1,870 €	1,87 €
3	U11-U13	Amplificador operacional rail-to-rail TLV170	0,486 €	1,46 €
			Total circuitos integrados:	3,53 €

1.3. Resistencias de 1/4 W

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
2	R1, R4	3,3 k	0,0106 €	0,02 €
1	R2	1,5 k	0,0106 €	0,01 €
23	R3, R6-R10, R17, R26, R27, R37, R38, R41, R44, R47, R51, R54, R58, R62, R69-R74	10 K	0,0106 €	0,24 €
1	R5	1,8 K	0,0106 €	0,01 €
1	R11	330 Ohm	0,0106 €	0,01 €
10	R28, R40, R43, R46, R49, R53, R57, R61, R65, R68	1 k	0,0106 €	0,11 €
3	R36, R75, R76	100 Ohm	0,0106 €	0,03 €

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
3	R39, R42, R45	2,7 K	0,0106 €	0,03 €
5	R50, R55, R59, R63, R67	470 Ohms	0,0106 €	0,05 €
1	RV1	Potenciómetro de 100 K	0,1470 €	0,15 €
1	LDR1	Fotoresistencia LDR de 1M con 0,1 lux y 0,1 Ohms con 10.000 lux	0,0196 €	0,02 €
			Total resistencias:	0,69 €

1.4. Condensadores

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
18	C1-C3, C6, C26, C28, C29, C35-C39, C46, C52-C56	10 uF	0,065 €	1,16 €
26	C4, C5, C7- C10, C12, C27, C30-C34, C40-C45, C47-C51, C57-C59	0,1 uF	0,006 €	0,15 €
			Total condensadores:	1,31 €

1.5. Transistores

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
6	Q2-Q7	Transistor NPN BC847	0,006 €	0,04 €
1	Q8	Fototransistor receptor de IR TSOP4038	0,086 €	0,09 €
			Total transistores:	0,12 €

1.6. Diodos

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
6	D1, D3, D6, D8, D10, D13	Diodo LED rojo, Vf = 1,8 V	0,048 €	0,29 €
2	D2, D14	Diodo LED IR VSMY2943SL	0,012 €	0,02 €
6	D4, D5, D7, D9, D11, D12	Diodo de protección de 5V VESD05A1	0,058 €	0,35 €
			Total diodos:	0,66 €

1.7. Relés

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
5	R48, R52, R56, R60, R66	Relé de potencia de 10 A tipo A con bobina de 5 V G5LE-1A DC5	0,761 €	3,81 €
1	R64	Relé de potencia de doble salida tipo C con bobina de 9V JW2HN-DC9V	2,300 €	2,30 €
			Total relés:	6,11 €

1.8. Conectores

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	J1	microSD socket	0,029 €	0,03 €
2	J2, J6	5,5mm DC Jack hembra	0,030 €	0,06 €
1	J19	microUSB tipo B hembra	0,042 €	0,04 €
5	J3, J4, J8, J16, J20	Conector de 3 pines macho 2,54 mm	0,054 €	0,27 €
1	J5	Conector de 4 pines macho 2,54 mm	0,057 €	0,06 €
10	J7, J9-J15, J17-J18	Terminal de 2 entradas de 5,08 mm para PCB, máx. 300 V 20 A.	0,050 €	0,50 €
			Total conectores:	0,96 €

1.9. Interfaz y alimentación

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	BUZ1	Zumbador piezoeléctrico pasivo	0,028 €	0,03 €
10	P1-P10	Pulsador	0,090 €	0,90 €
1	S2	Altavoz de 8 Ohm, 15 W de rango completo	5,490 €	5,49 €
1	U1	Micrófono MEMS ICS40180 de 3,3V y salida analógica	1,190 €	1,19 €
1	Externo	Tarjeta microSD 8 GB	0,950 €	0,95 €
4	Externo	Transformador de 230 VAC a 12 VDC de 2 A con salida por Jack 5,5 mm	1,590 €	6,36 €
1	Externo	Cable USB - microUSB tipo B	0,880 €	0,88 €
			Total interfaz y alimentación:	15,80 €

1.10. Resumen de materiales

Módulos	28,97 €
Circuitos integrados	3,53 €
Resistencias	0,69 €
Condensadores	1,31 €
Transistores	0,12 €
Diodos	0,66 €
Relés	6,11 €
Conectores	0,96 €
Interfaz y alimentación	15,80 €

Coste total de los materiales: 58,14 €

Nota: para algunos componentes, se están usando los precios de packs de varias unidades (las resistencias, por ejemplo, de 100 en 100). Para una producción de lote, los componentes se comprarían en mayores cantidades y su precio unitario disminuiría sensiblemente.

Las referencias y links de compra de cada componente formarán el *anexo 30* del proyecto.

2. Materiales por módulos

2.1. Controlador Principal

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	U3	Regulador de voltaje LM11173V3	0,538 €	0,54 €
1	U4	Módulo ESP-WROOM-32	2,200 €	2,20 €
1	U5	Placa de desarrollo de prototipos Arduino Nano	1,480 €	1,48 €
1	U6	Módulo de reconocimiento de voz V3,1 de elechouse	13,800 €	13,80 €
1	U9	Amplificador operacional de audio de potencia LM1875	1,870 €	1,87 €
1	U2	Amplificador operacional de audio TL081	0,206 €	0,21 €
2	R1, R4	3,3 k	0,0106 €	0,02 €
1	R2	1,5 k	0,0106 €	0,01 €
8	R3, R6-R10, R26, R27	10 K	0,0106 €	0,08 €
1	R5	1,8 K	0,0106 €	0,01 €
1	R11	330 Ohm	0,0106 €	0,01 €
1	R28	1 k	0,0106 €	0,01 €
1	RV1	Potenciómetro de 100 K	0,1470 €	0,15 €
1	LDR1	Fotoresistencia LDR de 1M con 0,1 lux y 0,1 Ohms con 10.000 lux	0,0196 €	0,02 €
6	C1-C3, C6, C27, C28	10 uF	0,065 €	0,39 €
7	C4, C5, C7-C10, C27	0,1 uF	0,006 €	0,04 €
1	D1	Diodo LED rojo, Vf = 1,8 V	0,048 €	0,05 €
1	J1	microSD socket	0,029 €	0,03 €
1	J2	5,5mm DC Jack hembra	0,030 €	0,03 €
1	J3	Conector de 3 pines macho 2,54 mm	0,054 €	0,05 €
2	P1, P2	Pulsador	0,090 €	0,18 €
1	S2	Altavoz de 8 Ohm, 15 W de rango completo	5,490 €	5,49 €
1	U1	Micrófono MEMS ICS40180 de 3,3V y salida analógica	1,190 €	1,19 €
1	Externo	Tarjeta microSD 8 GB	0,950 €	0,95 €
1	Externo	Transformador de 230 VAC a 12 VDC de 2 A con salida por Jack 5,5 mm	1,590 €	1,59 €
		Total Controlador principal:		**30,40 €**

2.2. Control de luces

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	U14	Regulador de voltaje LM11173V3	0,538 €	0,54 €
1	U10	Módulo ESP-WROOM-32	2,200 €	2,20 €
3	U11-U13	Amplificador operacional rail-to-rail TLV170	0,486 €	1,46 €
4	R37, R38, R41, R44	10 K	0,0106 €	0,04 €
3	R40, R43, R46	1 k	0,0106 €	0,03 €
1	R36	100 Ohm	0,0106 €	0,01 €
3	R39, R42, R45	2,7 K	0,0106 €	0,03 €
3	C29, C35, C36	10 uF	0,065 €	0,19 €
5	C30-C34	0,1 uF	0,006 €	0,03 €
1	D2	Diodo LED IR VSMY2943SL	0,012 €	0,01 €
1	J6	5,5mm DC Jack hembra	0,030 €	0,03 €
1	J4	Conector de 3 pines macho 2,54 mm	0,054 €	0,05 €
1	J5	Conector de 4 pines macho 2,54 mm	0,057 €	0,06 €
1	Externo	Transformador de 230 VAC a 12 VDC de 2 A con salida por Jack 5,5 mm	1,590 €	1,59 €
			Total control de luces:	**6,28 €**

2.3. Control de enchufes

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	U15	Regulador de voltaje LM11173V3	0,538 €	0,54 €
1	U16	Módulo ESP-WROOM-32	2,200 €	2,20 €
5	R47, R51, R54, R58, R62	10 K	0,0106 €	0,05 €
4	R49, R53, R57, R61	1 k	0,0106 €	0,04 €
4	R50, R55, R59, R63	470 Ohms	0,0106 €	0,04 €
3	C37-C39	10 uF	0,065 €	0,19 €
6	C40-C45	0,1 uF	0,006 €	0,03 €
4	Q2-Q5	Transistor NPN BC847	0,006 €	0,02 €
4	D3, D6, D8, D10	Diodo LED rojo, Vf = 1,8 V	0,048 €	0,19 €
4	D4, D5, D7, D9	Diodo de protección de 5V VESD05A1	0,058 €	0,23 €
4	R48, R52, R56, R60	Relé de potencia de 10 A tipo A con bobina de 5 V G5LE-1A DC5	0,761 €	3,04 €
1	J8	Conector de 3 pines macho 2,54 mm	0,054 €	0,05 €

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
6	J7, J9-J13	Terminal de 2 entradas de 5,08 mm para PCB, máx. 300 V 20 A.	0,050 €	0,30 €
4	P3-P6	Pulsador	0,090 €	0,36 €
1	Externo	Transformador de 230 VAC a 12 VDC de 2 A con salida por Jack 5,5 mm	1,590 €	1,59 €
		Total control de enchufes:		**8,90 €**

2.4. Controlador de motor de persiana

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	U18	Regulador de voltaje LM11173V3	0,538 €	0,54 €
1	U17	Módulo ESP-WROOM-32	2,200 €	2,20 €
4	R69-R72	10 K	0,0106 €	0,04 €
2	R65, R68	1 k	0,0106 €	0,02 €
1	R67	470 Ohms	0,0106 €	0,01 €
3	C46, C52, C53	10 uF	0,065 €	0,19 €
5	C47-C51	0,1 uF	0,006 €	0,03 €
2	Q6, Q7	Transistor NPN BC847	0,006 €	0,01 €
1	D13	Diodo LED rojo, Vf = 1,8 V	0,048 €	0,05 €
2	D11, D12	Diodo de protección de 5V VESD05A1	0,058 €	0,12 €
1	R66	Relé de potencia de 10 A tipo A con bobina de 5 V G5LE-1A DC5	0,761 €	0,76 €
1	R64	Relé de potencia de doble salida tipo C con bobina de 9V JW2HN-DC9V	2,300 €	2,30 €
1	J16	Conector de 3 pines macho 2,54 mm	0,054 €	0,05 €
4	J14, J15, J17, J18	Terminal de 2 entradas de 5,08 mm para PCB, máx. 300 V 20 A.	0,050 €	0,20 €
3	P7-P9	Pulsador	0,090 €	0,27 €
1	Externo	Transformador de 230 VAC a 12 VDC de 2 A con salida por Jack 5,5 mm	1,590 €	1,59 €
		Total controlador motor de persiana:		**8,48 €**

2.5. Control de temperatura

Cantidad	Referencia	Nombre / Valor	Precio unitario	Precio total
1	U19	Regulador de voltaje LM11173V3	0,538 €	0,54 €
1	U20	Módulo ESP-WROOM-32	2,200 €	2,20 €
2	R73, R74	10 K	0,0106 €	0,02 €
2	R75, R76	100 Ohm	0,0106 €	0,02 €
3	C54-C56	10 uF	0,065 €	0,19 €
3	C57-C59	0,1 uF	0,006 €	0,02 €
1	Q8	Fototransistor receptor de IR TSOP4038	0,086 €	0,09 €
1	D14	Diodo LED IR VSMY2943SL	0,012 €	0,01 €
1	J19	microUSB tipo B hembra	0,042 €	0,04 €
1	J20	Conector de 3 pines macho 2,54 mm	0,054 €	0,05 €
1	BUZ1	Zumbador piezoeléctrico pasivo	0,028 €	0,03 €
1	Externo	Cable USB - microUSB tipo B	0,880 €	0,88 €
			Total control de temperatura:	**4,09 €**

2.6. Resumen de módulos

Controlador principal	30,40 €
Control de luces	6,28 €
Control de enchufes	8,90 €
Controlador de motor de persiana	8,48 €
Control de temperatura	4,09 €
Coste total de los módulos:	**58,14 €**

3. Costes de diseño

3.1. Mano de obra

Horas	Referencia	Precio / hora	Precio total
50	Ingeniero electrónico: diseño de hardware	24,11 €	1.205,50 €
150	Ingeniero electrónico / Informático: diseño de software	24,11 €	3.616,50 €
40	Ingeniero electrónico: montaje del prototipo y testeo	16,29 €	651,60 €
60	Ingeniero técnico: redacción del proyecto	14,71 €	882,60 €
300 horas totales		**Total mano de obra:**	**6.356,20 €**

3.2. Herramientas y costes adicionales

Cantidad	Referencia	Precio unitario	Precio total
3	Placa de prototipos protoboard de 840 puntos de conexión	4,19 €	12,57 €
3	Placa de prototipos protoboard de 400 puntos de conexión	2,69 €	8,07 €
1	5x PCB de 2 capas de 200 mm x 120 mm x 1,6 mm con el diseño de los 5 módulos de Xana y plantilla para pasta de soldar	28,76 €	28,76 €
1	Licencia del software de programación	- €	- €
1	Licencia del software de diseño de hardware*	- €	- €
-	Costes directos complementarios (electricidad, productos de soldadura, cables, gastos de envíos, amortización de herramientas, etc.)	1%	64,64 €
		Total herramientas y costes adicionales:	**114,04 €**

* El software utilizado en este proyecto ha sido Proteus Design Suite 8,9 ya que es accesible de forma gratuita para sus alumnos a través de los servidores de la UPV. Para un proyecto sin esta accesibilidad se utilizaría un software de pago más económico como Altium Designer por 2.159 €/año o uno gratuito como KiCad EDA.

4. Resumen

Resumen del presupuesto	
1. y 2. Coste de materiales	58,14 €
3.1. Coste de mano de obra	6.356,20 €
3.2. Coste de herramientas y costes adicionales	114,04 €
SUBTOTAL PEM	6.528,38 €
Beneficio industrial (6%)	391,70 €
IVA (21%)	1.453,22 €

Presupuesto de contrata (PEC): 8.373,30 €

Anexo 00

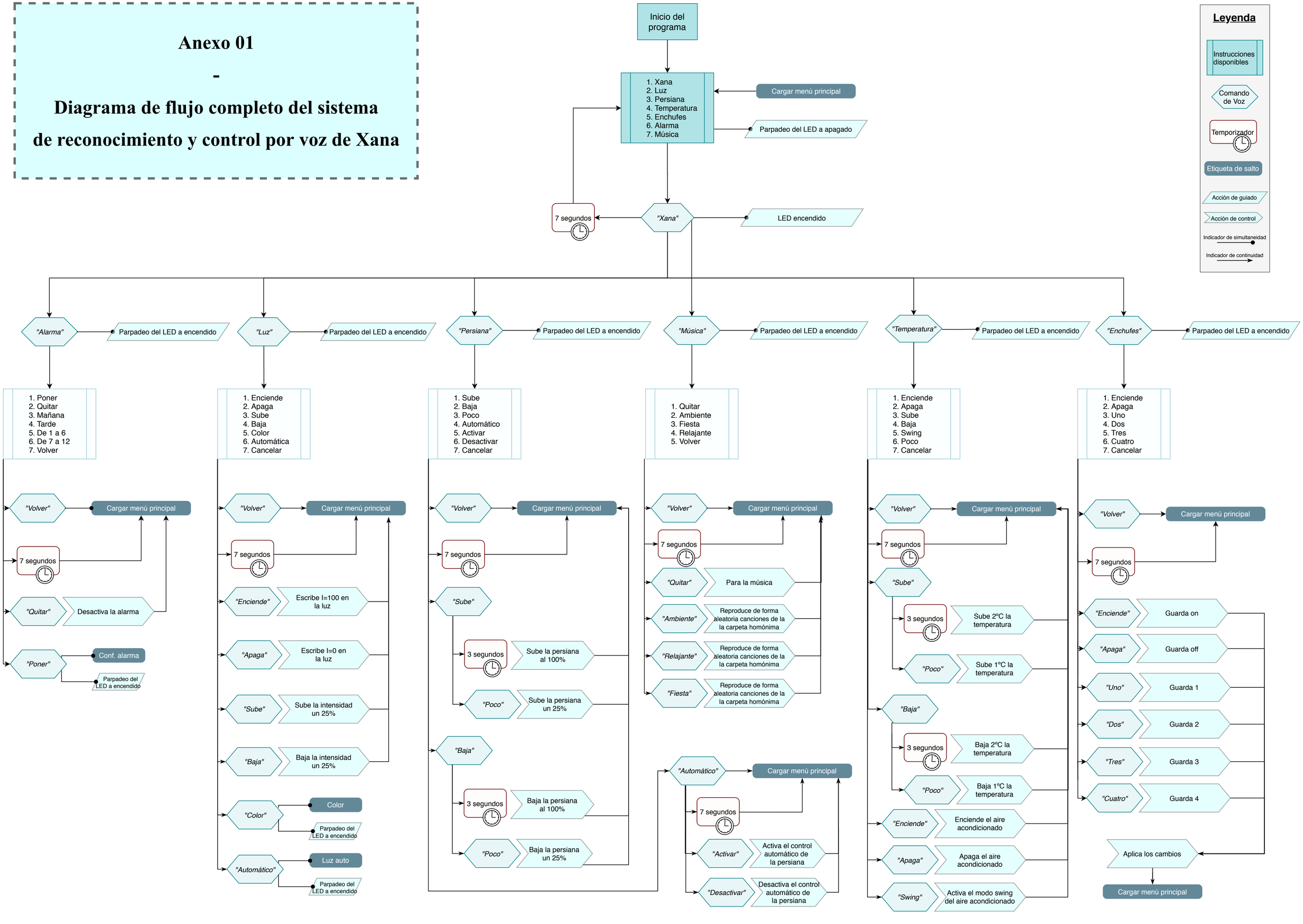
-

Índice de anexos

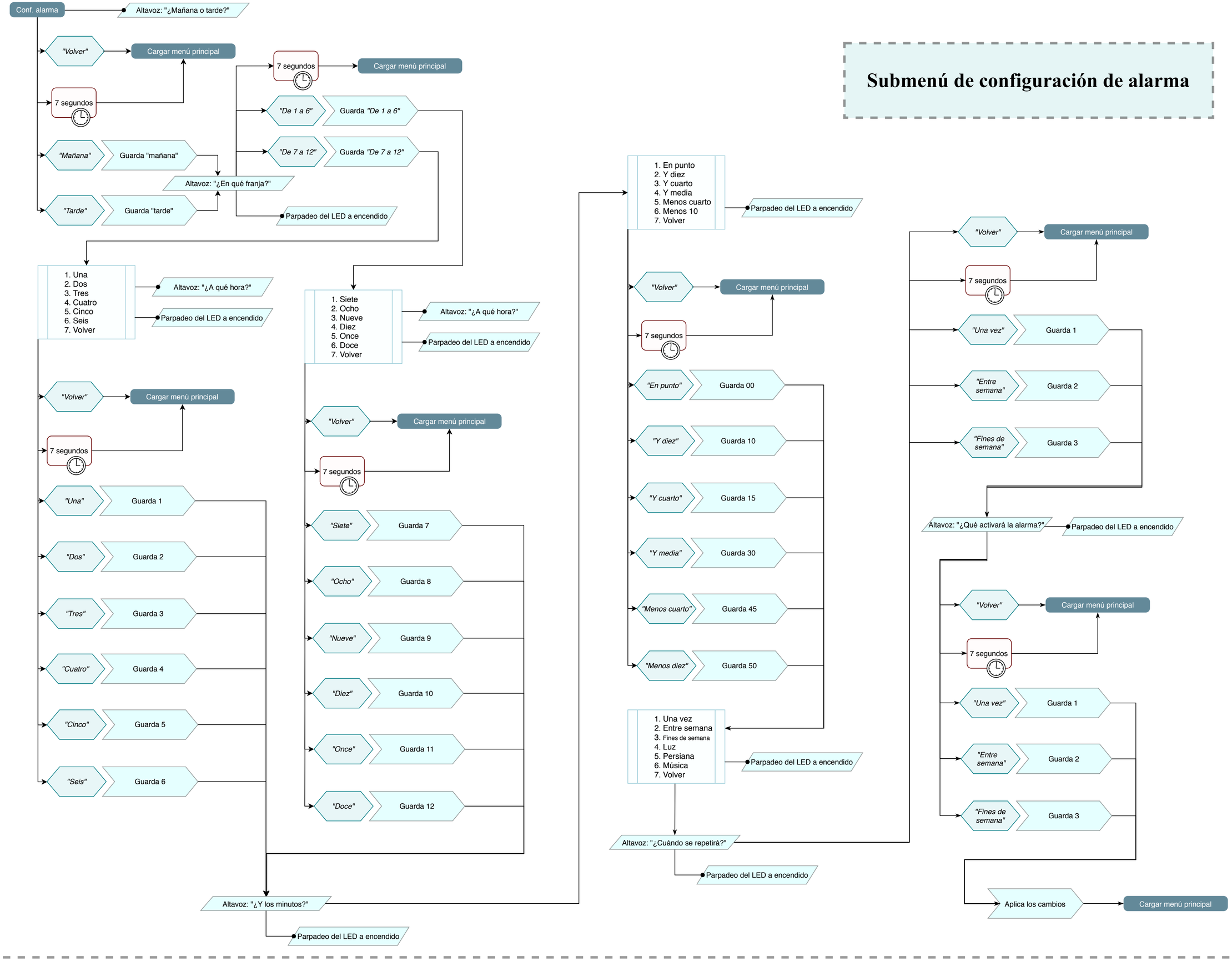
Anexo 01

-

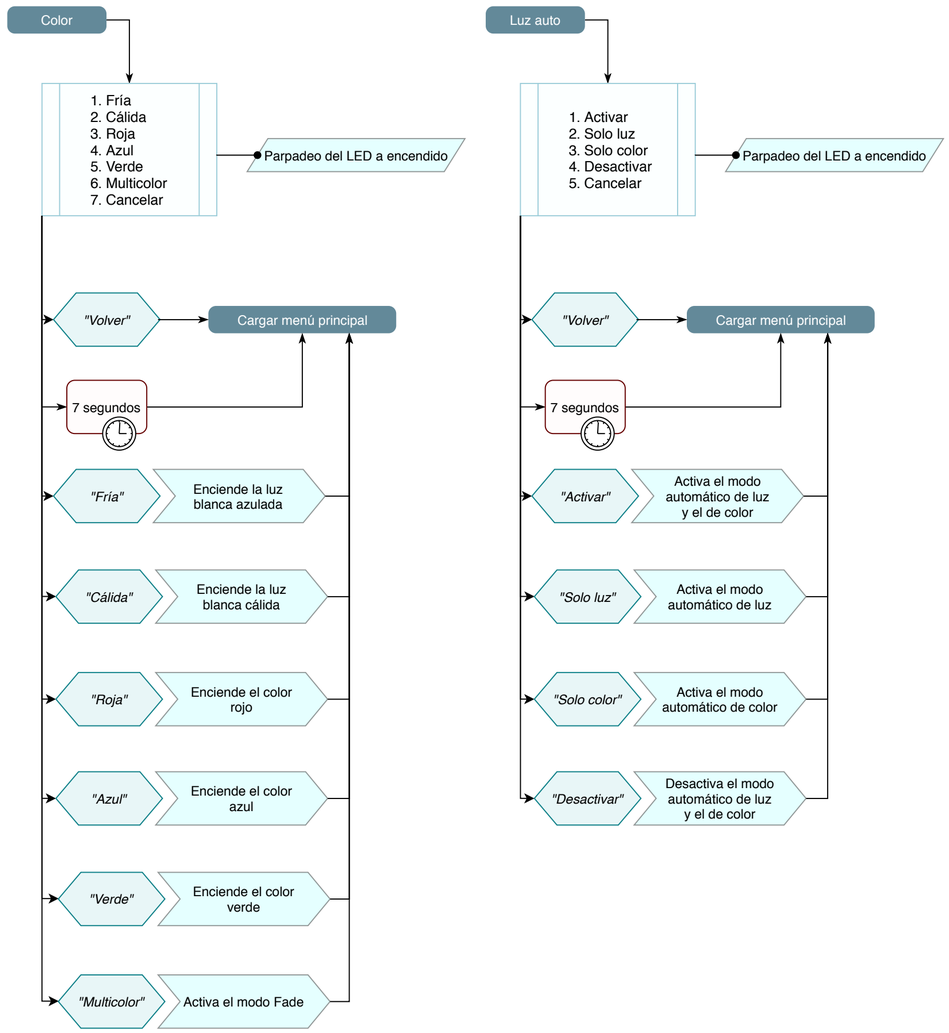
Diagrama de flujo completo del sistema de reconocimiento y control por voz de Xana



Submenú de configuración de alarma



Submenús de modos de color



Anexo 02

-

Instrucciones.h

```
/**
*********************************************************************
  * @file    Instrucciones.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    15/06/2020
  * @brief   Define las etiquetas de todas las instrucciones de voz
  *          de Xana.
*********************************************************************
  */

/** tiempo de espera máximo dentro de un menú antes de volver al
principal */
#define esperaMax 6000

/** Funciones principales */
```

```
#define insVolver        (0)
#define insXana          (1)

/** Menú principal */
#define insLuz           (2)
#define insMusica        (3)
#define insAlarma        (4)
#define insPersiana      (5)
#define insEnchufe       (6)
#define insTemp          (7)

/** Menú Luz */
#define insEnciende      (8)
#define insApaga         (9)
#define insSube         (10)
#define insBaja         (11)
#define insColor        (12)
#define insAuto         (13)

/** Submenú Luz -> Color */
#define insFria         (14)
#define insCalida       (15)
#define insAzul         (16)
#define insRoja         (17)
#define insVerde      (18)
#define insMulticolor   (19)

/** Submenú Luz -> Auto */
```

```
#define insActivar      (14)
#define insDesactivar   (15)
#define insSoloLuz      (16)
#define insSoloColor    (17)

/** Menú Música */
#define insAmbiente     (18)
#define insFiesta       (19)
#define insRelax        (20)
// **** insQuitar       (24)                //Ya definida
#define insBluetooth    (22)

/** Menú Alarma */
#define insPoner        (23)
#define insQuitar       (24)
#define insAM           (25)
#define insPM           (26)
#define ins1to6         (27)
#define ins7to12        (28)

/** Submenú Alarma -> Hora1 */
#define ins1            (29)
#define ins2            (30)
#define ins3            (31)
#define ins4            (32)
#define ins5            (33)
#define ins6            (34)
```

```
/** Submenú Alarma -> Hora2 */
#define ins7            (35)
#define ins8            (36)
#define ins9            (37)
#define ins10           (38)
#define ins11           (39)
#define ins12           (40)

/** Submenú Alarma -> Min */
#define insEnPunto      (41)
#define insY10          (42)
#define insY15          (43)
#define insY30          (44)
#define insY45          (45)
#define insY50          (46)

/** Submenú Alarma -> Repetir */
#define ins1vez         (47)
#define insSemana       (48)
#define insFindes       (49)
// **** insLuz          ( 2)                //Ya definida
// **** insMusica       ( 3)                //Ya definida
// **** insPersiana     ( 5)                //Ya definida

/** Menú Persiana */

#define insPoco         (50)
// **** insSube         (10)                //Ya definida
```

```
// **** insBaja         (11)              //Ya definida
// **** insAuto         (13)              //Ya definida
// **** insActivar      (14)              //Ya definida
// **** insDesactivar   (15)              //Ya definida

/** Menú Temperatura */
#define insSwing        (51)
// **** insEnciende     ( 8)              //Ya definida
// **** insApaga        ( 9)              //Ya definida
// **** insSube         (10)              //Ya definida
// **** insBaja         (11)              //Ya definida
// **** insPoco         (50)              //Ya definida

/* Menú Enchufe */
// **** insEnciende     ( 8)              //Ya definida
// **** insApaga        ( 9)              //Ya definida
// **** ins1            (29)              //Ya definida
// **** ins2            (31)              //Ya definida
// **** ins3            (31)              //Ya definida
// **** ins4            (32)              //Ya definida

//-----------------------------------------------------------------
-
//Fin de Instrucciones.h
```

Anexo 03

-

XanaVRM.h

```
/**
*****************************************************************
  * @file    XanaVRM.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    15/06/2020
  * @brief   Incluye todos los comandos necesarios para el
  *          funcionamiento de Xana con el VRM
*****************************************************************
  * @section  BASED ON

    Parte de este proyecto está basado en las librerías del Elechouse
    Voice Sensor Module V3.1 de Elechouse Team y Jiapeng Li.

*****************************************************************
  */
```

```
/*----------------------- Includes y defines -----------------------
*/

#include "VoiceRecognitionV3.h"
#include "Instrucciones.h"

#include "SD.h"
#include "TMRpcm.h"
#include "SPI.h"
#define SDPinSelect 4
TMRpcm altavoz;

#define botonVolUp 10
#define botonVolDw 7
uint8_t volumen = 5;

uint8_t records[7]; // save record
uint8_t buf[64];
bool xanaLista = false;

SoftwareSerial EspSerial(5,6);

unsigned long marcaTemp = 0;



/*---------------- LED de guía por los menús de voz -----------------
*/

#define VRMLED 8
```

```
void VRMLedOn(){
    digitalWrite(VRMLED, HIGH);
}

void VRMLedOff(){
    digitalWrite(VRMLED, LOW);
}

void VRMLedBlink(){
    VRMLedOff();
    delay(100);
    VRMLedOn();
    delay(100);
    VRMLedOff();
    delay(100);
    VRMLedOn();
}

/*---------- Escritura de instrucciones en el canal serie -----------
*/

/** Configura los pines de comunicacción: VR myVR(RX, TX); */
VR myVR(2, 3);

/**
  @brief   Print signature, if the character is invisible,
           print hexible value instead.
  @author  JiapengLi
```

```
  @param   buf     --> command length
           len     --> number of parameters
*/
void printSignature(uint8_t *buf, int len)
{
    Serial.print("\t");

    int i;
    for (i = 0; i < len; i++)
    {
        if (buf[i] > 0x19 && buf[i] < 0x7F)
        {
            Serial.write(buf[i]);
        }
        else
        {
            Serial.print("[");
            Serial.print(buf[i], HEX);
            Serial.print("]");
        }
    }
}

/**
  @brief   Print signature, if the character is invisible,
           print hexible value instead.
  @author  JiapengLi
  @param   buf  -->  VR module return value when voice is recognized.
```

```
        buf[0]  -->  Group mode(FF: None Group, 0x8n: User,
0x0n:System
        buf[1]  -->  number of record which is recognized.
        buf[2]  -->  Recognizer index value of the recognized record.
        buf[3]  -->  Signature length
        buf[4]~buf[n] --> Signature
*/

void printVR(uint8_t *buf)
{
    if(xanaLista)
    {
        VRMLedBlink();
    }

    Serial.print("Instrucción reconocida:");

    /*
    Serial.println(F("VR Index\tTotal Index\tInstrucción"));

    Serial.print("    ");
    Serial.print(buf[2], DEC);
    Serial.print("\t");

    Serial.print("\t    ");
    Serial.print(buf[1], DEC);
    Serial.print("\t");
    */
```

```
    if (buf[3] > 0)
    {
        printSignature(buf + 4, buf[3]);
    }
    else
    {
        Serial.print("NONE");
    }

    Serial.println(F("\r\n"));
}

/*------------ Carga y reconocimiento de comandos de voz ------------
*/


/** Carga una instrucción en la lista de las 7 reconocibles */
void cargarInstruccion(uint8_t instruccion)
{
    myVR.load((uint8_t)instruccion);
}

/** Devuelve el número de instrucciones de voz recibidas*/
int comprobarVoz()
{
    int cola;
    cola = myVR.recognize(buf, 50);
    return cola;
}
```

```
/*------------------- Funciones de inicialización -------------------
*/

void cargarMenuPrincipal()
{
    VRMLedOff();

    myVR.clear();
    Serial.println(F("\n"));

    cargarInstruccion(insXana);
    Serial.println(F("\tCargada instrucción XANA"));

    cargarInstruccion(insLuz);
    Serial.println(F("\tCargada instrucción LUZ"));

    cargarInstruccion(insMusica);
    Serial.println(F("\tCargada instrucción MÚSICA"));

    cargarInstruccion(insAlarma);
    Serial.println(F("\tCargada instrucción ALARMA"));

    cargarInstruccion(insPersiana);
    Serial.println(F("\tCargada instrucción PERSIANA"));

    cargarInstruccion(insEnchufe);
    Serial.println(F("\tCargada instrucción ENCHUFE"));
```

```
    cargarInstruccion(insTemp);
    Serial.println(F("\tCargada instrucción TEMPERATURA"));

}

void initXanaVRM()
{
    pinMode(VRMLED, OUTPUT);

    myVR.begin(9600);

    Serial.begin(9600);
    Serial.println(F("Asistente de hogar XANA: VISUALIZADOR DE "
    "INSTRUCCIONES del canal serie.\r\n\tMódulo de reconocimiento de "
    "voz: Elechouse Voice Recognition V3 Module"));

    if (myVR.clear() == 0)
    {
        Serial.println(F("\nReconocedor de instrucciones activado. "
        "Cargando instrucciones:"));
    }
    else
    {
        Serial.println(F("\n******** ERROR: ******** Not find "
        "VoiceRecognitionModule."));
        Serial.println(F("\tPlease check connection and "
        "restart Arduino."));
        while (1);
```

```
    }

    /** Carga las instrucciones del programa principal */
    cargarMenuPrincipal();

    VRMLedOn();
    delay(100);
    VRMLedOff();
    delay(100);
    VRMLedOn();
    delay(100);
    VRMLedOff();

}

void initAltavoz()
{
    if (!SD.begin(SDPinSelect))
    {
        Serial.println("fallo de conexion a SD");
    }
    altavoz.speakerPin = 9;
    altavoz.quality(1);
    altavoz.setVolume(5);

    pinMode(botonVolUp, INPUT);
    pinMode(botonVolDw, INPUT);
```

```
    randomSeed(analogRead(0));
}

/*------------------- Funciones de temporización --------------------
*/
/** Guarda una marca de tiempo para comprobar el tiempo transcurrido
 * con "tiempoTranscurrido"
 * */
void guardarTiempo()
{
    marcaTemp = millis();
}

/** Devuelve true si ha pasado  "tiempo_ms" desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
bool comprobarTiempo(int tiempo_ms)
{
    if (abs(millis() - marcaTemp) >= tiempo_ms)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/** Devielve el tiempo (en ms) transcurrido desde la última marca de
```

```
 * tiempo guardada con "guardarTiempo"
 * */
unsigned long tiempoDesdeMarca()
{
    return (millis() - marcaTemp);
}

//-------------------------------------------------------------------------
-
//Fin de XanaVRM.h
```

Anexo 04

-

XanaVRM.cpp

```
/**

*****************************************************************
  * @file    XanaVRM.cpp
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    15/06/2020
  * @brief   Asistente de hogar con control de voz.
  */


/*---------------------------- Includes ----------------------------
*/
#include <Arduino.h>
#include <SoftwareSerial.h>
#include "XanaVRM.h"


#include "menuLuz.h"
#include "menuAlarma.h"
#include "menuTemperatura.h"
#include "menuEnchufe.h"
```

```
#include "menuPersiana.h"
#include "menuMusica.h"

/*---------------- Accionamiento de eventos por voz ----------------
*/

/** Atiende a las instrucciones de voz del VRM en el menú principal y
 *  redirige al asuario a los distintos menús de actuadores
 *  */
void crearEventos()
{
  // Escribe la instrucción recibida por el canal serie
  printVR(buf);

  // Actúa en función de la instrucción recibida
  switch (buf[1])
  {
  case insXana:

    xanaLista = true;
    VRMLedOn();

    guardarTiempo();
    break;

  case insLuz:
    if (xanaLista)
    {
      menuLuz::cargarMenu();
```

```
      menuLuz::esperarInstruccion();

      //Vuelve a cargar las instrucciones del menú principal
      cargarMenuPrincipal();

      // Vuelve a hacer falta decir Xana para acceder a los otros
menús
      xanaLista = false;
    }
    else
    {
      Serial.println(F("--- Por favor, di Xana para acceder al menú "
      "principal ---\n\n"));
    }
    break;

  case insMusica:

    if (xanaLista)
    {
      menuMusica::cargarMenu();

      //Vuelve a cargar las instrucciones del menú principal
      cargarMenuPrincipal();

      // Vuelve a hacer falta decir Xana para acceder a los otros
menús
      xanaLista = false;
    }
```

```
    else
    {
      Serial.println(F("--- Por favor, di Xana para acceder al menú "
      "principal ---\n\n"));
    }
    break;

  case insAlarma:

    if (xanaLista)
    {
      menuAlarma::cargarMenu();
      menuAlarma::esperarInstruccion();

      //Vuelve a cargar las instrucciones del menú principal
      cargarMenuPrincipal();

      // Vuelve a hacer falta decir Xana para acceder a los otros
menús
      xanaLista = false;
    }
    else
    {
      Serial.println(F("--- Por favor, di Xana para acceder al menú "
      "principal ---\n\n"));
    }
    break;

  case insPersiana:
```

```
    if (xanaLista)
    {
      menuPersiana::cargarMenu();
      menuPersiana::esperarInstruccion();

      //Vuelve a cargar las instrucciones del menú principal
      cargarMenuPrincipal();

      // Vuelve a hacer falta decir Xana para acceder a los otros
menús
      xanaLista = false;
    }
    else
    {
      Serial.println(F("--- Por favor, di Xana para acceder al menú "
      "principal ---\n\n"));
    }
    break;

  case insEnchufe:

    if (xanaLista)
    {
      menuEnchufe::cargarMenu();
      menuEnchufe::esperarInstruccion();

      //Vuelve a cargar las instrucciones del menú principal
      cargarMenuPrincipal();
```

```
      // Vuelve a hacer falta decir Xana para acceder a los otros
menús
      xanaLista = false;
    }
    else
    {
      Serial.println(F("--- Por favor, di Xana para acceder al menú "
      "principal ---\n\n"));
    }
    break;

  case insTemp:

    if (xanaLista)
    {
      menuTemp::cargarMenu();
      menuTemp::esperarInstruccion();

      //Vuelve a cargar las instrucciones del menú principal
      cargarMenuPrincipal();

      // Vuelve a hacer falta decir Xana para acceder a los otros
menús
      xanaLista = false;
    }
    else
    {
      Serial.println(F("--- Por favor, di Xana para acceder al menú "
```

```
      "principal ---\n\n"));
    }
    break;

  default:
    Serial.println(F("ERROR: Función no declarada en el menú "
    "principal"));
    break;
  }
}

/* ---------------------------- Setup ------------------------------
*/

void setup()
{
  // Inicializa la comunicación con el VRM
  initXanaVRM();

  // Configura el altavoz y la lectura de tarjeta SD
  initAltavoz();

  Serial.println(F("\nInstrucciones Cargadas. Hable ahora\n\n"));
}

/* ---------------------------- Loop -------------------------------
*/

void loop()
```

```
{


  //Detecta las instrucciones de voz para navegar por el menú
principal
  if (comprobarVoz() > 0)
  {
    crearEventos();
  }


  // A los esperaMax segundos, vuelve a hacer falta decir Xana para
  // acceder a los menús de actuador.
  if(comprobarTiempo(esperaMax))
  {
    xanaLista = false;
    VRMLedOff();
  }


  // Sube y baja el volumen del altavoz
  if(digitalRead(botonVolUp) && volumen <=10)
  {
    delay(80);
    volumen++;
    altavoz.setVolume(volumen);
  }
  if(digitalRead(botonVolDw) && volumen >=0)
  {
    delay(80);
    volumen--;
    altavoz.setVolume(volumen);
```

```
  }


}


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

//Fin de XanaVRM.cpp
```

Anexo 05

-

menuAlarma.h

```
/**

*****************************************************************
  * @file    menuAlarma.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la navegación por el menú "alarma" de Xana.
  *
  *          Requiere las librerías "alarma.h" e "instrucciones.h"
  * */

#include "Instrucciones.h"
#include "alarma.h"

namespace menuAlarma
{

    bool volver = false;
```

```
bool alarmaOn = false;
bool ampm = false;
bool franja = false;
bool am;

#define LUZ 1
#define MUSICA 2
#define PERSIANA 3

/** Carga las instrucciones del menú "alarma" de Xana */
void cargarMenu()
{
    myVR.clear();

    cargarInstruccion(insPoner);
    Serial.println(F("\tCargada instrucción PONER"));

    cargarInstruccion(insQuitar);
    Serial.println(F("\tCargada instrucción QUITAR"));

    cargarInstruccion(insAM);
    Serial.println(F("\tCargada instrucción AM"));

    cargarInstruccion(insPM);
    Serial.println(F("\tCargada instrucción PM"));

    cargarInstruccion(ins1to6);
    Serial.println(F("\tCargada instrucción DE UNA A SEIS"));
```

```
    cargarInstruccion(ins7to12);
    Serial.println(F("\tCargada instrucción DE SIETE A DOCE"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));

    volver = false;
    ampm = false;
    franja = false;
}

/** Carga las instrucciones de un submenú de "alarma" */
void cargarMenu1to6()
{
    myVR.clear();

    cargarInstruccion(ins1);
    Serial.println(F("\tCargada instrucción UNO"));

    cargarInstruccion(ins2);
    Serial.println(F("\tCargada instrucción DOS"));

    cargarInstruccion(ins3);
    Serial.println(F("\tCargada instrucción TRES"));

    cargarInstruccion(ins4);
    Serial.println(F("\tCargada instrucción CUATRO"));
```

```
    cargarInstruccion(ins5);
    Serial.println(F("\tCargada instrucción CINCO"));

    cargarInstruccion(ins6);
    Serial.println(F("\tCargada instrucción SEIS"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));
}

/** Carga las instrucciones de un submenú de "alarma" */
void cargarMenu7to12()
{
    myVR.clear();

    cargarInstruccion(ins7);
    Serial.println(F("\tCargada instrucción SIETE"));

    cargarInstruccion(ins8);
    Serial.println(F("\tCargada instrucción OCHO"));

    cargarInstruccion(ins9);
    Serial.println(F("\tCargada instrucción NUEVE"));

    cargarInstruccion(ins10);
    Serial.println(F("\tCargada instrucción DIEZ"));
```

```
    cargarInstruccion(ins11);
    Serial.println(F("\tCargada instrucción ONCE"));

    cargarInstruccion(ins12);
    Serial.println(F("\tCargada instrucción DOCE"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));
}

/** Carga las instrucciones de un submenú de "alarma" */
void cargarMenuMinutos()
{
    myVR.clear();

    cargarInstruccion(insEnPunto);
    Serial.println(F("\tCargada instrucción EN PUNTO"));

    cargarInstruccion(insY10);
    Serial.println(F("\tCargada instrucción Y DIEZ"));

    cargarInstruccion(insY15);
    Serial.println(F("\tCargada instrucción Y CUARTO"));

    cargarInstruccion(insY30);
    Serial.println(F("\tCargada instrucción Y MEDIA"));

    cargarInstruccion(insY45);
```

```
    Serial.println(F("\tCargada instrucción MENOS CUARTO"));

    cargarInstruccion(insY50);
    Serial.println(F("\tCargada instrucción MENOS DIEZ"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));
}

/** Carga las instrucciones de un submenú de "alarma" */
void cargarMenuRepeticion()
{
    myVR.clear();

    cargarInstruccion(ins1vez);
    Serial.println(F("\tCargada instrucción UNA VEZ"));

    cargarInstruccion(insSemana);
    Serial.println(F("\tCargada instrucción SEMANA"));

    cargarInstruccion(insFindes);
    Serial.println(F("\tCargada instrucción FINDES"));

    cargarInstruccion(insLuz);
    Serial.println(F("\tCargada instrucción LUZ"));

    cargarInstruccion(insPersiana);
    Serial.println(F("\tCargada instrucción PERSIANA"));
```

```
    cargarInstruccion(insMusica);
    Serial.println(F("\tCargada instrucción MUSICA"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));
}

/** Configura la franja horaria */
void elegirHora1to6()
{
    bool horaElegida = false;
    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false
    && horaElegida == false)
    {

        if (comprobarVoz() > 0)
        {
            //Escribe la instrucción recibida por el canal serie
            printVR(buf);

            //Actúa en función de la instrucción recibida
            switch (buf[1])
            {
            case insVolver:
                volver = true;
```

```
        horaElegida = true;

        break;

    case ins1:
        alarma::horaAux = 1;
        horaElegida = true;

        break;

    case ins2:
        alarma::horaAux = 2;
        horaElegida = true;

        break;

    case ins3:
        alarma::horaAux = 3;
        horaElegida = true;

        break;

    case ins4:
        alarma::horaAux = 4;
        horaElegida = true;

        break;
```

```
            case ins5:
                alarma::horaAux = 5;
                horaElegida = true;

                break;

            case ins6:
                alarma::horaAux = 6;
                horaElegida = true;

                break;

            default:
                Serial.println(F("Función no configurada"));
                break;
            }
        }
    }
}

/** Configura la franja horaria */
void elegirHora7to12()
{
    bool horaElegida = false;
    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false
    && horaElegida == false)
```

```
{

    if (comprobarVoz() > 0)
    {
        // Escribe la instrucción recibida por el canal serie
        printVR(buf);

        // Actúa en función de la instrucción recibida
        switch (buf[1])
        {
        case insVolver:
            volver = true;
            horaElegida = true;

            break;

        case ins7:
            alarma::horaAux = 7;
            horaElegida = true;

            break;

        case ins8:
            alarma::horaAux = 8;
            horaElegida = true;

            break;
```

```
case ins9:
    alarma::horaAux = 9;
    horaElegida = true;

    break;

case ins10:
    alarma::horaAux = 10;
    horaElegida = true;

    break;

case ins11:
    alarma::horaAux = 11;
    horaElegida = true;

    break;

case ins12:
    alarma::horaAux = 12;
    horaElegida = true;

    break;

default:
    Serial.println(F("Función no configurada"));
    break;
}
```

```
        }
    }
}

/** Configura los minutos */
void elegirMinutos()
{
    bool minElegidos = false;
    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false
    && minElegidos == false)
    {

        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el canal serie
            printVR(buf);

            // Actúa en función de la instrucción recibida
            switch (buf[1])
            {
            case insVolver:
                volver = true;
                minElegidos = true;

                break;
```

```
case insEnPunto:
    alarma::minutosAux = 0;
    minElegidos = true;

    break;

case insY10:
    alarma::minutosAux = 10;
    minElegidos = true;

    break;

case insY15:
    alarma::minutosAux = 15;
    minElegidos = true;

    break;

case insY30:
    alarma::minutosAux = 30;
    minElegidos = true;

    break;

case insY45:
    alarma::minutosAux = 45;
    minElegidos = true;
    alarma::horaAux--;
```

```
                break;

            case insY50:
                alarma::minutosAux = 50;
                minElegidos = true;
                alarma::horaAux--;

                break;

            default:
                Serial.println(F("Función no configurada"));
                break;
            }
        }
    }
}

/** Configura la repetición de la alarma */
void elegirRepeticion()
{
    bool modoElegido = false;
    bool repElegida = false;

    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false
    && (modoElegido == false || repElegida == false))
```

```
{

    if (comprobarVoz() > 0)
    {
        // Escribe la instrucción recibida por el canal serie
        printVR(buf);

        // Actúa en función de la instrucción recibida
        switch (buf[1])
        {
        case insVolver:
            volver = true;
            modoElegido = true;
            repElegida = true;

            break;

        case ins1vez:
            if (!repElegida)
            {
                alarma::repUnaAux = true;
                alarma::repSemanaAux = false;
                alarma::repFindesAux = false;

                guardarTiempo();

                repElegida = true;
            }
```

```
        else
        {
            Serial.println(F("Elige el modo (luz, música "
            "o persiana)"));
            guardarTiempo();
        }

        break;

    case insSemana:
        if (!repElegida)
        {
            alarma::repUnaAux = false;
            alarma::repSemanaAux = true;
            alarma::repFindesAux = false;

            guardarTiempo();

            repElegida = true;
        }
        else
        {
            Serial.println(F("Elige el modo (luz, música "
            "o persiana)"));
            guardarTiempo();
        }

        break;
```

```
case insFindes:
    if (!repElegida)
    {
        alarma::repUnaAux = false;
        alarma::repSemanaAux = false;
        alarma::repFindesAux = true;

        guardarTiempo();

        repElegida = true;
    }
    else
    {
        Serial.println(F("Elige el modo (luz, música "
        "o persiana)"));
        guardarTiempo();
    }

    break;

case insLuz:
    if (repElegida)
    {
        alarma::modoLuzAux = true;
        alarma::modoMusicaAux = false;
        alarma::modoPersianaAux = false;
```

```
            Serial.println(F("Guardando alarma"));

            modoElegido = true;
        }
        else
        {
           Serial.println(F("Elige la repetición (una vez"
           ", entre semana o findes)"));
           guardarTiempo();
        }

        break;

    case insPersiana:
        if (repElegida)
        {
            alarma::modoLuzAux = false;
            alarma::modoMusicaAux = false;
            alarma::modoPersianaAux = true;

            Serial.println(F("Guardando alarma)"));

            modoElegido = true;
        }
        else
        {
           Serial.println(F("Elige la repetición (una vez"
           ", entre semana o findes)"));
```

```
            guardarTiempo();
        }

        break;

    case insMusica:
        if (repElegida)
        {
            alarma::modoLuzAux = false;
            alarma::modoMusicaAux = true;
            alarma::modoPersianaAux = false;

            Serial.println(F("Guardando alarma)"));

            modoElegido = true;
        }
        else
        {
            Serial.println(F("Elige la repetición (una vez"
            ", entre semana o findes)"));
            guardarTiempo();
        }

        break;

    default:
        Serial.println(F("Función no configurada"));
        break;
```

```
            }
        }
    }
}

/** Atiende a las instrucciones de voz del VRM */
void esperarInstruccion()
{

    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false && volver == false)
    {

        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el canal serie
            printVR(buf);

            // Actúa en función de la instrucción recibida
            switch (buf[1])
            {

            case insVolver:
                volver = true;

                break;
```

```
case insPoner:
    ampm = true;
    Serial.println(F("¿AM o PM?"));
    guardarTiempo();

    break;

case insQuitar:
    alarmaOn = false;
    volver = true;

    break;

case insAM:
    if (ampm)
    {
        am = true;
        franja = true;
        ampm = false;
        Serial.println(F("¿De 1 a 6 o de 7 a 12?"));
        guardarTiempo();
    }
    else
    {
        Serial.println(F("Comando no disponible"));
        guardarTiempo();
    }
```

```
        break;

    case insPM:
        if (ampm)
        {
            am = false;
            franja = true;
            ampm = false;
            Serial.println(F("¿De 1 a 6 o de 7 a 12?"));
            guardarTiempo();
        }
        else
        {
            Serial.println(F("Comando no disponible"));
            guardarTiempo();
        }

        break;

    case ins1to6:
        if (franja)
        {
            cargarMenu1to6();
            elegirHora1to6();
            if (!am)
            {
                alarma::horaAux += 12;
            }
```

```
            if (!volver)
            {
                cargarMenuMinutos();
                elegirMinutos();
            }
            if (!volver)
            {
                cargarMenuRepeticion();
                elegirRepeticion();
            }
            volver = true;
        }
        else
        {
            Serial.println(F("Comando no disponible"));
            guardarTiempo();
        }

        break;

    case ins7to12:
        if (franja)
        {
            cargarMenu7to12();
            elegirHora7to12();
            if (!am)
            {
```

```
                alarma::horaAux += 12;
            }

            if (!volver)
            {
                cargarMenuMinutos();
                elegirMinutos();
            }
            if (!volver)
            {
                cargarMenuRepeticion();
                elegirRepeticion();
            }
            volver = true;
        }
        else
        {
            Serial.println(F("Comando no disponible"));
            guardarTiempo();
        }

        break;

    default:
        Serial.println(F("Función no configurada"));
        break;
    }
}
```

```
        }
    }


} // namespace menuAlarma
```

Anexo 06

-

Alarma.h

```
/**

*****************************************************************
  * @file    alarma.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la configuración de una alarma o rutina
  *          desde menuAlarma.h
  * */

namespace alarma
{

    uint8_t hora = 0;
    uint8_t horaAux = 0;
    uint8_t minutos = 0;
    uint8_t minutosAux = 0;
```

```
bool modoLuz = false;
bool modoLuzAux = false;
bool modoMusica = false;
bool modoMusicaAux = false;
bool modoPersiana = false;
bool modoPersianaAux = false;

bool repUna = false;
bool repUnaAux = false;
bool repSemana = false;
bool repSemanaAux = false;
bool repFindes = false;
bool repFindesAux = false;

/** Comprueba si la hora actual coincide con la programada para la
 * alarma
 * */
void comprobarAlarma()
{
    /*
    si horaAlarma=hour() and minutoAlarma...
    {
        switch modo
        {
            case persiana:
            break;
            ...
        }
```

```
        }
        */
    }

    /** Fija la alarma al completar la configuración desde el menú de
     * voz
     * */
    void configurarAlarma()
    {
        /*
        if horaAux == 24 then horaAux = 0;
        hora = horaAuX;
        */
    }

} // namespace alarma

//-------------------------------------------------------------------
-
//Fin de alarma.h
```

Anexo 07

-

menuEnchufe.h

```
/**

*****************************************************************
  * @file    menuEnchufe.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la navegación por el menú "enchufes" de Xana.
  *
  *          Requiere las librerías "enchufes.h" e "instrucciones.h"
  * */

#include "Instrucciones.h"
#include "enchufes.h"

namespace menuEnchufe
{

    bool volver = false;
```

```
bool numero = false; //¿Se ha configurado el estado del enchufe?
bool estado = false; //¿Se ha especificado el nº del enchufe?

/** Carga las instrucciones del menú "enchufes" de Xana */
void cargarMenu()
{
    myVR.clear();

    cargarInstruccion(insEnciende);
    Serial.println(F("\tCargada instrucción ENCIENDE"));

    cargarInstruccion(insApaga);
    Serial.println(F("\tCargada instrucción APAGA"));

    cargarInstruccion(ins1);
    Serial.println(F("\tCargada instrucción UNO"));

    cargarInstruccion(ins2);
    Serial.println(F("\tCargada instrucción DOS"));

    cargarInstruccion(ins3);
    Serial.println(F("\tCargada instrucción TRES"));

    cargarInstruccion(ins4);
    Serial.println(F("\tCargada instrucción CUATRO"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));
```

```
    volver = false;
    numero = false;
    estado = false;
}

/** Atiende a las instrucciones de voz del VRM */
void esperarInstruccion()
{

    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false && volver == false)
    {

        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el canal serie
            printVR(buf);

            // Actúa en función de la instrucción recibida
            switch (buf[1])
            {

            case insVolver:
                volver = true;
                break;
```

```
case insEnciende:
    enchufe::encender();
    estado = true;

    if (numero)
    {
        enchufe::configurar();
        volver = true;
    }
    else
    {
        guardarTiempo();
    }
    break;

case insApaga:
    enchufe::apagar();
    estado = true;

    if (numero)
    {
        enchufe::configurar();
        volver = true;
    }
    else
    {
        guardarTiempo();
    }
```

```
        break;

    case ins1:

        enchufe::setNumero(1);
        numero = true;

        if (estado)
        {
            enchufe::configurar();
            volver = true;
        }
        else
        {
            guardarTiempo();
        }
        break;

    case ins2:

        enchufe::setNumero(2);
        numero = true;

        if (estado)
        {
            enchufe::configurar();
            volver = true;
        }
```

```
    else
    {
        guardarTiempo();
    }
    break;

case ins3:

    enchufe::setNumero(3);
    numero = true;

    if (estado)
    {
        enchufe::configurar();
        volver = true;
    }
    else
    {
        guardarTiempo();
    }
    break;

case ins4:

    enchufe::setNumero(4);
    numero = true;

    if (estado)
```

```
                    {
                        enchufe::configurar();
                        volver = true;
                    }
                    else
                    {
                        guardarTiempo();
                    }
                    break;

                default:
                    Serial.println(F("Función no configurada"));
                    break;
                }
            }
        }
    }

} //namespace menuEnchufe

//-----------------------------------------------------------------------
-
//Fin de menuEnchufe.h
```

Anexo 08

-

Enchufes.h

```
/**
*****************************************************************
  * @file    enchufes.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para el control por voz de cuatro enchufes desde
  *          menuEnchufe.h
  * */

namespace enchufe
{

    bool modo;
    uint8_t numero;

    void encender()
    {
```

```
    modo = true;
}

void apagar()
{
    modo = false;
}

void setNumero(uint8_t n)
{
    numero = n;
}

/** Envía la orden de encender/apagar el enchufe configurado */
void configurar()
{
    //Cambia el puerto de comunicación serie del VRM al ESP32
    myVR.end();
    EspSerial.begin(115200);



    EspSerial.print(F("Orden enviada:\n"));
    EspSerial.print(F("ENC"));
    EspSerial.print(numero);
    if (modo)
    {
        EspSerial.println("1");
    }
```

```
        else
        {
            EspSerial.println("0");
        }

        //Devuelve el puerto de comunicación serie del ESP32 al VRM
        EspSerial.end();
        myVR.begin(9600);
    }

} // namespace enchufe

//-------------------------------------------------------------------------
-
//Fin de enchufes.h
```

Anexo 09

-

menuLuz.h

```
/**
*****************************************************************
  * @file    menuLuz.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la navegación por el menú "luz" de Xana.
  *
  *          Requiere las librerías "luz.h" e "instrucciones.h"
  * */

#include "Instrucciones.h"
#include "luz.h"

namespace menuLuz
{

    bool volver = false;
    bool multicolor = false;
    bool autoLuz = false;
```

```
bool autoColor = false;

/** Carga las instrucciones del menú "luz" de Xana */
void cargarMenu()
{
    myVR.clear();

    cargarInstruccion(insEnciende);
    Serial.println(F("\tCargada instrucción ENCIENDE"));

    cargarInstruccion(insApaga);
    Serial.println(F("\tCargada instrucción APAGA"));

    cargarInstruccion(insSube);
    Serial.println(F("\tCargada instrucción SUBE"));

    cargarInstruccion(insBaja);
    Serial.println(F("\tCargada instrucción BAJA"));

    cargarInstruccion(insColor);
    Serial.println(F("\tCargada instrucción COLOR"));

    cargarInstruccion(insAuto);
    Serial.println(F("\tCargada instrucción AUTOMÁTICO"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));

    volver = false;
}
```

```
/** Carga las instrucciones de un submenú de "luz" */
void cargarMenuColor()
{

    myVR.clear();

    cargarInstruccion(insFria);
    Serial.println(F("\tCargada instrucción FRÍA"));

    cargarInstruccion(insCalida);
    Serial.println(F("\tCargada instrucción CÁLIDA"));

    cargarInstruccion(insRoja);
    Serial.println(F("\tCargada instrucción ROJO"));

    cargarInstruccion(insAzul);
    Serial.println(F("\tCargada instrucción AZUL"));

    cargarInstruccion(insVerde);
    Serial.println(F("\tCargada instrucción VIOLETA"));

    cargarInstruccion(insMulticolor);
    Serial.println(F("\tCargada instrucción MULTICOLOR"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));

    volver = false;
}

/** Carga las instrucciones de un submenú de "luz" */
```

```
void cargarMenuAuto()
{
    myVR.clear();

    cargarInstruccion(insActivar);
    Serial.println(F("\tCargada instrucción ACTIVAR"));

    cargarInstruccion(insSoloLuz);
    Serial.println(F("\tCargada instrucción SOLO LUZ"));

    cargarInstruccion(insSoloColor);
    Serial.println(F("\tCargada instrucción SOLO COLOR"));

    cargarInstruccion(insDesactivar);
    Serial.println(F("\tCargada instrucción DESACTIVAR"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));

    volver = false;
}

/** Atiende a las instrucciones de voz del VRM */
void esperarInstruccion()
{

    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false && volver == false)
    {
```

```
if (comprobarVoz() > 0)
{
    // Escribe la instrucción recibida por el canal serie
    printVR(buf);

    // Actúa en función de la instrucción recibida
    switch (buf[1])
    {

    case insVolver:
        volver = true;

        break;

    case insEnciende:
        luz::enciende();

        break;

    case insApaga:
        luz::apaga();
        multicolor = false;
        break;

    case insSube:
        luz::mas();
        break;

    case insBaja:
        luz::menos();
        break;
```

```
case insColor:
    cargarMenuColor();
    guardarTiempo();

    /** Submenú COLOR */
    while (comprobarTiempo(esperaMax) == false
    && volver == false)
    {
        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el
            //canal serie
            printVR(buf);

            // Actúa en función de la instrucción
            //recibida
            switch (buf[1])
            {

            case insVolver:
                volver = true;
                break;

            case insFria:
                luz::fria();
                multicolor = false;
                volver = true;
                break;

            case insCalida:
```

```
            luz::calida();
            multicolor = false;
            volver = true;
            break;

        case insRoja:
            luz::roja();
            multicolor = false;
            volver = true;
            break;

        case insAzul:
            luz::azul();
            multicolor = false;
            volver = true;
            break;

        case insVerde:
            luz::verde();
            multicolor = false;
            volver = true;
            break;

        case insMulticolor:
            multicolor = true;
            volver = true;
            break;

        default:
            Serial.println(F("Función no "
            "configurada"));
```

```
                break;
            }
        }
    }

    volver = true;
    break;

case insAuto:

    cargarMenuAuto();
    guardarTiempo();

    /** Submenú AUTOMÁTICO */
    while (comprobarTiempo(esperaMax) == false
    && volver == false)
    {
        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el
            //canal serie
            printVR(buf);

            // Actúa en función de la instrucción
            //recibida
            switch (buf[1])
            {

            case insVolver:
                volver = true;
                break;
```

```
case insActivar:
    autoLuz = true;
    autoColor = true;
    volver = true;
    break;

case insSoloLuz:
    autoLuz = true;
    autoColor = false;
    volver = true;
    break;

case insSoloColor:
    autoLuz = false;
    autoColor = true;
    volver = true;
    break;

case insDesactivar:
    autoLuz = false;
    autoColor = false;
    volver = true;
    break;

default:
    Serial.println(F("Función no "
    "configurada"));
    break;
}
```

```
                            luz::enviaAuto(autoLuz, autoColor,
                            multicolor);
                        }
                    }
                    volver = true;
                    break;

                default:
                    Serial.println(F("Función no configurada"));
                    break;
                }
            }
        }
    }

} // namespace menuLuz

//-------------------------------------------------------------------
-
//Fin de menuLuz.h
```

Anexo 10

-

Luz.h

```
/**

*****************************************************************
  * @file    luz.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para el control por voz de bombillas y tiras LED RGB
  *          desde menuEnchufe.h
  * */

namespace luz
{

    uint8_t intensidad; //intensidad de la luz (de 0 a 100)
    uint8_t colorR = 255;
    uint8_t colorG = 255;
    uint8_t colorB = 255;
```

```
void escribir(uint8_t intens = intensidad, uint8_t red = colorR,
uint8_t green = colorG, uint8_t blue = colorB)
{
    //Cambia el puerto de comunicación serie del VRM al ESP32
    myVR.end();
    EspSerial.begin(115200);

    //escribe intens en la luz
    EspSerial.print(F("Orden enviada:\n"));
    EspSerial.print("LUZ");

    if (intens < 100)
    {
        EspSerial.print(F("0"));
        if (intens < 10)
        {
            EspSerial.print(F("0"));
        }
    }
    EspSerial.print(intens);

    if (red < 100)
    {
        EspSerial.print(F("0"));
        if (red < 10)
        {
            EspSerial.print(F("0"));
```

```
    }
}
EspSerial.print(red);

if (green < 100)
{
    EspSerial.print(F("0"));
    if (green < 10)
    {
        EspSerial.print(F("0"));
    }
}
EspSerial.print(green);

if (blue < 100)
{
    EspSerial.print(F("0"));
    if (blue < 10)
    {
        EspSerial.print(F("0"));
    }
}
EspSerial.print(blue);

//Devuelve el puerto de comunicación serie del ESP32 al VRM
EspSerial.end();
myVR.begin(9600);
```

```
}

/** Enciende la luz al 100% de su intensidad con el color
 *  configurado anteriormente
 *  */
void enciende()
{
    intensidad = 100;
    escribir(intensidad);
}

/** Apaga la luz */
void apaga()
{
    intensidad = 0;
    escribir(intensidad);
}

/** Incrementa en un 25% la intensidad de la luz si es posible */
void mas()
{
    if (intensidad > 75)
    {
        intensidad = 100;
    }
    else
    {
        intensidad += 25;
```

```
    }
    escribir(intensidad);
}

/** Decrementa en un 25% la intensidad de la luz si es posible */
void menos()
{
    if (intensidad < 25)
    {
        intensidad = 0;
    }
    else
    {
        intensidad -= 25;
    }

    escribir(intensidad);
}

/** Establece una tonalidad de luz fría con la intensidad
 *  configurado anteriormente
 *  */
void fria ()
{
    colorR = 201;
    colorG = 226;
    colorB = 255;
    escribir(intensidad, colorR, colorG, colorB);
```

```
}

/** Establece una tonalidad de luz fría con la intensidad
 *  configurado anteriormente
 *  */
void calida ()
{
    colorR = 255;
    colorG = 197;
    colorB = 143;
    escribir(intensidad, colorR, colorG, colorB);
}

/** Enciende la luz roja con la intensidad
 *  configurada anteriormente
 *  */
void roja ()
{
    colorR = 255;
    colorG = 0;
    colorB = 0;
    escribir(intensidad, colorR, colorG, colorB);
}

/** Enciende la luz azul con la intensidad
 *  configurada anteriormente
 *  */
void azul ()
```

```
{
    colorR = 0;
    colorG = 0;
    colorB = 255;
    escribir(intensidad, colorR, colorG, colorB);
}

/** Enciende la luz verde con la intensidad
 *  configurada anteriormente
 *  */
void verde ()
{
    colorR = 0;
    colorG = 255;
    colorB = 0;
    escribir(intensidad, colorR, colorG, colorB);
}

/** Activa y desadctiva los modos automáticos de intensidad y
 * color
 * */
void enviaAuto(bool intens, bool color, bool multi){

    //Cambia el puerto de comunicación serie del VRM al ESP32
    myVR.end();
    EspSerial.begin(115200);

    //escribe los modos automáticos de la luz
```

```
EspSerial.print(F("Orden enviada:\n"));
EspSerial.print("AUTL");

if (intens)
{
    EspSerial.print(F("1"));
}
else
{
    EspSerial.print(F("0"));
}

if (color)
{
    EspSerial.print(F("1"));
}
else
{
    EspSerial.print(F("0"));
}

if (multi)
{
    EspSerial.print(F("1"));
}
else
{
    EspSerial.print(F("0"));
```

```
        }

        //Devuelve el puerto de comunicación serie del ESP32 al VRM
        EspSerial.end();
        myVR.begin(9600);

    }


} // namespace luz

//-------------------------------------------------------------------------
-
//Fin de luz.h
```

Anexo 11

-

menuMusica.h

```
/**

*****************************************************************
  * @file    menuMusica.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la navegación por el menú "música" de Xana.
  *
  *          Requiere las librerías "musica.h" e "instrucciones.h"
  * */

#include "Instrucciones.h"
#include "musica.h"

namespace menuMusica
{

    bool volver = false;
```

```
/** Carga las instrucciones del menú "luz" de Xana */
void cargarMenu()
{
    myVR.clear();

    cargarInstruccion(insAmbiente);
    Serial.println(F("\tCargada instrucción AMBIENTE"));

    cargarInstruccion(insFiesta);
    Serial.println(F("\tCargada instrucción DE FIESTA"));

    cargarInstruccion(insRelax);
    Serial.println(F("\tCargada instrucción RELAJNBTE"));

    cargarInstruccion(insQuitar);
    Serial.println(F("\tCargada instrucción QUITAR"));

    cargarInstruccion(insBluetooth);
    Serial.println(F("\tCargada instrucción BLUETOOTH"));

    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));

    volver = false;
}

/** Atiende a las instrucciones de voz del VRM */
```

```
void esperarInstruccion()
{
    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false && volver == false)
    {

        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el canal serie
            printVR(buf);

            // Actúa en función de la instrucción recibida
            switch (buf[1])
            {

            case insVolver:
                volver = true;

                break;

            case insAmbiente:
                musica::reproducirDesde(ambiente);
                volver = true;

                break;

            case insFiesta:
```

```
        musica::reproducirDesde(fiesta);
        volver = true;

        break;

    case insRelax:
        musica::reproducirDesde(relajante);
        volver = true;

        break;

    case insBluetooth:
        musica::activarBluetooth();
        volver = true;

        break;

    case insQuitar:
        musica::quitar();
        volver = true;

        break;

    default:
        Serial.println(F("Función no configurada"));

        break;
    }
```

```
            }
        }
    }


} // namespace menuMusica


//-------------------------------------------------------------------------
-
//Fin de menuMusica.h
```

Anexo 12

-

musica.h

```
/**

*****************************************************************
  * @file    musica.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la reproducción de archivos de audio desde una
  *          tarjeta SD a través de menuEnchufe.h
  * */

#define relajante "Relax"
#define ambiente "Ambiente"
#define fiesta "Fiesta"

uint16_t maxRand = 65535;
uint16_t numCancion;
String cancion;
bool conectado = false;
```

```
namespace musica
{
    /** reproduce un archivo ".wav" aleatoriamente con un nombre entre
     * "1.wav" y "65535.wav"
     */
    void reproducirDesde(String carpeta)
    {
        //Chequea la conexión con la tarjeta micro SD
        if(!conectado)
        {
            Serial.println("Reconectando con tarjeta microSD");
            conectado = SD.begin(SDPinSelect);
        }
        if(conectado)
        {
            do
            {
                numCancion = random(1, maxRand);

                cancion = carpeta;
                cancion += "/";
                cancion += numCancion;
                cancion += ".wav";

                //Si no existe ese archivo, descarta todos los que se
                //llamen un número superior en la siguiente búsqueda
                maxRand = numCancion;
```

```
                Serial.print("Buscando archivo ");
                Serial.println(cancion);

            } while (!SD.exists(cancion) && numCancion != 1);

            if(!SD.exists(cancion) && numCancion == 1)
            {
                Serial.println("Error: carpeta vacía o no
existente.");
            }
            else
            {
                const char *archivo = cancion.c_str();
                altavoz.play(archivo);
                Serial.print("Reproduciendo archivo ");
                Serial.println(cancion);
            }

            //Resetea el límite superior de búsqueda
            maxRand = 65535;
        }
    }

    //Para la reproducción de audio
    void quitar()
    {
        altavoz.stopPlayback();
        //Descativar bluetooth
```

```
    }

    //Activa la recepción de audio por Bluetooth Low Energy
    void activarBluetooth()
    {
        //No disponible en esta versión
    }

} // namespace musica

//-------------------------------------------------------------------------
-
//Fin de musica.h
```

Anexo 13

-

menuPersiana.h

```
/**

*****************************************************************
  * @file    menuPersiana.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la navegación por el menú "persiana" de Xana.
  *
  *          Requiere las librerías "persiana.h" e "instrucciones.h"
  **/

#include "Instrucciones.h"
#include "persiana.h"

namespace menuPersiana
{

    bool volver = false;
```

```
bool poco = false;
bool modoAuto = false;
// Será true si se va a subir la persiana o false si se va a bajar
bool direccion;

/** Carga las instrucciones del menú "luz" de Xana */
void cargarMenu()
{
    myVR.clear();

    cargarInstruccion(insSube);
    Serial.println(F("\tCargada instrucción SUBE"));

    cargarInstruccion(insBaja);
    Serial.println(F("\tCargada instrucción BAJA"));

    cargarInstruccion(insPoco);
    Serial.println(F("\tCargada instrucción UN POCO"));

    cargarInstruccion(insAuto);
    Serial.println(F("\tCargada instrucción MODO AUTO"));

    cargarInstruccion(insActivar);
    Serial.println(F("\tCargada instrucción ACTIVAR"));

    cargarInstruccion(insDesactivar);
    Serial.println(F("\tCargada instrucción DESACTIVAR"));
```

```
    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));

    volver = false;
    poco = false;
    modoAuto = false;
}

/** Atiende a las instrucciones de voz del VRM */
void esperarInstruccion()
{
    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false && volver == false)
    {

        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el canal serie
            printVR(buf);

            // Actúa en función de la instrucción recibida
            switch (buf[1])
            {

            case insVolver:
                volver = true;
```

```
    break;

case insSube:
    if (!modoAuto)
    {
        direccion = true;
    }

    break;

case insBaja:
    if (!modoAuto)
    {
        direccion = false;
    }

    break;

case insPoco:
    if (!modoAuto)
    {
        poco = true;
    }

    break;

case insAuto:
    modoAuto = true;
```

```
            guardarTiempo();

            break;

        case insActivar:
            if (modoAuto)
            {
                persiana::controlAutomatico(true);
            }

            break;

        case insDesactivar:
            if (modoAuto)
            {
                persiana::controlAutomatico(false);
            }

            break;

        default:
            Serial.println(F("Función no configurada"));
            break;
        }
    }
}

if (!modoAuto)
```

```
        {
            persiana::mover(direccion, poco);

        }
    }

} //namespace menuPersiana

//-------------------------------------------------------------------------
-
//Fin de menuPersiana.h
```

Anexo 14

-

persiana.h

```
/**

*****************************************************************
  * @file    persiana.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para el control por voz de un motor de persiana desde
  *          desde menuPersiana.h
  * */


namespace persiana
{
    /** Activa el movimiento ascendente o descendente de la persiana
     * según la variable "sube" e indica si hasta su tope o solamenete
     * un 20% desde su posición actual según la variable "poco"
     */
    void mover(bool sube, bool poco)
```

```
{
    //Cambia el puerto de comunicación serie del VRM al ESP32
    myVR.end();
    EspSerial.begin(115200);



    EspSerial.print(F("Orden enviada:\n"));


    if (sube)
    {
        EspSerial.print(F("PERsube"));
    }
    else
    {
        EspSerial.print(F("PERbaja"));
    }


    if (poco)
    {
        EspSerial.println(F("x"));
    }
    else
    {
        EspSerial.println(F("1"));
    }


    //Devuelve el puerto de comunicación serie del ESP32 al VRM
    EspSerial.end();
```

```
        myVR.begin(9600);
    }

    /** Activa y desactiva el movimiento automático de las persianas
*/
    void controlAutomatico(bool x)
    {
         //Cambia el puerto de comunicación serie del VRM al ESP32
        myVR.end();
        EspSerial.begin(115200);



        EspSerial.print(F("Orden enviada:\n"));
        EspSerial.print(F("AUTP"));

        if (x)
        {
            EspSerial.println(F("1"));
        }
        else
        {
            EspSerial.println(F("0"));
        }

        //Devuelve el puerto de comunicación serie del ESP32 al VRM
        EspSerial.end();
        myVR.begin(9600);
    }
```

```
} // namespace persiana


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

//Fin de persiana.h
```

Anexo 15

-

menuTemperatura.h

```
/**

*****************************************************************
  * @file    menuTemperatura.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para la navegación por el menú "temperatura" de Xana.
  *
  *          Requiere las librerías "temperatura.h" e
"instrucciones.h"
  * */


#include "Instrucciones.h"
#include "temperatura.h"


namespace menuTemp
{
```

```
bool volver = false;
// Será true para subir la temperatura y false para bajarla
bool direccion;
bool subeBaja = false;

/** Carga las instrucciones del menú "luz" de Xana */
void cargarMenu()
{
    myVR.clear();

    cargarInstruccion(insEnciende);
    Serial.println(F("\tCargada instrucción ENCIENDE"));

    cargarInstruccion(insApaga);
    Serial.println(F("\tCargada instrucción APAGA"));

    cargarInstruccion(insSube);
    Serial.println(F("\tCargada instrucción SUBE"));

    cargarInstruccion(insBaja);
    Serial.println(F("\tCargada instrucción BAJA"));

    cargarInstruccion(insSwing);
    Serial.println(F("\tCargada instrucción MUCHO"));

    cargarInstruccion(insPoco);
    Serial.println(F("\tCargada instrucción POCO"));
```

```
    cargarInstruccion(insVolver);
    Serial.println(F("\tCargada instrucción VOLVER\n"));

    volver = false;
    subeBaja = false;
}

/** Atiende a las instrucciones de voz del VRM */
void esperarInstruccion()
{
    guardarTiempo();

    while (comprobarTiempo(esperaMax) == false && volver == false)
    {

        if (comprobarVoz() > 0)
        {
            // Escribe la instrucción recibida por el canal serie
            printVR(buf);

            // Actúa en función de la instrucción recibida
            switch (buf[1])
            {

            case insVolver:
                volver = true;
                break;
```

```
case insEnciende:
    temp::encender();
    volver = true;
    break;

case insApaga:
    temp::apagar();
    volver = true;
    break;

case insSube:
    //Sube dos grados la temperatura
    temp::mas();
    delay(400);
    temp::mas();
    subeBaja = true;
    direccion = true;
    guardarTiempo();
    break;

case insBaja:
    //Baja dos grados la temperatura
    temp::menos();
    delay(400);
    temp::menos();
    subeBaja = true;
    direccion = false;
    guardarTiempo();
```

```
    break;

case insSwing: //Activa el modo swing
    temp::swing();
    volver = true;
    break;

case insPoco:
    //Sube o baja 1 grado la temperatura en dirección
    //contraria a la orden original para compensar
    //los dos grados que ésta "mueve".
    if (subeBaja)
    {
        if (direccion)
        {
            temp::menos();
        }
        else
        {
            temp::mas();
        }
        volver = true;
    }
    break;

default:
    Serial.println(F("Función no configurada"));
    break;
```

```
                }
            }
        }
    }

} // namespace menuTemp


//-------------------------------------------------------------------------
-
//Fin de menuTemperatura.h
```

Anexo 16

-

temperatura.h

```
/**
*****************************************************************
  * @file    temperatura.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    16/06/2020
  * @brief   Incluye un namespace con todos los comandos necesarios
  *          para el control por voz de un sistema de aire
  *          acondicionado desde menuTemperatura.h
  * */

namespace temp
{
    /** Enciende el sistema de aire acondicionado */
    void encender()
    {
        //Cambia el puerto de comunicación serie del VRM al ESP32
        myVR.end();
        EspSerial.begin(115200);
```

```
    EspSerial.print(F("Orden enviada:\n"));
    EspSerial.println(F("TEMonn"));

    //Devuelve el puerto de comunicación serie del ESP32 al VRM
    EspSerial.end();
    myVR.begin(9600);
}

/** Apaga el sistema de aire acondicionado */
void apagar()
{
    //Cambia el puerto de comunicación serie del VRM al ESP32
    myVR.end();
    EspSerial.begin(115200);

    EspSerial.print(F("Orden enviada:\n"));
    EspSerial.println(F("TEMoff"));

    //Devuelve el puerto de comunicación serie del ESP32 al VRM
    EspSerial.end();
    myVR.begin(9600);
}

/** Aumenta un grado la temperatura objetivo del sistema de aire
 *  acondicionado
 *  */
void mas()
```

```
{
    //Cambia el puerto de comunicación serie del VRM al ESP32
    myVR.end();
    EspSerial.begin(115200);

    EspSerial.print(F("Orden enviada:\n"));
    EspSerial.println(F("TEMmas"));

    //Devuelve el puerto de comunicación serie del ESP32 al VRM
    EspSerial.end();
    myVR.begin(9600);
}

/** Disminuye un grado la temperatura objetivo del sistema de aire
 *  acondicionado
 *  */
void menos()
{
    //Cambia el puerto de comunicación serie del VRM al ESP32
    myVR.end();
    EspSerial.begin(115200);

    EspSerial.print(F("Orden enviada:\n"));
    EspSerial.println(F("TEMmen"));

    //Devuelve el puerto de comunicación serie del ESP32 al VRM
    EspSerial.end();
    myVR.begin(9600);
```

```
    }

    /** Activa el modo de esparsión del aire mediante el movimiento de
     * las paletas del sistema de aire acondicionado
     *  */
    void swing()
    {
        //Cambia el puerto de comunicación serie del VRM al ESP32
        myVR.end();
        EspSerial.begin(115200);

        EspSerial.println(F("TEMswg"));

        //Devuelve el puerto de comunicación serie del ESP32 al VRM
        EspSerial.end();
        myVR.begin(9600);
    }

} // namespace temp

//-----------------------------------------------------------------------
-
//Fin de temperatura.h
```

Anexo 17

-

XanaClient.cpp

```
/**

*********************************************************************
  * @file    XanaClient.cpp
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    10/07/2020
  * @brief   El ESP32 recibe las órdenes del Arduino y el VRM y las
  *          comunica a los servidores de los actuadores remotos.
  * */

/** ---------------------- Includes y defines ----------------------
*/

#include <Arduino.h>
#include "Xana_ClientLib.h"

bool persianaAuto = false;
bool autoColor = false;
bool autoLuz = false;
```

```
// Variable para la persiana:
bool altura;

// Variables para los enchufes:
uint32_t enchufe;
bool estado;

//Cadenas auxiliares
String recibido;
String orden;
String parte;

/** ---------------------------- Setup ----------------------------
*/

void setup()
{
  Serial.begin(115200);
  Serial.println("\n\n\nComunicación arduino - ESP32 preparada\n\n");

  analogSetWidth(9);
  pinMode(LDR,INPUT);
}

/** ---------------------------- Loop ----------------------------
*/

void loop()
```

```
{

/** -- Recibe y clasifica las instrucciones del canal serie -- */
  if (Serial.available() > 0)
  {
    // lee la cadena de entrada:
    recibido = Serial.readStringUntil('\n');

    //Si es un comando, ejecuta la orden solicitada
    if (recibido == "Orden enviada:")
    {
      Serial.println("\n--- Orden recibida: ---");

      //Lee la orden
      orden = Serial.readStringUntil('\n');
      Serial.println(orden);

      //Lee el código de 3 dígitos correspondiente al actuador
      parte = orden.substring(0, 3);
      Serial.println("Actuador: " + parte);

      //Elige entre los 4 actuadores:
      const char *codigo = parte.c_str();
      switch (str2int(codigo))
      {
      /**
       * Código: LUZXXXRRRGGGBBB
       * XXX corresponde a la intensidad (0..100)
```

```
 * RRR, GGG y BBB son los parámetros del color (0..255)
 */
case str2int("LUZ"):
  Serial.println("Parámetros:");

  parte = orden.substring(3, 6);
  intensidad = parte.toInt();
  Serial.print("-- Intensidad: ");
  Serial.println(intensidad);

  parte = orden.substring(6, 9);
  red = parte.toInt();
  Serial.print("-- R: ");
  Serial.println(red);

  parte = orden.substring(9, 12);
  green = parte.toInt();
  Serial.print("-- G: ");
  Serial.println(green);

  parte = orden.substring(12, 15);
  blue = parte.toInt();
  Serial.print("-- B: ");
  Serial.println(blue);

  serverLuz::enviar(intensidad, red, green, blue);
  break;
```

```
        /**
       * Código: PERXXXXY
       * XXXX será "sube" o "baja"
       * Si Y es 1, se subirá bajará la persiana al 100%. Si no, un
20%
       */
      case str2int("PER"):
        Serial.println("Parámetros:");

        parte = orden.substring(7);
        if(parte){
          altura = true;
        }
        else
        {
          altura = false;
        }

        parte = orden.substring(3, 7);
        Serial.println("Dirección: " + parte);
        if(altura){
          Serial.println("Posición: del todo");
        }
        else{
          Serial.println("Posición: 25% más");
        }

        serverPer::enviar(parte, altura);
```

```
  break;

  /**
 * Código: ENCXY
 * X corresponde al número de identificación del enchufe (1..4)
 * Y corresponde al estado del enchufe (0, 1)
 */
case str2int("ENC"):
  Serial.println("Parámetros:");

  parte = orden.substring(3, 4);
  enchufe = parte.toInt();
  Serial.print("-- Enchufe ");
  Serial.println(enchufe);

  parte = orden.substring(4, 5);
  estado = parte.toInt();
  Serial.print("-- Estado: ");
  if (estado)
  {
    Serial.println("on");
  }
  else
  {
    Serial.println("off");
  }

  serverEnc::enviar(enchufe, estado);
```

```
  break;


  /**
 * Códigos posibles:
 * TEMonn: enciende
 * TEMoff: apaga
 * TEMmas: sube la temperatura objetivo
 * TEMmen: baja la temperatura objetivo
 * TEMswg: actva/desactiva el modo swing
 */
case str2int("TEM"):
  parte = orden.substring(3, 6);


  if (parte == "onn")
  {
    Serial.println("Encendiendo sistema de temperatura");
  }
  else if (parte == "off")
  {
    Serial.println("Apagando sistema de temperatura");
  }
  else if (parte == "mas")
  {
    Serial.println("Temperatura objetivo aumentada");
  }
  else if (parte == "men")
  {
```

```
          Serial.println("Temperatura objetivo disminuida");
         }
         else if (parte == "swg")
         {
          Serial.println("Cambiando el modo swing");
         }
         serverTem::enviar(parte);

         break;

         /**
        * Códigos posibles:
        * AUTLXYZ: donde X es el valor (0 o 1) del control automático
de
        * la intensidad de la luz, Y del color y Z del efecto
multicolor
        * AUTPX: donde X es el valor (0 o 1) del control automático
        * de la persiana
        */
       case str2int("AUT"):
         parte = orden.substring(3, 4);

         if (parte == "L")
         {
          Serial.println("Luz automática: ");

          parte = orden.substring(4,5);
          if(parte){
            Serial.println("Intensidad: sí");
```

```
  autoLuz = true;
}
else
{
  Serial.println("Intensidad: no");
  autoLuz = false;
}

parte = orden.substring(5,6);
if(parte){
  Serial.println("Color: sí");
  autoColor = true;
}
else
{
  Serial.println("Color: no");
  autoColor = false;
}

parte = orden.substring(6,7);
if(parte){
  Serial.println("Multicolor: sí");
  //Si se envía una intensidad del 137%, en lugar de
  //cambiar el color, se activará el modo FADE del mando IR
  serverLuz::enviar(137);
}
else
{
```

```
      Serial.println("Multicolor: no");
      red = 255;
      green = 255;
      blue = 255;
      serverLuz::enviar(80);
    }

  }

  else if (parte == "P")
  {
    Serial.print("Persiana automática: ");
    parte = orden.substring(4, 5);
    if(parte){
      Serial.println("activada");
      persianaAuto = true;
    }
    else
    {
      Serial.println("desactivada");
      persianaAuto = false;
    }

  }

  break;

default:
```

```
        Serial.println("----- ERROR: Tipo de actuador no ·"
                       "identificado ------\n");
        break;
      }
    }
  }

  if (persianaAuto)
  {
    control::persiana();
  }

  if (autoLuz)
  {
    control::luz();
  }

  if (autoColor)
  {
    control::color();
  }

}

//-------------------------------------------------------------------
-
//Fin de XanaClient.cpp
```

Anexo 18

-

XanaWifi.h

```
/**

*****************************************************************
  * @file    XanaWifi.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    06/07/2020
  * @brief   Incluye todos los comandos necesarios para el
  *          funcionamiento de las conexiones wifi del ESP32
  * */

/** --------------------------- Includes ---------------------------
*/

#include <Arduino.h>
#include <WiFi.h>
#include <HTTPClient.h>

/** ---------------------- Código del cliente ----------------------
*/
```

```
HTTPClient http;


/** Se conecta al servidor  serverSSID con contraseña serverPassword*/

void connectToServer(const char *serverSSID,const char
*serverPassword)

{


    Serial.print("\nConectando a: ");

    Serial.print(serverSSID);

    Serial.print(" ");


    WiFi.begin(serverSSID, serverPassword);


    while ((!(WiFi.status() == WL_CONNECTED)))

    {

        delay(80);

        Serial.print(".");

    }

    Serial.print("\nConectado con IP ");

    Serial.println((WiFi.localIP()));

}


/** Se desconecta del servidor */

void disconnectFromServer()

{

    WiFi.disconnect();

    Serial.println("\nSe ha desconectado del servidor");

}
```

```
/** Envía una petición HTTP request. Escribe la respuesta del servidor
 * por el canal serie si se solicita.
 * */
void sendRequest(const char *request, bool feedback = false)
{

    if (WiFi.status() == WL_CONNECTED)
    {
        //Request que envía al servidor
        http.begin(request);

        //Recibe la respuesta del servidor
        int8_t httpResponseCode = http.GET();

        if (feedback)
        {
            if (httpResponseCode > 0)
            {
                Serial.print("Código de respuesta del servidor: ");
                Serial.println(httpResponseCode);
                Serial.println();
            }
            else
            {
                Serial.print("Error del servidor: ");
                Serial.println(httpResponseCode);
                Serial.println();
            }
```

```
        }

        //Libera los recursos
        http.end();
    }
    else
    {
        Serial.println("Conexión con el servidor no establecida");
        return;
    }
}

/** ----------------------- Código del servidor --------------------
*/

WiFiServer server(80);

/** Variable que guarda la request HTTP del cliente */
String header;

/** Crea un punto de acceso wifi llamado serverSSID con contraseña
* serverPassword
* */
void createServer(const char *serverSSID, const char *serverPassword)
{
    // Crea la red WiFi
    Serial.print("Creando la red del servidor...");
    WiFi.softAP(serverSSID, serverPassword);
```

```
    //Escribe la IP estática de la red wifi
    IPAddress IP = WiFi.softAPIP();
    Serial.print("\nDirección IP del servidor: ");
    Serial.println(IP);

    //Activa el servidor
    server.begin();
}

//-------------------------------------------------------------------
-
//Fin de XanaWifi.h
```

Anexo 19

-

Xana_ClientLib.h

```
/**

*****************************************************************
  * @file    Xana_ClientLib.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    10/07/2020
  * @brief   Incluye todos los comandos necesarios para comunicar las
  *          órdenes de voz a los servidores de los actuadores
  * */

/** ---------------------- Includes y defines ----------------------
*/

#include <Arduino.h>
#include "XanaWifi.h"

#define LDR 23
#define voltajeMinimoPersiana 1.0
#define voltajeMinimoLuz 2.5
```

```
#define voltajeMaximoLuz 3.0

/** --------------------- Variables auxiliares ---------------------
*/

uint8_t intensidad;

uint8_t red;

uint8_t green;

uint8_t blue;

float voltaje = 3.3;

/** ---------------------- Credenciales Wi-Fi ----------------------
*/

const char *ssidLuz = "Xana_Luz_server";

const char *ssidPersiana = "Xana_Pers_server";

const char *ssidEnchufe = "Xana_Enchufes_server";

const char *ssidTemperatura = "Xana_Temp_server";

const char *password = "123456789";

/** --------------------- Casteo de variables ----------------------
*/

/** Convierte una cadena en el número que representa */

constexpr unsigned int str2int(const char *str, int h = 0)

{

  return !str[h] ? 5381 : (str2int(str, h + 1) * 33) ^ str[h];
```

```
}

/** ----------------- Peticiones a los servidores ------------------
*/

/** ------- Server LUZ ------------------------------------- */

namespace serverLuz

{

  /** Envía al servidor LUZ una petición HTTP con los
   *  datos facilitados
   *  */
  void enviar(uint8_t i = intensidad, uint8_t r = red,
              uint8_t g = green, uint8_t b = blue)
  {
    connectToServer(ssidLuz, password);

    String direccionIP = "http://192.168.4.1/LUZ";
    uint8_t parametro[4] = {i, r, g, b};
    uint8_t cont = 0;

    //Crea una dirección IP de la siguiente forma:
    //http://192.168.4.1/LUZIIIRRRGGGBBB
    for (cont = 0; cont < 4; cont++)
    {

      if (parametro[cont] < 10)
      {
        direccionIP += "00";
```

```
        direccionIP += parametro[cont];
      }
      else if (parametro[cont] < 100)
      {
        direccionIP += "0";
        direccionIP += parametro[cont];
      }
      else
      {
        direccionIP += parametro[cont];
      }
    }

    //Pasa la IP al tipo const char para poder enviar la request
    const char *charIP = direccionIP.c_str();
    sendRequest(charIP);
    Serial.println("Request enviada a " + direccionIP);

    //Se desconecta del servidor para ahorrar recursos
    disconnectFromServer();
  }

} // namespace serverLuz

/** ------- Server PERSIANA ---------------------------------- */
namespace serverPer
{
```

```
/** Envía al servidor PERSIANA una petición HTTP con los
 *  datos facilitados
 *  */
void enviar(String i, bool h)
{

  connectToServer(ssidPersiana, password);

  String direccionIP = "http://192.168.4.1/PER";

  //Crea una dirección IP de la siguiente forma:
  //http://192.168.4.1/PERIIIIH
  direccionIP += i;
  if(h){
    direccionIP += "1";
  }
  else
  {
    direccionIP += "x";
  }

  //Pasa la IP al tipo const char para poder enviar la request
  const char *charIP = direccionIP.c_str();
  sendRequest(charIP);
  Serial.println("Request enviada a " + direccionIP);

  //Se desconecta del servidor para ahorrar recursos
  disconnectFromServer();
```

```
  }

} // namespace serverPer

/** ------- Server ENCHUFE ----------------------------------- */
namespace serverEnc
{

  /** Envía al servidor ENCHUFE una petición HTTP con los
   *  datos facilitados
   *  */
  void enviar(uint8_t n, bool e)
  {
    Serial.println("He recibido: enchufe, estado"); //borrar
    Serial.print(n); //borrar
    Serial.println(e); //borrar



    connectToServer(ssidEnchufe, password);

    //Crea una dirección IP de la siguiente forma:
    //http://192.168.4.1/NE
    String direccionIP = "http://192.168.4.1/ENC";
    direccionIP += n;

    if (e)
    {
      direccionIP += "1";
```

```
    }
    else
    {
      direccionIP += "0";
    }

    //Pasa la IP al tipo const char para poder enviar la request
    const char *charIP = direccionIP.c_str();
    sendRequest(charIP);
    Serial.println("Request enviada a " + direccionIP);

    //Se desconecta del servidor para ahorrar recursos
    disconnectFromServer();
  }
} // namespace serverEnc

/** ------- Server TEMPERATURA ------------------------------- */
namespace serverTem
{

  /** Envía al servidor TEMPERATURA una petición HTTP con los
   *  datos facilitados
   *  */
  void enviar(String p)
  {
    connectToServer(ssidTemperatura, password);

    //Crea una dirección IP de la siguiente forma:
```

```
    //http://192.168.4.1/xxx
    String direccionIP = "http://192.168.4.1/" + p;

    //Pasa la IP al tipo const char para poder enviar la request
    const char *charIP = direccionIP.c_str();
    sendRequest(charIP);
    Serial.println("Request enviada a " + direccionIP);

    //Se desconecta del servidor para ahorrar recursos
    disconnectFromServer();
  }
} // namespace serverTem

/** -------------------- Controles automáticos ---------------------
*/

/** Escribe en "voltaje" la tensión recibida en el pin del LDR */
void leerVoltaje (){
  voltaje = float(analogRead(LDR)) * 3.3 / 512.0;
  Serial.print("Voltaje LDR (V)= ");
  Serial.println(voltaje);
}

namespace control
{
  /** Ejecuta el control automático de la persiana */
  void persiana()
  {
    leerVoltaje();
```

```
  if(voltaje < voltajeMinimoPersiana)
  {
    serverPer::enviar("baja", 1);
  }
}

/** Ejecuta el control automático de la intensidad de la luz
 *  aumentándola y disminuyéndola en saltos del 5%
 * */
void luz()
{
  leerVoltaje();
  if(voltaje < voltajeMinimoLuz && intensidad <= 95)
  {
    serverLuz::enviar(intensidad + 5, red, green, blue);
  }
  else if(voltaje > voltajeMaximoLuz && intensidad >= 5)
  {
    serverLuz::enviar(intensidad - 5, red, green, blue);
  }

}

/** Ejecuta el control automático del color de la luz,
 *  variando su tonalidad de cálida a fría conforme avanza el día
 * */
void color()
{
```

```
    //Control automático de color no disponible en esta versión
  }


}// namespace control


//-------------------------------------------------------------------------
-
//Fin de Xana_ClientLib.h
```

Anexo 20

-

Xana_ServerLuzLib.h

```
/**

*****************************************************************
  * @file    Xana_ServerLuzLib.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    23/07/2020
  * @brief   Incluye todos los comandos necesarios para controlar
  *          tiras o bombillas LED RGB
  * */

/** ---------------------- Includes y defines ----------------------
*/

#include <Arduino.h>
#include "IRremote.h"

#define ledR 32
#define ledG 33
#define ledB 25
```

```
//El ESP32 tiene 16 canales PWM a los que asignar cualquier GPIO
#define pwmRedChannel 0
#define pwmGreenChannel 1
#define pwmBlueChannel 2

#define ledFreq 12000
#define PWMBitResolution 8

#define onCode    0xF7C03F
#define offCode   0xF7408F
#define masCode   0xF700FF
#define menosCode 0xF7807F
#define redCode   0xF720DF
#define greenCode 0xF7A05F
#define blueCode  0xF7609F
#define coldCode  0xF7E01F
#define warmCode  0xF728D7
#define fadeCode  0xF7C837
IRsend emisor;

/** --------------------- Variables auxiliares ---------------------
*/

bool multicolor = false;

//Se asignan 16 bits para poder multiplicar R, G y B por la intensidad
uint16_t intensidad, red, green, blue;
uint16_t intensidad_anterior = intensidad;
```

```
uint16_t red_anterior = red;
uint16_t green_anterior = green;
uint16_t blue_anterior = blue;

String auxStr;

uint32_t marcaTemp = 0;

/** ------------------------ Inicialización ------------------------
*/

/** Asigna a las GPIO ledR, ledG y ledB los canales PWM pwmRedChannel,
 *  pwmGreenChannel y pwmBlueChannel con una frecuencia de ledFreq kHz
 *  y una resolución del ciclo de trabajo de PWMBitResolution bits
 *  */
void initLedsPWM()
{

    ledcSetup(pwmRedChannel, ledFreq, PWMBitResolution);
    ledcSetup(pwmGreenChannel, ledFreq, PWMBitResolution);
    ledcSetup(pwmBlueChannel, ledFreq, PWMBitResolution);

    ledcAttachPin(ledR, pwmRedChannel);
    ledcAttachPin(ledG, pwmGreenChannel);
    ledcAttachPin(ledB, pwmBlueChannel);
}

/** Muestra los colores blanco, rojo, verde y azul */
void demoInicial()
```

```
{

    ledcWrite(pwmRedChannel,   0  );
    ledcWrite(pwmGreenChannel, 0  );
    ledcWrite(pwmBlueChannel,  0  );
    delay(400);

    ledcWrite(pwmRedChannel,   255);
    ledcWrite(pwmGreenChannel, 255);
    ledcWrite(pwmBlueChannel,  255);
    delay(200);

    ledcWrite(pwmRedChannel,   0  );
    ledcWrite(pwmGreenChannel, 255);
    ledcWrite(pwmBlueChannel,  255);
    delay(400);

    ledcWrite(pwmRedChannel,   255);
    ledcWrite(pwmGreenChannel, 0  );
    ledcWrite(pwmBlueChannel,  255);
    delay(400);

    ledcWrite(pwmRedChannel,   255);
    ledcWrite(pwmGreenChannel, 255);
    ledcWrite(pwmBlueChannel,  0  );
    delay(400);

    ledcWrite(pwmRedChannel,   255);
```

```
    ledcWrite(pwmGreenChannel, 255);
    ledcWrite(pwmBlueChannel,  255);
}

/** -------------------------- Actuación ---------------------------
*/

/** Envía un código IR a la bombilla LED en función del cambio
 *  producido en las variables de control
 *  */
void escribirIR ()
{
    //si la intensidad es 0, apaga.
    if(intensidad == 0)
    {
        emisor.sendNEC(offCode, 32);
        multicolor = false;
    }

    //si el color es igual, lo que cambia es la intensidad
    else if(red == red_anterior && blue == blue_anterior
            && green == green_anterior)
    {
        if(intensidad > intensidad_anterior)
        {
            //si la intensidad pasa de 0 a algo, se enciende
            if(intensidad_anterior == 0)
            {
                emisor.sendNEC(onCode, 32);
```

```
        }
        //si no, solo sube el brillo
        else
        {
            emisor.sendNEC(masCode, 32);
        }
    }
    else if(intensidad < intensidad_anterior)
    {
        emisor.sendNEC(menosCode, 32);
    }

}

//Si cambia el color, se configura el color que toca
else if(red == 255 && blue == 0 && green == 0)
{
    emisor.sendNEC(redCode, 32);
    multicolor = false;
}
else if(red == 0 && blue == 255 && green == 0)
{
    emisor.sendNEC(blueCode, 32);
    multicolor = false;
}
else if(red == 0 && blue == 0 && green == 255)
{
    emisor.sendNEC(greenCode, 32);
```

```
        multicolor = false;
    }
    else if(red == 201 && blue == 255 && green == 206)
    {
        emisor.sendNEC(coldCode, 32);
        multicolor = false;
    }
    else if(red == 255 && blue == 143 && green == 197)
    {
        emisor.sendNEC(warmCode, 32);
        multicolor = false;
    }
}

/** Descompone la orden recibida por el cliente en la intensidad y
 *  colores solicitados y los escribe tanto por el canar +12V,R,G,B
 *  como por el canal IR
 *  */
void escribirLeds(String request)
{
    String url = request.substring(request.indexOf("/") + 1,
                 request.indexOf("HTTP") - 1);

    auxStr = url.substring(3, 6);
    intensidad = auxStr.toInt();
    Serial.print("-- Intensidad: ");
    Serial.println(intensidad);
```

```
// Si se recibe una intensidad del 137%, en lugar de actuar como
// en otros casos, se activará el modo multicolor del mando IR
if(intensidad == 137)
{
    emisor.sendNEC(fadeCode, 32);
    multicolor = true;
}
else
{
    //Guarda las variables para compararlas en la finción IR
    intensidad_anterior = intensidad;
    red_anterior = red;
    green_anterior = green;
    blue_anterior = blue;

    //Lee las nuevas variables
    auxStr = url.substring(6, 9);
    red = auxStr.toInt();
    Serial.print("-- R: ");
    Serial.println(red);
    //Multiplicamos R por el factor de intensidad solicitado
    red = red * intensidad / 100;
    //Invertimos el color para ofrecer una salida a ánodo común.
    red = 255 - red;

    auxStr = url.substring(9, 12);
    green = auxStr.toInt();
    Serial.print("-- G: ");
```

```
        Serial.println(green);
        //Multiplicamos R por el factor de intensidad solicitado
        green = green * intensidad / 100;
        //Invertimos el color para ofrecer una salida a ánodo común.
        green = 255 - green;

        auxStr = url.substring(12);
        blue = auxStr.toInt();
        Serial.print("-- B: ");
        Serial.println(blue);
        //Multiplicamos R por el factor de intensidad solicitado
        blue = blue * intensidad / 100;
        //Invertimos el color para ofrecer una salida a ánodo común.
        blue = 255 - blue;

        //Escribe la luz en la salida +12V,R,G,B (para tiras LED)
        ledcWrite(pwmRedChannel,   red);
        ledcWrite(pwmGreenChannel, green);
        ledcWrite(pwmBlueChannel,  blue);

        //Escribe la luz en la salida IR (para bombillas)
        escribirIR();
    }
}

/* ------------------------ Temporización ------------------------
*/
```

```
/** Guarda una marca de tiempo para comprobar el tiempo transcurrido
 * con "tiempoTranscurrido"
 * */
void guardarTiempo()
{
    marcaTemp = millis();
}

/** Devuelve true si ha pasado  "tiempo_ms" desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
bool comprobarTiempo(int tiempo_ms)
{
    if (abs(millis() - marcaTemp) >= tiempo_ms)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/** Devielve el tiempo (en ms) transcurrido desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
unsigned long tiempoDesdeMarca()
{
```

```
    return (millis() - marcaTemp);
}


//-------------------------------------------------------------------
-
//Fin de Xana_ServerLuzLib.h
```

Anexo 21

-

Xana_ServerEnchufeLib.h

```
/**

*****************************************************************
  * @file    Xana_ServerEnchufeLib.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    25/07/2020
  * @brief   Incluye todos los comandos necesarios para controlar
  *          cuatro relés
  * */

/** ---------------------- Includes y defines ----------------------
*/

#include <Arduino.h>

#define boton1 16
#define boton2 17
#define boton3 18
#define boton4 19
```

```
#define rele1 26
#define rele2 25
#define rele3 33
#define rele4 32

/** --------------------- Variables auxiliares ---------------------
*/

bool estado1 = false;
bool estado2 = false;
bool estado3 = false;
bool estado4 = false;

uint32_t marcaTemp = 0;

uint8_t rele, estado;
String auxStr;

/** ------------------------- Actuación ---------------------------
*/

/** Conmuta el relé num */
void encenderEnchufe (uint8_t num)
{
    if(num == 1)
    {
        Serial.println("Encendido enchufe 1");
        digitalWrite(rele1, HIGH);
```

```
        estado1 = true;
    }
    else if(num == 2)
    {
        Serial.println("Encendido enchufe 2");
        digitalWrite(rele2, HIGH);
        estado2 = true;
    }
    else if(num == 3)
    {
        Serial.println("Encendido enchufe 3");
        digitalWrite(rele3, HIGH);
        estado3 = true;
    }
    else if(num == 4)
    {
        Serial.println("Encendido enchufe 4");
        digitalWrite(rele4, HIGH);
        estado4 = true;
    }
}

/** Deja de conmutar el reé num */
void apagarEnchufe (uint8_t num)
{
    if(num == 1)
    {
        Serial.println("Apagado enchufe 1");
```

```
        digitalWrite(rele1, LOW);
        estado1 = false;
    }
    else if(num == 2)
    {
        Serial.println("Apagado enchufe 2");
        digitalWrite(rele2, LOW);
        estado2 = false;
    }
    else if(num == 3)
    {
        Serial.println("Apagado enchufe 3");
        digitalWrite(rele3, LOW);
        estado3 = false;
    }
    else if(num == 4)
    {
        Serial.println("Apagado enchufe 4");
        digitalWrite(rele4, LOW);
        estado4 = false;
    }
}

/** Descompone la orden recibida por el cliente en la el número de
 *  relé a controlar y su estado objetivo y actúa en consecuencia
 *  */
void escribirReles(String request)
{
```

```
String url = request.substring(request.indexOf("/") + 1,
             request.indexOf("HTTP") - 1);

auxStr = url.substring(3, 4);
rele = auxStr.toInt();
Serial.print("-- Enchufe: ");
Serial.println(rele);

auxStr = url.substring(4, 5);
estado = auxStr.toInt();
Serial.print("-- Estado: ");

switch (rele)
{
case 1:
    if (estado)
    {
        Serial.println("Encendido");
        encenderEnchufe(1);
    }
    else
    {
        Serial.println("Apagado");
        apagarEnchufe(1);
    }
    break;

case 2:
```

```
    if (estado)
    {
        Serial.println("Encendido");
        encenderEnchufe(2);
    }
    else
    {
        Serial.println("Apagado");
        apagarEnchufe(2);
    }
    break;

case 3:
    if (estado)
    {
        Serial.println("Encendido");
        encenderEnchufe(3);
    }
    else
    {
        Serial.println("Apagado");
        apagarEnchufe(3);
    }
    break;

case 4:
    if (estado)
    {
```

```
            Serial.println("Encendido");
            encenderEnchufe(4);
        }
        else
        {
            Serial.println("Apagado");
            apagarEnchufe(4);
        }
        break;

    default:
        Serial.println("Enchufe no existente");
        break;
    }
}

/* ------------------------- Temporización -------------------------
*/



/** Guarda una marca de tiempo para comprobar el tiempo transcurrido
 * con "tiempoTranscurrido"
 * */
void guardarTiempo()
{
    marcaTemp = millis();
}

/** Devuelve true si ha pasado  "tiempo_ms" desde la última marca de
```

```
 * tiempo guardada con "guardarTiempo"
 * */
bool comprobarTiempo(int tiempo_ms)
{
    if (abs(millis() - marcaTemp) >= tiempo_ms)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/** Devielve el tiempo (en ms) transcurrido desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
unsigned long tiempoDesdeMarca()
{
    return (millis() - marcaTemp);
}

//-----------------------------------------------------------------------
-
//Fin de Xana_ServerEnchufeLib.h
```

Anexo 22

-

Xana_ServerPersianaLib.h

```
/**

*****************************************************************
  * @file    Xana_ServerPersianaLib.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    27/07/2020
  * @brief   Incluye todos los comandos necesarios para controlar
  *          el motor de fase partida de una persiana eléctrica
  * */

/** ---------------------- Includes y defines ----------------------
*/

#include <Arduino.h>
#include <EEPROM.h>

#define pulsadorSet 25
#define pulsadorUp 32
#define pulsadorDown 33
```

```
#define releMover 34
#define releSwitch 35

/** --------------------- Variables auxiliares ---------------------
*/

// La altura es el tiempo (ms) que tarda la persiana en moverse de un
// extremo al otro
uint32_t altura = 0;

// La posicion es el tiempo (ms) que tardaría la persiana en alcanzar
// su tope inferior
int32_t posicion = 0;

// Será true si fue el de subida y false si fue el de bajada
bool ultimoPulsado;
bool moviendo = false;

uint32_t marcaTemp = 0;

String auxStr;

/* ------------------------- Temporización -------------------------
*/



/** Guarda una marca de tiempo para comprobar el tiempo transcurrido
 * con "tiempoTranscurrido"
 * */
```

```
void guardarTiempo()
{
    marcaTemp = millis();
}

/** Devuelve true si ha pasado  "tiempo_ms" desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
bool comprobarTiempo(int tiempo_ms)
{
    if (abs(millis() - marcaTemp) >= tiempo_ms)
    {
        return true;
    }
    else
    {
        return false;
    }
}

/** Devielve el tiempo (en ms) transcurrido desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
unsigned long tiempoDesdeMarca()
{
    return (millis() - marcaTemp);
}
```

```
/** ------------------------- Actuación ---------------------------
*/


/** Lee la altura total y posición actual de la persiana desde la
 *  memoria de programa (EEPROM)
 *  */
void cargarDatosDesdeMemoria() {


     // Accederemos a 8 bytes de de la memoria Flash: 4 bytes (32
bits)
     // por cada dato
    EEPROM.begin(8);


    //Lee las direcciones (bytes) 0, 1, 2 y 3 de la memoria Flash
    altura = EEPROM.readLong(0);
    //Lee las direcciones (bytes) 4, 5, 6 y 7 de la memoria Flash
    posicion = EEPROM.readLong(4);


    Serial.print("Altura total cargada: ");
    Serial.println(altura + " ms");
    Serial.print("Posición cargada: ");
    Serial.println(posicion + " ms");


    //Cerramos la comunicación con la EEPROM para ahorrar recursos
    EEPROM.end();
}


/** Guarda la altura total y posición actual de la persiana en la
 *  memoria de programa (EEPROM)
```

```
 *  */
void EscribirDatosEnMemoria(){

     // Accederemos a 8 bytes de de la memoria Flash: 4 bytes (32
bits)
     // por cada dato
    EEPROM.begin(8);

    //Escribe en las direcciones (bytes) 0 a la 3 de la memoria Flash
    EEPROM.writeLong(0, altura);
    //Escribe en las direcciones (bytes) 4 a la 7 de la memoria Flash
    EEPROM.writeLong(4, posicion);

    //Hacemos efectivos los cambios ejecutados
    EEPROM.commit();

    Serial.println("Altura guardada en la memoria del programa.");

    //Cerramos la comunicación con la EEPROM para ahorrar recursos
    EEPROM.end();
}

/** Escribe "0" en el relé de alimentación del motor */
void parar()
{
    digitalWrite(releMover, LOW);

    if(ultimoPulsado){
        posicion += tiempoDesdeMarca();
```

```
        if(posicion > altura)
        {
            posicion = altura;
        }
    }
    else
    {
        posicion -= tiempoDesdeMarca();
        if(posicion < altura)
        {
            posicion = 0;
        }
    }
    EscribirDatosEnMemoria();

}

/** Escribe "0" en el relé de arranque y "1" en el de la
 *  alimentación del motor
 *  */
void subir()
{
    guardarTiempo();
    digitalWrite(releSwitch, LOW);
    digitalWrite(releMover, HIGH);

    ultimoPulsado = true;
}
```

```
/** Escribe "1" en los relés de arranque y alimentación del motor */
void bajar()
{
    guardarTiempo();
    digitalWrite(releSwitch, HIGH);
    digitalWrite(releMover, HIGH);
    ultimoPulsado = false;
}

/** Sube la persiana hasta su tope o solo un 25% */
void subirHasta(bool tope = false)
{
    guardarTiempo();
    subir();

    uint32_t diferencia;

    if(tope)
    {
        diferencia = altura - posicion; //Sube hasta arriba
        while(!comprobarTiempo(diferencia))
        {
            //espera subir la diferencia
        }
    }
    else
    {
```

```
        diferencia = altura/4; //Sube un  25% de la altura máxima
        while(!comprobarTiempo(diferencia))
        {
            //espera subir la diferencia
        }
    }

    parar();

}

/** Baja la persiana hasta su tope o solo un 25% */
void bajarHasta(bool tope = false)
{
    guardarTiempo();
    subir();

    uint32_t diferencia;

    if(tope)
    {
        diferencia = posicion; //Baja hasta arriba
        while(!comprobarTiempo(diferencia))
        {
            //espera bajar la diferencia
        }
    }
    else
```

```
    {
        diferencia = altura/4; //Baja un  25% de la altura máxima
        while(!comprobarTiempo(diferencia))
        {
            //espera bajar la diferencia
        }
    }

    parar();

}

//-------------------------------------------------------------------
-
//Fin de Xana_ServerLuzPersiana.h
```

Anexo 23

-

Xana_ServerTemperaturaLib.h

```
/**

*****************************************************************
  * @file    Xana_ServerTemperaturaLib.h
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    29/07/2020
  * @brief   Incluye todos los comandos necesarios para controlar
  *          el motor de fase partida de una persiana eléctrica
  * */


/** ---------------------- Includes y defines ----------------------
*/


#include <Arduino.h>
#include <EEPROM.h>
#include "IRremote.h"


// No se define ledEmisor como 26 ya que esto se hace desde la
librería
// IRremoteBoardDefs.h
```

```
#define ledReceptor 25
#define pulsador 33
#define buzzer 32

#define la3 440
#define mi4 660

//ESP32 tiene 16 canales PWM a los que asignar cualquier GPIO
#define buzzerChannel 0
#define buzzerFreq 2000
#define PWMBitResolution 8

/** ---------------------- Variables auxiliares ----------------------
*/

//Define las clases del emisor y receptor IR
IRrecv receptor(ledReceptor);
decode_results recibido;
IRsend emisor;

uint32_t codigoOn, codigoMas, codigoMenos, codigoSwing;

unsigned long marcaTemp = 0;

String auxStr;

/** ------------------------ Inicialización ------------------------
*/
```

```
/** Inicializa el buzzer */
void initBuzzer()
{
    ledcSetup(buzzerChannel, buzzerFreq, PWMBitResolution);
    ledcAttachPin(buzzer, buzzerChannel);
}

/** Inicializa el receptor de IR */
void initReciver()
{
    receptor.enableIRIn();
}

/** ------------------------ Configuración ------------------------
*/

/** Lee codigoOn, codigoMas, codigoMenos y codigoSwing de la memoria
 *  EEPROM
 *  */
void LeerCodigosDesdeMemoria() {

    // Accederemos a 16 bytes de de la memoria Flash: 4 bytes (32
bits)
    // por cada código
    EEPROM.begin(16);

    //Lee las direcciones (bytes) 0, 1, 2 y 3 de la memoria Flash
    codigoOn   =  EEPROM.readLong(0);
    //Lee las direcciones (bytes) 4, 5, 6 y 7 de la memoria Flash
```

```
    codigoMas  =  EEPROM.readLong(4);
    //Lee las direcciones (bytes) 8, 9, 10 y 11 de la memoria Flash
    codigoMenos  =  EEPROM.readLong(8);
    //Lee las direcciones (bytes) 12, 13, 14 y 15 de la memoria Flash
    codigoSwing = EEPROM.readLong(12);

    Serial.println("Códigos cargados:");
    Serial.println("-- ON: " + codigoOn);
    Serial.println("-- MÁS: " + codigoMas);
    Serial.println("-- MENOS: " + codigoMenos);
    Serial.println("-- SWING: " + codigoSwing);

    //Cerramos la comunicación con la EEPROM para ahorrar recursos
    EEPROM.end();
}

/** Escribe codigoOn, codigoMas, codigoMenos y codigoSwing en la
 *  memoria EEPROM
 *  */
void EscribirCodigosEnMemoria(){

    // Accederemos a 16 bytes de de la memoria Flash: 4 bytes (32
bits)
    // por cada código
    EEPROM.begin(16);

    //Escribe en las direcciones (bytes) 0 a la 3 de la memoria Flash
    EEPROM.writeLong(0, codigoOn);
    //Escribe en las direcciones (bytes) 4 a la 7 de la memoria Flash
```

```
    EEPROM.writeLong(4, codigoMas);
    //Escribe en las direcciones (bytes) 8 a la 11 de la memoria Flash
    EEPROM.writeLong(8, codigoMenos);
    //Escribe en las direcciones (bytes) 12 a la 15 de la memoria
Flash
    EEPROM.writeLong(12, codigoSwing);

    //Hacemos efectivos los cambios ejecutados
    EEPROM.commit(); //Hacemos efectivos los cambios guardados

    Serial.println("Códigos Guardados en la memoria del programa.");

    //Cerramos la comunicación con la EEPROM para ahorrar recursos
    EEPROM.end();
}

/** Envía la señal IR de encender/apagar el sistema de temperatura */
void escribirOn()
{
    emisor.sendNEC(codigoOn, 32);
}

/** Envía la señal IR de subir la temperatura */
void escribirMas()
{
    emisor.sendNEC(codigoMas, 32);
}

/** Envía la señal IR de bajar la temperatura */
```

```
void escribirMenos()
{
    emisor.sendNEC(codigoMenos, 32);
}

/** Envía la señal IR de activar/desactivar el mpdp swing */
void escribirSwing()
{
    emisor.sendNEC(codigoSwing, 32);
}

/* ------------------------- Temporización -------------------------
*/


/** Guarda una marca de tiempo para comprobar el tiempo transcurrido
 * con "tiempoTranscurrido"
 * */
void guardarTiempo()
{
    marcaTemp = millis();
}

/** Devuelve true si ha pasado  "tiempo_ms" desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
bool comprobarTiempo(int tiempo_ms)
{
    if (abs(millis() - marcaTemp) >= tiempo_ms)
```

```
    {
        return true;
    }
    else
    {
        return false;
    }
}

/** Devielve el tiempo (en ms) transcurrido desde la última marca de
 * tiempo guardada con "guardarTiempo"
 * */
unsigned long tiempoDesdeMarca()
{
    return (millis() - marcaTemp);
}

//-----------------------------------------------------------------------
-
//Fin de Xana_ServerTemperaturaLib.h
```

Anexo 24

-

Xana_ServerLuz.cpp

```
/**

*****************************************************************
  * @file    Xana_ServerLuz.cpp
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    23/07/2020
  * @brief   Recibe las peticiones HTTP de los clientes para controlar
  *          tiras o bombillas LED RGB
  * */

/** --------------------------- Includes ---------------------------
*/

#include <Arduino.h>
#include "Xana_ServerLuzLib.h"
#include "XanaWifi.h"

/** ---------------- Credenciales de la red a crear ----------------
*/
```

```
const char *ssid = "Xana_Luz_server";
const char *password = "123456789";

/** ---------------------------- Setup ----------------------------
*/

void setup()
{
  Serial.begin(9600);

  //Configura los LEDs
  initLedsPWM();

  //Programa de inicio
  demoInicial();

  //Crea el punto de acceso WiFi
  createServer(ssid, password);
}

/** ---------------------------- Loop ----------------------------
*/

void loop()
{
  // Busca clientes conectados
  WiFiClient client = server.available();

  if (client)
```

```
  {
    Serial.println("Nuevo cliente conectado.");
    // Crea una cadena para la request del cliente
    String currentLine = "";

    while (client.connected())
    {
      // Si el cliente envía un mensaje, entrará en el reconocimiento
      if (client.available())
      {

        char c = client.read();
        Serial.write(c);
        header += c;

        if (c == '\n')
        {
          // Si recibe una línea en blanco, el cliente ha terminado
          //la petición
          if (currentLine.length() == 0)
          {
            // Los HTTP headers empiezan con un código de respuesta
            // (por ejemplo, HTTP/1.1 200 OK), una línea de contenido
y
            // una línea en blanco. La variable header tendrá un
            // formato "GET /xxxx HTTP/1.1". donde /xxxx corresponde
            // con la orden del cliente (por ejemplo, "/led19/on").

            //Respuesta al cliente
```

```
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println("Connection: close");
    client.println();

    /** -- Actuación en función de la request recibida -- */

    //Si después de "192.168.4.1" aparece "/LUZ"...
    if (header.indexOf("GET /LUZ") >= 0)
    {
      escribirLeds(header);
    }

    /** -- Fin de la respuesta al cliente ------------- */

    // La respuesta al cliente acaba con una línea en blanco
    client.println();
    break;
  }
  else
  {
    currentLine = "";
  }
}
else if (c != '\r')
{
  currentLine += c;
}
```

```
      }
    }

    /** ------- Desconexión del cliente ---------------------- */

    // Borra la variable de request cuando el cliente se desconecta
    header = "";

    // Libera la comunicación para acoger al siguiente cliente
    client.stop();
    Serial.println("Cliente desconectado.");
    Serial.println("");
  }

    /** ------- Función de multicolor ------------------------ */

  if(multicolor)
  {
    //Código para crear un efecto multicolor en la salida +12V, R, G,
B
    //no dispobnible en esta versión
  }
}

//---------------------------------------------------------------------
-
//Fin de Xana_ServerLuz.cpp
```

Anexo 25

-

Xana_ServerEnchufe.cpp

```
/**

*****************************************************************
  * @file    Xana_ServerEnchufe.cpp
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    25/07/2020
  * @brief   Recibe las peticiones HTTP de los clientes para controlar
  *          cuatro enchufes
  * */


/** --------------------------- Includes ---------------------------
*/


#include <Arduino.h>
#include "Xana_ServerEnchufeLib.h"
#include "XanaWifi.h"


/** ---------------- Credenciales de la red a crear ----------------
*/
```

```
const char *ssid = "Xana_Enchufes_server";
const char *password = "123456789";

#define t_cooldown 600
bool cooldown = false;

/** ---------------------------- Setup ----------------------------
*/

void setup()
{
  Serial.begin(9600);

  //Configura las salidas a los relés
  pinMode(rele1, OUTPUT);
  pinMode(rele2, OUTPUT);
  pinMode(rele3, OUTPUT);
  pinMode(rele4, OUTPUT);
  apagarEnchufe(1);
  apagarEnchufe(2);
  apagarEnchufe(3);
  apagarEnchufe(4);

  //Crea el punto de acceso WiFi
  createServer(ssid, password);
}
```

```
/** ----------------------------- Loop ------------------------------
*/


void loop()
{
  // Busca clientes conectados
  WiFiClient client = server.available();


  if (client)
  {
    Serial.println("Nuevo cliente conectado.");
    // Crea una cadena para la request del cliente
    String currentLine = "";


    while (client.connected())
    {
      // Si el cliente envía un mensaje, entrará en el reconocimiento
      if (client.available())
      {


        char c = client.read();
        Serial.write(c);
        header += c;


        if (c == '\n')
        {
          // Si recibe una línea en blanco, el cliente ha terminado
          //la petición
          if (currentLine.length() == 0)
```

```
            {
              // Los HTTP headers empiezan con un código de respuesta
              // (por ejemplo, HTTP/1.1 200 OK), una línea de contenido
y
              // una línea en blanco. La variable header tendrá un
              // formato "GET /xxxx HTTP/1.1". donde /xxxx corresponde
              // con la orden del cliente (por ejemplo, "/led19/on").

              //Respuesta al cliente
              client.println("HTTP/1.1 200 OK");
              client.println("Content-type:text/html");
              client.println("Connection: close");
              client.println();

              /** -- Actuación en función de la request recibida -- */

              //Si después de "192.168.4.1" aparece "/ENC"...
              if (header.indexOf("GET /ENC") >= 0)
              {
                escribirReles(header);
              }

              /** -- Fin de la respuesta al cliente ------------- */

              // La respuesta al cliente acaba con una línea en blanco
              client.println();
              break;
            }
            else
```

```
        {
          currentLine = "";
        }
      }
      else if (c != '\r')
      {
        currentLine += c;
      }
    }
  }

  /** ------- Desconexión del cliente ----------------------- */

  // Borra la variable de request cuando el cliente se desconecta
  header = "";

  // Libera la comunicación para acoger al siguiente cliente
  client.stop();
  Serial.println("Cliente desconectado.");
  Serial.println("");
}

  /** ------- Entradas por pulsador ------------------------- */

// cambia el estado del enchufe si se pulsa el botón
if(digitalRead(boton1) && !cooldown)
{
  delay(80);
```

```
  cooldown = true;
  guardarTiempo();

  if(estado1)
  {
    apagarEnchufe(1);
  }
  else
  {
    encenderEnchufe(1);
  }
}

if(digitalRead(boton2) && !cooldown)
{
  delay(80);
  cooldown = true;
  guardarTiempo();

  if(estado2)
  {
    apagarEnchufe(2);
  }
  else
  {
    encenderEnchufe(2);
  }
}
```

```
if(digitalRead(boton3) && !cooldown)
{
  delay(80);
  cooldown = true;
  guardarTiempo();

  if(estado3)
  {
    apagarEnchufe(3);
  }
  else
  {
    encenderEnchufe(3);
  }
}

if(digitalRead(boton4) && !cooldown)
{
  delay(80);
  cooldown = true;
  guardarTiempo();

  if(estado4)
  {
    apagarEnchufe(4);
  }
  else
```

```
    {
      encenderEnchufe(4);
    }
  }

  //Evita lecturas repetidas si no se suelta el botón
  if(cooldown)
  {
    if(comprobarTiempo(t_cooldown))
    {
      cooldown = false;
    }
  }
}

//---------------------------------------------------------------------
-
//Fin de Xana_ServerEnchufe.cpp
```

Anexo 26

-

Xana_ServerPersiana.cpp

```
/**

*****************************************************************
  * @file    Xana_ServerPersiana.cpp
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    27/07/2020
  * @brief   Recibe las peticiones HTTP de los clientes para controlar
  *          un motor de persiana de corriente alterna
  * */


/** --------------------------- Includes ---------------------------
*/


#include <Arduino.h>
#include "Xana_ServerPersianaLib.h"
#include "XanaWifi.h"


/** --------------------- Variables auxiliares ---------------------
*/
```

```
bool contando = false;
bool pulsando = false;

/** ---------------- Credenciales de la red a crear ----------------
*/

const char *ssid = "Xana_Pers_server";
const char *password = "123456789";



/** ---------------------------- Setup -----------------------------
*/

void setup()
{
  Serial.begin(9600);

  //Configura las GPIO
  pinMode(pulsadorUp, INPUT);
  pinMode(pulsadorDown, INPUT);
  pinMode(pulsadorSet, INPUT);
  pinMode(releMover, OUTPUT);
  pinMode(releSwitch, OUTPUT);



  cargarDatosDesdeMemoria();

  //Crea el punto de acceso WiFi
  createServer(ssid, password);
```

```
}

/** ---------------------------- Loop ----------------------------
*/

void loop()
{
  // Busca clientes conectados
  WiFiClient client = server.available();

  if (client)
  {
    Serial.println("Nuevo cliente conectado.");
    // Crea una cadena para la request del cliente
    String currentLine = "";

    while (client.connected())
    {
      // Si el cliente envía un mensaje, entrará en el reconocimiento
      if (client.available())
      {

        char c = client.read();
        Serial.write(c);
        header += c;

        if (c == '\n')
        {
          // Si recibe una línea en blanco, el cliente ha terminado
```

```
            //la petición
            if (currentLine.length() == 0)
            {
              // Los HTTP headers empiezan con un código de respuesta
              // (por ejemplo, HTTP/1.1 200 OK), una línea de contenido
y
              // una línea en blanco. La variable header tendrá un
              // formato "GET /xxxx HTTP/1.1". donde /xxxx corresponde
              // con la orden del cliente (por ejemplo, "/led19/on").

              //Respuesta al cliente
              client.println("HTTP/1.1 200 OK");
              client.println("Content-type:text/html");
              client.println("Connection: close");
              client.println();

              /** -- Actuación en función de la request recibida -- */

              //Si después de "192.168.4.1" aparece "/PERsube1"
              if (header.indexOf("GET /PERsube1") >= 0)
              {
                bool tope = true;
                //Sube la persiana hasta el tope
                subirHasta(tope);
              }

              //Si después de "192.168.4.1" aparece "/PERsubex"
              else if (header.indexOf("GET /PERsubex") >= 0)
              {
```

```
    //Sube la persiana un 25%
    subirHasta();
  }

  //Si después de "192.168.4.1" aparece "/PERbaja1"
  else if (header.indexOf("GET /PERbaja1") >= 0)
  {
    bool tope = true;
    //Baja la persiana hasta el tope
    bajarHasta(tope);
  }

  //Si después de "192.168.4.1" aparece "/PERbajax"
  else if (header.indexOf("GET /PERbajax") >= 0)
  {
    //Baja la persiana un 25%
    bajarHasta();
  }

  /** -- Fin de la respuesta al cliente ------------- */

  // La respuesta al cliente acaba con una línea en blanco
  client.println();
  break;
}
else
{
  currentLine = "";
```

```
        }
      }
      else if (c != '\r')
      {
        currentLine += c;
      }
    }
  }

  /** ------- Desconexión del cliente ----------------------- */

  // Borra la variable de request cuando el cliente se desconecta
  header = "";

  // Libera la comunicación para acoger al siguiente cliente
  client.stop();
  Serial.println("Cliente desconectado.");
  Serial.println("");
}

  /** ------- Calibración de la altura ---------------------- */

//Comienza a calibrar la altura de la persiana
if (digitalRead(pulsadorSet))
{
  delay(80);

  guardarTiempo();
```

```
    contando = true;
  }

  //Al soltar el botón, guarda el tiempo que tarda en bajar la
persiana
  if(contando && !digitalRead(pulsadorSet))
  {
    delay(80);

    contando = false;
    altura = tiempoDesdeMarca();

    if(ultimoPulsado)
    {
      posicion = altura;
    }
    else
    {
      posicion = 0;
    }

    EscribirDatosEnMemoria();

  }

  //Sube la persiana si se pulsa el botón
  if (digitalRead(pulsadorUp))
  {
    delay(80);
```

```
    pulsando = true;
    ultimoPulsado = true;
    subir();
  }

  //Baja la persiana si se pulsa el botón
  if (digitalRead(pulsadorDown))
  {
    delay(80);

    pulsando = true;
    ultimoPulsado = false;
    bajar();
  }

  //Para el motor si no se pulsa ningún botón
  if(pulsando
  && !digitalRead(pulsadorUp)
  && !digitalRead(pulsadorDown)
  && !moviendo)
  {
    delay(80);

    parar();
  }

}
```

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

//Fin de Xana_ServerPersiana.cpp
```

Anexo 27

-

Xana_ServerTemperatura.cpp

```
/**
  ******************************************************************
  * @file    Xana_ServerTemperatura.cpp
  * @author  Jorge Álvarez Pedrón
  * @version V1.0
  * @date    29/07/2020
  * @brief   Recibe las peticiones HTTP de los clientes para
  *          controlar un mando de aire acondicionado
  * */
/** ------------------ Includes y defines ------------------- */

#include <Arduino.h>
#include "Xana_ServerTemperaturaLib.h"
#include "XanaWifi.h"
#define esperaMax 7000

/** ------------ Credenciales de la red a crear ------------- */

const char *ssid = "Xana_Temp_server";
const char *password = "123456789";

/** ----------------------- Setup -------------------------- */
void setup(){
  Serial.begin(9600);
```

```
  //Configura las GPIO
  pinMode(ledReceptor, INPUT);
  pinMode(pulsador, INPUT);
  pinMode(buzzer, OUTPUT);
  initBuzzer();
  initReciver();
  LeerCodigosDesdeMemoria();

  //Crea el punto de acceso WiFi
  createServer(ssid, password);
}
/** ------------------------ Loop -------------------------- */

void loop(){
  // Busca clientes conectados
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Nuevo cliente conectado.");
    // Crea una cadena para la request del cliente
    String currentLine = "";

    while (client.connected()){
 // Si el cliente envía un mensaje, entrará en el reconocimiento
      if (client.available()){

        char c = client.read();
        Serial.write(c);
        header += c;

        if (c == '\n'){
     // Si recibe una línea en blanco, el cliente ha terminado
          //la petición
```

```
   if (currentLine.length() == 0){
// Los HTTP headers empiezan con un código de respuesta
// (por ejemplo, HTTP/1.1 200 OK), una línea de contenido
// y una línea en blanco. La variable header tendrá un
// formato "GET /xxxx HTTP/1.1". donde /xxxx corresponde
// con la orden del cliente (por ejemplo, "/led19/on").

//Respuesta al cliente
     client.println("HTTP/1.1 200 OK");
     client.println("Content-type:text/html");
     client.println("Connection: close");
     client.println();

 /** -- Actuación en función de la request recibida -- */

     //Si después de "192.168.4.1" aparece "/onn"...
     if (header.indexOf("GET /onn") >= 0) {
       escribirOn();
     }

     //Si después de "192.168.4.1" aparece "/off"...
     else if (header.indexOf("GET /off") >= 0){
       escribirOn();
     }

     //Si después de "192.168.4.1" aparece "/mas"...
     else if (header.indexOf("GET /mas") >= 0){
       escribirMas();
     }

     //Si después de "192.168.4.1" aparece "/men"...
     else if (header.indexOf("GET /men") >= 0){
       escribirMenos();
     }
```

```
          //Si después de "192.168.4.1" aparece "/swg"...
          else if (header.indexOf("GET /swg") >= 0) {
            escribirSwing();
          }

        /** -- Fin de la respuesta al cliente ------------- */

        // La respuesta al cliente acaba con una línea en blanco
           client.println();
           break;
         }
         else {
           currentLine = "";
         }
        }
        else if (c != '\r'){
         currentLine += c;
        }
      }
    }

    /** ------- Desconexión del cliente ---------------------- */

    // Borra la variable request cuando el cliente se desconecta
    header = "";

    // Libera la comunicación para acoger al siguiente cliente
    client.stop();
    Serial.println("Cliente desconectado.");
    Serial.println("");
}

    /** ------- Configuración del mando ---------------------- */
```

```
if (digitalRead(pulsador)){
  delay(80);

  bool timeout = false;
  bool errorReproducido = false;

  Serial.println("\n----  CONFIGURACIÓN DE COMANDOS  ----\n");

  //Sonido ascendente
  ledcWriteTone(buzzerChannel, la3);
  delay(150);
  ledcWriteTone(buzzerChannel, mi4);
  delay(250);
  ledcWriteTone(buzzerChannel, 0);

  guardarTiempo();

  // Espera a que se reciba señal o se pase el tiempo
     while(!receptor.decode(&recibido) && !comprobarTiempo(esperaMax)) {}

  //Si no ha pasado el tiempo, se ha recibido una señal
  if(!comprobarTiempo(esperaMax)) {
    Serial.print("Código recibido: " + recibido.value);
    Serial.println(". El código se guardará como 'ON/OFF'.");

    codigoOn = recibido.value;
  }
  else{
    timeout = true;
    if(!errorReproducido) {
      errorReproducido = true;

      //Sonido descendente
```

```
    ledcWriteTone(buzzerChannel, mi4);
    delay(150);
    ledcWriteTone(buzzerChannel, la3);
    delay(250);
    ledcWriteTone(buzzerChannel, 0);
  }
}

// Si no se ha acabado el tiempo en la lectura anterior,
//realiza la siguiente lectura
if(!timeout) {
//Pitido
  ledcWriteTone(buzzerChannel, mi4);
  delay(250);
  ledcWriteTone(buzzerChannel, 0);

guardarTiempo();

// Espera a que se reciba señal o se pase el tiempo
   while(!receptor.decode(&recibido) &&   !comprobarTiempo(esperaMax)) {}

//Si no ha pasado el tiempo, se ha recibido una señal
if(!comprobarTiempo(esperaMax)) {
  Serial.print("Código recibido: " + recibido.value);
  Serial.println(". El código se guardará como 'MAS'.");

  codigoMas = recibido.value;
}
else {
  timeout = true;
  if(!errorReproducido) {
    errorReproducido = true;

    //Sonido descendente
```

```
      ledcWriteTone(buzzerChannel, mi4);
      delay(150);
      ledcWriteTone(buzzerChannel, la3);
      delay(250);
      ledcWriteTone(buzzerChannel, 0);
    }
  }
}

  // Si no se ha acabado el tiempo en la lectura anterior,
  //realiza la siguiente lectura
  if(!timeout) {
  //Pitido
    ledcWriteTone(buzzerChannel, mi4);
    delay(250);
    ledcWriteTone(buzzerChannel, 0);

    guardarTiempo();

  // Espera a que se reciba señal o se pase el tiempo
     while(!receptor.decode(&recibido) && !comprobarTiempo(esperaMax)) {}

  //Si no ha pasado el tiempo, se ha recibido una señal
  if(!comprobarTiempo(esperaMax)) {

    Serial.print("Código recibido: " + recibido.value);
    Serial.println(". El código se guardará como 'MENOS'.");

    codigoMenos = recibido.value;
  }
  else{
    timeout = true;
    if(!errorReproducido) {
      errorReproducido = true;
```

```
      //Sonido descendente
      ledcWriteTone(buzzerChannel, mi4);
      delay(150);
      ledcWriteTone(buzzerChannel, la3);
      delay(250);
      ledcWriteTone(buzzerChannel, 0);
    }
  }
}

  // Si no se ha acabado el tiempo en la lectura anterior,
  //realiza la siguiente lectura
  if(!timeout) {
  //Pitido
    ledcWriteTone(buzzerChannel, mi4);
    delay(250);
    ledcWriteTone(buzzerChannel, 0);
    guardarTiempo();

  // Espera a que se reciba señal o se pase el tiempo
   while(!receptor.decode(&recibido) &&    !comprobarTiempo(esperaMax)) {}

  //Si no ha pasado el tiempo, se ha recibido una señal
  if(!comprobarTiempo(esperaMax)){
    Serial.print("Código recibido: " + recibido.value);
    Serial.println(". El código se guardará como 'SWING'.");
     codigoSwing = recibido.value;
  }
  else{
    timeout = true;
    if(!errorReproducido) {
      errorReproducido = true;
      ledcWriteTone(buzzerChannel, mi4);
```

```
        delay(150);
        ledcWriteTone(buzzerChannel, la3);
        delay(250);
        ledcWriteTone(buzzerChannel, 0);
      }
    }
  }
    // Si no se ha acabado el tiempo en la lectura anterior,
    //realiza la siguiente lectura
    if(!timeout){
      EscribirCodigosEnMemoria();

      //3 pitidos indican que se ha configurado correctamente
      ledcWriteTone(buzzerChannel, mi4);
      delay(150);
       ledcWriteTone(buzzerChannel, 0);
      delay(50);
      ledcWriteTone(buzzerChannel, mi4);
      delay(150);
      ledcWriteTone(buzzerChannel, 0);
      delay(50);
      ledcWriteTone(buzzerChannel, mi4);
      delay(150);
      ledcWriteTone(buzzerChannel, 0);
    }
  }
}
//---------------------------------------------------------------
//Fin de Xana_ServerTemperatura.cpp
```

Anexo 28

-

Tabla de resistencias normalizadas

Los valores de la siguiente tabla representan la resistividad de las resistencias normalizadas de la serie E12 (aquellas cuya tolerancia es de un ± 5%):

x 1	x 10	x 100	x 1.000 (K)	x 10.000 (10K)	x 100.000 (100K)	x 1.000.000 (M)
1 Ω	10 Ω	100 Ω	1 KΩ	10 KΩ	100 KΩ	1 M Ω
1,2 Ω	12 Ω	120 Ω	1K2 Ω	12 KΩ	120 KΩ	1M2 Ω
1,5 Ω	15 Ω	150 Ω	1K5 Ω	15 KΩ	150 KΩ	1M5 Ω
1,8 Ω	18 Ω	180 Ω	1K8 Ω	18 KΩ	180 KΩ	1M8 Ω
2,2 Ω	22 Ω	220 Ω	2K2 Ω	22 KΩ	220 KΩ	2M2 Ω
2,7 Ω	27 Ω	270 Ω	2K7 Ω	27 KΩ	270 KΩ	2M7 Ω
3,3 Ω	33 Ω	330 Ω	3K3 Ω	33 KΩ	330 KΩ	3M3 Ω
3,9 Ω	39 Ω	390 Ω	3K9 Ω	39 KΩ	390 KΩ	3M9 Ω
4,7 Ω	47 Ω	470 Ω	4K7 Ω	47 KΩ	470 KΩ	4M7 Ω
5,6 Ω	56 Ω	560 Ω	5K6 Ω	56 KΩ	560 KΩ	5M6 Ω
6,8 Ω	68 Ω	680 Ω	6K8 Ω	68 KΩ	680 KΩ	6M8 Ω
8,2 Ω	82 Ω	820 Ω	8K2 Ω	82 KΩ	820 KΩ	8M2 Ω
						10M Ω

Anexo 29

-

Organización por carpetas del proyecto

El presente proyecto consta de seis microcontroladores, por lo que se han creado seis proyectos distintos en Visual Studio Code. Éstos son: uno para el código del Arduino nano, que recibirá las instrucciones del VRM; otro para el ESP32 del controlador principal y los otros cuatro para cada uno de los actuadores remotos: el controlador de luces, el de enchufes, el de temperatura y el del motor de la persiana.

Cada uno de los proyectos está compuesto por las siguientes carpetas:

- **.pio**: en esta carpeta se guardan los archivos de compilación de la extensión Platformio. Es una carpeta privada del compilador y no debe manipularse.
- **.vscode**: en esta carpeta guarda el IDE su configuración, información sobre las extensiones, etc. Es una carpeta privada del entorno de desarollo y no debe manipularse.
- **include**: en esta carpeta se han incluido las librerías generales que se ingluyen en las cabeceras del programa principal.
- **lib**: en esta carpeta se incluyen todas las demás librerías privadas que necesitan las generales para funcionar.
- **src**: en esta carpeta se incluye el código principal, aquel archivo *.cpp* formado por una función de inicialización (setup) y un bucle que se repetirá constantemente a lo largo del funcionamiento del microcontrolador.
- **test**: esta carpeta es para realizar pruebas de funcionamiento y variaciones durante el desarrollo del código. Al finalizar el proyecto, deberán haberse eliminado o integrado a *include*, *lib* o *src* para dejar esta carpeta vacía.

Además, en la carpeta de cada proyecto, hay una serie de archivos como *.gitignore* o *VS_Workspace* con funciones organizativas ajenas al proyecto en sí. Esta organización por carpetas es meramente organizativa y no supone una parte esencial para el funcionamiento del proyecto sino para su desarrollo.

La siguiente tabla muestra los archivos de código incluidos en los anexos anteriores que contiene cada carpeta de cada proyecto:

Carpeta / *Proyecto*	**include**	**lib**	**src**	**test**
Arduino nano	*Instrucciones.h, XanaVRM.h*	*alarma.h, menuAlarma.h, enchufes.h, menuEnchufes.h, luz.h, meuLuz.h, musica.h, menuMusica.h, persiana.h, menuPersiana.h, SD (carpeta de librerías), temperatura.h, menuTemperatura.h, TMRpcm-master (carpeta de librerías), VoiceRecognitionV3 -master (carpeta de librerías)*	*XanaVRM.cpp*	-
ESP32 - Cliente	*XanaVRM.h, Xana_ClientLib.h, XanaWifi.h*	*SD (carpeta de librerías), temperatura.h, menuTemperatura.h, TMRpcm-master (carpeta de librerías)*	*XanaClient .cpp*	-
ESP32 - Servidor - Luces -	*XanaWifi.h, Xana_ServerLuzLib.h*	*Arduino-IRremote (carpeta de librerías)*	*Xana_ ServerLuz.cpp*	-
ESP32 - Servidor - Enchufes -	*XanaWifi.h, Xana_ ServerEnchufeLib.h*	-	*Xana_ ServerEnchufe Lib.cpp*	-

ESP32 - Servidor - Persiana -	*XanaWifi.h, Xana_Server Persiana.h*	-	*Xana_ Server Persiana.cpp*	-
ESP32 - Servidor - Temperatura -	*XanaWifi.h, Xana_ServerLuz Temperatura.h*	*Arduino-IRremote (carpeta de librerías)*	*Xana_Server Temperatura .cpp*	-

Anexo 30

-

Enlaces de compra y referencias de los componentes

Los componentes se muestran en el mismo orden que siguen en el *apartado 1* del *Documento 3 - presupuesto:*

- Regulador de voltaje LM11173V3: Referencia 926-LM1117IMP3.3NOPB en Mouser
- Módulo ESP-WROOM-32: Referencia 356-ESP32WRM32E132PH en Mouser
- Placa de desarrollo de prototipos Arduino Nano: https://es.aliexpress.com/item/4001261534060.html?spm=a2g0o.productlist.0.0.3ec27072TVbeYc&algo_pvid=9a4dcff4-8f03-444b-91ac-854519336e72&algo_expid=9a4dcff4-8f03-444b-91ac-854519336e72-15&btsid=0b0a050b15989835591554428e6d2d&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Módulo de reconocimiento de voz V3,1 de elechouse: https://es.aliexpress.com/item/4001082972028.html?spm=a2g0o.productlist.0.0.644b221a3a0Uj3&algo_pvid=53fefd0e-6bcc-4aa0-8f4d-dfe61f1058bf&algo_expid=53fefd0e-6bcc-4aa0-8f4d-dfe61f1058bf-12&btsid=0b0a0ad815989836334138570e177c&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Amplificador operacional de audio TL081: Referencia 511-TL081IDT en Mouser
- Amplificador operacional de audio de potencia LM1875: Referencia 926-LM1875T/LF02 en Mouser
- Amplificador operacional rail-to-rail TLV170: Referencia 595-TLV170IDBVR en Mouser
- Resistencias 1/4W: Serie RMCF1206JT en DigiKey
- Potenciómetro de 100 K: Referencia 581-601040 en Mouser

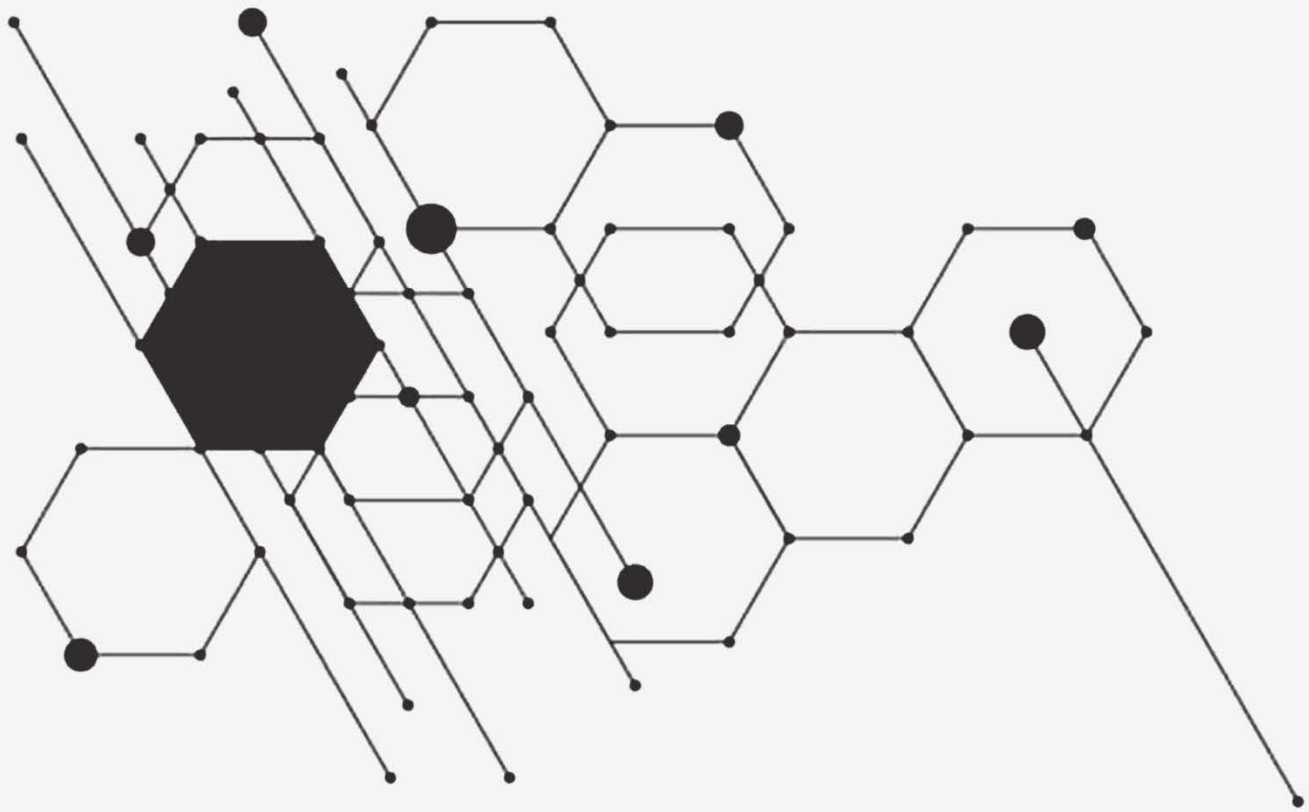
- Fotoresistencia LDR de 1M con 0,1 lux y 0,1 Ohms con 10.000 lux: https://es.aliexpress.com/item/4000055759348.html?spm=a2g0o.detail.1000013.4.76fc71f4kS7pBw&gps-id=pcDetailBottomMoreThisSeller&scm=1007.13339.169870.0&scm_id=1007.13339.169870.0&scm-url=1007.13339.169870.0&pvid=f9feb17c-b861-45cb-b538-26539161535b&_t=gps-id:pcDetailBottomMoreThisSeller,scm-url:1007.13339.169870.0,pvid:f9feb17c-b861-45cb-b538-26539161535b,tpp_buckets:668%230%23131923%2314_668%23808%234094%23198_668%23888%233325%236_668%232846%238113%23610_668%232717%237566%23801_668%231000022185%231000066056%230_668%233468%2315617%23819
- Condensadores 10 uF: Referencia 1276-2893-1-ND en DigiKey
- Cobndensadores 0.1 uF: Referencia 1276-1004-1-ND en DigiKey
- Transistor NPN BC847: https://es.aliexpress.com/item/33048376022.html?spm=a2g0o.productlist.0.0.62f75f93F62P74&s=p&ad_pvid=202009011114598572253023524960000426214_2&algo_pvid=e1deb1e6-b79a-4c51-bec1-8d8c146840fe&algo_expid=e1deb1e6-b79a-4c51-bec1-8d8c146840fe-1&btsid=0b0a01f815989840995703411e0482&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Fototransistor receptor de IR TSOP4038: https://es.aliexpress.com/item/32803615826.html?spm=a2g0o.productlist.0.0.51c2ee2eUoKgwN&algo_pvid=d4ebe415-7fba-41d3-a6fa-a4b08689205c&algo_expid=d4ebe415-7fba-41d3-a6fa-a4b08689205c-0&btsid=0b0a0ac215989840328582268e65aa&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Diodo LED rojo, Vf = 1,8 V: Referencia 1080-1597-3-ND en DigiKey
- Diodo LED IR VSMY2943SL: https://es.aliexpress.com/item/4000838376382.html?spm=a2g0o.productlist.0.0.56de48a6pm5CZe&s=p&ad_pvid=202009011116181212570628994354000043157 9_3&algo_pvid=49542212-65d7-4ad0-89ab-e3eb1bbf9042&algo_expid=49542212-65d7-4ad0-89ab-e3eb1bbf9042-2&btsid=0b0a1879159898417870223 12ec531&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Diodo de protección de 5V VESD05A1: Referencia 78-VESD05A1-02V-G308 en mouser
- Relé de potencia de 10 A tipo A con bobina de 5 V G5LE-1A DC5: Referencia 653-G5LE-1ADC5 en Mouser
- Relé de potencia de doble salida tipo C con bobina de 9V JW2HN-DC9V: Referencia 769-JW2HN-DC9V en Mouser
- microSD socket: https://es.aliexpress.com/item/10000240869668.html?spm=a2g0o.productlist.0.0.1ad8a742t6VScx&algo_pvid=8bb2fd6a-d58a-4c0b-98a4-1874e8cd37b9&algo_expid=8bb2fd6a-d58a-4c0b-98a4-1874e8cd37b9-

6&btsid=0b0a0ad815990025282505803e168c&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_

- 5,5mm DC Jack hembra: https://es.aliexpress.com/item/4001264341107.html?spm=a2g0o.productlist.0.0.5b243429HoRuwT&algo_pvid=d0dc8ee7-9479-470a-ad4c-d694d899cb7b&algo_expid=d0dc8ee7-9479-470a-ad4c-d694d899cb7b-0&btsid=0b0a187b15990026055061604ea1a3&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- microUSB tipo B hembra: https://es.aliexpress.com/item/4000449163807.html?spm=a2g0o.productlist.0.0.1ad8a742t6VScx&s=p&ad_pvid=202009011619421212241882170315000072503 2_4&algo_pvid=1ad1131f-beec-4944-a5c0-f2088350f208&algo_expid=1ad1131f-beec-4944-a5c0-f2088350f208-3&btsid=0b0a0ad815990023826866646e16c9&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Conector de 3 pines macho 2,54 mm: https://es.aliexpress.com/item/33028739982.html?spm=a2g0o.productlist.0.0.55656e10zPk2kr&algo_pvid=01d1395d-1bfc-4b59-bd16-acb56968fa3c&algo_expid=01d1395d-1bfc-4b59-bd16-acb56968fa3c-34&btsid=0b0a182b15990030084755714ed751&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Conector de 4 pines macho 2,54 mm: https://es.aliexpress.com/item/33028739982.html?spm=a2g0o.productlist.0.0.55656e10zPk2kr&algo_pvid=01d1395d-1bfc-4b59-bd16-acb56968fa3c&algo_expid=01d1395d-1bfc-4b59-bd16-acb56968fa3c-34&btsid=0b0a182b15990030084755714ed751&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Terminal de 2 entradas de 5,08 mm para PCB, máx. 300 V 20 A.: https://es.aliexpress.com/item/32865614871.html?spm=a2g0o.productlist.0.0.4496146fKwpuNH&algo_pvid=f81bf2fd-4eda-4b3b-af2a-15a2daa1bba6&algo_expid=f81bf2fd-4eda-4b3b-af2a-15a2daa1bba6-4&btsid=0b0a187915990028981347344ec4f2&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Zumbador piezoeléctrico pasivo: https://es.aliexpress.com/item/4000203106698.html?src=google&albch=shopping&acnt=439-079-4345&isdl=y&slnk=&plac=&mtctp=&albbt=Gploogle_7_shopping&aff_atform=google&aff_short_key=UneMJZVf&gclsrc=aw.ds&&albagn=888888&&ds_e_adid=438858099979&ds_e_matchtype=&ds_e_device=c&ds_e_network=u&ds_e_product_group_id=743612850714&ds_e_product_id=es4000203106698&ds_e_product_merchant_id=107330331&ds_e_product_country=ES&ds_e_product_language=es&ds_e_product_channel=online&ds_e_product_store_id=&ds_url_v=2&ds_dest_url=https://es.aliexpress.com/item/4000203106698.html?&albcp=10191226958&albag=10

2259630456&gclid=CjwKCAjwnK36BRBVEiwAsMT8WEjRTHJ2mq7WW34ahwsDjjKD-F7FUKf96Hekl0X0pemEcNpaGEOc2xoC1M0QAvD_BwE

- Pulsador: https://www.e-merchan.com/pup65r.html
- Altavoz de 8 Ohm, 15 W de rango completo: https://es.aliexpress.com/item/4000159320927.html?spm=a2g0o.productlist.0.0.4e8a5f0fBOQjBT&algo_pvid=9eacec6e-2192-4919-a2d5-d6a808a39385&algo_expid=9eacec6e-2192-4919-a2d5-d6a808a39385-1&btsid=0b0a119a15990020008217969ec212&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Micrófono MEMS ICS40180 de 3,3V y salida analógica: Referencia 410-ICS-40180 en Mouser
- Tarjeta microSD 8 GB: https://www.ebay.es/itm/Tarjeta-Memoria-Micro-SD-SDHC-8GB-Card-Envio-desde-ESPANA-v52/352208602272?hash=item520144c4a0:g:Ng8AAOxy4dNSrx~9
- Transformador de 230 VAC a 12 VDC de 2 A con salida por Jack 5,5 mm: https://es.aliexpress.com/item/32953326560.html?spm=a2g0o.productlist.0.0.14647318KaSWMT&algo_pvid=54795ff9-01ea-4699-8fb4-5be3a9e5e65c&algo_expid=54795ff9-01ea-4699-8fb4-5be3a9e5e65c-8&btsid=0b0a0ae215989852075022336eb25a&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
- Cable USB - microUSB tipo B: https://www.ebay.es/itm/Cable-Cargador-Micro-USB-2-0-de-carga-y-datos-para-Movil-Elija-color-y-tamano/183412340707?hash=item2ab43a37e3:g:93QAAOSwOgRbjA~k
- Placa de prototipos protoboard de 840 puntos de conexión: https://es.farnell.com/multicomp/mc01000/breadboard-840-pin/dp/2503747?gclid=Cj0KCQjwhb36BRCfARIsAKcXh6FIU01SL53m_fkodx4UEzezTkDHyI4XuYi2aCyxh2CaTzU6JQ2jY4QaAmvTEALw_wcB&gross_price=true&mckv=s_dc|pcrid|449499896844|plid||kword||match||slid||product|2503747|pgrid|105470367580|ptaid|aud-123476735829:pla-926717601599|&CMP=KNC-GES-GEN-SHOPPING_Tools-Production-supplies-High-Bids-24-JUL-20
- Placa de prototipos protoboard de 400 puntos de conexión: https://es.farnell.com/pro-signal/psg-bb-400/breadboard-400-pin-white/dp/2503765?gclid=Cj0KCQjwhb36BRCfARIsAKcXh6GDRapKVGDKkBL7pBgljV9ENfu4jYjwMCh5TT3X_-lZvoaEaAjPjRwaAiIxEALw_wcB&gross_price=true&mckv=s_dc|pcrid|449543969278|plid||kword||match||slid||product|2503765|pgrid|105470367780|ptaid|aud-123476735829:pla-991323251757|&CMP=KNC-GES-GEN-SHOPPING_Tools-Production-supplies-Low-Bids-24-JUL-20
- 5x PCB de 2 capas de 200 mm x 120 mm x 1,6 mm con el diseño de los 5 módulos de Xana y plantilla para pasta de soldar: www.jlcpcb.com



Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA